Painterly Rendering for Animation

Barbara J. Meier
Walt Disney Feature Animation

Abstract

We present a technique for rendering animations in a painterly style. The difficulty in using existing still frame methods for animation is getting the paint to "stick" to surfaces rather than randomly change with each frame, while still retaining a hand-crafted look. We extend the still frame method to animation by solving two major specific problems of previous techniques. First our method eliminates the "shower door" effect in which an animation appears as if it were being viewed through textured glass because brush strokes stick to the viewplane not to the animating surfaces. Second, our technique provides for frame-to-frame coherence in animations so that the resulting frames do not randomly change every frame. To maintain coherence, we model surfaces as 3d particle sets which are rendered as 2d paint brush strokes in screen space much like an artist lays down brush strokes on a canvas. We use geometric and lighting properties of the surfaces to control the appearance of brush strokes. This powerful combination of using 3d particles, surface lighting information, and rendering 2d brush strokes in screen space gives us the painterly style we desire and forces the brush strokes to stick to animating surfaces. By varying lighting and choosing brush stroke parameters we can create many varied painterly styles. We illustrate the method with images and animated sequences and present specific technical and creative suggestions for achieving different looks.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.5 [Computer Graphics]: Three-Dimensional Graphics and Realism – Color, Shading, Shadowing, and Texture.

Key Words: painterly rendering, non-photorealistic rendering, particle systems, painting, abstract images.

Author's current affiliation: Hammerhead Productions.
email: bjm@gg.caltech.edu or barb@hammerhead.com

1 Introduction

A painting reduces a subject to its essence. The process of painting is an artist's interpretation of the world, real or imagined, to a two-dimensional canvas. By not depicting every detail, the painter allows the viewer to complete the picture, to share in the interpretative process. Of course the process begins with the painter who, by abstracting a scene, can direct the viewer's eye to the area of interest by simplifying unimportant details. A painter can exaggerate the effect of light to create a wide tonal range that creates richness and drama at the center of interest. By using the largest brush stroke possible to represent small forms and textures, the painter creates a shorthand for conveying details. The character of brush strokes define the character of a surface and how light is reflected from it; surfaces that are well-blended imply smoothness or softness while direct, unblended strokes imply stronger lighting or more pronounced surface texture. Painters use varying edge definition, edges that are distinct in one place and lose themselves in another, to add rhythm to a composition. Letting brush strokes cross edge boundaries can also help unify an entire composition. By varying brush stroke texture, size, and direction, the artist can not only define forms, but also provide rhythm and energy that help direct the viewer's eye. Larger, smoother brush strokes tend to recede in depth while small, textured strokes depict foreground detail. A painter can even use brush strokes to represent light and atmosphere. Whatever the painting style, a certain amount of abstraction, or economy of description, strengthens the composition and provides focus [5, 6, 13].

Computer rendering provides an easy, automated way to render everything in a scene with fine detail. This creates static images that do not invite the viewer into the process. In particular, when creating images for animation, focus and simplification are essential to showing action in a clear way since the temporal nature of the image gives the viewer much less time to let their eyes wander about the scene [16]. Certainly focus and simplicity can be achieved with computer rendering tools by carefully controlling lighting and surface attributes and unnecessary detail can be obscured using hierarchical modeling, but it is still difficult to obtain the level of abstraction that is evident in a good painting. Even the brush strokes of a painting contribute to the abstraction of its subject and add another dimension to which a viewer can respond. One could not imagine looking at a Van Gogh painting without experiencing the energy of his brush strokes. Hand-drawn and hand-painted animations have an energetic quality that is lacking in most computer-rendered animation. Often when computer methods try to mimic the wavering quality of hand-drawn animation, too much randomness creeps in and makes the animation noisy. A human artist drawing each frame is better able to control frame-to-frame coherence, while maintaining a hand-crafted look.

