Hierarchical Motion Brushes for Animation Instancing

Antoine Milliez[1,2*] Gioacchino Noris[2] Ilya Baran[2,3] Stelian Coros[2] Marie-Paule Cani[4]
Maurizio Nitti[2] Alessia Marra[2] Markus Gross[1,2] Robert W. Sumner[1,2]

[1] ETH Zürich [2] Disney Research Zurich [3] Onshape Inc.
[4] Laboratoire Jean Kuntzmann (University of Grenoble, CNRS), Inria

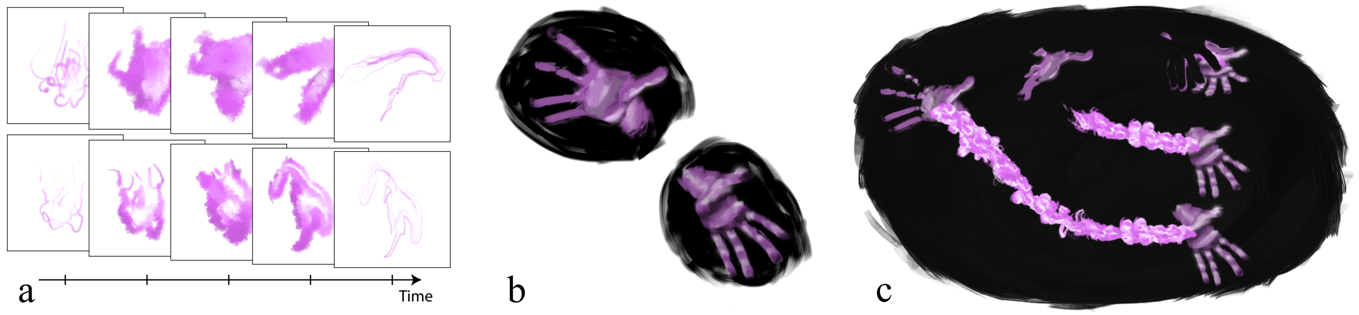

Figure 1: *Hierarchical motion brushes are a powerful tool for creating stylized animated content such as the fire of this example. The user painted a set of motion brushes (a) and uses them in a scene containing two hands (b). By only painting one stroke, a complex fire effect is created that erupts from one hand and reaches the other over time (c).*

Abstract

Our work on “motion brushes” provides a new workflow for the creation and reuse of 3D animation with a focus on stylized movement and depiction. Conceptually, motion brushes expand existing brush models by incorporating hierarchies of 3D animated content including geometry, appearance information, and motion data as core brush primitives that are instantiated using a painting interface. Because motion brushes can encompass all the richness of detail and movement offered by animation software, they accommodate complex, varied effects that are not easily created by other means. To support reuse and provide an effective means for managing complexity, we propose a hierarchical representation that allows simple brushes to be combined into more complex ones. Our system provides stroke-based control over motion-brush parameters, including tools to effectively manage the temporal nature of the motion brush instances. We demonstrate the flexibility and richness of our system with motion brushes for splashing rain, footsteps appearing in the snow, and stylized visual effects.

CR Categories: I.3.4 [Computer Graphics]: Graphics Utilities—Paint systems;

Keywords: sketch-based interaction, hierarchical animation, 3D painting

*e-mail:antoine@disneyresearch.com

1 Introduction

The dominant brush model used in 2D digital painting is remarkably simple yet remarkably powerful: a brush image is stamped repeatedly along a drawn curve to form a painted stroke. Parameters such as spacing, rotation jitter, or pen pressure modulation allow customization, but the core operation of image stamping persists. Armed with his or her favorite brushes (custom brush images and parameter settings), a talented artist can create 2D digital masterpieces that attest to the power and flexibility of this concept of “controllable reuse”, where the brush stamps are reused over and over under the direct control of the artist.

The world of 3D animation departs from the painting metaphor with a vast set of tools to control the shape, appearance, and movement of characters and objects within a 3D environment. Animation software provides a huge degree of flexibility to craft and fine tune every minute detail, allowing animators to create extremely rich 3D worlds and bring them to life through movement. Unfortunately, creating high-quality animation is notoriously difficult, time consuming, and expensive. The flexibility over the many details of shape, appearance, and motion that animation software offers often translates into complexity that becomes unwieldy to manage and unintuitive to control. When art direction calls for a stylized look that departs from standard depiction techniques, the problem can be exacerbated since extra care and attention must be given to ensure that the proper visual style is achieved.

Our work combines the core concept of controllable reuse from the traditional 2D brush model with the full power and flexibility of 3D animation to offer a new authoring tool for animated effects. We propose the concept of “motion brushes,” which expand the brush stamp idea, replacing the static brush image with a 3D animated scene comprised of geometry, appearance information, and motion. This animated content is instantiated via a painting interface directly in the 3D environment, giving the artist fast and intuitive control analogous to 2D digital painting. However, because the motion brush can encompass all the richness and detail of appearance and motion offered by animation software, complex and varied animated effects such as footsteps appearing and disappearing in the snow, stylized fire traveling in an arc through the air (Figure 1), or rain drops falling from a cloud and splashing on the ground below can be instantiated with a single painted stroke. Although the core concept of motion brushes is general in nature, we focus on the case of stylized depiction where brush-based interaction is especially fitting and natural.

On a technical level, the primary challenge of our system lies in developing a method to package complex animated content into a brush-based painting interface while still providing adequate control over the authoring process, over parameters relevant to instantiation, and over the temporal nature of the data. In order to effectively manage the complexity inherent in animated content, the core representation of our motion brush system is hierarchical in nature, both in space and time. Simple motion brushes, such as a drop falling through the air or a splash on the ground can be authored independently and then combined into a composite motion brush to create a drop falling and then splashing. This hierarchical construction provides a founded means to deal with complexity during motion brush authoring. In order to achieve an effective instantiation process, our system derives parameter values from scene information such as surface normals while supporting parameter customization via the same intuitive painting interface. Control over random jittering of values such as orientation, position, and visibility ensures a varied, non-repetitive appearance. Since our motion brushes are animated, our system also encompasses control over the temporal nature of instantiated copies, allowing varied start times and temporal advection.

