A Lightweight, Procedural, Vector Watercolor Painting Engine

Stephen DiVerdi* Adobe Systems Inc.
Aravind Krishnaswamy† Adobe Systems Inc.
Radomir Mech‡ Adobe Systems Inc.
Daichi Ito§ San Jose State University

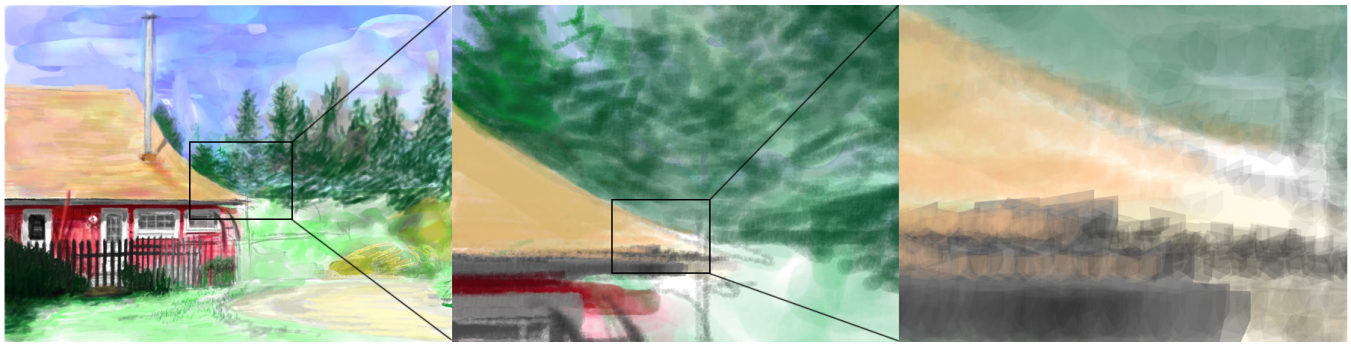

Figure 1: *A vector watercolor painting made in our interactive iPad painting application, displaying complex texture and color blending. Insets zoom in to show stroke detail and resolution independence.*

Abstract

Existing natural media painting simulations have produced high quality results, but have required powerful compute hardware and have been limited to screen resolutions. Digital artists would like to be able to use watercolor-like painting tools, but at print resolutions and on lower end hardware such as laptops or even slates. We present a procedural algorithm for generating watercolor-like dynamic paint behaviors in a lightweight manner. Our goal is not to exactly duplicate watercolor painting, but to create a range of dynamic behaviors that allow users to achieve a similar style of process and result, while at the same time having a unique character of its own. Our stroke representation is vector-based, allowing for rendering at arbitrary resolutions, and our procedural pigment advection algorithm is fast enough to support painting on slate devices. We demonstrate our technique in a commercially available slate application used by professional artists.

CR Categories: I.3.4 [Computer Graphics]: Graphics Utilities—Paint Systems; 1.6.8 [Simulation and Modeling]: Types of Simulation—Animation;

Keywords: watercolor painting, vector graphics, real-time

1 Introduction

Despite recent advances in digital painting, there remains a large contingent of traditional media artists. Natural media such as watercolors, oil paints, bristle brushes, and charcoals remain relevant to modern artists because of their expressive and serendipitous workflows. The wide range of complex textures and qualities that can be created using, e.g. watercolor paints, is difficult to match in today's digital media.

*e-mail: steved@adobe.com
†e-mail: aravind@adobe.com
‡e-mail: rmech@adobe.com
§e-mail: daich@mac.com

I3D 2012, Costa Mesa, CA, March 9 – 11, 2012.
© 2012 ACM 978-1-4503-1194-6/12/0003 $10.00

Natural media simulation therefore is an active area of research in the computer graphics community. Towards this end, interactive systems have been developed [Chu and Tai 2005; Laerhoven and Reeth 2005] to allow artists to create compositions more in the style of their traditional tools. Unfortunately, to date none of these research prototypes have been able to make their way into commercial systems. Either their compute requirements have been too high, or they have been unable to support the document resolutions necessary for professional work.

Rather than attempt yet another exact replication of watercolor painting behaviors, we pursue another direction. Our goal instead is to create a painting application with a wide range of dynamic behaviors that are inspired by core watercolor effects such as color blending, feathering, and edge darkening. With a similar fundamental style, our painting application can achieve the same wide variety of complex and serendipitous results as watercolor, but with a unique character that is specific to our algorithm.

We present a novel formulation of an interactive watercolor-like paint algorithm that departs from existing research in many significant ways. First, our formulation uses a particle-based model of pigment flow, rather than grid-based. Second, the particle representation is vector instead of raster, allowing for rendering at arbitrary resolutions. Third, the particle update step is a physically-inspired procedural algorithm that is very fast to calculate. The result of these differences in approach means that unlike previous work, our algorithm can be used to generate high resolution output on low-powered devices, while still recreating a useful subset of interactive watercolor paint behaviors. We are able to achieve common watercolor effects such as edge darkening, non-uniform pigment density, granulation, and backruns. Furthermore, by varying model parameters, we can simulate a variety of different types of brush and pigment types.

To demonstrate the performance and real-world utility of our formulation, we shipped a reduced form of the algorithm for the Apple iPad as Adobe Eazel, in conjunction with the release of Adobe Photoshop CS 5.5. Artists are able to paint on their iPads and then transfer the composition to a desktop machine where Photoshop can re-render it at a higher resolution. Due to the novelty of our algorithm, professional artists have been able to create unique paintings with the quality of watercolors that rival their work with other painting tools, but on devices with much less computational power, confirming the success of our approach.

