Authoring and Animating Painterly Characters

KATIE BASSETT
Disney Research Zurich and ETH Zurich
ILYA BARAN
Disney Research Zurich
JOHANNES SCHMID and MARKUS GROSS
Disney Research Zurich and ETH Zurich
and
ROBERT W. SUMNER
Disney Research Zurich

Artists explore the visual style of animated characters through 2D concept art, since it affords them a nearly unlimited degree of creative freedom. Realizing the desired visual style, however, within the 3D character animation pipeline is often impossible, since artists must work within the technical limitations of the pipeline toolset. In order to expand the range of possible visual styles for digital characters, our research aims to incorporate the expressiveness afforded by 2D concept painting into the computer animation pipeline as a core component of character authoring and animation. While prior 3D painting methods focus on static geometry or simple animations, we develop tools for the more difficult task of character animation. Our system shows how 3D stroke-based paintings can be deformed using standard rigging tools. We also propose a configuration-space keyframing algorithm for authoring stroke effects that depend on scene variables such as character pose or light position. During animation, our system supports stroke-based temporal keyframing for one-off effects. Our primary technical contribution is a novel interpolation scheme for configuration-space keyframing that ensures smooth, controllable results. We demonstrate several characters authored with our system that exhibit painted effects difficult to achieve with traditional animation tools.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*; G.1.1 [**Numerical Analysis**]: Interpolation—*Interpolation formulas*

General Terms: Design, Algorithms

Additional Key Words and Phrases: Painterly animation, expressive animation, nonphotorealistic rendering

Authors' addresses: K. Bassett (corresponding author), Disney Research Zurich and ETH Zurich; email: kmbassett@gmail.com; I. Baran, Disney Research Zurich; J. Schmid and M. Gross, Disney Research Zurich and ETH Zurich; R. W. Sumner, Disney Research Zurich.

DOI: http://dx.doi.org/10.1145/2484238

ACM Reference Format:
Bassett, K., Baran, I., Schmid, J., Gross, M., and Sumner, R. W. 2013. Authoring and animating painterly characters. ACM Trans. Graph. 32, 5, Article 156 (September 2013), 12 pages.
DOI: http://dx.doi.org/10.1145/2484238

1. INTRODUCTION

The visual design of the characters in a computer animated film greatly influences the film's emotional and comedic impact, making character design a critical component of animation production. During the design phase, concept artists explore a character's visual style using traditional media or 2D digital painting tools. By creating artwork that expresses a spectrum of different "looks," the artists search for a style and representation that perfectly captures the character's personality, background, and role in the film. A delicate, vulnerable character may be depicted using soft, pastel strokes, while a villain's design could focus on bold, angular lines. These conceptual works represent the artistic vision for the character's look and presence in the film.

Unfortunately, realizing this artistic vision within the constraints of the modern computer animation pipeline is often impossible. The animation pipeline encompasses a set of powerful tools for modeling, rigging, texturing, posing, lighting, and rendering that make animation creation tractable. However, while concept artists have nearly unlimited creative freedom when working directly with pencils, pastels, brushes, and paint, the animation pipeline provides only indirect influence over the character's final look through the toolset of the pipeline software. These tools may not accomodate the visual style envisioned for the character, requiring alterations to conform to the pipeline's technical limitations. As a result, the final look of the character in the finished animation may deviate significantly from the artist's original vision, with those soft strokes and bold lines lost in the translation.

Our research aims to incorporate the expressiveness afforded by concept painting into the computer animation pipeline in order to expand the range of possible visual styles for digital characters. We approach this task using a few guiding principles. First, we want to bring painting into the animation pipeline as a core component of character authoring so that painted styles can extend beyond mere static looks to directly determine a character's appearance in different poses, from different camera views, under different lighting conditions, and at different times. Second, we focus on providing direct control to the artist by ensuring that individual

Fig. 1. Poses of a painterly ballerina authored and animated with our system. © Disney

strokes painted during character authoring are faithfully rendered in the final frame. Third, we want to augment and improve the existing animation pipeline, rather than replacing it all together. In this way, more subtle painterly styles can be accomplished by adding painted embellishments on top of traditional digital animation methods. Or, for full expressive flexibility, the artist can shape the character's appearance entirely and directly through painting. The painterly ballerina in Figure 1 is an example of our work.

To accomplish these goals, we extend ideas from *static* stroke-based 3D painting with a focus on painterly character authoring and animation. First, we show how to associate the movement of painted strokes with the movement of a character's mesh so that 3D paintings can be deformed using standard rigging tools, regardless of the particular rigging algorithm employed. Next, we propose a configuration-space keyframing algorithm for authoring pose-dependent stroke effects. This mechanism allows stroke opacity or movement to be keyframed to positions in a configuration space that includes character pose and other scene variables such as light position. With this mechanism, artists can create pose-dependent touch-ups to fine-tune the look of a character, or larger-scale effects such as an animated facial expression. Finally, during animation, our system supports stroke-based temporal keyframing for one-off effects.

Our primary contribution is a system and workflow for painterly character authoring and animation that provides direct control over expressive, animated character appearance. In realizing this system, we make several technical contributions, of which the most significant is a novel configuration-space interpolation algorithm. We also describe a stroke-skinning algorithm, stroke-based deformation tools, and a stroke-based temporal keyframing function. We demonstrate several characters authored with our system that exhibit painted effects difficult to achieve with traditional animation tools.

2. RELATED WORK

Researchers have long recognized the importance of expressive stylization in computer animation, and the NonPhotorealistic Rendering (NPR) community has developed an impressive body of work that achieves different stylizations through novel rendering or video processing methods. Rendering algorithms target impressionism [Meier 1996], watercolor [Curtis et al. 1997; Bousseau et al. 2006], hatching [Praun et al. 2001], Seuss-esque illustrations [Kowalski et al. 1999], and many other media and styles [Hertzmann 2003] while addressing techincal challenges such as temporal coherence [Bénard et al. 2011] and interactivity [Markosian et al. 1997]. Early work in video stylization [Litwinowicz 1997; Hertzmann and Perlin 2000] shows how to transform a sequence of images into a "painted video." Recent methods in this area (e.g., Lin et al. [2010] and Lu et al. [2010]) deliver compelling representations of different painterly styles.

Although these rendering and video processing methods can convincingly mimic a wide variety of styles and media, customizing the result to achieve a particular look remains challenging. The techniques are algorithmic in nature and provide only indirect influence over the final result by tuning parameters such as stroke shape, texture, size, orientation, and density. Since these variables may be unintuitive to an artist, Zhao and Zhu [2011] develop a perceptual basis for stylization control variables in order to make parameter tuning more intuitive. Sophisticated user interfaces also help make customization easier [O'Donovan and Hertzmann 2012]. Other approaches use example-based control by copying pixels from an exemplar image [Hertzmann et al. 2001] or by learning hatching styles from a drawn example [Kalogerakis et al. 2011]. In all cases, the primary focus is controlling stroke synthesis, rather than direct specification of the final result.

In response to the need for direct control over NPR stylization, researchers have focused on 3D painting or drawing interfaces. Work in ab initio design uses drawing [Cohen et al. 2000; Bourguignon et al. 2001; Tolba et al. 2001; Keefe et al. 2007; Rivers et al. 2010] or painting [Keefe et al. 2001] to create objects and environments from scratch, but lacks the painterly aesthetic possible with traditional 2D digital painting systems. This limitation is addressed by embedding painted strokes on [Teece 1998; Daniels 1999] or near [Schmid et al. 2011] a 3D proxy shape and replacing traditional rendering with a "repainting" algorithm that ensures every painted stroke is faithfully represented in the final imagery. These methods present the artist with a blank slate on which to create an object's look, permitting a high degree of expressive freedom with little algorithmic interference. However, they support only camera animation, and thus cannot accomodate animated characters that move and deform.

As an alternative, Kalnins and colleagues [2002] strike an intermediate balance with a system that combines more traditional NPR

methods such as toon shading with a powerful painting interface to directly control the stylization of annotations for silhouette strokes, creases, hatching, and painted decals. By combining traditional NPR shaders with authored annotations, animated characters and level-of-detail operations are supported, albeit with some indirection between the authored style and the rendered result. Similarly, Maya Paint Effects [2011] allows the creation of 3D structures that can be rendered in a painterly fashion, but the focus is on geometry instancing rather than tools for stroke-level animations tailored for painterly characters.

Our work represents a change in core focus compared to previous NPR research. Whereas existing work centers on new rendering or image processing styles, we place the spotlight on character authoring and explore how stylized depiction can be incorporated into the character rigging and animation stages. We give the artist tools to author dynamic, stylized looks for animated characters that are as expressive as the static looks currently possible with concept painting. We take inspiration from the level-of-detail authoring concept in WYSIWYG NPR [Kalnins et al. 2002] and build upon existing 3D painting [Schmid et al. 2011] and rendering [Baran et al. 2011] algorithms, with a focus on authoring and animating painterly characters. To accomplish this goal, we propose a configuration-space keyframing system for painted strokes and extend existing pose-space interpolation algorithms [Lewis et al. 2000; Ngo et al. 2000; Sloan et al. 2001] to allow high-quality stroke movement.

