Real-time Motion Retargeting to Highly Varied User-Created Morphologies

Chris Hecker* Bernd Raabe†
Ryan W. Enslow† John DeWeese†
Maxis/Electronic Arts

Jordan Maynard‡
Trion World Network

Kees van Prooijen‡
Total Immersion Software

Abstract

Character animation in video games—whether manually keyframed or motion captured—has traditionally relied on codifying skeletons early in a game's development, and creating animations rigidly tied to these fixed skeleton morphologies. This paper introduces a novel system for animating characters whose morphologies are unknown at the time the animation is created. Our authoring tool allows animators to describe motion using familiar posing and key-framing methods. The system records the data in a morphology-independent form, preserving both the animation's structural relationships and its stylistic information. At runtime, the generalized data are applied to specific characters to yield pose goals that are supplied to a robust and efficient inverse kinematics solver. This system allows us to animate characters with highly varying skeleton morphologies that did not exist when the animation was authored, and, indeed, may be radically different than anything the original animator envisioned.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: character animation, motion retargeting, user generated content, inverse kinematics, procedural animation, games

1 Introduction

User generated content is an increasingly popular way to give players meaningful creative input into video games [Edery et al. 2007], and enabling players to create the fundamental art assets—the characters, vehicles, buildings, and planets—is one of the primary design goals of the game *Spore* [Wright 2005]. Allowing players to create art assets after the game has shipped forces the code to be flexible and robust enough to deal with the new, never-before-seen content. Characters present a particularly challenging problem, requiring the synthesis of player created morphologies with authored animations for in-game actions (e.g. picking fruit from a tree, throwing a spear, or dancing a jig). Preserving the style and quality of professionally authored animations as they are retargeted at runtime to extremely varied player-created morphologies required a new approach to character animation.

Our solution preserves the traditional animation workflow found in tools such as Maya [Aut 2008], while enabling keyframe animators to create automatically retargetable, yet stylized, motion. We bring the animators directly into the retargeting loop and have them specify the semantics of various aspects of the animation during authoring. We use this semantic data to record the motion in a *generalized* form, and at runtime we *specialize* it onto the different character morphologies to generate the individual retargeted motions. The success of this method shows the power of explicitly specified semantic information in solving the motion retargeting problem.

The remainder of this section presents a high level summary of our methods for authoring and playing animations on unknown morphologies, a list of our contributions, and an overview of the terminology used to describe the animated characters. Section 2 discusses related work. Sections 3 and 4 discuss in detail the authoring and playback of the character animations. Finally, Section 5 describes the results both qualitatively and quantitatively, as well as how the system is tested in production and its limitations, and discusses future work and the applicability of the system to other games and animation challenges.

1.1 Overview

Authoring In our OpenGL-based authoring tool *Spasm*, the animators pose the characters, set keys, edit curves, and preview animations, much like in traditional game animation (Fig. 1). Animations contain an arbitrary number of *channels*, and for each channel, the animator tells *Spasm* which parts of the character to select and which aspects of the motion are important. This semantic information is used to create an invertible function G that is used to move from the *specialized* pose on a specific character to a *generalized* space that is character-independent, and back again (using $S = G^{-1}$). The semantic specification phases for both selection and movement are the key differences—from the animator's perspective—between authoring an animation in *Spasm* versus a traditional character animation tool.

Playback At runtime, the *generalized* animation curves are *specialized* onto the character using S and a combinatoric technique—called *variants*—for controlling the playback. The resulting pose goals preserve the overall motion and stylistic details of the authored animation. Stylized locomotion is synthesized for the player-created leg morphology and layered onto the goals. The goals are fed into an inverse kinematics (IK) solver tuned to handle

*checker@d6.com
†{braabe,renslow,jdeweese}@ea.com
‡{jordan.maynard,keesvp}@gmail.com

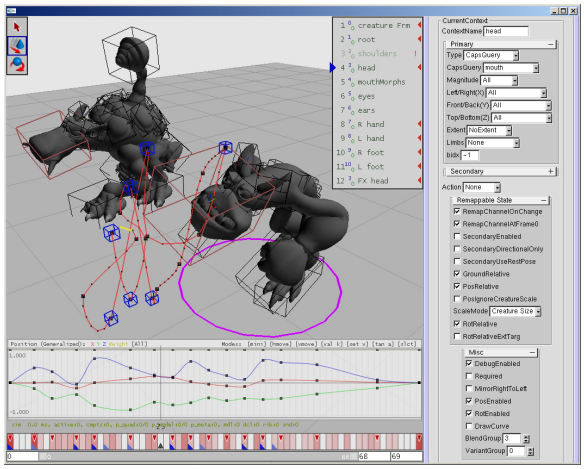


Figure 1: Spasm *screenshot with two preview characters.*

conflicting objectives while attaining natural solution poses. Finally, passive secondary animation is added for aesthetic appeal.

Contributions The semantic specification concept and details for both selection and movement, the decomposition of the problem into generalized and specialized spaces, and the various ways of formulating G based on the animator-controlled semantics are the main contributions of this work to the authoring phase of the retargeting problem. The contributions to the playback phase include the variant technique for exposing the combinatoric variety of animation playback options, the locomotion synthesizer for generating stylized gaits on arbitrary leg morphologies, the robust, high performance IK solver with its emphasis on ad hoc tunability and preconditioners for attaining natural character poses, and the passive secondary animation subsystem.

1.2 Character Terminology

The player creates new characters in *Spore* by manipulating a malleable clay-like torso containing the spine, and attaching limbs and deformable anatomical body parts chosen from a palette [Willmott et al. 2007], including various mouths, eyes, graspers, feet, spikes, armor, etc. The final character is composed of *bodies* (which include the anatomical parts, spine vertebrae, and limbs), meshes, and textures. Characters usually contain between 20 and 80 bodies. Figure 2 shows the relationship between a character's mesh (the top row) and its bodies (the bottom row).

The bodies contain the position and rotation transforms, bounding boxes, and hierarchical parent-child information traditionally associated with *bones* in other animation systems. The bodies also contain tags—called *capabilities* or *caps*—describing the body's semantics to the animation system. For example, the body associated with a hand mesh has the *grasper* capability. Bodies can have scalar *deform curves* that control low level mesh animations on the body. Examples include opening and closing hands, mouths, or eyes, ears drooping, toes bending, etc. The deforms are standardized across the various types of anatomical body meshes (e.g. all mouth bodies respond to the open/closed deform). This allows the deforms to be opaque to the animation system, so we ignore them for the remainder of this paper.

The character bodies form a directed acyclic graph, with a serial chain of spine bodies at the root of the tree.[1] The *root body* of the tree is a unique spine body chosen by a heuristic based on the maximum number of incident leg limbs and the position of the body.

The position and orientation of the bodies at the time the player creates the character is called the *rest pose*. Traditionally, the rest pose of a game character is a known standard pose, but we cannot count on any specific configuration for the rest pose. We simply assume the player created the character in a "reasonable" rest configuration,[2] not hyper-extended or curled into a ball, etc.

2 Related Work

Traditional game character animation blends static animation data onto fixed skeletons, with occasional IK for foot placement or head lookat [Mizuguchi et al. 2001]. When the morphology of the character is unknown, these methods are clearly inapplicable.

Motion retargeting is not a new problem, but most existing work is offline, not efficient enough for real-time games, or requires a large database of example motion, and no previous work considers the range of morphologies we do. Gleicher [1998] presents an offline method using spacetime constraints on example motion. The paper briefly considers different morphologies, but requires similar sizes, and only retargets to characters with fewer degrees of freedom (DOF). Many other offline methods exist with similar limitations [Lee and Shin 1999; Popović and Witkin 1999]. Choi and Ko [2000] and Shin et al. [2001] develop online methods using IK, but both only consider identical skeletal topologies. Kulpa et al. [2005] support different numbers of bones along limbs by using IK in real time, but only on humanoid topologies.