2 Related Work

The creation of watercolor painting-like images has been extensively studied. Much of the work has focused on the automatic conversion of photographs into paintings [Hertzmann 2003], by analyzing the image contents and placing a series of brush strokes. These techniques have been extended to work on videos and in real-time [Hertzmann and Perlin 2000; Lu et al. 2010]. Bousseau et al. [2006] took a different approach, by combining many different image filters applied to photographs or 3D renderings, to create compelling watercolor depictions. While these results are pleasingly artistic, the lack of interactive, artist-driven control limits their utility for painting applications.

Some commercially available applications provide interactive watercolor tools, such as Corel Painter [2011] and Ambient Design ArtRage [2011]. In Painter's case, the artist paints a brush stroke and then waits for a processing routine to compute the diffusion effect before another stroke can be placed, thus interrupting the painting workflow. ArtRage's watercolor brush allows many brush strokes in rapid succession, but does not compute on-canvas motion of the pigment once it has been deposited. Ultimately, there is not yet a commercially available painting application that captures the interactive and dynamic nature of real watercolor painting.

To create more faithfully realistic reproductions of watercolor paintings, Curtis et al. model the underlying processes of water and pigment advection in their system [Curtis et al. 1997]. They achieved impressive simulations of real watercolor behaviors, but due to the complexity of the algorithm were unable to achieve a fully interactive system. More recently, Chu and Tai [2005] and separately, Van Laerhoven and Van Reeth [2005] demonstrated similar levels of realism in interactive watercolor painting systems that heavily utilized GPUs to compute their effects in real-time. Because of the high compute and resource requirements of these algorithms, they have yet to find their way into commercially shipping products, largely because they have not been demonstrated to be able to handle the resolutions necessary for professional work on commonly available hardware.

In a similar vein, interactive oil painting has been simulated as well, including thick paint application [Baxter et al. 2004] and complex brush paint deposition and dirtying [Chu et al. 2010]. However, oil paint simulation is fundamentally different from watercolor, in that once deposited on the canvas, oil paints do not tend to flow arbitrarily across the canvas, while this is a main effect of watercolors. Therefore, the tradeoffs made in handling performance, resolution, and quality are fundamentally different in our work.

Because of the complex textures found in watercolor painting, vector output is much less common than raster. Applications like Adobe Illustrator [2010] can be used to create watercolor-like vector compositions by using static vector artwork of scanned watercolor brush strokes, but it lacks all of the dynamic aspects of watercolor painting. Diffusion curves [Orzan et al. 2008] also allow the creation of complex vector artwork, but it is not clear how to control them with a brush stroke input interface. Finally, the work of Ando and Tsuruno [2010] uses an adaptive vector representation to track the front of a pigment fluid mixing in water. However, because of the lack of diffusion support, over time the front's complexity becomes unbounded, limiting the viable duration of the simulation.

3 Algorithm

The intuition behind our model is that pixel grid-based simulations are dense whereas watercolor stroke effects are generally sparse, and that the specific path pigment particles take through canvas media is not as important as the generally complex dynamic behavior. In fact, each particle in real watercolor paint can be thought of as taking a random walk, and the aggregate behavior is that of watercolor paint. Based on this intuition, we adopt a sparse representation for our paint pigment, and we use a random walk algorithm to update its position each time step. By carefully selecting the model and update equation, we can recreate a variety of interesting behaviors.

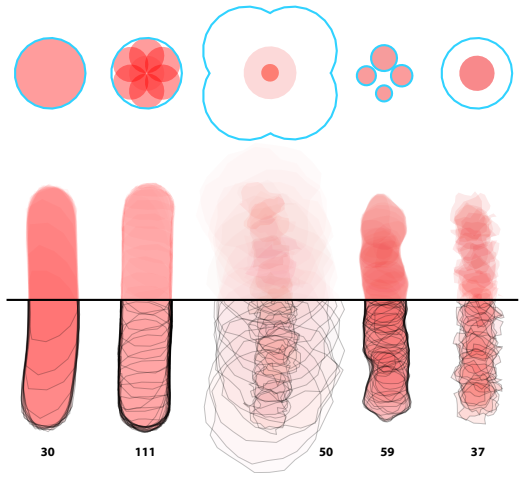

Figure 2: *Initial splat configurations and resulting stroke for each brush type.* Left to right: *simple, wet-on-dry, wet-on-wet, blobby, crunchy. Cyan outlines indicate the water region per-stamp. Black outlines on the bottom half of each stroke indicate final splat shapes. The total number of splats in each stroke is also indicated.*

At the core of our model, we represent paint pigment as a collection of dynamic 'splat' particles, where each splat is a complex polygon of n vertices. Many such splats are placed along the trajectory of a stroke based on rules controlled by the current brush type, and then the vertices of these splats are advected according to another set of rules at each time step. Each splat has its own opacity value, and splats are rendered back to front with standard transparency blending. With this approach, we are able to reproduce a variety of dynamic watercolor paint behaviors, including edge darkening, granulation, backruns, and different types of brush strokes.