Our core contribution is the motion brush concept that allows detailed animated sequences to be packaged into hierarchical brushes that can be reused in new scenes via painting. This conceptual development is supported by the technical decisions related to hierarchical management, parameter control, and temporal evaluation necessary to make the motion brush concept effective. We demonstrate a number of examples where motion brushes encapsulate complex animation that can be easily instantiated and customized via painting, demonstrating an effective new contribution and workflow in the computer animation pipeline.

2 Related Work

Stroke-based interfaces have been widely used in computer graphics due to their intuitive nature and expressive power. The stamp-based brush model, a core component of modern 2D digital painting software [Photoshop 2012; GIMP 2013], repeatedly applies an image stamp along a drawn curve. By randomizing or cycling different image stamps used (e.g., “image pipes” or “image hoses” in GIMP), interesting 2D brush effects are possible. In [Měch and Miller 2012], an extension to that concept is provided through interactive tools for generating procedural patterns.

The skeletal strokes system [Hsu et al. 1993] builds upon this concept by showing how to effectively deform an image along an entire stroke, rather than simply stamping copies. Other work focuses on simulated media [Lu et al. 2013; Chu et al. 2010] or using stroke operations to fine-tune brush parameters after their application [Kazi et al. 2012]. Our work takes inspiration from these 2D methods to propose a new authoring workflow for 3D animation. Our motion brushes enhance the traditional 2D stamp model to incorporated 3D animated scenes into a stroke-based instancing system. Brush parameters and control of the temporal nature of motion brushes are derived from environmental information or fine-tuned via additional stroke operations.

Painting interfaces have also been used for instancing geometry in 3D applications. ZBrush [ZBrush 2013] allows users to “paint” with 3D meshes in order to add geometric detail to a sculpted object, while the GeoBrush system [Takayama et al. 2011] supports interactive geometry cloning from one mesh to another. Maya Paint Effects [Maya 2014] offers a system to instance sophisticated procedurally-defined geometry into a scene via a stroke-based painting interface. The instanced objects are defined by a customizable branching structure that naturally supports organic shapes such as flowers, trees, and other plants. XGen [Thompson et al. 2003], which is also incorporated into Maya, can instantiate curves or full 3D geometry to fill the surface of primitives, making it especially appropriate for hair, fur, forests or other situations where many copies of the same object are needed to fill a region. “Grooming” tools are provided to fine tune parameters. Some existing methods such as [Schwarz et al. 2007a] provide paint-inspired gestures to instantiate predefined geometry, and let users repaint parameters over the instantiated geometry. This presents similarities to our method, while not offering the ability of content reuse and coarse-to-fine animation control hierarchical motion brushes natively provide.

These methods address the transfer of static geometric detail or the instancing of static geometric copies of an object. For example, Maya Paint Effects instances static copies of procedural geometry. The parameters used to define the geometry can be animated after the fact, but the representation does not allow custom movement to be packaged into the effects themselves. Likewise, XGen’s support of new primitives is restricted to static shapes; keyframe animation is discarded when new XGen primitives are created. Our research places the core focus on animation. As such, motion brushes support arbitrary keyframe animation along with geometric and appearance information, allowing artists to hand craft the movement of primitives over time and package them into a reusable brush. Incorporating animation in this way allows motion brushes to encapsulate a rich set of animated effects that are not easily achieved with existing systems.

Painting parameters in a scene was explored in various areas of research. The system presented in [Ulicny et al. 2004] lets users paint over crowd agents to set their mood or color. However this interaction is only local and no means are provided to edit large numbers of crowd agents at once. In [Schwarz et al. 2007b], a variety of properties can be painted over instantiated geometry in 2D by editing buffers storing the parameters. This method does not scale well to using instances that move in the scene, since the buffers do not change. This is a major drawback for animation applications. Moreover, considering that the set of parameter painting gestures is sparse within a scene, scaling up to 3D applications using such buffers does not offer the best performances in memory. [Salisbury et al. 1997], [Olsen et al. 2005], and more recently [Kazi et al. 2014] demonstrate a variety of field painting tools used in 2D for non-photorealistic rendering which were an inspiration for this work.

The Dynamic Element Textures work presented in [Ma et al. 2013] is very relevant to this paper. They observed that when animating groups of primitives such as a school of fish, their motion can be decomposed into a hierarchy of coarse to fine scale motions. A low-frequency motion control governs the overall direction and shape of the group, while the finer scale motion controls depict the animation and variation of individual elements. This backs up this work, where our framework inherently provides control over coarse-to-fine motion. Fine-scale motion can be controlled when defining a motion brush, which can be used to paint over a mesh depicting a coarser motion, which can itself be reused at an even coarser level, and so on. Editing motion on such hierarchical levels was explored in [Kazi et al. 2014], where elements with granular motion move in motion fields painted by users, and in [Dontcheva et al. 2003] where users can use physical widgets to direct the motion of different parts of virtual characters. The technique presented in this paper however provides tools for reusing content and authoring the motion of multiple characters at once.

Finally, painting interfaces also play a strong role in research that targets stylized depiction such as the WYSIWYG NPR system [Kalnins et al. 2002] or OverCoat [Schmid et al. 2011; Bassett et al. 2013]. The painterly aesthetic offered by these systems aligns well with the concept of motion brushes. Whereas WYSIWYG NPR and OverCoat focus on stylizing objects or characters through painting, motion brushes provide a mechanism to enhance the brush model itself. Combined together, these complementary ideas permit strong aesthetic control over both the brush depiction and movement. As a result, we develop the motion brush system using ideas from the OverCoat framework, allowing the creation of brushes for stylized nonphotorealistic fire and other art-directed visual effects. In this way, our motion brush system contributes to the overall field of nonphotorealistic rendering [Bénard et al. 2011] with a new tool for creating stylized animation.