3. METHOD

Our work focuses on incorporating the expressiveness afforded by traditional 2D concept painting into the 3D character authoring process. We build upon existing research in stroke-based 3D painting since it addresses similar goals for static scenes. In particular, the OverCoat system of Schmid and colleagues [2011] permits a very painterly aesthetic, and we use it as a starting point for our work. However, other stroke-based 3D painting systems [Daniels 1999; Kalnins et al. 2002] could also provide a strong foundation on which to build our painterly character authoring system.

In OverCoat, the artist creates a simple proxy version of the object she or he wishes to paint using traditional 3D modeling tools. This proxy object defines an implicit 3D canvas, allowing the artist to paint strokes into the space near the object using a familiar 2D tablet for input. Strokes are stored as polylines in space and carry additional information about color and brush texture that allows the entire painting to be "repainted" from different viewpoints. However, OverCoat paintings are entirely static, and are therefore limited to models in a single pose with no support for animated characters that move and deform.

Our system uses the 3D paint and embedding methods of OverCoat and extends beyond immobile 3D paintings to encompass the authoring and animation of deformable characters with stroke-level control. Specifically, we add stroke-skinning, editing, and keyframing capabilities. We allow keyframing both in time for one-off effects and in configuration space for creating a painterly "rig" analogous to pose-space deformation [Lewis et al. 2000]. Our tools for editing paint strokes, called "smudge" and "expand," work using a paint-stroke metaphor themselves.

3.1 Workflow

Traditionally, character authoring includes modeling, rigging, and texturing a character, while animation involves setting the values of its rig parameters over time in order to create movement. We leverage the traditional character animation pipeline and show how to enhance it with stroke-based painting.

Fig. 2. A comparison of proxy geometry (left), a result after skinning deformation (middle), and after skinning and configuration-space keyframing (right). Configuration-space keyframing allows the artist to add stroke animation for features that are not present in the geometry or rig, such as the eyebrows, the cheeks, the pointed hairs, and the eye motion of this dog. © Disney

Painting. During the authoring phase, the artist creates a 3D proxy model and rig for the character using traditional techniques. Since much of the character's detail will ultimately come from painting, the character's proxy geometry and rig need only be an approximate representation. The artist then shapes the overall appearance of the character by painting it. Each painted stroke is embedded in the space near the model according to the toolset of the static 3D painting system. During this initial painting phase the artist is no longer confined to painting the "rest pose" of the model, as with OverCoat [Schmid et al. 2011], but is now able to paint the character in any pose due to our skinning deformation algorithm (Section 3.2). This algorithm can automatically move the painted strokes together with the proxy geometry as the character is positioned, allowing the artist to see and work with the model in any configuration.

Our system links to Maya [2012] in order to receive mesh pose and animation data. During the initial painting phase, the artist can move and pose the character using traditional rig and animation tools in Maya while the character is simultaneously updated in our interface. Our skinning algorithm (Section 3.2) stores paint-stroke information in the rest pose regardless of the pose it was painted in, allowing the artist to freely alternate between posing in Maya and painting in our system. The initial state of the character is shaped in this way, until the figure is satisfactory in any configuration the artist may need.

Stroke Rigging. The artist shapes the overall appearance by painting the character model as described earlier, but may desire more fine-scale changes to take place at the stroke level when a particular pose is achieved. For example, when activating a facial blend shape, the artist may wish to include painted changes in the character's facial expression, as demonstrated in Figure 2. Our configuration-space keyframing system (Section 3.3) supports this functionality. The character's rig parameters, together with other scene variables such as light and camera positions, define the scene's configuration space. The artist can modify and key the opacity or position of any stroke to the current scene configuration. Our configuration-space interpolation algorithm (Section 4) ensures that the keyframes are smoothly and predictably interpolated. To author these stroke-level behaviors, the artist need only pose the character in Maya, modify the strokes as desired, and set a configuration-space key.

Like other graphics tools, our system supports the concept of layers, which, in our case, are collections of strokes. Because strokes that are grouped together (logically, not necessarily spatially) will typically be animated together, our stroke-based keyframing

mechanisms operate per-layer: a keyframe applies to all strokes in the layer for which it is set. For the keyframing to be simple, it is therefore important that the artist organize the scene elements into layers. For example, the hair strokes on top of the head in Figure 2 are authored as a single layer. While the keyframes are set for each layer, any stroke manipulation can be applied to individual strokes, selected by color, layer, or region, or they can be applied across the entire layer. The artist can change stroke opacity, and can modify stroke geometry using the smudge and expand tools (Section 3.3). Both the smudge and expand tools are based on the painting input metaphor by using the same embedding facilities as the paint strokes. Once the paint strokes have been modified in the current pose, a key can be set, locking the stroke attributes in the active layer to that pose and configuration.

Animation. In the animation phase, the character proxy geometry is animated using traditional animation software like Maya. During playback, our system deforms the painted strokes and interpolates any configuration-space keyframes on stroke movement or opacity that were created in the authoring phase. Our system supports an additional keyframing mechanism that allows the opacity and point positions of strokes to be keyed to the animation timeline (Section 3.4). The procedure is the same as with setting configuration-space keys: the artist moves the timeline in Maya to the frame to be modified, makes the desired changes to the strokes, and sets a time-specific key in our system. This functionality permits one-off effects relevant to the particular context of the animation. Once the character has been animated, final renders are created using a mixed-order compositing algorithm that ensures temporal coherence [Baran et al. 2011].

3.2 Skinning Deformation

Skinning deformation connects the movement of strokes to the deformation of the proxy object, allowing 3D paintings to be deformed by traditional character rigs. We employ an algorithm based on linear-blend skinning [Akenine-Möller et al. 2008] to accomplish this task. While linear-blend skinning is traditionally used in the context of a skeleton, we blend transformations of surface elements, similar to Singh and Kokkevis [2000]. For each vertex of the proxy geometry, we compute a transformation that captures the space deformation in a local neighborhood from a designated rest pose to the target pose. For a given vertex v_i and its one-ring V_i^1, our algorithm uses Procrustes analysis to find a least-squares optimal rigid motion 3-by-4 matrix $\mathbf{M}_i$ that aligns all vertices in V_i^1 from the rest pose to the target pose. These $\mathbf{M}_i$'s serve as "bones."

For the actual skinning deformation, paint strokes are transformed from the rest pose by a convex combination of $\mathbf{M}_i$, where each point on a paint stroke has its own set of weights.

$$\mathbf{M} = \sum_i w_i \mathbf{M}_i \tag{1}$$

While computing good skinning weights is difficult in some applications, in our case paint strokes are typically located relatively close to the proxy geometry's surface. As a result, simply associating each stroke with the closest geometric primitive of the mesh has proven sufficient. If the closest point on the surface lies within a triangle, the barycentric coordinates of the closest point are used as weights w_i. When the closest point lies exactly on an edge or vertex, the barycentric coordinates of any shared triangle delivers the correct weights. As long as the proxy surface deforms smoothly, transformations on adjacent vertices are similar, and blending them does not result in common linear-blend skinning artifacts. Figure 3 shows a simple result of this skinning deformation method.

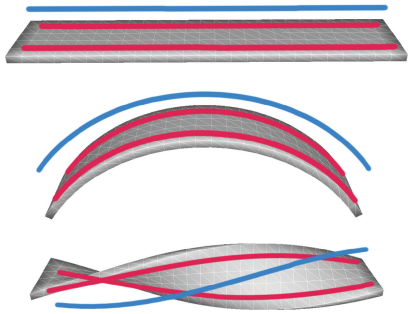


Fig. 3. This figure shows a simple result of our skinning deformation algorithm. The red strokes are embedded on the surface, while the blue stroke is embedded above it. Since we compute skinning transformations based on rotations around the mesh vertices, they naturally extend to the space surrounding the object.

Newly applied paint strokes are painted with respect to the currently active pose, but stored with respect to the rest pose. We compute the skinning weights in the active pose, and then apply $\mathbf{M}^{-1}$ to each stroke point to find its position in the rest pose. An alternative would be to store the paint strokes with respect to the pose in which they were painted, but doing so would require storing each pose in which paint strokes were created as well as computing separate transformations from them to the currently active pose.

3.3 Configuration-Space Keyframing

The skinning deformation described in the previous section allows control over the gross movement of strokes. However, a core advantage of 3D painting is that it gives the artist the power to directly paint details and subtle expressive elements that are not present in the proxy geometry and difficult to realize through modeling operations. Since animating such elements with traditional rigs may be cumbersome or impossible, we give artists detailed stroke-level control using a configuration-space stroke keyframing system.