3.1 Paint Initialization

During stroke input, stamps are placed in uniform path length increments along the stroke path, so that slow and fast strokes covering the same distance will result in the same number of stamps placed. Each stamp is a set of one or more splats (initially circles), arranged according to the current brush type. We use the 'wet-on-dry' brush as an example, which consists of seven splats of $n = 25$ vertices arranged with one centered on the stamp position and six placed around its perimeter (see Figure 2). Each splat stores motion vectors including a motion bias vector $\mathbf{b}$, and a per-vertex velocity vector $\mathbf{v}$. Additionally, a splat stores its age a (in steps), as well as parameters of the brush type: roughness r (in pixels) and flow f (percentage). These parameters are similar in effect to those used in MoXi [Chu and Tai 2005], but we use them differently in our algorithm.

At each stamp, an amount of water is also added to the canvas. Water does not move on canvas and is stored separately from the list of splats, as a rasterized 'wet map' consisting of a 2D grid of cells. We set the wet map resolution to match that of the display being painted on, so on an iPad 2, it is 1024x768 cells. The brush type specifies the size and shape of the region that is wetted by rasterizing a constant value into the wet map. This value represents the remaining amount of time each cell will stay wet. If the cell is already wet, the max is stored.

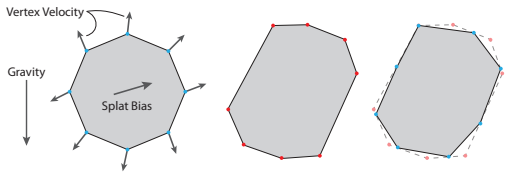

Figure 3: *Each splat's motion is dictated by a per-vertex velocity, a splat bias, and a global gravity.* From left to right: *A splat's initial configuration, after advection, and after boundary resampling.*

The user selects a brush stroke size in pixels, which is then used to scale the splats and wet region based on roughly how large the final resulting stroke will be, to achieve the specified size.

3.2 Pigment Advection

At each time step, the set of live splats is iterated across, updating each vertex position $\mathbf{x_t}$ to $\mathbf{x_{t+1}}$ based on the equations,

$$\mathbf{d} = (1-\alpha)\mathbf{b} + \alpha\frac{1}{\mathbf{U}(1,1+r)}\mathbf{v} \quad (1)$$

$$\mathbf{x}^* = \mathbf{x_t} + f\mathbf{d} + \mathbf{g} + \mathbf{U}(-r,r) \quad (2)$$

$$\mathbf{x_{t+1}} = \begin{cases} \mathbf{x}^* & \text{if } w(\mathbf{x}^*) > 0 \\ \mathbf{x_t} & \text{otherwise} \end{cases} \quad (3)$$

where $\mathbf{g}$ is a global gravity vector, α is a tuning parameter that blends the splat motion bias with per-vertex velocity, which we set to 0.33, $\mathbf{U}(a,b)$ is a uniform random variable between a and b, and $w(\mathbf{x})$ is the wet map value at position $\mathbf{x}$. Based on splat parameters and random numbers, a new candidate position $\mathbf{x}^*$ is computed, and then if the canvas is wet at that position, the vertex is updated; otherwise, the vertex does not move. See Figure 3 for an illustration. Once a splat's vertices have all been updated, its opacity is recomputed by conserving the "amount of pigment", defined as the opacity times the area.

Conceptually, these equations embody the biased random walk behavior depicted by particles in real watercolor paint flow. Each vertex's motion is first a combination of its own velocity and the splat's bias velocity — this allows the splat to continue to expand in area, while also effectively moving according to some water velocity. The vertex velocity is varied by the roughness to reproduce semi-permeability of the canvas media. The overall motion vector $\mathbf{d}$ can be restricted by the flow pigment property which corresponds to the inverse of paint viscosity. Gravity further biases all splats to provide a preferred direction for paint to flow in. The final random parameter, determined by roughness, is the source of the random walk behavior that encompasses all the branching and roughening aspects of watercolor paint.

After all splats have been updated, the wet map is updated as well by decrementing all the non-zero values stored in it. This effectively reduces the remaining wetness at each cell, and when a cell reaches zero, pigment will no longer be able to flow into that portion of the canvas.

3.3 Sampling Management

Over time, the different advection directions of each vertex in a splat can result in splats that have very non-uniformly sampled boundaries, which creates unrealistically straight hard edges. Over many splats, this artifact can create a faceted, polygonal appearance in strokes, especially as they turn corners or have fine water features that cause neighboring splats to have very different wetness values. We have two strategies for dealing with this artifact.

The first is to constrain the motion of vertices within a splat to not be too far from their neighbors. For a splat, each vertex i's new position is computed, $\mathbf{x_{t+1}^i}$, and then compared to its neighbors' old positions, $\mathbf{x_t^{i-1}}$ and $\mathbf{x_t^{i+1}}$, and if the distances between either pair is above some threshold, vertex i's position is not updated (that is, $\mathbf{x_{t+1}^i} = \mathbf{x_t^i}$).

The second strategy is to periodically re-sample each splat's boundary. To do this, the splat's total perimeter is computed, and divided by the number of vertices to compute the arc length per vertex. Then starting at an arbitrary location, the splat's vertices are moved to uniform arc length increments along the boundary.

Both strategies also have the effect of somewhat smoothing the splat boundary, while limiting the polygonal artifacts that may appear in the final stroke appearance. The constrained motion is fast to perform and has the desired effect, but also ultimately limits the propagation of the paint somewhat, which can alter a stroke's appearance in other ways. Boundary resampling is slower by comparison, but achieves higher quality final output so this is what we use. Figure 3 shows the effect of boundary resampling on splat shape.