Many of these efforts try to automatically derive the semantic structure of the motion and its constraints from example data, whereas we have the animators *a priori* describe the important factors to our system. In this sense, our system is *illustrative* instead of *example based*. In fact, abstracting the motion, for both the constraints and the style, is a common thread throughout the related work. For example, while not focused on motion retargeting, both Chi et al. [2000] and Neff and Fiume [2005] form parameterized semantic models for animation style on fixed human skeletons. By contrast, our system gives the animator direct control over the model for the motion constraints for each animation, and then records the stylized animation curves within this model.

IK for character animation is also a very well studied problem, both synthetic [Girard and Maciejewski 1985; Welman 1993; Zhao and Badler 1994] and example-based [Grochow et al. 2004]. Unlike most IK solvers for character animation, which use explicit representations of the angular DOF, our solver draws from the molecular dynamics (MD) literature and works with cartesian DOF and length constraints [Kastenmeier and Vesely 1996; Tang et al. 1999]. Following the SHAKE algorithm [Ryckaert et al. 1977], most of these MD-inspired IK techniques use a linearized error fixup during constraint iteration, but we use a nonlinear length correction, similar to Jakobsen [2001]. Our IK solver differs from previous work in its two-phase architecture and its explicit emphasis on ad hoc tunability and custom preconditioners, which we have found invaluable for finding natural poses under conflicting goals.

Synthetic locomotion has often focused on bipeds [Bruderlin and Calvert 1989; Bruderlin and Calvert 1996; Sun and Metaxas 2001].

[1] Early prototypes for the *Spore* character creator supported more general structures, including loops, but testing revealed the interface complexity of a general graph was actually a hindrance to player creativity.

[2] This assumption can obviously prove to be false, but in our testing, players create the characters in acceptable poses, and sometimes adjust the rest pose after seeing their character animate.

Girard and Maciejewski [1985] develop a system for arbitrary independent legs, but do not address stylistic variation or animator input. Procedural secondary animation, both physical [O'Brien et al. 2000] and fake [Barzel 1997; Barzel et al. 1996] has been discussed, but our system dynamically discovers the sub-trees to be simulated and integrates—but does not interfere—with the authored data.

3 Animation Authoring

Traditional keyframed animation workflow proceeds as follows:

1. **Select.** The animator chooses which parts of the character he or she will animate.
2. **Pose.** The animator moves the selected parts of the character to a new position and orientation.
3. **Key.** The animator records a key frame for the selected and posed parts of the character.
4. **Preview.** The animator plays back the animation, or a portion of it, to check the work, repeating the process based on the visual feedback.

Spasm preserves this basic workflow, enabling animators comfortable with traditional tools to create retargetable motion. We now discuss the modifications to each step in detail.

3.1 Selecting

In traditional character animation, the animator clicks the mouse on the character skeleton to select the bone or locator he or she wants to pose via forward or inverse kinematics. The selected object is usually referenced by a simple index or name in the animation. When the runtime skeleton is not known at author-time, this direct indexing is impossible, so we require the animator to *describe* which bodies are selected using a semantic query. After a channel's selection is specified, the animator can then click on any one of the selected bodies of a character in the viewport to *activate* it.

3.1.1 Contexts

In our system, the animator uses a channel's *context* to describe the bodies the channel poses. Contexts are constraint-based filters over the bodies of the character. The context selects zero or more bodies by intersecting the specified constraints, in a manner similar to e-mail and music filters [Moz 2008; App 2008]. We use the phrase *selected bodies* to refer to the results of the context query, and *active body* to refer to the unique selected body with which the animator is currently interacting in *Spasm*.

Type The animator builds up a *primary context query* to select bodies on the character by first specifying the *type* of the query. The type specifies the body capability (*grasper*, *mouth*, *spine*, *root*, etc.) to be selected. This query will select all the bodies on the character that have the specified cap, so if the query was *grasper* and the character has four graspers, four bodies will be selected.

Spatial Queries The animator can narrow the selection by specifying different *spatial* constraints for the selection, including **Front/Center/Back**, **Left/Center/Right**, and **Top/Center/Bottom**. Each of these spatial constraints can be relative to the character's bounds (the front or back halfspaces of the entire character's bounding box, with the side constraints being inclusive of the center zone), or relative to the *setspace* of bodies with the given capability. For example, if a character has four graspers, but all are in the front halfspace of the character, then the regular front constraint would choose all of the graspers, but a front setspace query would make a bounding box of the graspers, then choose the graspers in the front halfspace of those local bounds. The setspace option allows the animator to separate out bodies even when they are clumped relative to the character as a whole.

Extent Queries There is also an *extent* constraint, which can be **FrontMost**, **BackMost**, **RightMost**, etc. and will always select zero or one body, the most distant body in the given extent direction (ties are broken arbitrarily).

Limb Modifier Finally, there is a *limb modifier*, which performs limited skeletal hierarchy traversal. The limb modifier can be set to **SpineSegment**. When the unmodified query selects bodies, this modifier will "walk up" any parent limbs to the first bodies with the *spine* capability, effectively finding an approximation to the clavicle/shoulder (or hip/pelvis) bones for a given selection. This, for example, allows the animator to select and pose the appropriate clavicle(s) for another channel's selected graspers.

3.1.2 Discussion

The game code also constructs and evaluates context queries on the characters for use in gameplay AI reasoning, inventory, etc. Again, in traditional games the skeleton is known and the code can simply refer to explicit bone names or indices, but in *Spore* the code must describe the semantics of the selection just as the animators do.

We looked into creating context queries automatically by clicking on character bodies in *Spasm*, but we ruled this out as combinatorically infeasible. There are many different ways to select the same body with completely different context queries, so the benefits of an automatic interface were outweighed by the potential errors introduced trying to resolve the ambiguities. From a workflow standpoint, the animators set up their preferred channels once, and reuse them in multiple animations, so this has proven not to be a burden.

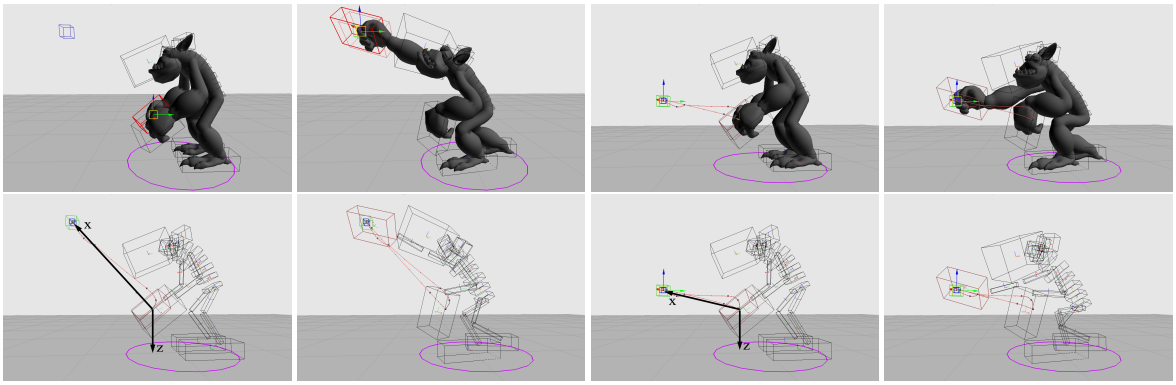
3.2 Posing

As with selection, the semantics of the motion—called the *movement mode*—is specified by the animator so the motion can be retargeted. The movement mode dictates how the pose data is interpreted for the channel by describing the important characteristics of the motion (e.g. whether the movement is relative to the ground, or depends on the size of the character or the length of the limb, etc.). At a low level, it is a specification for the coordinate frame in which movement is recorded, and it is used to construct the generalization function G and its inverse, the specialization function S. Once a body is selected and activated, the animator can pose the active body with Maya-style position and rotation manipulators. If multiple bodies are selected by the primary context query, all of the bodies will move as the animator manipulates the active body. The active body will follow the mouse manipulator, and the other selected bodies will move with the active body based on the movement mode of the channel as described below.