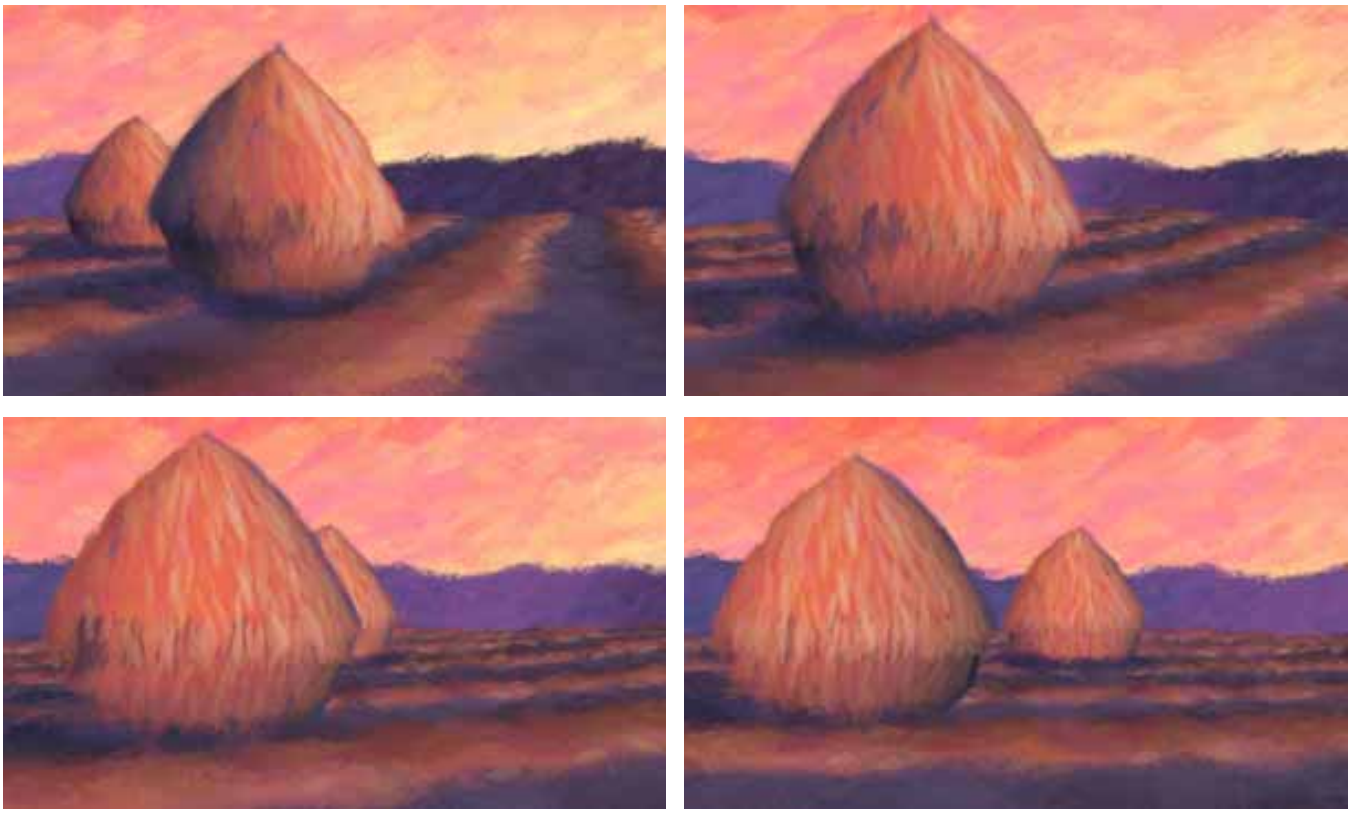

Figure 1: **Frames from a painterly rendered animation.** The painterly renderer is particularly well-suited for abstracting natural textures like the cloudy sky, hay, and plowed ground in this example. Note that the haystack texture does not exhibit the problems of traditional texture-mapping in which the gift-wrapped texture gets dense near silhouette edges. The overlapping brush strokes on the plowed ground imply volume rather than flat, painted texture as the view animates, even though the surface is planar. We use the largest brush strokes to paint the sky, using brush texture and random hue variation to create clouds that do not exist in the color reference picture. The original haystack geometry is simply a cone resting on a cylinder. We represent the hay with a brush stroke shorthand that eliminates the need to model and color every piece of hay.

We want to take advantage of the benefits of a painterly look on computer-rendered animating geometry. Aesthetically, a still frame should have the characteristics of an oil or pastel painting: details should be abstracted by shorthand brush strokes, the roundness of forms should be defined by brush stroke directions, color should break the boundaries of surfaces to create rhythm in the composition, brush stroke size and texture should be varied according to the kind of surface being depicted, and the effects of light should be exaggerated to help provide focus, all as if an artist had painted on a physical canvas. Technically, the rendered images should maintain coherence in animated sequences and should not change in a random way every frame. Images should not have the gift-wrapped look of painted textures that are mapped onto the geometry using traditional methods. Our goal is not to eliminate the need for observational understanding and artistic vision, but rather to provide a tool that automates the drawing of brush strokes, but leaves the artistic decisions about lighting, color, and brush stroke characteristics to the user.