3 Method

In this section, we present the motion brush concept and its associated workflow. After this technical discussion, we give details on the implementation and interface in 4. Finally, sections 5 and 6 provide results and a forward-looking discussion of the method.

3.1 Overview

Motion brushes package animated content in a representation that can be intuitively applied in new contexts. In traditional digital painting, a stamp-based brush takes an input image and stamps it over and over along a painted stroke. Motion brushes are similar, but instead of a single image, they instantiate an animated scene into a 3D world. We refer to a particular instance of the animated scene as a "motion stamp."

In the simplest case, a motion brush could be defined by a single plane texture mapped with an image. When used, this motion brush would behave similarly to a regular stamp-based brush: the user draws a stroke and the motion stamps–made up of the single texture map–are generated along the stroke. However, our framework supports much more complex motion brush definitions. The scene could be more detailed, having various pieces of geometry, different textures, and be animated over time. Similarly to stamp-based brushes, the user can parametrize the spacing and orientation while generating motion stamps, ultimately controlling the visual result. However, unlike traditional stamp-based brushes, parameters of the motion stamps related to space and time remain editable. Moreover, at any stage of creation, the user can export the current scene as a new motion brush and use it to create new scenes or to define new, more complex, motion brushes.

Regardless of the complexity of the animation being produced, the following steps typically apply in our proposed workflow:

Authoring motion brushes. To create a new motion brush, the user builds an animated scene. Conceptually, no restriction on tools and formats apply. However, our implementation favors stylized motion brushes and is designed to work with [Maya 2014] for animation definition and OverCoat [Schmid et al. 2011] for painterly stylization. Thus authoring a motion brush entails creating and animating 3D geometry in Maya and stylizing its appearance using OverCoat's 3D painting tools.

Painting with motion brushes. Motion brushes are used to instantiate animated geometry in a 3D scene via a painting interface. Given a 2D stroke drawn with a stylus or other input device, we embed this stroke into the 3D scene using an embedding optimisation [Schmid et al. 2011] that allows strokes to conform to object surfaces or depart from the surface into the surrounding space. Once the stroke has been embedded within the scene, it is sampled and motion stamp instances are created according to parameters such as brush size and spacing. Jittering parameters add randomness to orientation, scale, animation start time, and other values. We describe these steps in section 3.4.

M	Motion brush associated with this motion stamp.
o	Opacity.
T	Transformation matrix defining position, orientation and scale.
t_{start}	Animation starting frame for this instance.
loop	Specifies if the animation loops.

Table 1: *Parameters P for a motion stamp m.*

Painting parameters. After painting with a motion brush, the user can edit the result by redefining instantiation parameters related to its orientation, size, animation starting time, or other relevant parameters. The interface we provide for editing the parameters is based on painting gestures, as described in 3.5.

Building hierarchies. A motion brush may instantiate a scene containing other motion brushes. This hierarchical construction make it possible to build complex, detailed motion brushes from simpler components. The only formal restriction we place on the hierarchical representation is that it form a tree structure. For more details, see section 3.6.

3.2 Definitions

A motion brush is an elementary digital scene comprised of geometry that moves over time. More formally, let M refer to a motion brush associated with an animated scene. We denote $|M|$ the duration of the brush animation. For time $t \in [1, |M|]$, M_t is the state of M at the time t, which corresponds to the t^{th} frame of the brush animation. We use S to denote a motion stamp, which is an instance of a motion brush M created as the user paints. Besides the content of the scene, a motion stamp stores a number of parameters P that characterize this particular instance. Table 1 enumerates the parameters used in our implementation. Please note that our method is not restricted to the set of parameters presented in this paper. As described in Section 4, extending this set requires little implementation.

3.3 Authoring motion brushes

Our authoring framework for motion brushes involves two core components. First 3D geometry is created and animated using traditional animation tools. Our implementation works with Maya [Maya 2014] and supports polygonal models animated with any keyframe or simulation method. This flexibility gives artists a great deal of creative freedom. We fix the convention that the origin of the Maya scene will be aligned with the point at which the scene is instantiated as a motion stamp. Likewise, the x-axis direction will align with the direction of painting. In this way, artists can anticipate how animation should be developed in order to achieve a desired effect when it is used as a motion brush.

The animated scene can be used as-is to define a motion brush, and we present one example (Figure 5) showing this functionality. However, we support a second component that makes use of the OverCoat 3D painting system [Schmid et al. 2011; Bassett et al. 2013] for stylization. Animated geometry created in Maya is imported into OverCoat where its appearance can be stylized using OverCoat's 3D painting functions. OverCoat paint strokes are as-

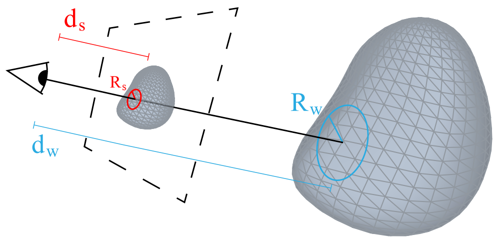

Figure 2: *Radius Definition. R_s is projected onto the embedding mesh obtaining a world space brush size R_w which is used to create the motion stamps. To do so, we search for the intersection between a camera ray centered at the brush and the surface, obtaining the distance d_w. Then, we apply: $R_w = R_s \cdot d_w / d_s$*

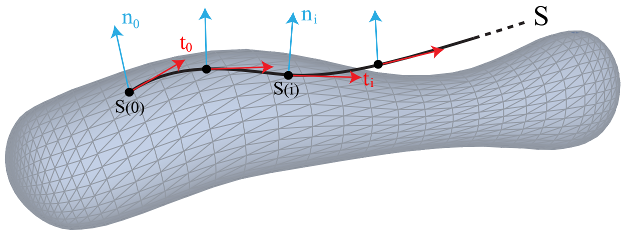

Figure 3: *Stroke Sampling in 3D. The stroke S defines the tangent (red). The normal (blue) is taken from the embedding surface. If the stroke extends off the surface, the normal is propagated using rotation minimizing frames.*

sociated with the geometry and can be rendered to create highly stylized looks.