3.2.1 Generalization and Specialization

The position and rotation of a given body b_i on a specific character in character-relative Euclidean space (i.e. what is displayed in the *Spasm* viewport) is referred to as the *specialized* coordinates or pose of the body, $\mathbf{q}_{si}$. The function G takes the specialized pose to the body-independent *generalized* position and rotation coordinates, $\mathbf{q}_g$, by $\mathbf{q}_g = G(b_i, \mathbf{q}_{si}, m)$. Similarly, given $\mathbf{q}_g$ and a body b_i, the function S produces the $\mathbf{q}_{si}$ for that body, $\mathbf{q}_{si} = S(b_i, \mathbf{q}_g, m)$. The m parameter controls *mirroring* across the sagittal plane, and is discussed in Section 4.1.2.

During body manipulation and posing in *Spasm*, the active body's $\mathbf{q}_s$ are continually generalized to $\mathbf{q}_g$, and then the resulting $\mathbf{q}_g$ is



(a) Rest 1 (b) Reach 1 (c) Rest 2 (d) Reach 2

Figure 2: Secondary Relative. *Moving the green external target affects the reach pose and the red spline, but not the rest pose.*

specialized onto each of the channel's selected bodies b_i yielding $\mathbf{q}_{si}$. This process includes the active body, and because $G = S^{-1}$, the active body appears to the animator to be directly manipulated in Euclidean specialized space, but it is not treated as a special case internally. The active movement modes determine the specific definition of G, which in turn determines how the other non-active selected bodies move with the active body.

3.2.2 Movement Modes

The movement modes describe the semantics of the motion—what's important about it for expressing a desired intent. They are ad hoc, in the sense that additional movement modes are added on an as-needed basis when the animators are unable to express an intended movement so it *generalizes* to a large set of characters, but they are usually designed to work together and layer when possible. All movement modes have the character rest pose as a sort of "origin", such that if one character is at its rest pose for a given $\mathbf{q}_g$, all characters will be at their rest poses, regardless of the mode.

Identity The most basic movement mode is the identity, $G = S = \mathbf{1}$, also called *absolute mode*. In this mode, the $\mathbf{q}_s$ of the manipulated body are simply copied to the $\mathbf{q}_g$ by G, which are then copied to the $\mathbf{q}_{si}$ of the other bodies by S—all selected bodies have the same character-relative position and rotation. This mode is not very useful for animating positions, but it can be useful for rotations, allowing the animator to specify a uniform end orientation for selected bodies regardless of their rest orientation. Posing a grasper to hold a platter of food up would be appropriate use of absolute mode for rotation.

Rest Relative In contrast to absolute mode, in *rest relative* mode G computes the $\mathbf{q}_g$ as a delta from the rest pose of the body. The bodies move relative to their rest poses—if the manipulated body is moved 1 unit to the left of its rest position, all of the bodies move 1 unit to the left of their respective rest positions, and similarly for rotation. Both position and rotation can have rest relative mode set independently. A hand wave animation might use rest relative position mode to pose a grasper because the absolute position isn't important, just the relative waving motion, but absolute rotation mode so the grasper is pointed up regardless of its rest pose.

Scale Animators can specify a *scale mode* that affects how G and S compute position data. If no scale mode is set, the $\mathbf{q}_{si}$ position curves are the same scale on all characters. If **CreatureSize** mode is enabled, S performs a nonuniform scale to the $\mathbf{q}_{si}$, proportional to the bounding box of the character. Small characters get small curves; large characters get large curves. **LimbLength** mode causes S to scale the pose based on the *limb length* of the posed body. The limb length of a given body is the path length from the body to the nearest spine segment, forming an approximation to the workspace of the body. Specialization will result in large movements on long limbs and small movements on short limbs.

Ground Relative The *ground relative* mode distorts the coordinate frame of the movement such that the z-axis is vertical and scaled 0 to 1 from the rest pose to the ground. In generalized coordinates, $\mathbf{q}_{g_z} = 0.0$ specifies the rest pose height and $\mathbf{q}_{g_z} = 1.0$ specifies the ground height. This way, when $\mathbf{q}_{g_z} = 1.0$ is specialized onto different bodies, each will hit the ground, so an animation of picking up a rock or pounding the ground will generalize across characters with different grasper heights with the same timing.[3]

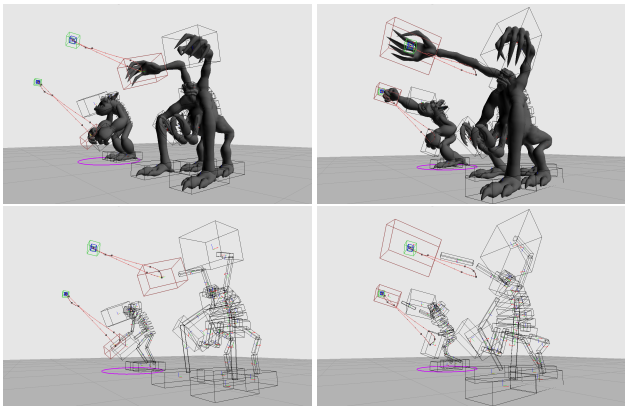
Secondary Relative Movement A *secondary context query* lets the animator specify another set of bodies to which the primary body movement is relative for expressing motions like "put the hand to the mouth" or "clap hands" across characters with different relative positions of hands, mouths, etc. The secondary context also allows an **ExternalTarget** query type, giving the game dynamic input into the specialization of the animations. For example, the code can set the target to a fruit location for a "pick fruit" animation or have two characters "shake hands" by setting the target of each character to the other character's grasper.

G distorts the motion frame so the x-axis is the vector between the rest position of the body and the target, with 0.0 at the rest position and 1.0 at the target.[4] This frame is updated during evaluation of G, so a pose that puts the body at the target will change as the target moves, even if the $\mathbf{q}_g$ for the pose is not changing. Figures 2(a) and (c) show the affect of moving the target while in the rest pose; the two character poses are identical because the rest pose is independent of G as mentioned above. Figures 2(b) and (d) show the poses for one $\mathbf{q}_g$ but with different target positions; *these two different specialized poses are generated from the same generalized pose*. In the lower wireframe images in Figure 2 you can see the channel's red animation curve being distorted as the target moves,

[3] We plan to add a *sagittal relative* mode soon that will do the same distortion for the sagittal plane, allowing animators to express hand claps and other animations that have bodies that cross from left to right with increased generalization quality.

[4] There is a modifier called **SecondaryDirectionalOnly** that does *not* rescale the frame's x-axis, so that the direction changes as the target moves but the scaling of the pose does not. This modifier allows the animators to express a punch movement in the direction of the target, without the punch changing shape as the target moves closer to the body.

while keeping the local style information intact. Figure 3 shows the example for two characters with very different morphologies.



(a) Rest (b) Reach

Figure 3: *Two different characters posed with the same* $\mathbf{q}_g$.

Lookat We also allow a *lookat controller* to be activated when secondary relative mode is enabled. This mode aims the rotation frame's forward vector towards the target object. The head body in Figure 2 has a lookat controller applied to it; notice the head following the target during the reach. The lookat mode defines a frame for recording relative rotation poses—it does not completely specify the rotation of the body—so a "nodding head" movement can be performed while looking at the [potentially moving] target. Soft joint limits are applied to the lookat controller to damp its influence as the target moves into the half space behind the selected body, allowing characters with multiple heads facing in different directions to use an appropriate head to follow a target.