The focus of most rendering research in the last two decades has been on the creation of photorealistic imagery. These methods are quite sophisticated, but tend to create imagery that is mechanical-looking because detail is represented very accurately. Recently there has been a movement toward more creative and expressive imagery in computer graphics but few techniques that provide ways to achieve different looks, especially for animation. Some computer painting tools can mimic successfully the hand-drawn line quality, painterly look, and energy of traditional media, but these tools typically work only for still frames. These tools and related work are discussed in section 2.

Our solution, presented in section 3, is to generate a set of particles that describe a surface, depth-sort the particles in camera space, and render them as 2d brush strokes in screen space using a painter's algorithm [7]. The look of the 2d brush strokes, including color, size, and orientation, is derived from the geometry, surface attributes, and lighting characteristics of the surface. These attributes are designed by the user and either associated directly with the particles or encoded in rendered images of the geometry, called reference pictures. We illustrate our work with images and animations that have been successfully rendered to achieve a painterly look using this algorithm (Figures 1, 5, and 7), and in section 4 we discuss the images. Finally, in section 5, we present aesthetic techniques and technical considerations for creating various image styles.

2 Related Work

Our work combines core ideas from two areas of previous work: 1) painterly rendering of still images from reference pictures and 2) particle rendering. From the first research area, our work was most directly inspired by [4]. Haeberli described a system for creating painterly images from a collection of brush strokes that obtain

```
create particles to represent geometry
for each frame of animation
    create reference pictures using geometry, surface
        attributes, and lighting
    transform particles based on animation parameters
    sort particles by distance from viewpoint
    for each particle, starting with furthest from viewpoint
        transform particle to screen space
        determine brush stroke attributes from
            reference pictures or particles and randomly
            perturb them based on user-selected parameters
        composite brush stroke into paint buffer
    end (for each particle)
end (for each frame)
```

Figure 2: **Painterly rendering algorithm.**

their attributes, such as position, color, size, and orientation, from synthetically rendered or photographic reference pictures. Several commercial systems, such as [3] and [8], have incorporated the idea of reference pictures, and Saito and Takahashi use a similar concept, the G-buffer, to create simplified illustration-type images [11]. Our system also uses reference pictures to obtain brush stroke attributes.

In Haeberli's system brush stroke positions are randomly distributed, so successive frames of an animation would change randomly. Alternatively, the positions and sizes of brush strokes could remain constant over the animation, but this creates the "shower door" effect, because brush strokes are effectively stuck to the viewplane not to the animating surfaces. The University of Washington illustration systems [12, 17] provide methods for rendering images in a pen-and-ink style, but again, the randomness that is employed to achieve the hand-drawn look would cause successive frames to change randomly.

We solve the temporal randomness problem by using particle rendering methods. If we treat brush strokes as particles that are stuck to surfaces, we eliminate both the "shower door" effect and random temporal noisiness. Reeves first presented an algorithm for rendering particles without using traditional 3D models to represent them, instead drawing them as circles and motion-blurred line segments in screen space [10]. We also render particles in screen space, but use 2d brush stroke shapes instead of circles and line segments.

Rendering 2d shapes in screen space is one of the core concepts of our work. Fleischer et al. [2] described a similar method, except they place *3d* geometric elements on surfaces in *model* space, which are then rendered traditionally as geometric textures such as scales, feathers, and thorns. The appearance of their 3d shapes compared to our 2d brush strokes is quite different.

Finally, Strassmann presented a technique for modeling brush strokes as splines for Sumi-E style painting, a Japanese brush-and-ink technique [14]. This system is designed primarily for still images, but does provide a simple method for animation. The user specifies key frames for each brush stroke that are interpolated over time. Our approach is different in that we provide a rendering technique rather than an interactive system and we are emulating a more impressionistic style of painting with short paint dabs rather than long graceful strokes.

3 Painterly Rendering

In this section, we describe our painterly rendering algorithm as shown in Figure 2.