3.4 Lifetime Management

A splat has three stages of life: flowing, fixed, and dried. When first added to the canvas, a splat is flowing. After some number of steps (a parameter of the brush type), the splat stops being advected and is in the fixed state. While fixed, the splat can potentially be re-wetted to resume advection. After a period of time without moving, the pigment is determined to have permanently stained the canvas and the splat is considered dried. Once dried, the splat is rasterized into a dry pigment buffer and removed from the simulation. This keeps the rendering cost for the canvas capped as the artist continues to make many strokes in a painting.

The addition of water can re-wet fixed splats to achieve effects like back runs and feathered edges. When water is applied to the canvas, fixed splats' vertices are checked to see if they overlap with the new water, and if they do, some of those vertices are unfixed and the splat's age is reset. To determine if a re-wetted vertex becomes unfixed, a random number is tested against a threshold based on how close the splat is to being totally dry and how strong the water's unfixing property is set.

Once a splat has become fixed, its advection under its initial momentum is complete and so the splat's per-vertex velocity vectors are set to zero, along with the splat's bias vector. When the splat is re-wetted, the direction of the vertex motion is determined by the water placement, mimicking the effect of the flow of real water through an absorptive canvas. When the wet regions are rasterized, they also write a radially outward pointing per-pixel 'water velocity' vector to the wet map. Then when the re-wetted vertices are updated via Equation 1, the splat vertex's $\mathbf{v}$ vector is replaced with the vector sampled from the water velocity at the vertex's location, $\mathbf{v_w}(\mathbf{x_t})$.

In addition to controlling splat advection, the age of the splat also determines the impact of the granulation texture. As a splat dries, granulation has a larger impact on its appearance, so a texture (generated from scanned, real watercolor brush strokes) is used to apply a small delta to the splat's opacity, with a weight based on how close to dry the splat is. The granulation texture is a single image that covers the entire canvas, and each splat uses its canvas coordinates plus a small random jitter to sample from it.

3.5 Brush Types

We implemented five brush types in our model that reproduce a variety of characteristic watercolor strokes, though many more vari-

ations are possible. The brush types are simple, wet-on-dry, wet-on-wet, blobby, and crunchy. Each uses a different arrangement of splats per stamp (see Figure 2), and has different settings for the brush parameters. Each stroke in Figure 2 has a target width $w = 45$ pixels. The wet map cells are set to 255 when wetted, and the simulation is run at 30Hz, so they take 8.5 seconds to dry. Unless otherwise specified, splats have life $l = 30$ (so 1 second), roughness of $r = 1$ pixel, and flow of $f = 100\%$. The gravity vector is $\langle 0, 0 \rangle$.

The simple brush uses a single splat per stamp, and demonstrates non-uniform pigment distribution because of the random nature of the splat evolution, and blending between strokes due to the advection into surrounding wet portions of the canvas. The splat's diameter $d = w$ and its bias vector is $\mathbf{b} = \langle 0, 0 \rangle$.

Wet-on-dry places seven splats per stamp, with six arranged around the perimeter of the seventh. The center splat has $\mathbf{b} = \langle 0, 0 \rangle$, while the six perimeter splats at $\theta = \{0, \frac{\pi}{6}, ..., \frac{5\pi}{6}\}$ have $\mathbf{b} = \langle \frac{d}{2} \cos\theta, \frac{d}{2} \sin\theta \rangle$, where $d = \frac{w}{2}$. This causes the perimeter splats to collect around the edges of the stroke, adding additional pigment for an edge darkening appearance.

Wet-on-wet places a small splat ($d = \frac{w}{2}$) inside a larger splat ($d = \frac{3w}{2}$) for each stamp. This results in a progressively darker region towards the center of the stroke for a feathered appearance characteristic of applying pigment to a wet canvas. Both splats have $r = 5$ pixels, $l = 15$ steps and $\mathbf{b} = \langle 0, 0 \rangle$.

The blobby brush places four randomly sized splats ($d \in [\frac{w}{3}, w]$) in a cross pattern per stamp, with $l = 15$ steps and $\mathbf{b} = \langle 0, 0 \rangle$. Along with adding some noise to the splats' color, this creates a more heavily non-uniform stroke shape and pigment density, similar to watercolor paint applied to a rougher canvas or with more granular pigment.

Finally, the crunchy brush places one splat per stamp ($d = w$), but sets $r = 5$ pixels, $f = 25\%$, and $l = 15$ steps to reduce how far the vertices propagate while increasing the influence of the random walk component of the advection. The result is a stroke appearance that exhibits more significant branching and broken stroke texture, with very rough edges.

See Figure 4 for a comparison between real watercolor paint and the types of effects our algorithm is able to generate.

4 Application

In order to create a usable painting application based on our algorithm, a number of considerations are necessary. We have made both desktop and iPad applications implementing our algorithm for real-time dynamic watercolor painting.

4.1 Interactive Rendering

Our interactive painting application uses OpenGL to render the splats and dried pigment buffer. Since splats can be concave or even self-intersecting polygons, standard rendering approaches are not appropriate. A typical solution is to tessellate the polygon, but since our polygons are changing shape every frame, the overhead of tessellation is too high. Instead, we use a two pass stencil buffer approach per-splat. For the first pass, the color buffer is masked out and the stencil operation is set to invert, and the splat vertices are rendered as a triangle fan. This results in the stencil buffer storing a value of one everywhere the polygon is filled and zero elsewhere. The second pass writes to the color buffer and sets the stencil test to equal to one and the stencil operation to zero, and renders a quad that covers only the bounding box of the single splat. This colors the pixels interior to the polygon with proper winding rules, and also resets the stencil buffer to all zeroes for the next splat, avoiding the need for additional passes.