3.2.3 Blending

Multiple channels can select the same bodies, so a priority and grouping system is in place to give the animator control over which channels contribute to a body's pose. The first channel to select a body takes ownership of the body[5] for its animator-specifiable *blend group*. Later channels can share ownership of the body if the animator assigns them a matching blend group. The resulting $\mathbf{q}_s$ for each channel are blended using traditional weighted position and rotation blends. Because the movement modes are set at author time and fixed for a given channel, the animators use blends between channels to affect a continuous change from one movement mode to another. For example, a character can grab a piece of fruit at the external target using a grasper, and then put the fruit in the character's mouth in a single fluid movement by blending between external target-relative and internal target-relative channels over time using varying weights on each channel.

3.3 Keying

The animation key frames record the generalized $\mathbf{q}_g$ for each enabled *key data curve* for a channel, and intra-key interpolation occurs on these $\mathbf{q}_g$. The key data curves include position, rotation, the various deform curves, weights on each curve, and miscellaneous discrete keyable information.[6] The animators have explicit Hermite spline control over the keyed $\mathbf{q}_g$ curves. The raw splines on the $\mathbf{q}_g$ can be manipulated, or the curves can be specialized by *Spasm* onto the given character and active body, and the splines can be manipulated in specialized space (which is then generalized back to the stored keys). Some animators prefer to manipulate the generalized curves directly (which control the actual data being interpolated, but can be nonintuitive), while other prefer the specialized curves (which appear to be traditional Euclidean curves, but are different for each selected body), so we provide both options.

Key Remapping When an animator changes the movement mode during authoring (e.g. to experiment with the different options to increase generalization of the animation, or to repurpose an animation for a different but similar movement), G for the channel changes. This invalidates any existing $\mathbf{q}_g$ keys saved with the old G, so we *remap* the existing keys such that the active body b's $\mathbf{q}_s$ do not change. We construct S_{old} from the pre-modification movement mode, and G_{new} from the post-modification movement mode, and then remap each key's $\mathbf{q}_g$ by $\mathbf{q}_{g_{new}} = G_{new}(b, S_{old}(b, \mathbf{q}_{g_{old}}, false), false)$. This leaves b's $\mathbf{q}_s$ curve and Euclidean motion exactly the same, while the $\mathbf{q}_g$ are converted to the new movement mode. By keeping the active body's specialized curve unchanged, we give the animator direct control over what aspects of the old versus new movement modes he or she would like to preserve. For example, by activating a body at the end of a long limb, and then switching **LimbLength** scaling off, the large movement will be preserved on the active body while the corresponding old small movements on short limbs will become large movements. By contrast, activating a body on a short limb and then removing **LimbLength** scaling would perform the opposite remapping: all of the curves would become small.

3.4 Preview

Spasm allows the animator to load multiple characters simultaneously, all bound to the same animation data as shown in Figures 1 and 3. The animator can edit the animation data on any loaded character at any time and instantly see the changes reflected on the other characters, because the $\mathbf{q}_s$ for all the characters are derived from the shared $\mathbf{q}_g$ (Section 3.2.1). This enables the animator to preview and edit the animation on many different morphologies simultaneously, and directly compare how changes to one character's pose generalize to other characters. This allows the animator interactive control over the quality of the motion retargeting across a set of example characters and greatly tightens the authoring feedback loop. Preview in *Spasm* uses the same code as in-game playback to support WYSIWYG[7] authoring.

4 Animation Playback

At runtime, the *Spore* animation system presents a typical animation Application Programming Interface [Muratori et al. 2008; Desmecht and Dachary 2006] to the game, including playback, asynchronous animation loading, caching, queing, layering animations with weights, etc. The first time an animation is played on a given character, a special *bind phase* occurs. Then, every frame, for any animations playing on a character, the $\mathbf{q}_s$ for the channel's selected bodies are computed using S and blended together. The resulting pose goals are composed with any synthesized locomotion and fed to the IK solver, and then secondary animation is applied.

4.1 Binding

When an animation is played on a character for the first time, the animation and the character go through a *bind phase* to determine *if*

[5] More precisely, ownership is allocated separately to each animated degree of freedom, so ownership of just the rotation is possible.

[6] Examples include per-key visual, sound, and data events.

[7] What You See Is What You Get

and *how* the playback will occur, using techniques called *branching* and *variants*, respectively.

4.1.1 Branching

The bind phase begins by evaluating a series of *branch predicates* on the character. These three-valued predicates ($true$, $false$, $ignore$) constrain the set of characters to which the animation will bind, allowing the animators to split up the space of character morphologies. For example, one branch predicate is **UprightSpine**, which evaluates a heuristic for whether the character's spine is predominantly prone or upright. Other predicates include **HasGraspers** and **HasFeet**. If the animators cannot create a single sufficiently general animation—and we cannot modify the system to generalize for the action—they can create multiple distinct "branched" animations for the original movement. For example, animations for tool usage are branched on whether the character has graspers to manipulate objects, or whether the character uses its mouth because it has no graspers. Branching animations is a solution of last resort, because it increases content creation costs by forcing multiple animations to be authored and tested for an action.

4.1.2 Variants and Mirroring

As described in Section 3.2.3, blend group ownership and priority information is computed during the bind phase. The system then computes "how many different ways the animation can play on the character". Each "way" is called a *variant*. The game code choses which variant is played at runtime as described below. Variants are best described using examples. Figure 4 depicts an abstract character with five bodies, labeled A, B, 1, 2, and 3.

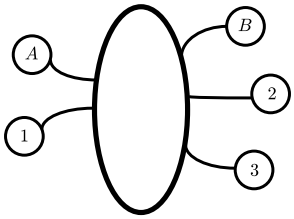
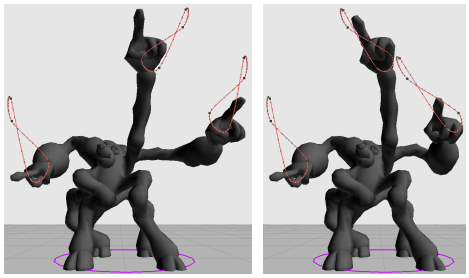


Figure 4: *Abstracted character and bodies, viewed from the front.*

Single Variant Group If we want to create an animation for grabbing a fruit from a tree with a grasper, the character only needs a single grasper to grab the fruit, but there is no way at author time to know which grasper should be used—the position of the fruit relative to the character and the number and distribution of the character's graspers are unknown. We allow the animator to mark a channel so its curves are played on only *one* of its selected bodies at a time. A distinct *variant* for the animation is generated for each grasper selected by the marked channel. The game can play the variant for the most appropriate grasper dynamically based on arbitrary game-specific criteria: for example, which grasper is closest to the fruit, which one is not already holding another object, etc. Concretely, if all the bodies in Figure 4 are graspers, five variants will be generated, one for each body: A, B, 1, 2, and 3.

Sagittal Mirroring An animation is created with arbitrary chirality, and the system automatically generates its *sagittal mirror* during variant generation. The animator can control how S specializes curves when mirroring. Figure 5 shows the two mirroring possibilities on an asymmetric character for the same $\mathbf{q}_g$. Figure 5(a) shows the $\mathbf{q}_{si}$ with m (from Section 3.2.1) set to false on the left side, and true on the right side, while Figure 5(b) shows m set to false for all bodies. Notice how the positions and orientations of the graspers in Figure 5(a) are mirrored appropriately across the sagittal plane.