We begin by creating a particle set that represents geometry such as a surface. The particles are transformed to screen space and sorted in order of their distance from the viewpoint. We use a painter's algorithm to render particles as 2d brush strokes starting with the particles furthest from the viewpoint, and continuing until all particles are exhausted. Each brush stroke renders one particle. The look of the rendered brush strokes, including color, shape, size, texture, and orientation, is specified by a set of reference pictures or by data that is stored with the particles. Reference pictures are rendered pictures of the underlying geometry that use lighting and surface attributes to achieve different looks. The attributes for a particle are looked up in the reference pictures in the same screen space location at which a particle will be rendered finally. Figure 3 illustrates the painterly rendering pipeline.

In the following sections, we begin by discussing particle placement. Next we explain brush stroke attributes, how they are applied, and how the reference pictures that encode the attributes are created. Finally, we present various ways of manipulating the brush stroke attributes to produce painterly images.

3.1 Generating Particles

There are many methods of populating a surface with particles, such as those described in [15] and [18]. We employ a simple method that starts with a parametric surface and a desired number of particles. We tessellate the surface into triangles that approximate the surface. Then, for each triangle, we compute its surface area and randomly distribute particles within it. The number of particles for a triangle is determined by the ratio of its surface area to the surface area of the entire surface. The particle placer may store additional information with the particles such as color, size, and orientation. After the initial particle placement, these additional attributes or the particles' positions may be modified by performing various functions on them. Alternatively, the entire particle set can be generated from a particle system simulation [9].

3.2 Specifying and Applying Brush Attributes

In order to render a brush stroke, we need the following attributes: **image, color, orientation, size,** and **position.**

The brush **image** is a color image with alpha. The image may be solid or it may contain texture as shown in Figure 4. A single image may be used as is or it may be used to cut a shape from a random position in a sheet of texture, providing each brush stroke with unique texture. Although the brush can be a full color image, we typically use monochrome images that are the same in all channels so that the brush itself does not impart color, just texture.

Orientation, color, and **size** are either stored with the individual particles or obtained from reference pictures. If these attributes are associated with the particles, then they are used directly by the renderer; otherwise, the attributes are sampled from reference pictures which encode information about surface geometry and lighting characteristics by screen space location. Reference pictures can be generated in several ways, but typically are rendered images of the particle set or surface. After a particle's position is transformed to screen space, we use the 2d transformed position to look up color, orientation, and size information in the same 2d location in the appropriate reference pictures. Example reference pictures for these attributes are shown in Figure 3.

The reference picture used for **color** information is typically a smooth-shaded rendered image of the surface with appropriate color attributes and lighting. Texture maps are generally not necessary except to describe broad color changes across the surface. The painterly rendering will provide texture and high frequency variations in color.