OpenGL creates aliased polygons that appear jagged on screen, but full screen anti-aliasing can be expensive and may not be supported on slate devices. To smooth the results in our interactive painting application, we use a fast post-processing filter that performs an adaptive per-pixel directional blur based on a set of heuristics [Lottes 2011]. For offline rendering at arbitrary resolutions, we use a software polygon rasterizer that includes high quality, analytical anti-aliasing.

In our interactive system, feedback is provided about the current state of the wet map by darkening the document in wet regions. This allows the user to know when their strokes will be affected by pre-existing canvas wetness. Further feedback about splat life stage is given by additional darkening of splats that are currently flowing. In this way, the effect of re-wetting can be predicted, as dried splats will not respond to additional water, whereas merely fixed splats will.

4.2 User Interface

The basic form of our desktop application presents the canvas alongside a GUI for controlling the paint parameters, such as selecting brush type and size, or changing the canvas texture. Users can paint with a mouse in the standard case, or they can use a more sophisticated input device such as a Wacom tablet. The stylus pressure can be mapped to the brush size, so that users can dynamically control the size of the mark made by varying the stroke pressure. Similarly, stylus tilt or rotation can be mapped to influence the stroke splats' bias vectors to create strokes with a directional preference for advection. This can be useful for applying paint that makes a hard edge on one side and a smooth falloff on the opposite side.

Conversely, our iPad application has different affordances and a correspondingly different interface. Only finger-based input is available, so no pressure or tilt sensitivity is possible. On the other hand, the iPad's accelerometer can be used to control the global gravity vector, to make a physical mapping between the orientation of the iPad and the pigment's flow. Furthermore, the iPad's small screen encourages maximizing screen real estate for painting, so we do not present any on-screen interface and let the canvas occupy the entire space. Settings are called up using a specific gesture on the canvas, which allows the user to control a subset of the features of the full algorithm.

4.3 Vector Output

When a canvas is saved, two output files are generated: a bitmap of the rendered image as a PNG, and a vector representation of all the splats (including dried) as an SVG [Dahlström et al. 2011]. See Figure 5 for an example of the SVG vector data directly embedded in this document.

While the SVG specification includes support for advanced features such as image filter effects and blend modes that could duplicate our granulation texture shading math, the unfortunate state of affairs is that no commonly available SVG viewer supports those features. In fact, the complexity of the SVGs we generate, even without texture but just with many overlapping transparent paths, is such that available viewers may struggle to display them efficiently. Conversely, our interactive application is able to display the same data significantly faster, due to its use of GPU acceleration for rendering. Therefore, for now, we limit our SVG output to store only transparent filled paths, and leave the granulation effects for a time when more capable SVG viewers are available.