(a) Mirrored (b) Not mirrored

Figure 5: *An asymmetric character with and without sagittal plane mirroring on a grasper channel. The grasper curve to the left in each picture is the unmirrored curve.*

Many Variant Groups A more complex example involves passing a fruit from one grasper to another using two grasper channels. Variants allow the game code to arbitrarily choose both the giving and receiving graspers. Each combination of graspers participating in the handoff generates a single variant. If we assume the bodies labeled A and B are the only two graspers on the character, two variants, AB and BA, are generated. The first hands the fruit from A to B, and the second from B to A using sagittal mirroring. The variants do not have to have the same cap: if A and B are mouths, and 1, 2, and 3 are graspers, an animation to put a fruit in a mouth could generate $1A$, $1B$, $2A$, $2B$, $3A$, and $3B$. The animator can specify the spatial relationship of the varying channels, so if 1, 2, and 3 are graspers, and the two channels are the same-side-constrained, two variants will be generated: 23 and 32. If the channels are opposite-side-constrained, the system will generate four variants: 12, 13, 21, and 31. No spatial constraints will generate all six permutations: 12, 13, 21, 23, 31, 32.

Variant Product This modified cartesian product is called the *variant product* of the bodies. There is no limitation on the channel arity of the variant product or on the types of the channels involved. The animator can also group channels so they vary together and do not generate additional variants. For example, an animation might have a grasper channel and a grasper-shoulder channel using a *limb modifier*. Setting these two channels to the same variant group will cause the shoulder to co-vary with the appropriate grasper.

4.2 Gaits

The characters undergo legged locomotion across uneven terrain and along curved paths with discontinuous input velocities. The animators can control the feet for non-locomoting animations, but we use a *gait system* to generate the locomotion to ensure we meet the aesthetic requirements that feet not slip and that the leg movement be plausible based on the character translation and rotation.

Leg Groups The player generated characters can have any number of legs of different lengths in arbitrary arrangements. A *leg* is defined as a path through the tree of connected limb segments with a *foot* body at the leaf and a spine segment at the base of the tree—called the *hip*. Figure 6 shows some examples of player created leg configurations with various branching structures. The legs are clustered into one or more groups of roughly equal length. The leg groups are *harmonized* by approximating their length ratios with small rational numbers. The ratios are used to compute the relative frequency of the gait cycle applied to each group.

Foot and Hip Posing For a given leg group, the feet attached to each unique hip are ordered, and the gait system generates a cyclical pattern of foot movements by assigning values to the locomo-

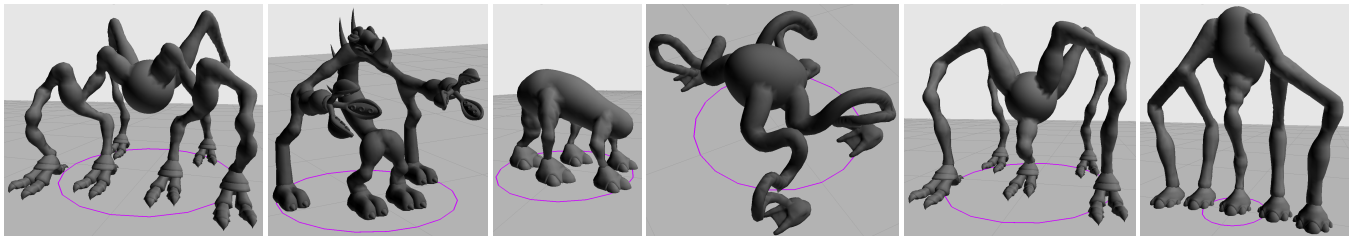


Figure 6: *A variety of player-created leg configurations.*

tion *duty factor* and *step trigger* parameters. The duty factor is the fraction of the duration of the overall gait cycle time that the foot is on the ground [Alexander 2003]. The step trigger is the offset from the overall gait cycle at which the foot begins its particular cycle [Rotenberg 2004]. The hips are translated and rotated as the feet are moved for believable torso motion. The foot's path during its flight phase is controlled by the animators in *Spasm* using either a custom set of parameters for the arc, or a graphical editor for the position and rotation of the foot in a normalized space. The normalized path is then applied to each foot, taking into account the leg length. This system is separate from the generalization and specialization system described above for historical reasons. We have considered merging them but have not had time.

Gait Styles The system can handle multiple gait styles layered on a character at the same time. The animators author a mapping between character movement speed and the gait parameters for groups with $1 \leq n_{feet} \leq 6$, and the system procedurally generates the parameters styles for $n_{feet} \geq 7$. The movement speed is used to interpolate between sets of parameters authored for various velocities. Additional gait styles can be applied to a character for special effects, like limping, or a lumbering gait for large characters. Different leg groups could be in different gait styles simultaneously on the same character. For example, two short legs could be running while four long legs are trotting.

A heuristic is used to determine if a character with no feet should float above the ground or crawl along the ground. Crawling characters have some spine bodies converted into *pseudo-feet* and an inch-worm-like gait is generated for these bodies.

4.3 Particle IK Solver

The animation system's *Particle IK Solver* uses the body pose goals set by the various subsystems already discussed, and attempts to satisfy them by forming a system of constraints over the linked character bodies. Because a *Spore* character usually has many fewer goals than DOF, the IK equations form an *underdetermined system*. In general, underdetermined systems admit infinite solutions, so the solver uses the extra DOF to optimize secondary objectives. The Particle IK Solver has many competing high level objectives:

1. **High Performance.** It will be run every frame on every procedurally animating character in the game.
2. **Accuracy and Naturalness in Workspace.** It should compute an accurate and natural looking solution pose if the goals are within the *workspace* of the character.
3. **Graceful Failure.** There are three main failure modes, and each should result in a natural pose: First, ***goals can be outside the workspace***; the character should reach for them instead of mechanically hyperextend. Second, ***goals can conflict*** and lead to overdetermined constraints for sub-trees of the character. Third, the ***goals may be implausible***; they are strictly within the character workspace, but satisfying them might cause the character to attain an unnatural pose.
4. **Path Independence.** IK solvers for underdetermined systems, especially with preconditioners relying on frame-coherence, can give *path dependent* solutions—the solution at timestep t depends on the solutions at previous timesteps [Klein and Huang 1983; Yan et al. 1999]. This is undesirable for a number of reasons (e.g. introducing unpredictability into playback [Tolani et al. 2000]). Highly redundant manipulators with path dependent solvers can eventually tie themselves in knots in our experience.

Overview Robustly satisfying the objectives above led us to create a new IK solver with several differences from traditional solvers. The Particle IK Solver treats the character skeleton as a set of 3DOF particles with 1DOF length constraints between them. A simple iterative constraint solver is run over the particles with various preconditioners and parameters for tuning its behavior both statically and dynamically. The final body DOF are *reconstructed* from the particle positions and length constraints after the solve. It is a two-phase solver; the first phase solves for the spine pose, and the second phase solves for the limb poses, treating the spine as fixed. We attain more natural poses with the two-phase solver than we did with monolithic one-phase solvers because we can tune each phase to the particular idiosyncrasies of spine versus limb movement. In fact, the Particle IK Solver is specifically designed to support flexible tuning and the addition of many *ad hoc* preconditioners and subtree solvers. We implemented several mathematically more complex iterative and nonlinear equation-based IK solvers (including several flavors of Cyclic Coordinate Descent, Jacobian methods, Constrained Dynamics, etc.), but in our experience these solvers were slower and less amenable to tuning due to their complexity and nonlinearity. This was a major impediment to our ability to tune the solver code to attain natural poses. The simple architecture and core implementation of the Particle IK Solver allows us to make specific ad hoc tuning adjustments and special cases, without compromising the quality of the solution in other areas of the pose. In some sense, the design of the solver gives us "local control" over the solution algorithm, a characteristic we feel was missing from our previous, more "advanced", solvers.

4.3.1 Initialization

The IK solver analyzes the character's body goals and morphology to find the complete sub-tree that "has IK". A body is said to "have IK" if it or any of its children have pose goals. If a sub-tree does not have IK, it is ignored by the solver and it becomes a candidate for procedural secondary animation as discussed in Section 4.4.