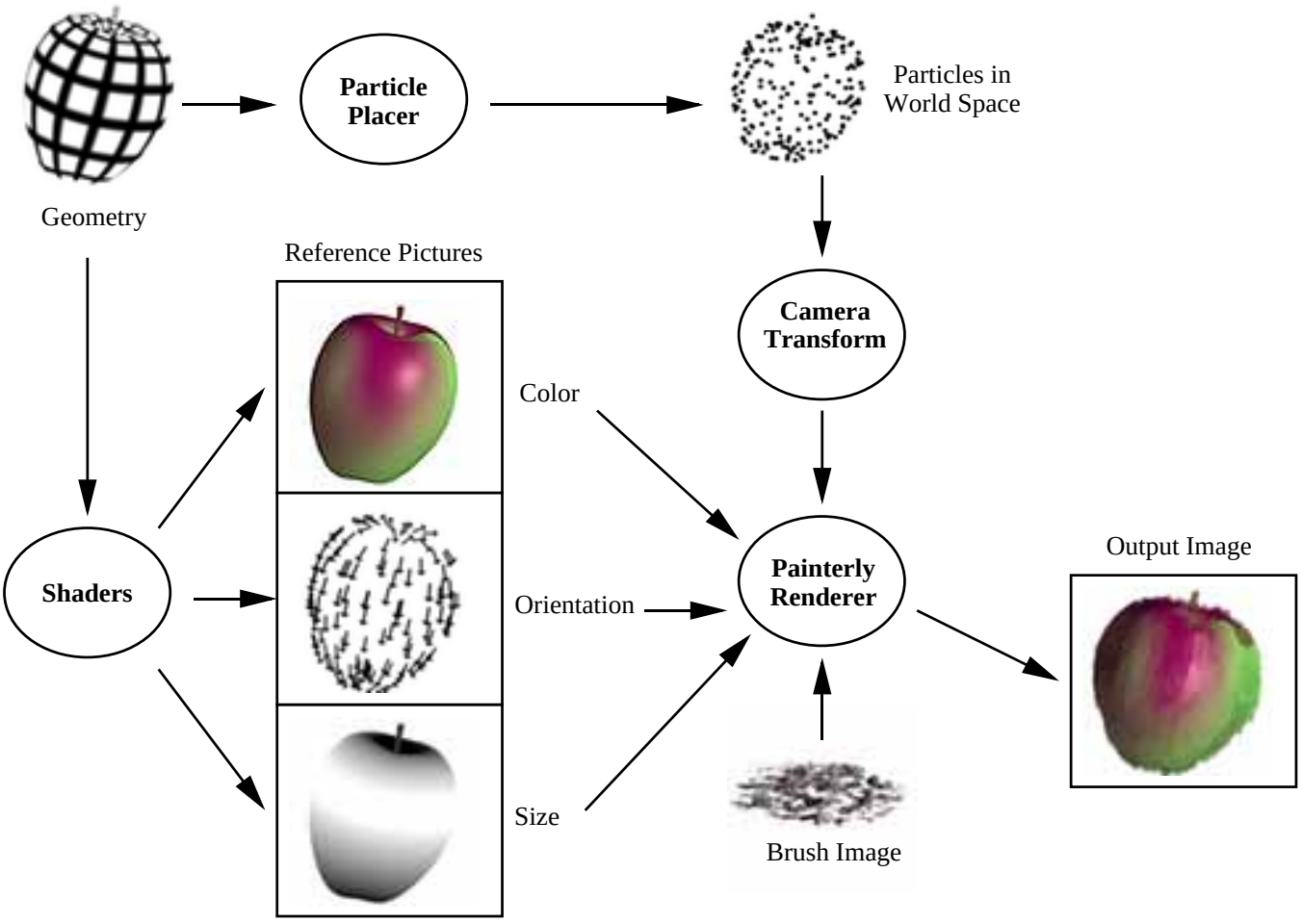

Figure 3: **An example of the painterly rendering pipeline.** The *particle placer* populates a surface with particles. The surface geometry is rendered using various *shaders* to create brush stroke attribute reference pictures. Note that the arrows in the orientation image are representational in this diagram; the orientations are actually encoded in the color channels of the image. The particles, which are transformed into screen space, the reference pictures, and the brush image are input to the *painterly renderer*. The renderer looks up brush stroke attributes in the reference pictures at the screen space location given by each particle's position and renders brush strokes that are composited into the final rendered image.

The reference picture that encodes **orientation** information is an image made with a specialized shader that encodes surface normals in the resulting image. This surface normal shader projects the 3d surface normals into two dimensions along the view vector or another specified vector. Alternatively, we may constrain orientations to line up with the direction of a surface parameter or texture coordinate.

Finally, the brush **size** reference picture is a scalar image that encodes x and y scaling information. We linearly map the range of values in the image to the range of user-specified sizes so that the areas with small values are painted with the smallest brushes and the areas with high values are painted with the largest brushes. Again, we can use lighting, texture maps, or specialized shaders to achieve the desired look.

Brush stroke **position** comes from the particle's position in screen space. Position may be modified by a function such as moving it in the direction of a velocity vector or adding noise.

To apply the attributes, the brush image is either used directly or cut from a sheet of texture, multiplied by the color and alpha, scaled by the size, and rotated to the orientation, each as specified in the corresponding reference picture or by data stored with the particle.

Figure 4: **Some brush images used to create the paintings in this paper.**

Once attributes are applied, brush strokes are composited into the final rendered image at the position specified by the particle.