The configuration space is a high-dimensional space defined by the character's rigging controls (joint angles, blend-shape variables, etc.) and other scene parameters such as light and camera positions. In this sense, it represents a generalization of Rademacher's [1999] work on view-dependent geometry. A keyframe is a point in this space together with desired stroke information at that point. For an arbitrary pose, the keyframes nearby in configuration space are interpolated (Section 4). Our system allows stroke opacity and stroke position and shape to be keyframed. We chose this set based on the needs encountered during the creation of our example results. However, other quantities such as color or stroke width could also be keyframed in the same manner.

Since a 3D painting may consist of thousands of paint strokes, we encourage the artist to partition the painting into layers that have a semantic meaning with respect to animation. For example, if an eyebrow is to be animated, all strokes belonging to the eyebrow should be placed in a separate layer. Keyframes are always set with respect to an entire layer. For example, a stroke position keyframe captures the positions of all paint stroke points in a layer.

Opacity keyframing is straightforward. It involves a single opacity measure that can be set to any value between 0 and 1 at any given point in the configuration space. This opacity value modulates the opacity of all strokes in the layer, allowing a whole group of paint strokes to fade in or out during the animation.

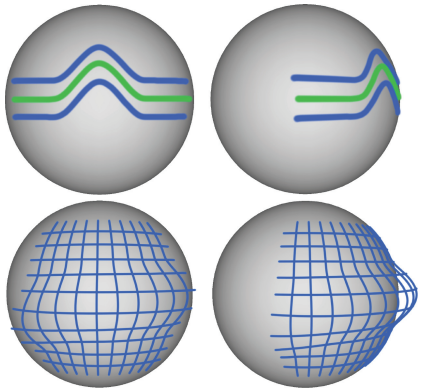


Fig. 4. This figure shows the result of applying the stroke smudging tool to three paint strokes that were originally straight (top row), and of applying the stroke expansion tool to a grid of strokes that was originally painted on the sphere's surface (bottom row).

Position keyframes are more complex because of their interaction with skinning. Following pose-space deformation [Lewis et al. 2000], we interpolate all stroke point positions linearly in the rest pose. Although more sophisticated gradient-domain blending can be used, we did not see a need for it. Specifying a positional keyframe involves providing all of the point positions that define the strokes in a layer. Since setting these point positions manually would be cumbersome, we provide the artist with two tools to deform paint strokes: smudging and expansion. Both tools share a common input metaphor with the painting process: they operate on stroke paths embedded in space using the 3D painting system. Smudging moves paint stroke points along the direction of the embedded smudge stroke, while expansion moves stroke points along the normal defined by the implicit canvas [Schmid et al. 2011]. Both tools only affect stroke points within a certain radius around the embedded path. The magnitude can optionally be modulated by a falloff function, such as the cubic step function $2r^3 - 3r^2 + 1$, where r is the distance normalized by the tool radius. Figure 4 illustrates the operations of these shaping tools.

3.4 Temporal Keyframing

The configuration-space keyframing system described earlier focuses on character authoring and is used to incorporate stroke-based effects that are common to character poses or other repeated scene conditions. When a particular animation is created, the artist may wish to add one-off effects that are specific to the context of the animation. We accomplish these edits by allowing stroke opacities and point positions to be keyed to the animation timeline.

Since animation takes place after authoring, temporal keyframes should override any existing configuration-space keys. To bound the regions where temporal keyframes are in effect, we employ the concept of "sentinel keys." In contrast to "normal" temporal keys, sentinel keys do not carry any data. Let t be the current animation time. If t is between two normal temporal keyframes, the keyframed result should be the interpolation between the two keyframe values (we use our interpolation method restricted to one dimension). If t is between two sentinel keys (or between a sentinel key and the beginning/end of the animation), temporal keyframing is inactive. In this case, the keyframed value is determined entirely by the configuration-space keyframes. To avoid sudden jumps around a temporally keyframed region, there needs to be a smooth transition between configuration-space and temporal keyframing. Therefore,

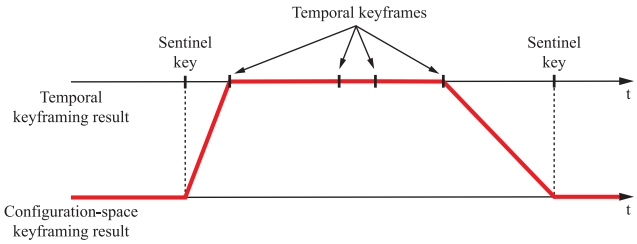


Fig. 5. Sentinel keys are used to bound the regions where temporal keyframing is active. In this figure, the horizontal line at the top represents the parameter values that result from interpolating the temporal keyframes, while the one at the bottom represents the parameter values resulting from the configuration-space keyframes. Between the sentinel keys and their neighboring temporal keyframes, the results of temporal keyframing and configuration-space keyframing are blended to ensure a smooth transition.

we blend the result of configuration-space keyframing and the value of the neighboring temporal keyframe if t is between a sentinel key and a temporal keyframe (Figure 5).

4. CONFIGURATION-SPACE INTERPOLATION

Given a set of keyframes in configuration space $\mathbb{R}^d$, our system needs to be able to interpolate them to all other points in configuration space. The interpolant has to generate natural-looking animation, which is difficult to define mathematically. Instead, we provide a number of simple requirements that should not be violated. Let $(\mathbf{x}_i, y_i)$ be n keyframes, with $\mathbf{x}_i$ the locations in configuration space and y_i the values (e.g., layer opacities or stroke positions). Let $\hat{y}(\hat{\mathbf{x}})$ be the interpolant.

REQUIREMENT 1. *The interpolant should interpolate the keyframes:* $\hat{y}(\mathbf{x}_i) = y_i$.

If the interpolant does not actually interpolate, controlling the output can be very frustrating for the artist.

REQUIREMENT 2. *The interpolant* $\hat{y}$ *should be* C^1 *in* $\hat{\mathbf{x}}$. *It should also be continuous in* $\mathbf{x}_i$ *and* y_i.

Small changes to the inputs that cause jumps in the output also make controlling animations difficult.

REQUIREMENT 3. *If* $\mathbf{x}_i$ *span an affine subspace of the configuration space and if* $\hat{\mathbf{x}} - \hat{\mathbf{x}}'$ *is orthogonal to that subspace, then* $\hat{y}(\hat{\mathbf{x}})$ *should equal* $\hat{y}(\hat{\mathbf{x}}')$.

If, for example, we key a layer to an elbow, we do not want that layer affected by changes in knee parameters.

A few requirements are harder to define formally, but are still desirable.

REQUIREMENT 4. *The interpolation scheme should work on high-dimensional configuration spaces.*

REQUIREMENT 5. *The interpolation scheme should not require the user to set too many parameters. It should naturally adapt to the spacing between keyframes and the different scales associated with different dimensions.*

REQUIREMENT 6. *Outside the region bounded by the keyframes, the interpolant should level off to the value at the closest keyframe.*

Extrapolation requires large negative weights, which tend to amplify defects in strokes, so we found it best to avoid it.

REQUIREMENT 7. *Changing a value y_i should not affect the interpolant "far away" from the associated $\mathbf{x}_i$. Changes to the location $\mathbf{x}_i$ should also not affect the interpolant far away.*

The notion of far away is intentionally fuzzy, as it needs to capture the user's perception of which keyframes are unrelated.

These requirements are simple individually, but are quite difficult to satisfy simultaneously. In fact, we are not aware of any prior interpolation scheme that satisfies Requirements 1–5. We present a new interpolation method that we call smooth Voronoi interpolation. It satisfies Requirements 1–5 and, to some degree, Requirements 6 and 7.

4.1 Previous Methods

Prior methods that needed configuration-space interpolation use radial-basis functions [Lewis et al. 2000; Igarashi et al. 2005], or even linear interpolation [Baran et al. 2009]. These techniques and other common methods violate one or more of our requirements. It is impossible to enumerate all interpolation methods that have been developed, but we discuss some of the most popular ones.

Geometric approaches like natural-neighbor coordinates [Sibson 1981], finite element thin-plate spline approaches [Hegland et al. 1997], and bounded biharmonic weights [Jacobson et al. 2011] are difficult to compute in higher dimensions, violating Requirement 4. Nearest-neighbor interpolation satisfies all requirements except smoothness.

Shepard interpolation [Shepard 1968], Moving Least Squares (MLS) [Levin 1998], radial-basis functions, and closely related kriging methods all require a kernel function $\phi_i(\mathbf{x})$ centered at each data point. Typically, the kernel functions are all the same and only depend on the distance from x_i: $\phi_i(\mathbf{x}) = \phi(\|\mathbf{x} - \mathbf{x}_i\|)$. The function $\phi(r)$ may be a Gaussian, $1/r^2$, or any of a number of other functions. We briefly review the schemes and then discuss which requirements are violated for which kernel functions.