Once modelling, rendering, and stylization has been completed, the user can decide to create a motion brush from the edited scene, and our system caches all relevant information, including mesh connectivity, vertex positions at each frame of the animation, and OverCoat painting information necessary to achieve the chosen stylized look.

3.4 Painting with motion brushes

Motion brushes are used to embellish 3D scenes with animated content. Painting with motion brushes is similar to traditional painting with stamp-based brushes. The user selects a motion brush and sets relevant brush parameters such as the brush size R_s, spacing r, or opacity o. The user then draws a stroke using a stylus or other input device. This stroke is represented as a polyline which is embedded into the 3D scene using the method described in [Schmid et al. 2011] and stored persistently for future modification. The value R_s is adjusted based on projection to determine the world-space brush size R_w of the motion stamps (Figure 2). The spacing value s is set by the user as a percentage of R_w to determine the sampling frequencey along the stroke.

Next, local coordinate frames are computed along the embedded stroke. When the embedded stroke lies on some 3D object in the scene, the frame is defined by the stroke tangent, the object's surface normal and the cross product of the two. In the more general case where strokes depart from an object's surface, our system propagates normal information using rotation minimizing frames [Wang et al. 2008].

Finally, motion stamps are instantiated in the scene along the stroke according to the stroke spacing value, aligning the origin of each stamp with the local coordinate frame of the stroke. As defined in 3.2, each of these stamps is characterized by a set of parameters P.

The parameters for each stamp are assigned values controlled by the user through the UI. While legacy parameters typical to 2D painting interfaces, such as radius, opacity, etc. are controlled similarly to within such software, the ones specific to motion brushes require specific UI components, as described in section 4.

When instantiating a motion stamp S_i using a hierarchical motion brush, a hierarchy of stamps will be instantiated in the scene. We propagate the parameter values used to instantiate S_i through the hierarchy tree of stamps, from the root down, to ensure an intuitive behavior for the user. Further details on propagating these values is provided in 4.

3.5 Parameter painting

The manipulation of motion stamp parameters unlocks a number of possibilities, drastically increasing the artistic freedom of motion brushes. Our method supports three types of editing based on painting gestures:

Direct editing. The most straightforward tool we provide the user with lets them directly repaint parameters. The user selects a parameter p from P, sets its new value v through UI elements, and uses the "parameter brush" to paint over existing motion stamps. The stamps below the cursor have their parameter p changed to the new value v.

Parametrization along the stroke. The structure of a stroke can be used to coordinate the parameters of motion stamps relatively to one another. For that purpose, every stroke carries an arc-length parametrization along its painting direction. The user can use this arc length value to control any scalar parameter associated with the motion brushes, such as the opacity o or start time t_{start}, obtaining motion stamps that gradually appear or disappear, or offsetting the start time of the animation, as shown in Figure 10. In our implementation, the user can specify a custom mapping using a piecewise linear curve widget.

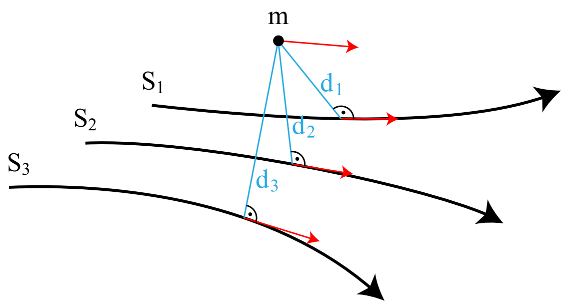

Figure 4: *The user paints three vector field strokes, S_1, S_2, and S_3. The field contribution at the location of the motion stamp m is computed as sum of the field values (or vectors) at the nearest points on S_i, weighted by the distances d_i using the Shepard interpolation scheme.*

Field parametrization. Our implementation provides a painting interface where the user draws field strokes from which either a scalar or a vector field is generated and mapped to a parameter of the selected motion stamps. Scalar fields can influence opacity and animation start time. Vector fields can change the orientation of the motion stamps (Figure 5), or be used to advect their position. The influence of the field stroke(s) on the motion stamps is shown in Figure 4. For a motion stamp m, we project its position onto each field stroke S_i. Naming the parameter value carried by S_i at the projected position $P(S_i)$ and d_i the distance from m to S_i, we

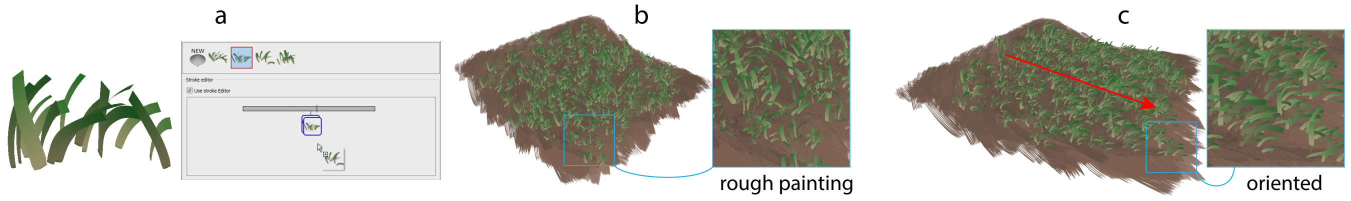

Figure 5: *The grass example demonstrates our vector field parametrization. (a) A few motion brushes made of animated blades of grass are created and (b) used to roughly paint the terrain. (c) The user then paints a vector field (red) orienting the grass as if moved by the wind.*

compute the value of the parameter P on m using Shepard interpolation of the field strokes:

$$P_m = \sum_{S_i \in Strokes} \frac{1}{d_i^2} P(S_i)$$

In addition to direct specification via painting, the user can import external data such as the velocity field from a physical simulation. Figure 12 shows an example from a fluid simulation created in Maya. The velocity field is used to advect motion brush strokes, while the density field is mapped to a scaling of the motion stamps.