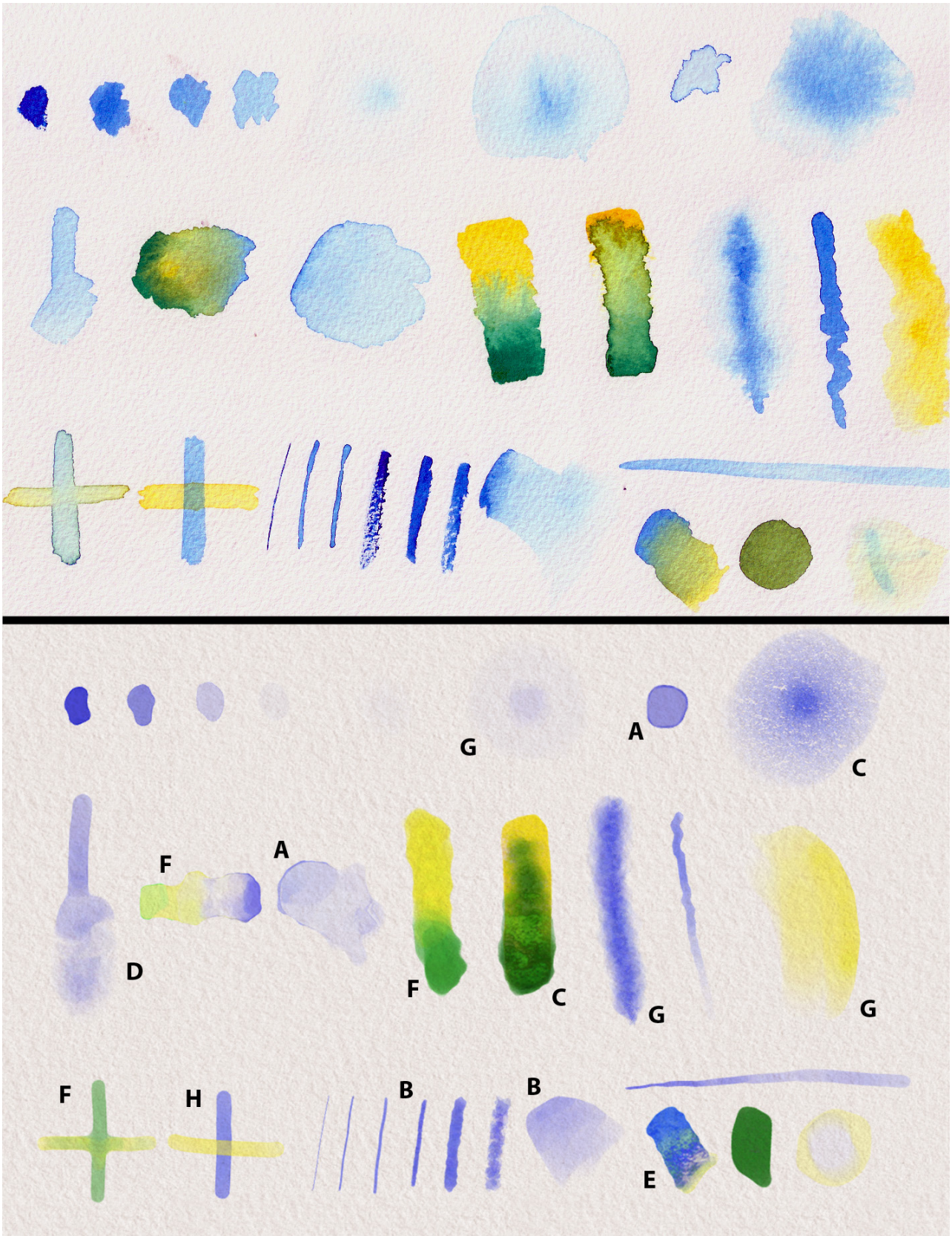

Figure 4: *A comparison of similar strokes made with real watercolor paint (top) versus our algorithm (bottom). Paper texture is added to our results for comparison purposes. The strokes are chosen to showcase a variety of characteristic watercolor behaviors, including edge darkening* (A)*, non-uniform pigment density* (B)*, granulation* (C)*, re-wetting* (D)*, back runs* (E)*, color blending* (F)*, feathering* (G)*, and glazing* (H)*. Strokes exemplifying particular effects have been labeled with the corresponding letter. While our algorithm does not make identical strokes, it exhibits the same range and depth of expressiveness as traditional watercolor.*

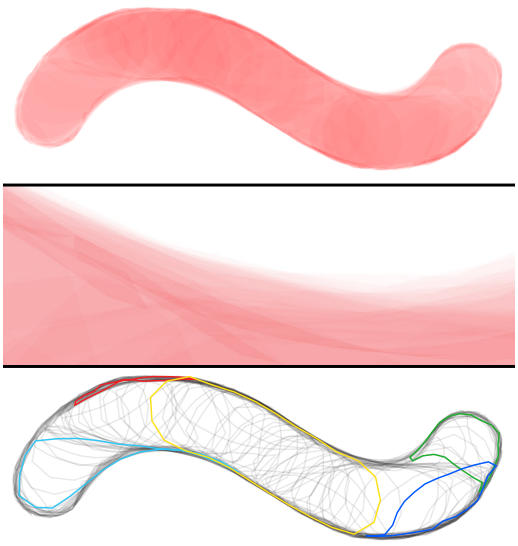

Figure 5: Top: *The vector output of a brush stroke made from left to right. Zoom in to see resolution independence.* Middle: *Rasterized version, zoomed in to 800% to show detail.* Bottom: *All 195 splats outlined instead of filled, with a few splats highlighted to show shape variations. Vertices on the edge of the wet region have a tendency to get 'stuck' as their motion is restricted, which can result in teardrop shapes.*

5 Results

Because of the low computational requirements of our algorithm, we were able to release a version of it as the iPad application called Adobe Eazel for Photoshop CS5 [2011]. This version includes a basic feature set (only wet-on-dry brush, no granulation, no re-wetting). It runs at the iPad's native resolution (1024x768) and has the ability to send the document to a desktop version of Photoshop to re-render the document at a higher resolution with high quality anti-aliasing.

As a result of this public release, our algorithm was used by many artists of many different experience levels and aesthetic styles, which has resulted in a wide variety of different compositions (see Figure 6).

5.1 Performance

The nature of our formulation means that performance in terms of resolution is not particularly important, because the interactive rasterization doesn't impact the final output. Instead, we are limited by the number of active splats being animated and rendered simultaneously. Specifically, rendering is the most costly step, especially on low-powered devices like the iPad.

We were able to improve the rendering performance by only drawing a subset of the flowing splats, at an adjusted opacity to create a similar bulk stroke appearance. Once splats dry, all splats are rendered into the dried buffer at the correct opacity, creating a smoother, higher-quality final stroke.

We tested our performance on an iPad 2 with a 1GHz Apple A5 processor and PowerVR SGX 543MP2 GPU, and on a MacBook Pro with a 2.2GHz Intel Core i7 processor and an AMD Radeon HD 6750M graphics card, to see how many splats could be rendered simultaneously while maintaining interactive frame rates. The resulting data is in Table 1. To put these numbers in context, the painting from Figure 1 is a typical example made on a desktop computer, which took 383,145 simulation steps and totals 685,898 splats over 18,192 strokes, with an average simultaneously active splat count of 134 and maximum of 6447.

ms/frame	iPad	MacBook Pro
33	400	4000
100	1000	10000

Table 1: *Approximate number of splats that can be simulated and rendered simultaneously at different frame rates and on different hardware.*

The fact that the number of active splats dictates performance implies that the way to manage runtime performance of the application is to limit that number. Towards that end, we set the max lifetime of the different brush types such that only a target number of splats are flowing at the same time for average painting speeds.