The RBF interpolant is

$$\hat{y}(\hat{\mathbf{x}}) = \sum_i w_i \phi_i(\hat{\mathbf{x}}), \tag{2}$$

where for each j, w_i's satisfy the equation: $\sum_i w_i \phi_i(\mathbf{x}_j) = y_j$, and can thus be found by solving a linear system. This interpolant is not affine invariant (adding a constant to all the y_i does not result in a shifted interpolant) so commonly a constant or a polynomial term is added. Alternatively, normalized RBFs can be used

$$\hat{y}(\hat{\mathbf{x}}) = \sum_i \frac{w_i \phi_i(\hat{\mathbf{x}})}{\sum_j \phi_j(\hat{\mathbf{x}})}, \tag{3}$$

with w_i chosen to guarantee interpolation.

MLS interpolation is defined using a class $\mathcal{F}$ of functions on $\mathbb{R}^d$, often linear functions. The interpolant is then $f_{\hat{\mathbf{x}}}(\hat{\mathbf{x}})$, where $f_{\hat{\mathbf{x}}}$ is a local least-squares fit to the data.

$$f_{\hat{\mathbf{x}}} = \arg\min_{f \in \mathcal{F}} \sum_i \phi_i(\hat{\mathbf{x}}) \| f(\mathbf{x}_i) - y_i \|^2 \tag{4}$$

If $\mathcal{F}$ is the class of constant functions, the MLS interpolant reduces to the Shepard interpolant.

$$\hat{y}(\hat{\mathbf{x}}) = \sum_i \frac{\phi_i(\hat{\mathbf{x}})}{\sum_j \phi_j(\hat{\mathbf{x}})} y_i \tag{5}$$

These schemes satisfy Requirement 4 and, as long as ϕ is smooth, Requirement 2. They generally do not satisfy Requirement 6. Whether the other requirements are satisfied depends on the kernel function.

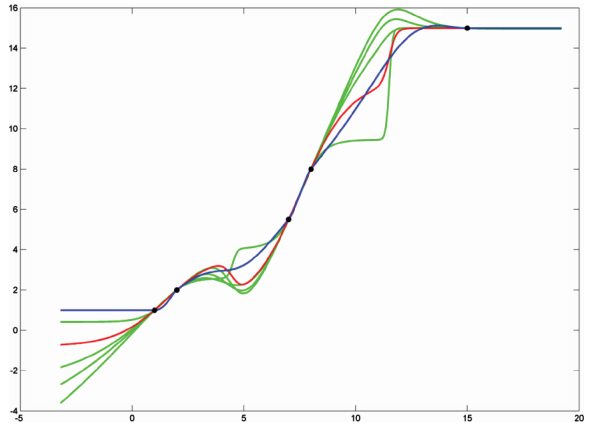


Fig. 6. A comparison of our method (blue) and normalized Gaussian radial-basis functions with kernel radii 1, 2, 3, 4, 5 (green and red). The RBF interpolant with kernel radius 2 is shown in red. This radius is both large enough to cause an overshoot after the point at $x = 2$ and small enough to have a step around $x = 12$, demonstrating that picking a good kernel radius can be an impossible task.

Interpolation (Requirement 1). Radial-basis functions guarantee interpolation by construction, but Shepard and MLS methods do not necessarily interpolate the data points, for example, if ϕ is a Gaussian. Interpolation for these methods is guaranteed if $\phi(r)$ goes to infinity as r approaches zero or if $\phi_i(\mathbf{x}_j) = 0$ for $i \neq j$ (only for the Shepard method).

Affine Subspace (Requirement 3). If the kernel is a Gaussian, for example, $\phi_i(\mathbf{x}) = e^{-\|\mathbf{x}-\mathbf{x}_i\|^2}$, and if $\mathbf{x}'$ is $\mathbf{x}$ displaced in a direction normal to $\mathbf{x} - \mathbf{x}_i$, so $(\mathbf{x} - \mathbf{x}_i) \cdot (\mathbf{x}' - \mathbf{x}) = 0$, then $\phi_i(\mathbf{x}') = e^{-\|\mathbf{x}'-\mathbf{x}\|^2} \phi_i(\mathbf{x})$. In other words, the value of ϕ_i is scaled by a function of only the displacement distance. This property ensures that Requirement 3 is satisfied when the kernel is a Gaussian for Shepard, MLS, and normalized RBFs. Other kernels, such as $1/r^2$, violate this requirement.

It is tempting to take an interpolant that does not satisfy Requirement 3 and apply it to the affine span of $\mathbf{x}_i$ computed explicitly, for example, using PCA. However, a slight perturbation of $\mathbf{x}_i$ can change the dimensionality of the relevant subspace, making such a scheme not continuous.

Parameters (Requirement 5). None of the interpolation methods we discuss (including ours) is invariant to anisotropic scaling of the configuration space. Therefore, the configuration-space variables should be scaled to roughly correspond to their relative importance. However, methods that use a fixed-radius Gaussian kernel are not invariant to *isotropic* scaling and are very sensitive to the kernel radius. Gaussian kernels that are too narrow lead to abrupt transitions in the animation, while overly wide kernels lead to overshooting and, for RBFs, poor conditioning. Figure 6 illustrates that a good kernel radius may not even exist if the keyframes are not uniformly spaced. Specifying a separate covariance matrix for each ϕ_i can address this issue, but is an undue burden on the user.

Local Influence (Requirement 7). If ϕ is nonzero everywhere, changes to $\mathbf{x}_i$ or y_i affect the interpolants produced by all three methods everywhere. If ϕ_i's are locally supported, Shepard and MLS interpolants at $\mathbf{x}$ do not depend on $\mathbf{x}_i$ or y_i if $\phi_i(\mathbf{x}) = 0$. However, care must be taken to ensure that at every point at least one ϕ_i is nonzero; otherwise, the interpolant is undefined. For radial-basis

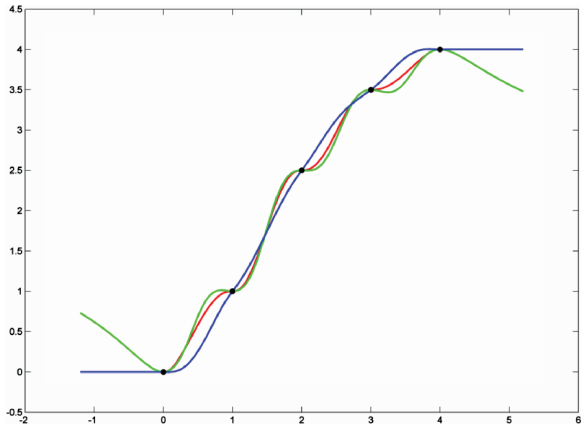


Fig. 7. A comparison of our method (blue), Shepard interpolation with the $1/d^2$ kernel (green), and using our ϕ's directly (red). The task is to interpolate the five regularly spaced points shown. The steps produced by these prior methods would manifest themselves as stuttering during animation playback.

functions, even if ϕ_i's are locally supported, as long as each $\mathbf{x}_i$ is in the support regions of neighboring ϕ's, changing y_i or $\mathbf{x}_i$ affects the interpolant everywhere: the inverse of the matrix with i, j^{th} entry $\phi_i(\mathbf{x}_j)$ is likely to be dense even if the matrix itself is sparse.

Derivatives. Shepard interpolation tends to have problems with derivatives at the keyframes (Figure 7) being zero, which leads to stuttering motions. These can be alleviated by blending local linear fits instead of the values y_i, for instance, using

$$\hat{y}(\hat{\mathbf{x}}) = \sum_i \frac{\phi_i(\hat{\mathbf{x}})}{\sum_j \phi_j(\hat{\mathbf{x}})} f_{\mathbf{x}_i}(\hat{\mathbf{x}}) \tag{6}$$

with $f_{\mathbf{x}_i}$ defined as in Eq. (4). Similar ideas with different kernel functions and local interpolants were explored for interpolation over $\mathbb{R}^2$ by Franke [1977]. This modification, however, does not address the requirement violations discussed before.

4.2 Smooth Voronoi Interpolation

We design our interpolant as a blend of local linear interpolants, similar to Eq. (6), but with important differences. Instead of a single kernel $\phi(r)$, we construct a specialized kernel $\phi_i(\mathbf{x})$ around each data point that is adapted to the distribution of keyframes around $\mathbf{x}_i$. This kernel is a smooth bump centered at $\mathbf{x}_i$ that falls off to zero towards every other keyframe, so $\phi_i(\mathbf{x}_j) = \delta_{ij}$, where δ_{ij} is the Kronecker delta. The linear fits around each keyframe are computed also using these kernels.