3.6 Hierarchical motion brushes

Motion brushes become particularly powerful when combined together. As previously stressed, we allow any scene to be used as a motion brush. This remains valid for complex scenes created using motion brushes. This formulation provides a natural way to manage complexity: simple brushes can be created, saved, and then used to form more complex ones. This provides a very intuitive way to reuse content at different levels of detail, but also proves useful for designing coarse-to-fine animations. In [Ma et al. 2013], complex animations comprising of elementary animated pieces such as a school of fish can be seen as combinations of coarse-scale and fine-scale motions. Hierarchical motion brushes give direct control to the user to create such complex animations. Designing elementary animated brushes corresponds to what will be a fine-scale motion in the final scene, while using those brushes to paint on a mesh animated at a lower frequency depicts the coarse-scale motion that will govern the overall shape of the animation.

4 Implementation and UI elements

4.1 Implementation

Core A motion brush is a digital scene containing meshes that potentially carry strokes painted with existing motion brushes. Those strokes contain motion stamps, that are potentially roots of subtrees in the scene hierarchy, as described in 3.6. See Figure 6 for an example of such a hierarchy. When editing a mesh that carries strokes, or individually editing such strokes, or when those are animated, a parsing of the tree is necessary to update every stamp in the scene. For example in the case of Figure 6, stretching the main mesh will stretch the stroke it carries using the method described in [Bassett et al. 2013]. The stamps on that stroke are moved to stay on the deformed stroke, and so on. However, when no such change is operated, parsing the tree becomes unnecessary. In particular when users move the camera around the mesh to paint on it from different angles, they expect an interactive feedback. Our solution on an implementation point of view is to keep a list of ”renderable” elements such as stamp geometry, and paint strokes in the case of our implementation using the Overcoat technology. The renderer then never accesses the tree and renders that list of elements directly.

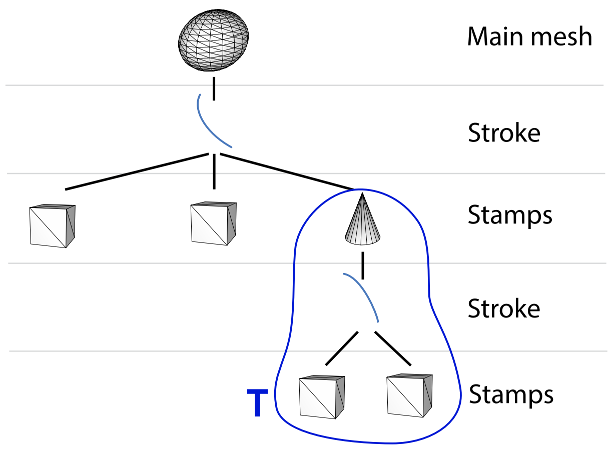

Figure 6: *Example of a scene graph using motion brushes. The user created a cube brush and used it to paint a stroke on a cone. The obtained scene was described by the tree T. The painted cone was then used as a motion brush along with the cube brush to paint on an rounder mesh. When instantiating the stamp using the cone brush, T was added as a subtree to the scene hierarchy.*

Parameters As previously stated, motion stamps instantiated through painting carry a set of parameters that need to be initialized. The higher level stamps (the ones directly owned by the painted stroke) have their parameters assigned values obtained through UI elements, as described in Section 4.2. However, when instantiating a hierarchical stamp, the parameters for its child stamps already have significant values by design. Therefore, we equip motion stamps with a `inherit()` method that computes values for its parameters depending on their parent stamp. For example, when designing the hierarchical brush in Figure 8, the drop was set to start at frame 1, and the splash at frame 10. When instantiating a stamp using this hierarchical brush in an animated scene so that it starts at frame 24, we want the drop to start at frame 24, and the splash to start at frame 33. So our `inherit()` method will contain the line `this.startFrame += parent.startFrame - 1;`.

Extending the motion brushes within any application comes down to extending the list of parameters P. This is made very easy by our implementation: one only needs to provide a UI control for initializing the new parameters, provide a formula for propagating them through the stamp hierarchy, and use the parameters in the renderer, which will access the stamps directly and use the parameters values for rendering.

4.2 Interface

In order to give full control to the user over the painted scene, we interface all the parameters on which motion stamps depend. In the same fashion as in traditional 2D painting software, sliders are used to control the scalar values such as the opacity, spacing, etc. As for the animation starting time, our system includes a timeline widget which is essentially a slider that lets the user set the current time and review the animation by moving over time. When painting with an animated motion brush, the start frame for the instantiated stamps' animation is the current one. This lets users move in time and paint when they want an animation to start.

In several use cases, the user might want to have a parameter vary along a stroke. While this is editable post-painting as described in 3.5, we also offer that option up front. A curve editor lets users design the shape of the parameter value along the strokes, as showcased in Figure 11.c where a user-designed curve is used to delay the start of the motion stamp animations along the stroke.

Furthermore, a powerful UI element we provide lets users chose which motion brushes will be used along the strokes they paint. Our interface displays a stroke profile, onto which users can drag and drop brushes. Through sliding delimiters, the user can chose which brush should be used on which portion of the strokes they are about to draw. This is particularly useful when several brushes are combined in a semantic way (beginning, middle end), or when the user wants to randomize which motion brush to use, by grouping several brushes on the same portion of the stroke.