Intuitively, thicker strokes made by larger brushes should be more costly to render because of their higher pixel count (especially for limited pixel fill-rate devices like the iPad), but they are actually faster because fewer splats are placed per path length of stroke. This is because stamp spacing is set as a percentage of stamp size, so thicker strokes have more distance between consecutive stamps and result in fewer active splats while painting. Very small brush strokes result in many more simultaneous active splats and therefore are much slower to render. To combat this effect, we reduce the lifetime of splats in small strokes, to keep the target number of active splats more manageable.

6 Discussion

As a result of our public release, we have been able to collect a large amount of feedback from users on the painting application, and also to see how a wide variety of users are successful with, or struggle with, creating their compositions.

6.1 User Feedback

In general, user reactions have been mixed. A significant amount of the iPad application feedback has focused on the interface and on other painting features such as undo and layers. A common lament was, "I wish I had some sort of pen or brush input device." Specifically regarding the actual dynamic paint behavior, while many users have been impressed by the appearance, there is difficulty in learning to control it — one user said, "I definitely need to work with it some more and I have a feeling I'll start catching on to how to better work with it." Another later said, "The interaction feels very natural to me now." This is quite similar to traditional watercolor paints, which are some of the most difficult artistic media to master. The artists who persevered and gained greater control over the app (see Figure 6 for examples) were able to use the blending behaviors to achieve complex color gradients and textures that would have been very difficult to make with other digital paint tools. Some of their comments on the behavior: "I like the 'flow' in the paint, works well when painting using opacity," "Its like trying to paint with superwet watercolor," and "Great water-colour-ish style here." It is difficult to objectively state whether the comments are 'good' or 'bad' — as with any other artistic medium, personal style of the artist is a dominant influence, and where some artists will find confusion and frustration, others find success and serendipity, just as with real watercolors.

6.2 Limitations

While having many advantages, our novel formulation also has disadvantages over other paint simulation methods. The sparse, vector representation of paint is good for reducing computational complexity, but it does raise the cost for dense detail, which means vector representations of granulation texture, etc. are infeasible with

purely vector data. Heavy branching, intricate flow, and complex textures are all difficult to represent with pure vector formats, which is why we rely on a raster technique for pigment granulation.

Furthermore, because of the procedural approach to the pigment evolution, simulating the full range of real paint behaviors becomes increasingly difficult and the algorithmic complexity does not scale well. Conversely, a physical simulation can add the physical parameters into a unified framework in a straightforward manner to reproduce other types of paint such as acrylics or oil paints, or to modify real physical properties of pigment such as viscosity or particle size. The fidelity with which real paint can be faithfully reproduced with a physical simulation is higher than with our approach. Our goal was to achieve a balance between interactivity and quality, so we focus on a reproducing a similar range of dynamic behaviors, but with a distinct expressive experience.

7 Conclusion

We have presented a novel formulation of a dynamic and interactive paint engine that reproduces many of the important behaviors of watercolor paint. The splat model is lightweight enough to allow low-powered devices such as iPads to easily run the painting application. The vector representation makes rendering at any resolution possible with high quality anti-aliasing for final output. We demonstrated that through exposed parameters of the model, a wide variety of different types of brushes and effects can be achieved, allowing a full range of expressive and creative possibilities.

To evaluate the utility of our algorithm, we commercially released an iPad application that implements a subset of the full algorithm's functionality, where it was downloaded over 4500 times the first day and achieved a position of seventh best selling paid application in the store.

While watercolor paint simulation was our inspiration, the difficulty of achieving an experience that matches artists' carefully trained expectations about paint behavior is extremely high, and unmet expectations can greatly sour the experience (a sort of uncanny valley effect). Therefore, the goal of the algorithm was not to duplicate the experience of watercolor painting on a computer, but to create a range of dynamic behaviors that allow users to achieve a similar style of process and result, while at the same time having a unique character of its own. With time and further development, the particular feeling of our algorithm may attain its own aesthetic that users strive for, in a way that other media have their adherents. In this respect, we have created something new in the spectrum of traditional to digital media.

Future work is to continue to explore the range of effects that are possible with this type of algorithm, and to see if other paint media can be emulated, or other useful artistic tools. The vector splat formulation may have potential in other simulation-inspired domains as well, where compute power is not available for full-fledged physical simulations. From the application development perspective, we hope to continue to iterate on our interface, including common painting features such as layers, an interface to let users define new brush types, and improving the ability for users to more easily work from photos or other source material.

Acknowledgements

We would like to thank all of the beta testers who provided invaluable feedback. Special thanks to our contributing artists: John Derry, Paul Kercal, Mike Miller, Benjamin Rabe, Don Shank, and Mike Shaw.

References

ADOBE SYSTEMS INC., 2010. Illustrator CS5. http://www.adobe.com/.

ADOBE SYSTEMS INC., 2011. Adobe Eazel for Photoshop CS5. http://itunes.apple.com/us/app/adobe-eazel-for-photoshop/id421302663.

AMBIENT DESIGN, 2011. ArtRage 3.5. http://www.ambientdesign.com/.

ANDO, R., AND TSURUNO, R. 2010. Vector fluid: a vector graphics depiction of surface flow. In *Proceedings of the International Symposium on Non-Photorealistic Animation and Rendering*, 129–135.

BAXTER, W., WENDT, J., AND LIN, M. C. 2004. IMPaSTo: a realistic, interactive model for paint. In *Proceedings of the international symposium on Non-photorealistic animation and rendering*, 45–148.

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

CHU, N. S.-H., AND TAI, C.-L. 2005. MoXi: real-time ink dispersion in absorbent paper. In *Proceedings of ACM SIGGRAPH*, 504–511.