The functions $\phi_i(\mathbf{x})$ are defined based on a smooth step function. A Hermite cubic step could be used, but we have found the quadratic-linear-quadratic step for $\gamma = 0.2$ to produce better results because its maximal derivative is smaller, which leads to smoother-looking interpolation.

$$S_\gamma(t) = \begin{cases} 0 & \text{if } t \le 0 \\ 1 & \text{if } t \ge 1 \\ \frac{t^2}{2\gamma - \gamma^2} & \text{if } 0 < t \le \gamma \\ 1 - \frac{(1-t)^2}{2\gamma - \gamma^2} & \text{if } 1-\gamma < t < 1 \\ 0.5 + \frac{t-0.5}{1-\gamma} & \text{if } \gamma < t \le 1-\gamma \end{cases} \tag{7}$$

Figure 8 shows the step for different values of γ.

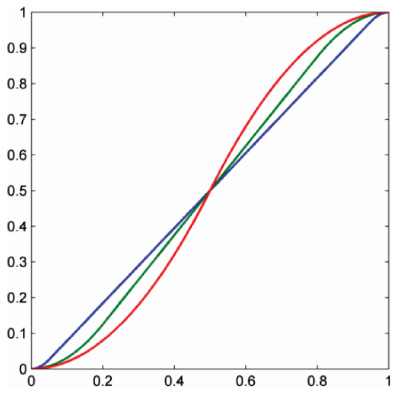


Fig. 8. The function $S_\gamma(t)$ for $\gamma = 0.05$ (blue), 0.2 (green), 0.5 (red).

To compute the kernel ϕ_i, we project $\hat{\mathbf{x}}$ onto the line connecting $\mathbf{x}_j$ to $\mathbf{x}_i$ for each $j \neq i$ and evaluate S for the resulting line parameter value, so that the result is 1 if the projection is at or past $\mathbf{x}_i$ and 0 if it is at or past $\mathbf{x}_j$. We then take the product of the smooth steps toward each keyframe

$$\phi_i'(\hat{\mathbf{x}}) = \prod_{j \neq i} S\left(\frac{(\hat{\mathbf{x}} - \mathbf{x}_j) \cdot (\mathbf{x}_i - \mathbf{x}_j)}{\|\mathbf{x}_i - \mathbf{x}_j\|^2} \right) \tag{8}$$

and normalize the result: $\phi_i(\hat{\mathbf{x}}) = \phi_i'(\hat{\mathbf{x}}) / \sum_j \phi_j'(\hat{\mathbf{x}})$. These kernels have several useful properties: they vary between 0 and 1, they are as smooth as S, and they are constant in directions orthogonal to the affine subspace spanned by $\mathbf{x}_i$'s. Also, $\phi_i(\hat{\mathbf{x}}) > 0$ precisely when $(\hat{\mathbf{x}} + \mathbf{x}_i)/2$ is in the Voronoi region of $\mathbf{x}_i$ (hence the name of the interpolation). However, because each ϕ_i is smooth and reaches its extreme values at the keyframes, the gradient of ϕ_i at each $\mathbf{x}_j$ is zero, and if we use Eq. (5) we will get the stepping artifacts discussed earlier (Figure 7).

To avoid the stepping artifacts, instead of blending y_i's directly, we blend linear functions that approximate the data near keyframes. To ensure interpolation, the linear function associated with keyframe i must pass through y_i, so we can write it as $f_{\mathbf{x}_i}(\mathbf{x}) = y_i + \mathbf{g}_i \cdot (\hat{\mathbf{x}} - \mathbf{x}_i)$, where $\mathbf{g}_i$ is the gradient of $f_{\mathbf{x}_i}$. Then Eq. (6) becomes

$$\hat{y} = \sum_{i=1}^{n} \phi_i(\hat{\mathbf{x}}) \left(y_i + \mathbf{g}_i \cdot (\hat{\mathbf{x}} - \mathbf{x}_i) \right). \tag{9}$$

It remains to estimate the gradients $\mathbf{g}_i$. For $f_{\mathbf{x}_i}(\mathbf{x}_j)$ to equal y_j, the directional derivative in the direction towards $\mathbf{x}_j$ must equal $\frac{y_j - y_i}{\|\mathbf{x}_j - \mathbf{x}_i\|}$. The directional derivative of $f_{\mathbf{x}_i}$ towards $\mathbf{x}_j$ is given by $\mathbf{g}_i \cdot \frac{\mathbf{x}_j - \mathbf{x}_i}{\|\mathbf{x}_j - \mathbf{x}_i\|}$. Ideally, the directional derivative would match the desired value exactly.

$$\mathbf{g}_i \cdot \frac{\mathbf{x}_j - \mathbf{x}_i}{\|\mathbf{x}_j - \mathbf{x}_i\|} = \frac{y_j - y_i}{\|\mathbf{x}_j - \mathbf{x}_i\|}$$

However, unless y_i's lie on a linear function of $\mathbf{x}$, we will not be able to satisfy all of these constraints. Instead, we treat these as least-squares constraints, and, combining them, we formulate a quadratic energy that we minimize to find $\mathbf{g}_i$.

$$E(\mathbf{g}_i) = \sum_{j \neq i} a_{ij}^2 \left(\mathbf{g}_i \cdot \frac{\mathbf{x}_j - \mathbf{x}_i}{\|\mathbf{x}_j - \mathbf{x}_i\|} - b_{ij} \frac{y_j - y_i}{\|\mathbf{x}_j - \mathbf{x}_i\|} \right)^2 \tag{10}$$

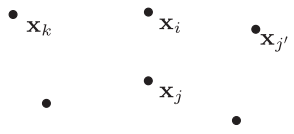
Here a_{ij} is used to ensure that only immediate neighbors have an influence on the gradient estimate and b_{ij} is used for forcing the gradient at extreme keyframes to zero to avoid overshooting and satisfy Requirement 6. To prevent distant keyframes from influencing the gradient, we smoothly reduce a_{ij} using the kernels ϕ_i as $\mathbf{x}_{ij}$ is moved away from $\mathbf{x}_i$. Specifically, we set

$$a_{ij} = \phi_i(\alpha \mathbf{x}_i + (1-\alpha)\mathbf{x}_j) \tag{11}$$

with $\alpha = 0.5$. To set the gradient to zero at extreme points, we determine whether there are keyframes on both sides of $\mathbf{x}_i$ in the $\mathbf{x}_j$ direction.

$$b_{ij} = S\left(-\beta \cdot \min_{k \neq i} \frac{(\mathbf{x}_k - \mathbf{x}_i)\cdot(\mathbf{x}_j - \mathbf{x}_i)}{\|\mathbf{x}_k - \mathbf{x}_i\|\|\mathbf{x}_j - \mathbf{x}_i\|}\right) \tag{12}$$

with $\beta = 5$. In the 2D configuration shown here, $\mathbf{x}_i$ is an extreme point with respect to $\mathbf{x}_j$, so $b_{ij} = 0$, but not with respect to $\mathbf{x}_{j'}$, so $b_{ij'} = 1$. The minimizing $\mathbf{x}_k$ is the same for both cases.



The problem of minimizing the energy E is typically underconstrained, as no information is available in directions orthogonal to the affine subspace spanned by the keyframes. To satisfy Requirement 3, the directional derivative needs to be zero in these directions. This requirement is achieved by adding the Tikhonov regularization term $\epsilon\|\mathbf{g}_i\|^2$ to the energy function with $\epsilon = 0.2$. We then minimize E using singular value decomposition.

Like all discussed schemes, this interpolation method only depends linearly on the y_i's because $\mathbf{g}_i$ is a linear function of y_i's. Therefore, we only need to compute basis functions $w_i(\mathbf{x})$ such that $\hat{y}(\hat{\mathbf{x}}) = \sum_i w_i(\hat{\mathbf{x}})y_i$ and use them to interpolate many different strokes or other y's. Note that these basis functions are different from the kernel functions $\phi_i(\mathbf{x})$ because w_i's incorporate the effect of using the gradient in the interpolation. Example basis functions generated with smooth Voronoi interpolation are shown in Figure 9. Because configuration space is hard to visualize, we show an application of our interpolation to spatial keyframing [Choi et al. 2008] in Figure 10. The accompanying video demonstrates the practical impact of the interpolant on a 3D painting of an elf's face, comparing our method to Gaussian kernel radial-basis functions and linear moving least squares with $1/r^2$ weights.

We now discuss the extent to which smooth Voronoi interpolation satisfies our requirements. Requirement 1 (interpolation) is satisfied because $\phi_i(\mathbf{x}_j) = \delta_{ij}$ and $f_{\mathbf{x}_i}(\mathbf{x}_i) = y_i$.