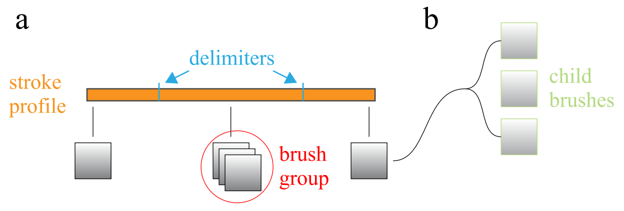

Figure 7: *This illustration combines various elements and is similar to the corresponding UI element in our implementation, shown in Figure 5. (a) The orange bar represents the paint stroke. The user places motion brushes on it. The delimiters in blue let the user indicate the desired distribution of the motion brushes along the stroke. Motion brushes can be stacked in groups (red). During the instantiation, motion stamps on this section of the stroke will be associated randomly with one of the brushes in the group. (b) Any motion brush can include a hierarchy of brushes.*

5 Results

The following examples target specific areas of this method, but share the intent of creating diverse visual styles. The reader is invited to watch the accompanying video to see the corresponding animated sequences.

5.1 Spatial and temporal hierarchy

To show the importance of spatial and temporal hierarchies, we propose a simple example shown in Figure 8. Notice that such hierarchy is used throughout all the results of this paper. To obtain the effect of rain, the user creates separate motion brushes for drops of rain and splashes on the ground. These two can then be combined into a single composite motion brush. Once created, the motion brush is then be painted across on the ground. By randomizing the start time of each instance, a stylized rain effect is achieved. Additional complexity is obtained in this example by generating multiple splash effects and randomly choosing among them when the drop hits the ground. Notice how all the components of this scene are very simple on their own, while complexity is obtained through hierarchy and instancing.

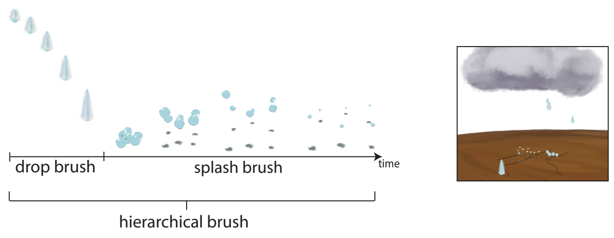

Figure 8: *Rain Example. This example shows how complex scenes can be constructed with a hierarchy of simple brushes. A drop and a splash brushes are designed, and combined into a single hierarchical motion brush. On the right, this motion brush is used to fill a scene with rain.*

5.2 Parametrization

For a system to be usable in practice, complexity must be subject to artistic control. As shown in section 3.4, our system provides control over a number of parameters in an intuitive way.

Figure 10 shows an example of parametrization along the stroke. A motion brush with footprints is created and applied to a number of strokes drawn on a ground. Without any parametrization, footprints would appear at the same time everywhere. The appropriate result is achieved by adding a delay on the animation start of the instances along the stroke. Figure 11 shows the same principle, this time applied on a motion brush painted over a moving proxy geometry.

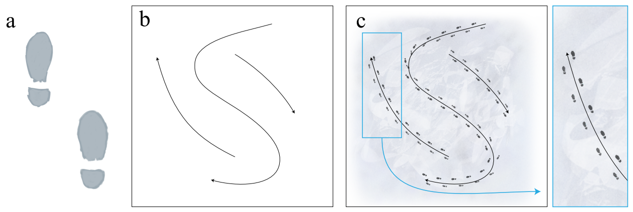

Figure 9: *Footprint example showing parametrization along the painting direction. (a) A motion brush with left and right footprints is created. (b) Three strokes are drawn on the ground, and the start time of the motion brush instances is parametrized along the stroke. (c) As a result, footprints are generated one after the other, as if invisible people where walking on the snow.*

In Figure 5, the user generates a grass-filled terrain using motion brushes with animated blades of grass. Through the dialog shown in Figure 5 (a), the user specifies that motion stamps should randomly chosen from four motion brushes available. After painting the terrain with grass (5 (b)), the orientation of the grass looks chaotic as it depends on the paint direction. To simulate the effect of wind, the user draws a vector field (red arrows in 5 (c)), which re-orient the motion stamps. The accompanying video shows a further editing step, where the user applies a scalar field parametrization to offset the animation start of the motion stamps, generating a wave effect.

Another example of field parametrization is shown in Figure 12. In this example, the goal of the user was to generate stylized smoke made of Santa sleds. For that purpose, motion brushes depicting Santa Claus and a reindeer were created. To create realistic smoke dynamics, a fluid simulation was computed in Maya, and the ve-

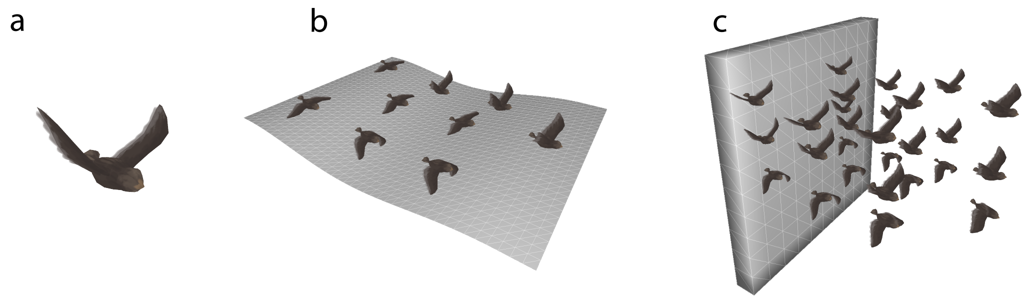

Figure 10: *This example demonstrates the edition of coarse-to-fine motion. A bird brush is first defined (a), then used to paint a mesh depicting a wave (b). That wave of birds is used on a plane moving forward (c) to create a flock of birds.*