The Root The root body is special-cased in the IK solver and always has IK. It is forward kinematically controlled by the anima-

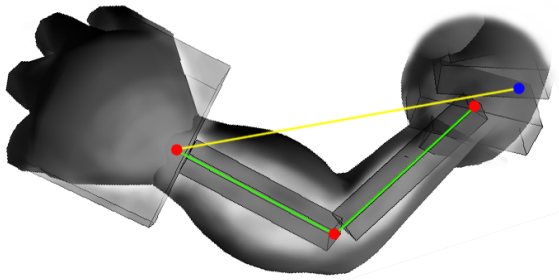


Figure 7: *Particle and constraint allocation for a limb.*

tors and it is not affected by the solver iterations.[8] The root's pose is computed and the non-root bodies are hierarchically transformed into the *root-relative rest pose*. This pose is used as a preconditioner, giving the root goal significant influence over the final pose of the character.

Delegated Goals Some body goals are *delegated* to the body's parent to improve the solved pose. For example, mouths attached to the spine delegate their pose goals to their parent spine segment. This keeps the mouth from moving independently of the spine—if the animator moves the mouth, the spine will bend to place the mouth at the goal and the motion will be more natural than if the mouth had pivoted at its attachment joint.

Particles and Constraints Particles and constraints are allocated to the joints and bodies depending on the caps and the solver phase. Details of the allocation strategy for each phase are below in Sections 4.3.2 and 4.3.3. Particles are 3DOF points in Euclidean space. Full 6DOF body goals are converted to 3DOF position goals at the joint position of the body, avoiding the need to handle 6DOF goals inside the solver [Meredith and Maddock 2004]. Constraints are 1DOF length constraints between two particles. A particle can have any number of constraints attached to it. Each constraint contains a "mass" value for each of its two endpoint particles, allowing the same particle to appear to have different masses for tuning purposes when accessed from different attached constraints. The constraints are enforced iteratively using a nonlinear length correction step [Jakobsen 2001], where the computed length error is used to adjust the position of each particle along the constraint axis in inverse proportion to its mass—a particle with a larger mass moves less during length correction. The mass values have no physical meaning, they are simply scalar values that control the impact of the constraint solution on each endpoint particle. The length error for each constraint is fed through a $C1$ continuous piecewise function that determines the resulting correction distance, enabling soft constraints. Constraints are allowed to stretch and compress to give an empirically more organic feel to the generated poses.

4.3.2 Spine Phase

In the spine phase, we allocate particles and constraints to the spine and to simplified versions of the limbs. Each body with a goal in a limb sub-tree is given a single length constraint directly to the spine. The distal particle is set to the body goal position, the proximal particle is set to the spine joint, and the constraint length is set to the rest chord length of the simplified chain. The yellow line in Figure 7 shows an example, with the blue dot showing the spine attachment point of the simplified limb constraint, and the red dot showing the grasper's pose goal.

[8] Immediate children of the root can have a slight influence on the root's final pose, but this influence is computed after initialization.

Spine Splines The character's torso is usually composed of many spine bodies as shown in Figure 8. Solving the spine directly with a particle at each joint produces kinks and unnatural poses. We avoid these problems by only allocating particles and constraints at IK *branch points* along the spine. Branch points are caused whenever more than one child of a spine body has IK, and at non-spine $\rightarrow$ spine transitions in the IK chains moving rootwards, such as limb attachment points. We generate a particle at each IK branch point on the spine, with a length constraint between each, no matter how many spine bodies exist between the branch points. In Figure 8's example, there are no branch points along the spine, so a single length constraint (the blue line) is generated for the entire spine. If the feet in the figure had goals, there would be branch points at the leg attachment bodies, and multiple constraints along the spine would be generated. Any spine particles are associated with the child spine body of the joint at which they are allocated. This is a unique map since the spine is a serial chain. At each spine particle, an additional constraint is allocated to the root-relative rest pose position for the *anti-buckling algorithm* described below.

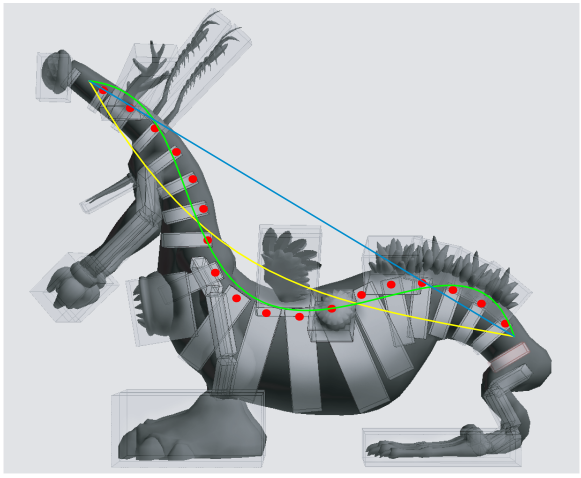


Figure 8: *A typical character spine and spine splines.*

Next, the IK solver fits a spline to the spine body joint positions between each pair of IK branch points, using linear least squares, and stores the spline parameters in the local space of the particle's associated body mentioned above. For a given spine spline segment, only state at the particle endpoints is available during reconstruction, so we must use a single spline to span the segment. It is common for player created spines to have many inflection points, and cubic splines did not fit adequately in many common cases, while quintic Hermite splines proved sufficient. In Figure 8, the red dots are the spine body positions, the yellow curve is the computed cubic spline, and the green curve is the quintic.[9] We then step along the spline—transporting a coordinate frame with its y axis parallel to the spline tangent—to a point near each body's position. We record the t parameter value along the spline and the *relative offset* and *relative orientation* of the body with respect to the transported frame. This information allows us to reconstruct the coordinates of the interior spine bodies given a posed spline, so the spine bodies interior to a spine constraint can be ignored during the solve.

Constraint Iteration and Anti-Buckling Once the particles, constraints, and spine splines have been constructed, the spine phase constraint solver iterations begin. The constraint solver has

[9] Note, the spline endpoints are the joint positions of the first and last constrained spine bodies, not the body positions.

inner and outer iterations. The inner iterations loop over the constraints, from the leaves inward.[10] The outer iterations loop over the inner iterations a fixed number of times (5 in our implementation).

The simplified limbs are allowed to compress to 10% of their rest lengths during this phase, under the assumption that the limb will bend to hit the pose during the limb phase. The shape of the spine is monitored during the outer constraint iterations, and a simple heuristic metric for whether it has *buckled* (folded over onto itself) into an unnatural pose is computed by comparing the angles of the neighboring spine constraints. If buckling is detected, the constraints to the spine's root-relative rest pose are smoothly enabled, which pulls the spine back to a known-good configuration. The end result is a blend between the buckled pose and the root-relative rest pose, which has proven to be a reasonable compromise between flexibility and robustness in our tests.

Reconstruction After the constraint iterations are completed, the spine pose is reconstructed from the particle positions with an outward iteration. The root's pose is known, as mentioned in Section 4.3.1. Each spine particle's position is adjusted by projection in case the constraint iterations did not converge to an allowed shrink/stretch factor (10%/120% in our implementation).

We reconstruct the orientations of spine bodies associated with IK particles by computing a tangent vector at both the rest and the posed position. We then construct the minimal-twist rotation between these tangents, and use it to rotate the root-relative rest orientation of the body. Any delegated orientation goals are then blended into the body. Once the position and orientation of the particle-associated spine bodies are computed, the intra-spline bodies can be reconstructed by the frame transport discussed above. The technique for computing the tangents at the particles depends on the constraint topology. If the particle is in two spine constraints, the tangent, v_i, is computed using a three-point-difference with the neighboring spine particles: $v_i = (p_{i+1} - p_{i-1})/2$. If the particle is an endpoint, we use the tangent of the rootward neighbor particle (which will either be the root or an interior particle, and therefor known) and solve for the tangent of a natural cubic spline given $v_i = 0$, yielding $v_i = 3(p_{i-1} - p_i)/2 - v_{i-1}/2$.