Requirement 2 (smoothness) is also satisfied. The interpolant is C^1 in $\hat{\mathbf{x}}$ because ϕ_i is C^1 in $\hat{\mathbf{x}}$. The interpolant is linear in y_i's and therefore smooth. Continuity in $\mathbf{x}_i$ depends on the minimum of E varying continuously with $\mathbf{x}_i$. The a_{ij} and b_{ij} are continuous in $\mathbf{x}_i$ and the regularization ensures the minimization left-hand side is nonsingular, which implies the continuity of the inverse.

Requirement 3 is satisfied because neither $\phi_i(\hat{\mathbf{x}})$ nor $\mathbf{g}_i \cdot \hat{\mathbf{x}}$ changes when $\hat{\mathbf{x}}$ is displaced in the direction orthogonal to the affine span of the keyframes.

Although the presented method has parameters α, β, γ, and ϵ, they do not need to be adjusted depending on keyframe positions, unlike the σ's of Gaussian kernels. We used the same parameter values for all of the results shown in this article ($\alpha = 0.5$, $\beta = 5$, $\gamma = 0.2$, $\epsilon = 0.2$). We therefore consider Requirement 5 satisfied.

Limitations. Like prior methods, our scheme does not always satisfy Requirement 6: the gradients at extreme keyframes may not end up zero and overshooting can occur. Although it is not hard to construct examples where this happens, we did not run into this issue when producing our animated characters.

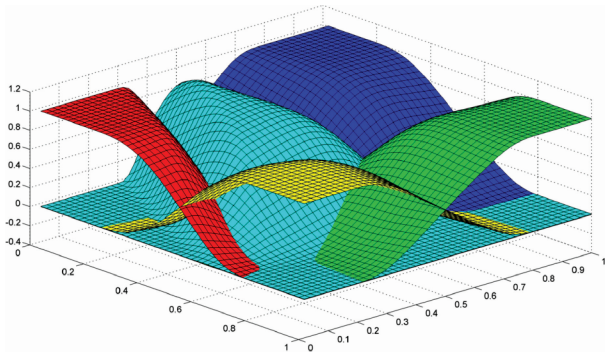


Fig. 9. Basis functions w generated by our interpolation scheme in 2D. The "keyframes" are at (0.2, 0.2), (0.8, 0.8), (0.2, 0.8), (0.8, 0.2), and (0.3, 0.4).

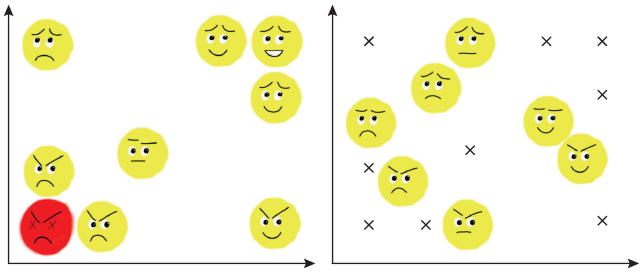


Fig. 10. A painted smiley face has been keyframed to show various expressions. The configuration space is defined by the 2D position of the face. The left figure shows all keyframes at their respective 2D positions. The right figure shows interpolated faces superimposed on the keyframes.

Requirement 7 is also partially satisfied. Changing $\mathbf{x}_i$ changes the function globally because each ϕ_i depends on all of the keyframe locations. On the other hand, dependence on y_i is more local. The kernels ϕ_i do not extend beyond the Voronoi neighbors of $\mathbf{x}_i$: the support regions for some of the ϕ_i's for Figure 10 are shown in Figure 11. The gradient $\mathbf{g}_i$ only depends on y_j's for which a_{ij} is nonzero, namely those at $\mathbf{x}_j$'s within the Voronoi cell of $\mathbf{x}_i$ scaled by $2/(1-\alpha)$. The interpolant at $\hat{\mathbf{x}}$ depends only on the gradients associated with the keyframes for which $\phi_i(\hat{\mathbf{x}})$ is nonzero. The basis functions thus go to zero away from their associated keyframes (Figure 9).

Another limitation of our interpolant is that it may have local extrema away from the keyframes, which can occasionally lead to unintuitive interpolation (Figure 12). All other interpolation methods discussed share this limitation.

Complexity. We perform different sets of computations as part of the interpolation method. When the user sets or moves a keyframe, we must recompute everything from scratch. When the user changes the current configuration $\hat{\mathbf{x}}$, we must recompute the w_i's, but do not need to recompute $\mathbf{g}_i$'s. To actually blend the stroke positions and opacities, we only need to use the already computed w_i's.

For n keyframes with a d-dimensional configuration space, evaluating $\phi_i'(\hat{\mathbf{x}})$ takes $O(nd)$ time. Evaluating all of the ϕ_i's at the same point $\hat{\mathbf{x}}$ thus takes $O(n^2d)$ time. To find the weights w_i, we must evaluate Eq. (9) for each i, with different y's. When the ϕ_i's and

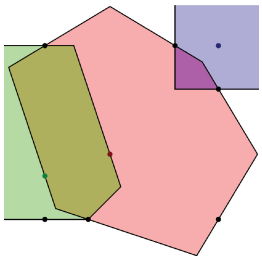


Fig. 11. The supports of the kernels ϕ_i are shown here for three of the keyframes in Figure 10. These are the Voronoi cells of the points, each scaled by a factor of two around its data point. Note that the regions of influence of the keyframes are larger due to the gradients.

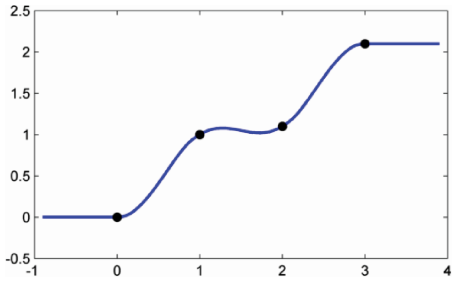


Fig. 12. An example in which the values at keyframes increase monotonically, but the interpolant does not.

$\mathbf{g}_i$'s are already computed, this calculation takes $O(n^2 + nd)$ time. Overall, when the user changes $\hat{\mathbf{x}}$, computing w_i's takes $O(n^2 d)$ time. When the user changes one or more of the $\mathbf{x}_i$'s, the $\mathbf{g}_i$'s need to be recomputed. Computing the a_{ij}'s is the bottleneck for this computation and each can be computed in $O(n^2 d)$ time (because all of the ϕ' must be computed to compute ϕ). The total time spent computing a_{ij}'s is thus $O(n^4 d)$. Many optimizations are possible, especially if one is allowed to vary the formulation slightly, but in all of our examples, we use fewer than 20 keyframes per layer and interpolation takes no noticeable time compared to the time required for rendering. On a single CPU thread of a Core i7 930 desktop, with $n = 20$ and $d = 100$, computing the interpolant from scratch takes 35ms, while only recomputing w_i's takes 120μs. For $n = 40$, the times go up to 540ms and 470μs, respectively, roughly as predicted by the asymptotics.

5. RESULTS

We authored and animated several example characters to demonstrate our skinning deformation system and the range of applications for our configuration-space and temporal keyframing methods. All characters are purely stroke-based paintings rendered using mixed-order compositing [Baran et al. 2011]. No traditional computer graphics elements are used.

The accompanying video shows the examples from Figures 3 and 10, as well as all other results discussed in this section. We found that proxy geometry animation is best for gross movement of many strokes in tandem. Configuration-space keyframing is best for fine-scale stroke control, to sculpt the way strokes move on a detailed level that is not easily represented by the proxy geometry.

In the smiley-face animation, the configration-space position is represented by a bomb icon, and the eyes of the face stay fixed on the bomb position thanks to the configuration-space keyframing.

Fig. 13. Painted lighting effects such as the highlight, shading, and shadow of this apple can be achieved by including the light's position in the configuration space and setting stroke position and stroke opacity keyframes to the light's position. © Disney

The dog example (Figure 2) demonstrates both skinning and configuration-space keyframing. In the rig's blend shape, only the snout, ears, and jaw move. We use stroke position keyframing to make the hair stand up, create eyebrows, animate the pupils, and pull the cheeks out. Opacity keyframing is used to add highlights and shadows around the cheeks and eyebrows.

Lighting is a crucial element in many animations. Figure 13 shows a painted lighting example in which a light source's position is included in the configuration space. Keyframes for stroke positions and opacities are added for different positions of the light source to create the illusion of a moving highlight, shadow, and shading. In this example, no actual lighting calculations are made. Everything is accomplished with configuration-space keyframing.