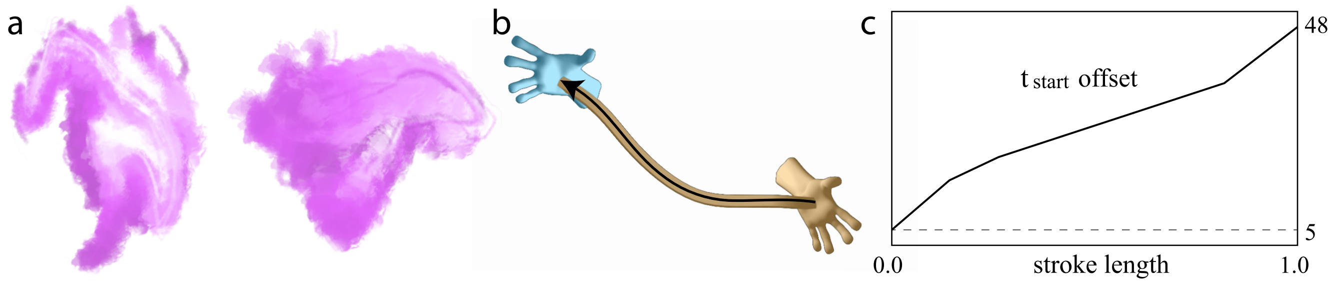

Figure 11: *This figure illustrates the creation of the fire example shown in Figure 1. From two textured planes depicting pink flames (a), a motion brush is created and used to paint over the pipe connecting the two hands (b). Notice that the pipe is rigidly attached to the yellow hand, while the blue hand slides along the pipe, reaching the position shown here. (c) shows the curve generated by the user to correctly time the delay of the flames, giving the illusion of the fire following the blue hand.*

locity and density field were imported into our application. The velocity field was used to advect the strokes, while the motion stamps were scaled based on the density field.

6 Conclusion

We have presented hierarchical motion brushes, a novel tool combining the controllable reuse of traditional brushes with the power and flexibility of 3D animation authoring tools. Motion brushes are not restricted to 3D applications and are suitable for 2D as well as 2.5D applications. We decided to demonstrate the reach of this method over a number of 3D examples that depict stylized animated effects encapsulated into a brush framework.

Limitations of our current method point to several areas of future work. One limitation of this method is that, while motion brushes may instantiate complex data, the coordination of such data must be orchestrated by the user. In technical terms, this limitation is expressed by the fact that the scene hierarchy is a tree with information propagating only from the root to the leaves. While this allows for coordination between direct neighbors in the tree hierarchy, separate branches remain unaware of one other. Abstractions to author the coordination between different motion brush animations is an interesting area of future research. In common applications such as video editing software, this can be achieved through a timeline widget containing several tracks. In the case of software using motion brushes, we need a more complex timeline letting users coordinate different animations at different hierarchical levels. A more elaborate approach would be to incorporate an event system to trigger Motion Brush animations, which would help a lot with authoring the relative timings of motion stamps.

A second limitation lies in the desire to have real-time feedback while instantiating large numbers of animations. In our implementation, we opted to preemptively cache as much data as possible, resulting in relatively long loading times (within one minute) but good responsiveness while working on any of the scenes shown in this paper. Similar to most 3D animation tools, one may consider a solution where the user operates with a fast real-time preview render, while the final results are generated offline using a production-quality render. In general, particular care should be taken in the implementation of this method, especially if targeting production environments, in order to minimize evaluations of the brush hierarchy.

One very interesting area of research is the one of contextualized painting, where motion brushes react differently when used on different parts of the canvas. Our current implementation would require users to paint various parameter fields in the scene and map those parameters to the motion brushes they use. This becomes a tedious process and automatically computing such fields would be very useful. For instance, one could think of a brush which paints grass on soil and moss on rocks. One could also conceive a system where motion stamps are aware of each other and present contextualized behaviors with respect to the presence of other motion stamps. This could be used, for instance, to paint a seaside scene, where water warriors riding the waves charge sand warriors waiting on the beach, the warrior animation being dependent on the proximity of other warriors. This could be implemented by having instantiated stamps alter the parameter fields in the scene for example. We believe that motion brushes provide a solid base upon which such directions can be explored, and hope to see in the near future new tools bringing more stylization and power to the tip of a brush.