4.3.3 Limb Phase

The limb phase of the IK solver uses the posed spine and solves for the limb configurations. In this phase, particles are allocated at every limb joint, and constraints are allocated for every limb body (the red points and green lines in Fig. 7). If a body has multiple direct children with IK goals, cross-constraints are added between them so they retain their relative positions.

Preconditioning The limb constraint solver uses an *aim preconditioner*. The spine attachment point of a limb sub-tree is computed given the newly posed spine, and then the limb sub-tree is distorted (or *aimed*) such that its IK particles are at their goal positions before the constraint iterations start.

We will describe the limb aim distortion for a serial chain limb first, where there is a single particle at the leaf with a pose goal. A vector v_r from the spine attachment point to the leaf particle in the root-relative rest pose is computed. Then a vector v_g from the spine attachment point to the IK goal for the particle is computed, and a minimal-twist rotation is generated to take v_r to v_g. Finally, a scale factor along v_g is composed with the rotation to create a transform that rotates and scales the limb particles such that the end particle is at the pose goal. This transform does not take the constraint lengths into account, nor does it scale in directions orthogonal to v_g, so if the pose goal is very far (resp. close) from the character, the limb will be highly stretched (resp. squashed) along v_g. The resulting length constraint violations will be handled during the constraint iteration. The aim preconditioner keeps the limb in a natural pose as it compresses and extends, and favors rotation at shoulders and hips. Simply running the constraint iterations from the root-relative rest pose leads to unnatural poses, as the goal error is distributed poorly along the limb. Other IK algorithms try to provide tuning parameters to ameliorate this problem [Welman 1993]. Our aim preconditioner leads to more natural poses in our experiments, but it relies on the temporary violation of the length constraints, which is not supported by most IK algorithms.

For branching limbs, the algorithm computes the aim distortion transform by forming a weighted average of the goals in a sub-tree. It then recurses outward to each child branch point, and repeats until it aims the leaf serial chains.

Reconstruction The constraint iteration is identical to the spine phase (without the anti-buckling algorithm), and after it is complete, the limb positions and orientations are reconstructed. Position reconstruction is also similar the spine phase. Because we have a full set of particles and constraints for the limb bodies, we only need to compute the twist around the constraint axis to complete a body's orientation. In the case of a body with no cross-constraints on its particles, we compute the minimal-twist rotation from the root-relative rest pose to the final pose and use it to transform the body's rest orientation to the final orientation. If there are cross-constraints, then the orientation is fully determined (or overdetermined) and we pick the first two constraint axes to form the posed frame. We have considered a minimization over the potential orientations similar to the *relative orientation* problem in photogrammetry [Horn 1990], but it has not proved necessary.

4.4 Jiggles

There is no robust way for an animator to select the bodies that are *not* required for a given motion during authoring, but these bodies should have secondary animation applied to them for aesthetic quality [Thomas and Johnston 1981; Lasseter 1987]. Therefor, if a sub-tree of the character does not "have IK", a very simple highly-damped pseudo-physical dynamics simulator—called *Jiggles*—is applied to the bodies based on a heuristic for flexibility, placement, and type. The important factors are that the "jigglable" sub-trees are dynamically determined based on the bodies the animation *does not* select, and the simulated movement is completely passive with respect to the keyed bodies and does not feed back to the rest of the character. This latter restriction keeps the animator's motion authoritative over the selected parts of the character, while the remainder of the character merely responds to the keyed movement with plausible secondary motion. Our original secondary animation system—called *Wiggles*—set goals for the IK solver and affected the animation pose goals, which negatively impacted the quality of the animators' work.

5 Results and Discussion

We have presented an animation system capable of playing animations on characters with wildly varying morphologies, characters that did not exist at the time the animations were authored. The animations retain stylistic details as they are played, and the system allows animators to control the way the animation is generalized across characters. The animators are able to use familiar work flow to create these animations. The character gaits are synthesized with stylistic variations tuned by the animators. An efficient IK solver, tuned to return natural organic movements, poses the character ev-

[10] Inward iteration distributes the position error immediately since the leaves contain all the initial error.

ery frame. A simple procedural system applies subtle secondary animation for believability.

Testing and Performance The animation system presented has been implemented and has undergone thorough testing. Qualitatively, players seem amazed when the character they just created "comes to life" and expresses emotions via animation. The in-game editor begins playing animations on the character as soon as the player adds feet, mouths, or graspers. Quantitatively, we stochastically test the animations against an ever-changing database of characters uploaded from the game. A team of animation testers plays the animations on a range of characters and reports failures using a spreadsheet as shown in Figure 9. The fail cases reveal plain old code bugs, aesthetic generalization issues that can be addressed by animators with the existing features, and generalization issues that need additional features or branches (Section 4.1.1) to solve. We have tested several hundred creatures and a thousand animations over the development of the system. The current pass rate is ~90% and is continually improving.

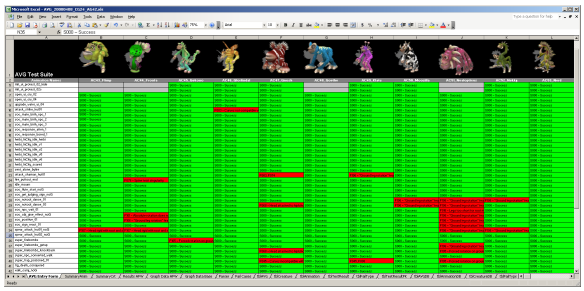


Figure 9: *An example Animation Validation Grid.*

The performance of the system is acceptable. A procedurally animating character with 25 bodies currently takes approximately 0.2ms per frame on a 1.7Ghz Pentium-M machine and optimization is ongoing. Approximately 35% of the time is spent in the IK solver, with the rest spread amongst keyframe interpolation, goal blending, etc.

Applicability and Usability Although this animation system was created specifically for *Spore* and its extreme character morphology range, there are several aspects of this work that are applicable to other games and animation problems. Most importantly, bringing animators into the loop with the semantic markup phase for selection and movement modes, and giving them realtime preview and editing across multiple different targets, is applicable to motion retargeting in general, and more broadly, any place parameterized animation is used. Although there is an appeal to algorithmically discovering semantics, we find having a human simply tell the system what is important is immensely powerful, efficient, and robust. Our target-relative movement modes are also generally applicable. For example, many existing systems use blending to interpolate poses for reaching targets or aiming weapons with style [Kovar and Gleicher 2004]. By using our movement modes to record the animation curves, we avoid the need to search and blend sampled data, and we can distort the curves directly to meet a goal while retaining authored style. The Particle IK Solver is another generally useful contribution of the work. Designing explicitly for ad hoc tuning and preconditioners to aid in finding natural pose solutions is valuable to all animation IK systems. Finally, as more games explore user generated content, whether allowing players to create characters from scratch like *Spore*, or just allowing adjustable height, weight, body shape, and other morphology parameters, real time motion retargeting will become more important to many games.

Animators learning the system need training on how to create context queries and use *Spasm*, but they create complete animations on a single character within a few hours. However, it takes weeks to build up an intuition about which kinds of motions generalize across a wide range of characters and which don't. The system is complicated, and there are multiple equivalent ways to author an animation on a single character that specialize completely differently onto other characters.

Limitations The current animation system has many limitations. The current selection and movement modes cannot effectively express a motion such as "hug each other" or "rub your chin" because the system has no understanding of volume and boundaries. We also completely ignore intra-character collision because it was too difficult to solve the nonlinear workspace problem for arbitrary morphologies within our time constraints, but it is clearly a flaw in the system. The Particle IK Solver's failures could be more graceful. The current anti-buckling technique works most of the time, but it is simplistic and can appear heavy-handed when it engages. Also, the path independence of the IK solver causes singularities in the solution space.[11] We push the singularities around to configurations the character is unlikely to hit, but the animations can still twist unrealistically if the pose comes close to the singularity.