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

COREL, 2011. Painter 12. http://www.corel.com/.

CURTIS, C. J., ANDERSON, S. E., SEIMS, J. E., FLEISCHER, K. W., AND SALESIN, D. H. 1997. Computer-generated watercolor. In *Proceedings of ACM SIGGRAPH*.

DAHLSTRÖM, E., DENGLER, P., GRASSO, A., LILLEY, C., MCCORMACK, C., SCHEPERS, D., AND WATT, J. 2011. Scalable vector graphics (SVG) 1.1 (second edition). W3C. http://www.w3.org/TR/2011/REC-SVG11-20110816/.

HERTZMANN, A., AND PERLIN, K. 2000. Painterly rendering for video and interaction. In *Proceedings of the international symposium on Non-photorealistic animation and rendering*, 7–12.

HERTZMANN, A. 2003. A survey of stroke-based rendering. *IEEE Computer Graphics and Applications 23* (July), 70–81.

LAERHOVEN, T. V., AND REETH, F. V. 2005. Real-time simulation of watery paint. *Computer Animation and Virtual Worlds 16* (July), 429–439.

LOTTES, T. 2011. FXAA. NVIDIA. http://developer.nvidia.com/sites/default/files/akamai/gamedev/files/sdk/11/FXAA_WhitePaper.pdf.

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

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: a vector representation for smooth-shaded images. In *Proceedings of ACM SIGGRAPH*, 92:1–92:8.

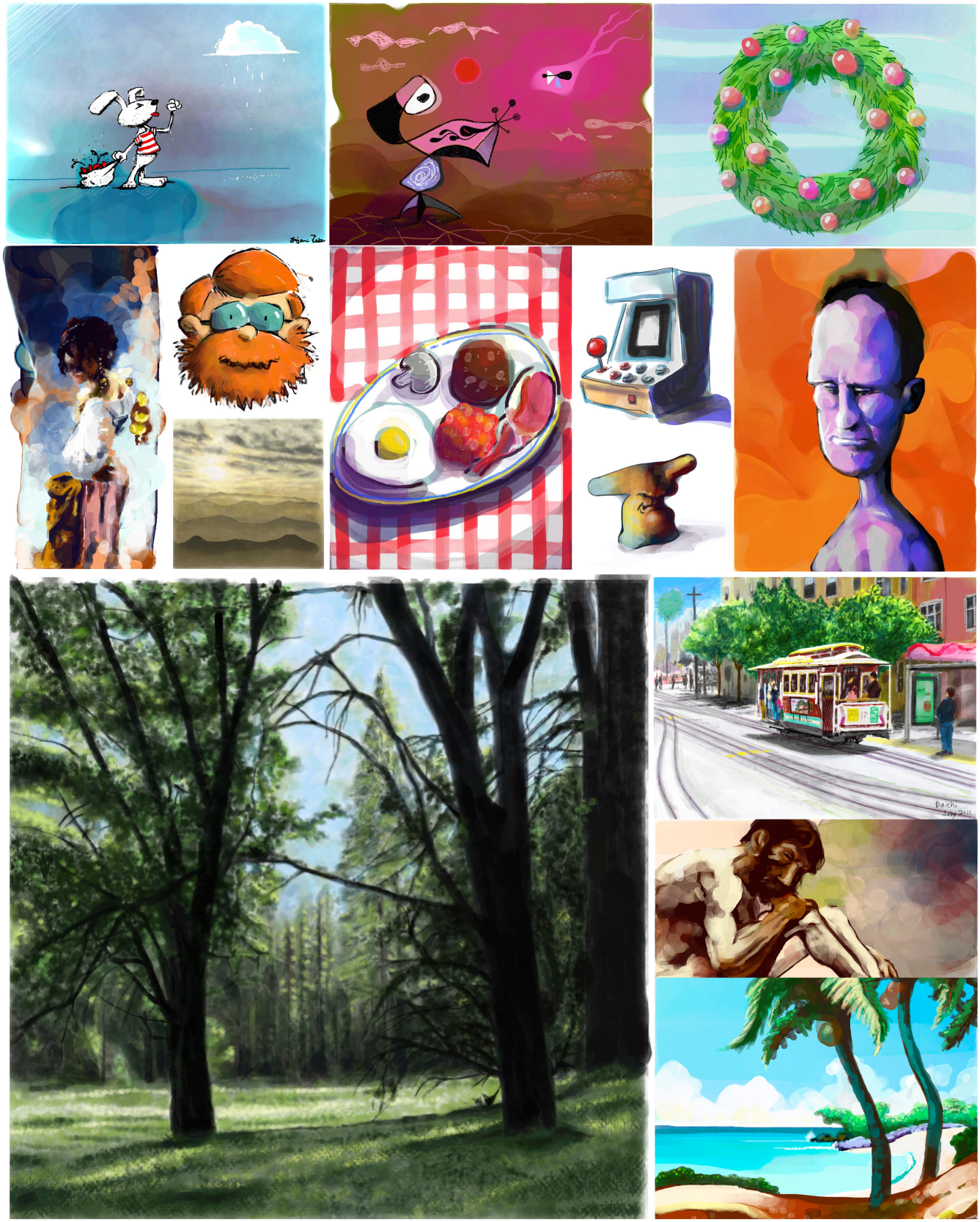

Figure 6: *Examples of paintings by professional artists exhibiting different styles characteristic of watercolor painting, using our desktop and iPad software. Note the range of achievable appearances of our algorithm. Used with permission.*