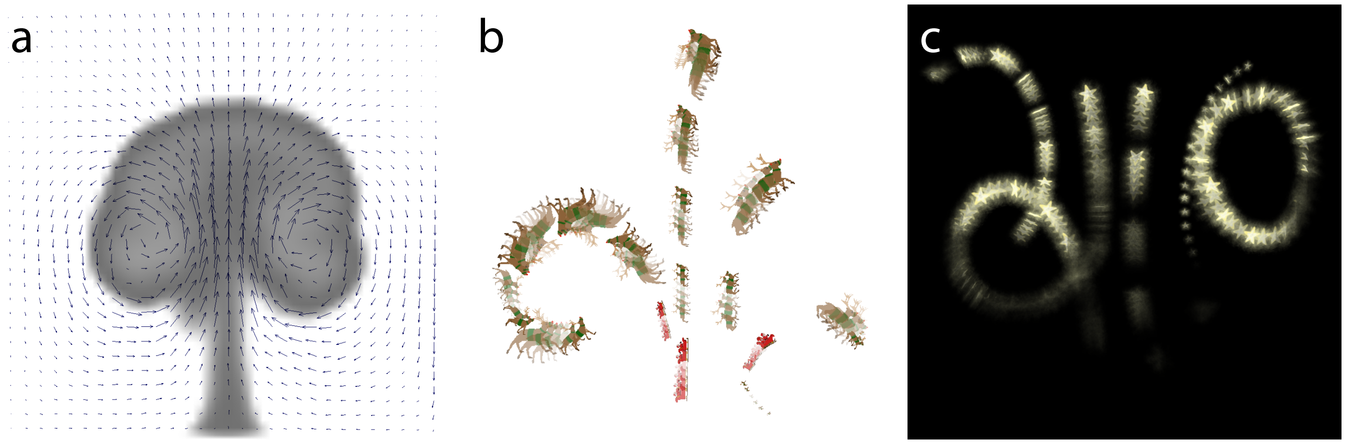

Figure 12: *Stylized smoke example. A fluid simulation (a) is computed in Maya. Two motion brushes–one with a Santa sled and reindeer, and one with spinning stars–are used to paint a number of strokes. The strokes are advected in space using the velocity field, while their size and opacity is parametrized by the density field. The resulting effects are shown in (b) and (c) respectively.*

References

BASSETT, K., BARAN, I., SCHMID, J., GROSS, M., AND SUMNER, R. W. 2013. Authoring and animating painterly characters. *ACM Trans. Graph. 32*, 5 (Oct.), 156:1–156:12.

BÉNARD, P., BOUSSEAU, A., AND THOLLOT, J. 2011. State-of-the-Art Report on Temporal Coherence for Stylized Animations. *Computer Graphics Forum 30*, 8 (Dec.), 2367–2386.

CHU, N., BAXTER, W., WEI, L.-Y., AND GOVINDARAJU, N. 2010. Detail-preserving paint modeling for 3d brushes. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*, ACM, NPAR '10, 27–34.

DONTCHEVA, M., YNGVE, G., AND POPOVIĆ, Z. 2003. Layered acting for character animation. *ACM Trans. Graph. 22*, 3 (July), 409–416.

GIMP, 2013. The GNU Image Manipulation Program. http://www.gimp.org/.

HSU, S. C., LEE, I. H. H., AND WISEMAN, N. E. 1993. Skeletal strokes. In *Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology*, ACM, New York, NY, USA, UIST '93, 197–206.

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 Transactions on Graphics (Proc. SIGGRAPH) 21*, 3 (July), 755–762.

KAZI, R. H., IGARASHI, T., ZHAO, S., AND DAVIS, R. 2012. Vignette: Interactive texture design and manipulation with freeform gestures for pen-and-ink illustration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, New York, NY, USA, CHI '12, 1727–1736.

KAZI, R. H., CHEVALIER, F., GROSSMAN, T., ZHAO, S., AND FITZMAURICE, G. 2014. Draco: Bringing life to illustrations with kinetic textures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, New York, NY, USA, CHI '14, 351–360.

LU, J., BARNES, C., DIVERDI, S., AND FINKELSTEIN, A. 2013. Realbrush: Painting with examples of physical media. *ACM Transactions on Graphics (Proc. SIGGRAPH)* (Aug.).

MA, C., WEI, L.-Y., LEFEBVRE, S., AND TONG, X. 2013. Dynamic element textures. In *SIGGRAPH 2013*, 90:1–90:10.

MAYA, 2014. Autodesk Maya. http://www.autodesk.com/maya.

MĚCH, R., AND MILLER, G. 2012. The *Deco* framework for interactive procedural modeling. *Journal of Computer Graphics Techniques (JCGT) 1*, 1 (Dec), 43–99.

OLSEN, S. C., MAXWELL, B. A., AND GOOCH, B. 2005. Interactive vector fields for painterly rendering. In *GI '05: Proceedings of the 2005 conference on Graphics interface*, Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 241–247.

PHOTOSHOP, 2012. Photoshop CS6 by Adobe Systems Incorporated. http://www.adobe.com/photoshop.

SALISBURY, M. P., WONG, M. T., HUGHES, J. F., AND SALESIN, D. H. 1997. Orientable textures for image-based pen-and-ink illustration. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., SIGGRAPH '97, 401–406.

SCHMID, J., SENN, M. S., GROSS, M., AND SUMNER, R. W. 2011. Overcoat: An implicit canvas for 3d painting. In *ACM SIGGRAPH 2011 Papers*, ACM, New York, NY, USA, SIGGRAPH '11, 28:1–28:10.

SCHWARZ, M., ISENBERG, T., MASON, K., AND CARPENDALE, S. 2007. Modeling with rendering primitives: An interactive non-photorealistic canvas. In *Proceedings of the 5th International Symposium on Non-photorealistic Animation and Rendering*, ACM, New York, NY, USA, NPAR '07, 15–22.

SCHWARZ, M., ISENBERG, T., MASON, K., AND CARPENDALE, S. 2007. Modeling with rendering primitives: an interactive non-photorealistic canvas. In *Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, ACM, NPAR '07, 15–22.

TAKAYAMA, K., SCHMIDT, R., SINGH, K., IGARASHI, T., BOUBEKEUR, T., AND SORKINE, O. 2011. Geobrush: Interactive mesh geometry cloning. *Computer Graphics Forum (proceedings of Eurographics) 30*, 2, 613–622.

THOMPSON, II, T. V., PETTI, E. J., AND TAPPAN, C. 2003. Xgen: Arbitrary primitive generator. In *ACM SIGGRAPH 2003 Sketches &Amp; Applications*, ACM, New York, NY, USA, SIGGRAPH '03, 1–1.

ULICNY, B., CIECHOMSKI, P. D. H., AND THALMANN, D. 2004. Crowdbrush: Interactive authoring of real-time crowd scenes. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '04, 243–252.

WANG, W., JÜTTLER, B., ZHENG, D., AND LIU, Y. 2008. Computation of rotation minimizing frames. *ACM Trans. Graph. 27*, 1 (Mar.), 2:1–2:18.

ZBRUSH, 2013. Zbrush, 4R6 by Pixologic. http://www.pixologic.com/zbrush.