The blowfish animation (Figure 14) makes use of all aspects of our animation system. The mesh animation was created with blend shapes and skeletal rigging and contains the overall deformation of the body and the movement of the fins. The eyes, however, do not move in the original animation and were animated by smudging and keyframing the pupils to a 2D eye target position in the configuration space. The spikes were folded out with the smudging tool and keyframed to the blend-shape weight that corresponds to the blown-up shape of the fish. Spike stretching is avoided by constraining all points along one spike to a single point on the surface. The blend-shape weight parameter also causes the cheeks and lips to turn red via opacity keyframing. As the fish inflates, the distances between paint strokes increase, leading to gaps in the surface. We filled these gaps with additional paint strokes that are keyframed to appear only as the body expands. The difference is visible in the blowfish comparison segment in the video. The blinking eyes were realized by keying opacity to a separate parameter in the configuration space. Figure 15 shows three frames of the blowfish animation with and without the configuration-space effects.

Finally, the ballerina character, shown in Figure 1, has a detailed rig that allows the animated painting to rely heavily on stroke skinning. However, the input animation exhibits a number of rigging artifacts, including the leg protruding through the skirt and the collapsing deformation in the shoulders. Such artifacts can easily be remedied with our configuration-space keyframing system, as illustrated in Figure 16. In the particular case of the shoulders, we modified the behavior of the problematic strokes by keying new stroke positions to the angles of the shoulder joints using 10 keyframes to fix both shoulders. The leg protrusion was fixed with three pose keyframes to move the skirt upward. Apart from alleviating issues in the input animation, we also added an embellishing effect animation that causes the ballerina's tutu to twist in reaction to her pirouette. This effect was achieved with three temporal keyframes during her spin that marks the maximum deformation of the fabric.

Table I gives complexity statistics for our results, including the number of layers, strokes, configuration-space position keyframes,

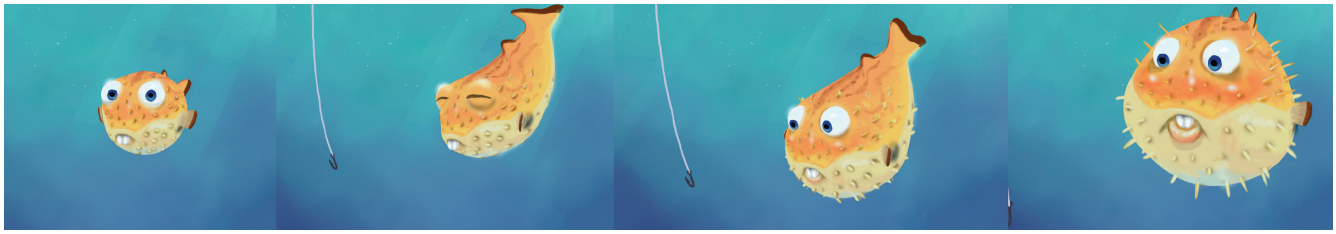


Fig. 14. A blowfish gets a shock when catching sight of a fishing hook. © Disney

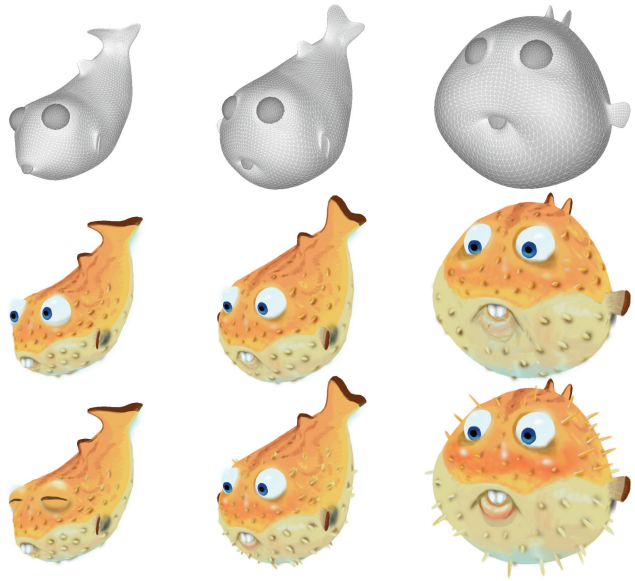


Fig. 15. An illustration of the different authoring steps for the blowfish example. The top row shows the animated proxy geometry, the middle row shows the painted result with skinning deformation only, and the bottom row shows the result with configuration-space keyframing. The middle row lacks the blinking and spike animations, and shows holes in the blown-up pose because the original paint strokes have moved too far apart. © Disney

Table I. Complexity Statistics for Our Results

Result	Layers	Strokes	CS Pos. Keys	CS Op. Keys	Temp. Keys
Dog	42	10,095	43	19	0
Blowfish	20	8,281	15	10	5
Ballerina	69	7,436	60	4	54
Apple	24	807	41	15	0

configuration-space opacity keyframes, and temporal keyframes for the dog, blowfish, ballerina, and apple examples.

6. CONCLUSION

We have presented a system for authoring and animating painterly characters that incorporates much of the expressive freedom of 2D concept painting into the character animation pipeline. In essence, our work upgrades painting from its restricted role in concept design and texturing to become an integrated piece of expressive depiction that impacts modeling, rigging, posing, lighting, and rendering. We show how painted strokes can be deformed together with the character's surface. Our configuration-space and temporal keyframing

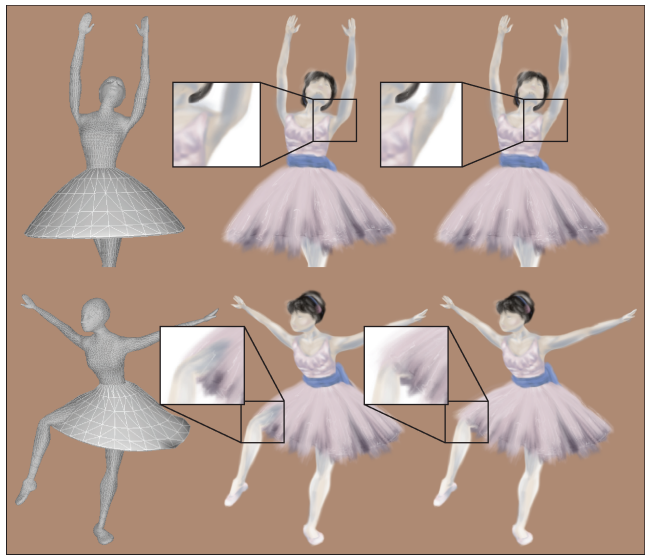


Fig. 16. In the ballerina example, we used configuration-space keyframing (right column) to fix issues that arose from the proxy geometry animation (left and middle columns), such as the skinning artifacts in the shoulder regions and the leg protruding through the skirt. © Disney

system allows artists to fine-tune the movement of strokes in order to accomplish stroke-level effects that are difficult or impossible to achieve using more traditional modeling and animation tools. We have demonstrated results for facial animation, animated lighting, and full body animation ranging in style from comical and cartoony to fine-art impressionism.

Although our system gives the artist new control over painterly animation, it also comes with many limitations that offer exciting opportunities for future work. The freedom to shape the depiction of painted strokes in different poses, under different lighting conditions, and at different times can mean that character authoring becomes more laborious. The extra time required for painting is somewhat balanced by the possibility of using a simpler geometric model and rigging deformations. Nonetheless, an interesting area of future work lies in exploring methods to fill a character with strokes without having to paint each and every one individually. Such a system must strike an effective balance between its automatic nature and the level of artistic control.

Lighting represents a challenge in and of itself. Our current system allows painted lighting effects, such as the moving shadow, highlight, and shading on the apple in Figure 13. This example is somewhat simplistic, and the amount of work involved in creating extremely complex lighting effects can make authoring them

impractical. Incorporating automatic lighting into our animation system is thus an important area of future work. As mentioned before, the key challenge involves automating the process without stealing the artist's expressive control.

Image-space effects, such as silhouettes, are not easily accomplished by our system. Incorporating the WYSIWYG annotation system [Kalnins et al. 2002] for silhouettes, creases, and hatching is thus an interesting future work direction. On the technical side, our configuration-space keyframe interpolation algorithm avoids stuttering when interpolating multiple keyframes, but does not provide precise control over the interpolation procedure. Generalizing the spline control typically found in animation packages to higher dimensions in an effective way is a challenging area of future work.

A final future work direction is more conceptual. An animated painting brings together two seemingly incompatible worlds since 3D movement must be conveyed using a medium that has been static for thousands of years. Our system attempts to give the animator explicit control over this movement. Discovering the boundaries of this control and how to move past them to create effects and styles we have not yet dreamed is a future direction that relies on art just as much as it relies on science.

ACKNOWLEDGMENTS

We thank Eakta Jain and Jessica Hodgins for providing us with the ballerina motion capture. Thanks to Maurizio Nitti for creating the blowfish example. We thank Alec Jacobson, Olga Sorkine, and Jovan Popović for helpful early discussions regarding smooth Voronoi interpolation. We also thank the anonymous reviewers for feedback that helped to greatly improve the article.

REFERENCES

AKENINE-MOLLER, T., HAINES, E., AND HOFFMAN, N. 2008. *Real-Time Rendering* 3rd Ed. A. K. Peters, Ltd., Natick, MA.