Future Work In addition to addressing the specific limitations above, the biggest area for improvement involves increasing the level of "reasoning" the system performs about the character morphologies. Examples would be the aforementioned boundary relative mode and a movement mode that models concepts like "close in" and "extended". Animators will need to be able to express much more complex and subtle semantics to the system before the animations will generalize in way that's competitive with a Maya animation on a single fixed character. The gait system's biggest challenge is how to provide believable locomotion with anticipation in the face of discontinuous player and code inputs.

Acknowledgements

Many thanks to Lucas Kovar for his incredible help in preparing this paper. James O'Brien, Zoran Popović, and the anonymous reviewers provided excellent feedback as well. We also thank the *Spore* animators, John Cimino, Tony Gialdini, Bob King, and Gal Roth, and the rest of the *Spore* team for putting up with our *unique*[12] animation system. Finally, thanks to the animation testers, Patrick Benjamin, David Kertesz, and Steven Sadler, and to producers Stephen Lim, Brodie Andersen, and Steve Eng.

References

ALEXANDER, R. M. 2003. *Principles of Animal Locomotion*. Princeton University Press.

APPLE, INC. 2008. *Apple iTunes Smart Playlists*. `http://www.apple.com/lae/itunes/smartplaylists.html`.

AUTODESK, INC. 2008. *Autodesk Maya*. `http://www.autodesk.com/maya`.

BARZEL, R., HUGHES, J. F., AND WOOD, D. N. 1996. Plausible motion simulation for computer graphics animation. In *Computer Animation and Simulation '96*, 183–197.

BARZEL, R. 1997. Faking dynamics of ropes and springs. *IEEE Comput. Graph. Appl. 17*, 3, 31–39.

[11] This is unavoidable for path independent solvers because of the Hairy Ball Theorem and its ilk.

[12] They might choose a different word.

BRUDERLIN, A., AND CALVERT, T. 1989. Goal-directed dynamic animation of human walking. In *Proceedings of ACM SIGGRAPH 89*, 233–242.

BRUDERLIN, A., AND CALVERT, T. 1996. Knowledge-driven, interactive animation of human running. In *Proceedings of Graphics Interface (GI'96)*, 213–221.

CHI, D., COSTA, M., ZHAO, L., AND BADLER, N. 2000. The emote model for effort and shape. In *Proceedings of ACM SIGGRAPH '00*, 173–182.

CHOI, K.-J., AND KO, H.-S. 2000. On-line motion retargeting. *Journal of Visualization and Computer Animation 11*, 223–243.

DESMECHT, L., AND DACHARY, L., 2006. Cal3d animation system. `http://home.gna.org/cal3d`.

EDERY, D., BROWN, M., PALLISTER, K., KOSTER, R., AND MUZYKA, R., 2007. Sharing control. Game Developers Conference 2007.

GIRARD, M., AND MACIEJEWSKI, A. 1985. Computational modeling for the computer animation of legged figures. In *Proceedings of ACM SIGGRAPH 1985*, 263–270.

GLEICHER, M. 1998. Retargeting motion to new characters. In *Proceedings of ACM SIGGRAPH 98*, Annual Conference Series, ACM SIGGRAPH, 33–42.

GROCHOW, K., MARTIN, S., HERTZMANN, A., AND POPOVIĆ, Z. 2004. Style-based inverse kinematics. *ACM Transactions on Graphics 23*, 3.

HORN, B. K. P. 1990. Relative orientation. *International Journal of Computer Vision 4*, 59–78.

JAKOBSEN, T., 2001. Advanced character physics. Game Developers Conference 2001. `http://www.teknikus.dk/tj/gdc2001.htm`.

KASTENMEIER, T., AND VESELY, F. J. 1996. Numerical robot kinematics based on stochastic and molecular simulation methods. *Robotica 14*, 329–337.

KLEIN, C., AND HUANG, C. 1983. Review of pseudoinverse control for use with kinematically redundant manipulators. *IEEE Trans. Systems, Man, and Cybernetics 13*, 2 (March/April), 245–250.

KOVAR, L., AND GLEICHER, M. 2004. Automated extraction and parameterization of motions in large data sets. *ACM Trans. Graph. 23*, 3, 559–568.

KULPA, R., MULTON, F., AND ARNALDI, B. 2005. Morphology-independent representation of motions for interactive human-like animation. *Computer Graphics Forum, Eurographics 2005 special issue 24*, 3, 343–352.

LASSETER, J. 1987. Principles of traditional animation applied to 3d computer animation. In *Proceedings of ACM SIGGRAPH '87*, 35–44.

LEE, J., AND SHIN, S. Y. 1999. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of ACM SIGGRAPH 99*, 39–48.

MEREDITH, M., AND MADDOCK, S. 2004. Using a half-jacobian for real-time inverse kinematics. In *Procedings of The 5th International Conference on Computer Games: Artificial Intelligence, Design and Education*, 81–88.

MIZUGUCHI, M., BUCHANAN, J., AND CALVERT, T. 2001. Data driven motion transitions for interactive games. In *Eurographics 2001 Short Presentations*.

MOZILLA FOUNDATION. 2008. *Mozilla Thunderbird Message Filters*. Tutorial at `http://opensourcearticles.com/thunderbird_15/english/part_07`.

MURATORI, C., ROBERTS, J., FORSYTH, T., AND MOORE, D., 2008. Granny 3d animation sdk. `http://www.radgametools.com/granny/sdk.html`.

NEFF, M., AND FIUME, E. 2005. Aer: aesthetic exploration and refinement for expressive character animation. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 161–170.

O'BRIEN, J. F., ZORDAN, V. B., AND HODGINS, J. K. 2000. Combining active and passive simulations for secondary motion. *IEEE Comput. Graph. Appl. 20*, 4, 86–96.

POPOVIĆ, Z., AND WITKIN, A. 1999. Physically based motion transformation. In *Proceedings of ACM SIGGRAPH 99*, 11–20.

ROTENBERG, S., 2004. Locomotion. UCSD CSE169: Computer Animation Lecture Notes. Available at `http://graphics.ucsd.edu/courses/cse169_w04/CSE169_13.ppt`.

RYCKAERT, J., CICCOTTI, G., AND BERENDSEN, H. 1977. Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes. *Journal of Computational Physics 23*, 327–341.

SHIN, H. J., LEE, J., GLEICHER, M., AND SHIN, S. Y. 2001. Computer puppetry: an importance-based approach. *ACM Transactions on Graphics 20*, 2 (Apr.), 67–94.

SUN, H. C., AND METAXAS, D. N. 2001. Automating gait generation. In *Proceedings of ACM SIGGRAPH '01*, 261–270.

TANG, W., CAVAZZA, M., MOUNTAIN, D., AND EARNSHAW, R. 1999. A constrained inverse kinematics technique for real-time motion capture animation. *The Visual Computer 15*, 7–8 (November), 413–425.

THOMAS, F., AND JOHNSTON, O. 1981. *Disney Animation, The Illusion of Life*. Abbeville Press.

TOLANI, D., GOSWAMI, A., AND BADLER, N. I. 2000. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical models 62*, 5, 353–388.

WELMAN, C. 1993. *Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation*. Master's thesis, Simon Fraser University.

WILLMOTT, A., QUIGLEY, O., CHOY, L., SHARP, B., AND INGRAM, R., 2007. Rigblocks: Player-deformable objects. SIGGRAPH 2007 Sketches.

WRIGHT, W., 2005. The future of content. Game Developers Conference 2005, March. `http://video.google.com/videoplay?docid=-262774490184348066`.

YAN, Y., OHNISHI, K., AND FUKUDA, T. 1999. Decentralized control of redundant manipulators: a control scheme that generates a cyclic solution to the inverse problem. In *Proceedings of 1999 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 404–409.

ZHAO, J., AND BADLER, N. 1994. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphics 13*, 4 (Oct.), 313–336.