BARAN, I., SCHMID, J., SIEGRIST, T., GROSS, M., AND SUMNER, R. W. 2011. Mixed-order compositing for 3d painting. *ACM Trans. Graph. 30*, 6, 132:1–132:6.

BARAN, I., VLASIC, D., GRINSPUN, E., AND POPOVIC, J. 2009. Semantic deformation transfer. *ACM Trans. Graph. 28*, 3, 36:1–36:6.

BENARD, P., BOUSSEAU, A., AND THOLLOT, J. 2011. State-of-the-art report on temporal coherence for stylized animations. *Comput. Graph. Forum 30*, 8, 2367–2386.

BOURGUIGNON, D., CANI, M.-P., AND DRETTAKIS, G. 2001. Drawing for illustration and annotation in 3d. *Comput. Graph. Forum 20*, 3, 114–122.

BOUSSEAU, A., KAPLAN, M., THOLLOT, J., AND SILLION, F. X. 2006. Interactive watercolor rendering with temporal coherence and abstraction. In *Proceedings of the 4th International Symposium on Non-Photorealistic Animation and Rendering*. 141–149.

CHOI, B., YOU, M., AND NOH, J. 2008. Extended spatial keyframing for complex character animation. *Comput. Anim. Virtual Worlds 19*, 3–4, 175–188.

COHEN, J. M., HUGHES, J. F., AND ZELEZNIK, R. C. 2000. Harold: A world made of drawings. In *Proceedings of the 1st International Symposium on Non-Photorealistic Animation and Rendering*. 83–90.

CURTIS, C. J., ANDERSON, S. E., SEIMS, J. E., FLEISCHER, K. W., AND SALESIN, D. H. 1997. Computer-generated watercolor. In *Proceedings of the Annual ACM SIGGRAPH Conference on Computer Graphics and Interactive Techniques*. 421–430.

DANIELS, E. 1999. Deep canvas in disney's tarzan. In *Proceedings of the ACM SIGGRAPH Conference Abstracts and Applications (SIGGRAPH'99)*. 200.

FRANKE, R. 1977. Locally determined smooth interpolation at irregularly spaced points in several variables. *IMA J. Appl. Math. 19*, 4, 471–482.

HEGLAND, M., ROBERTS, S., AND ALTAS, I. 1997. Finite element thin plate splines for surface fitting. In *Proceedings of the Conference on Computational Techniques and Applications (CTAC'97)*. 289–296.

HERTZMANN, A. 2003. A survey of stroke-based rendering. *IEEE Comput. Graph. Appl. 23*, 4, 70–81.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *Proceedings of the Annual ACM/SIGGRAPH Conference on Computer Graphics and Interactive Techniques*. 327–340.

HERTZMANN, A. AND PERLIN, K. 2000. Painterly rendering for video and interaction. In *Proceedings of the 1st International Symposium on Non-Photorealistic Animation and Rendering (NPAR'00)*. ACM Press, New York, 7–12.

IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. 2005. Spatial keyframing for performance-driven animation. In *Proceedings of the ACM/SIGGRAPH/Eurographics Symposium on Computer Animation*. 107–116.

JACOBSON, A., BARAN, I., POPOVIC, J., AND SORKINE, O. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph. 30*, 4, 78:1–78:8.

KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L.,WEBB, M., HUGHES, J. F., AND FINKELSTEIN, A. 2002. Wysiwyg npr: Drawing strokes directly on 3d models. *ACM Trans. Graph. 21*, 3, 755–762.

KALOGERAKIS, E., NOWROUZEZAHRAI, D., BRESLAV, S., AND HERTZMANN, A. 2011. Learning hatching for pen-and-ink illustration of surfaces. *ACM Trans. Graph. 31*, 1, 1:1–1:17.

KEEFE, D., ZELEZNIK, R., AND LAIDLAW, D. 2007. Drawing on air: Input techniques for controlled 3d line illustration. *IEEE Trans. Vis. Comput. Graph. 13*, 1067–1081.

KEEFE, D. F., FELIZ, D. A., MOSCOVICH, T., LAIDLAW, D. H., AND LAVIOLA, J. J. JR. 2001. Cavepainting: A fully immersive 3d artistic medium and interactive experience. In *Proceedings of the ACM Symposium on Interactive 3D Graphics*. 85–94.

KOWALSKI, M. A., MARKOSIAN, L., NORTHRUP, J. D., BOURDEV, L., BARZEL, R., HOLDEN, L. S., AND HUGHES, J. F. 1999. Art-based rendering of fur, grass, and trees. In *Proceedings of the Annual ACM/SIGGRAPH Conference on Computer Graphics and Interactive Techniques*. 433–438.

LEVIN, D. 1998. The approximation power of moving least-squares. *Math. Comput. 67,* 224, 1517–1532.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformations: A unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the Annual ACM/SIGGRAPH Conference on Computer Graphics and Interactive Techniques*. 165–172.

LIN, L., ZENG, K., LV, H., WANG, Y., XU, Y., AND ZHU, S.-C. 2010. Painterly animation using video semantics and feature correspondence. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering (NPAR'10)*. ACM Press, New York, 73–80.

LITWINOWICZ, P. 1997. Processing images and video for an impressionist effect. In *Proceedings of the Annual ACM/SIGGRAPH Conference on Computer Graphics and Interactive Techniques*. 407–414.

LU, J., SANDER, P. V., AND FINKELSTEIN, A. 2010. Interactive painterly stylization of images, videos and 3d animations. In *Proceedings of the ACM Symposium on Interactive 3D Graphics and Games*. 127–134.

MARKOSIAN, L., KOWALSKI, M. A., TRYCHIN, S. J., BOURDEV, L. D., GOLDSTEIN, D., AND HUGHES, J. F. 1997. Real-time nonphotorealistic rendering. In *Proceedings of the Annual ACM/SIGGRAPH Conference on Computer Graphics and Interactive Techniques*. 415–420.

MAYA. 2012. http://www.autodesk.com/maya.

MEIER, B. J. 1996. Painterly rendering for animation. In *Proceedings of the Annual ACM/SIGGRAPH Conference on Computer Graphics and Interactive Techniques*. 477–484.

NGO, T., CUTRELL, D., DANA, J., DONALD, B., LOEB, L., AND ZHU, S. 2000. Accessible animation and customizable graphics via simplicial configuration modeling. In *Proceedings of the Annual ACM/SIGGRAPH Conference on Computer Graphics and Interactive Techniques*. 403–410.

O'DONOVAN, P. AND HERTZMANN, A. 2012. Anipaint: Interactive painterly animation from video. *IEEE Trans. Vis. Comput. Graph. 18*, 3, 475–487.

PAINT EFFECTS. 2011. Painting in 3d using paint effects. In *Autodesk Maya Learning Resources*. http://download.autodesk.com/us/maya/2011help.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Realtime hatching. In *Proceedings of the Annual ACM/SIGGRAPH Conference on Computer Graphics and Interactive Techniques*. 579–584.

RADEMACHER, P. 1999. View-dependent geometry. In *Proceedings of the Annual ACM/SIGGRAPH Conference on Computer Graphics and Interactive Techniques*. 439–446.

RIVERS, A., IGARASHI, T., AND DURAND, F. 2010. 2.5d cartoon models. *ACM Trans. Graph. 29*, 4, 59:1–59:7.

SCHMID, J., SENN, M. S., GROSS, M., AND SUMNER, R. W. 2011. Overcoat: An implicit canvas for 3d painting. *ACM Trans. Graph. 30*, 4, 28:1–28:10.

SHEPARD, D. 1968. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 23rd ACM National Conference*. ACM Press, New York, 517–524.

SIBSON, R. 1981. A brief description of natural neighbor interpolation. In *Interpolating Multivariate Data*. John Wiley and Sons, 21–36.

SINGH, K. AND KOKKEVIS, E. 2000. Skinning characters using surface oriented free-form deformations. In *Proceedings of the Annual Graphics Interface Conference*. 35–42.

SLOAN, P.-P. J., ROSE, C. F. III, AND COHEN, M. F. 2001. Shape by example. In *Proceedings of the ACM Symposium on Interactive 3D Graphics*. 135–144.

TEECE, D. 1998. 3d painting for non-photorealistic rendering. In *ACM/SIGGRAPH Conference Abstracts and Applications*. ACM Press, New York, 248.

TOLBA, O., DORSEY, J., AND MCMILLAN, L. 2001. A projective drawing system. In *Proceedings of the ACM Symposium on Interactive 3D Graphics*. 25–34.

ZHAO, M. AND ZHU, S.-C. 2011. Customizing painterly rendering styles using stroke processes. In *Proceedings of the ACM/SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering (NPAR'11)*. ACM Press, New York. 137–146.

Received August 2012; revised February 2013; accepted May 2013