3.3 Animating Parameters and Randomness

It is possible to animate brush stroke attributes by animating characteristics of the reference pictures, but it is necessary for the reference

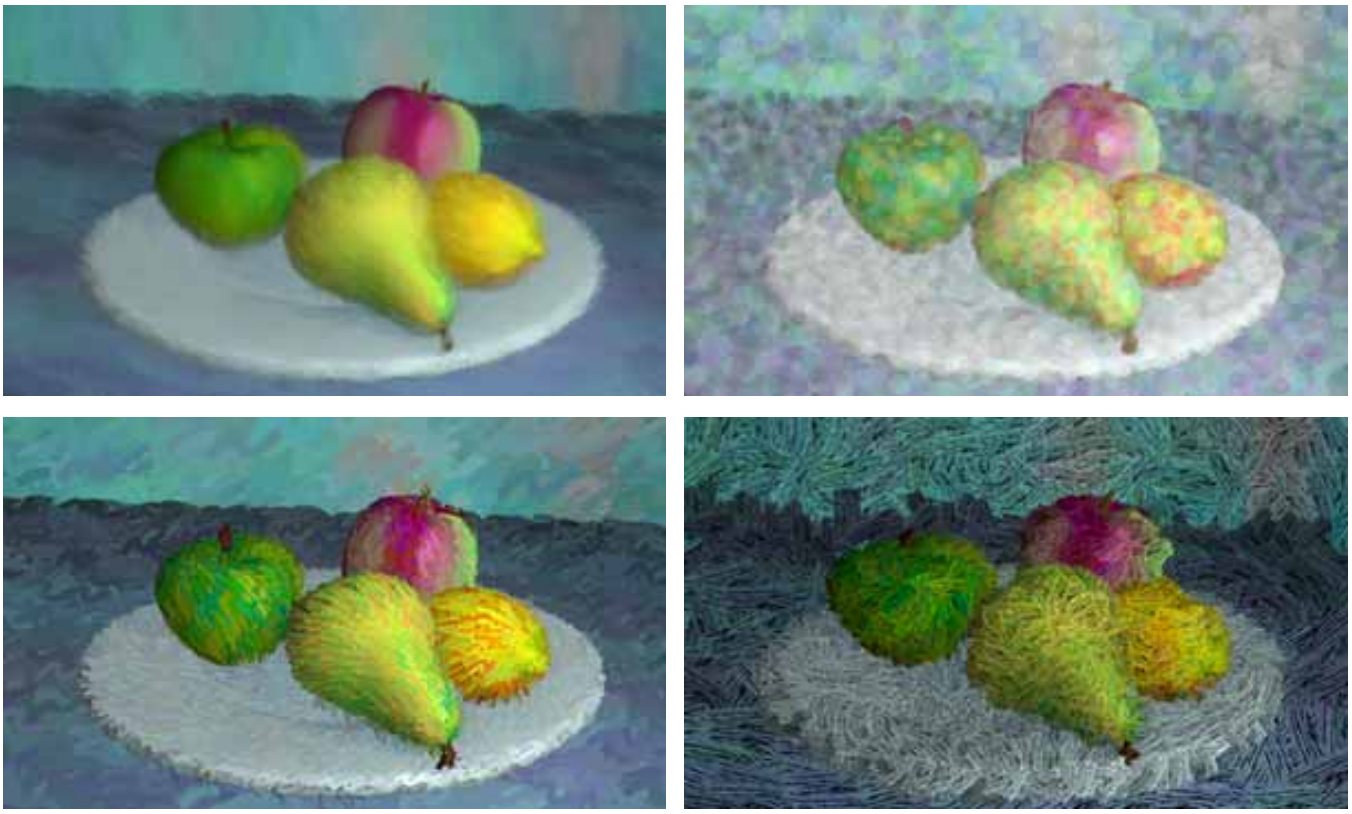

Figure 5: **Four styles of painterly rendered fruit.** By choosing different brush images and painting parameters, we have created four different looks from the same set of reference pictures. The upper left image has the soft, blended quality of a pastel painting. The pointillistic version, in the upper right, remaps the original saturations and values from the color reference picture to a new range. A squiggle brush image and increased hue variation were used to create marker-style strokes in the lower left image. The brush used to create the lower right contained some opaque black that helps to create a woodcut print style.

pictures to change smoothly over time so that the final rendered images are not temporally noisy.

Using randomness is important in achieving a hand-crafted look; therefore, we can randomly perturb the brush stroke attributes based on user-selected parameters. Figure 6 illustrates the lack of richness and texture that results when randomness is not used.

To maintain coherence, a seed is stored with each particle so that the same random perturbations will be used for a particular particle throughout an animation. The user specifies the amount of randomness by choosing a range about the given attribute. For example, we may specify that brush rotations be determined by an orientation reference picture, but to eliminate the mechanical look of the brushes lining up perfectly, we specify that we are willing to have brush orientations fall within the range of -10 to +20 degrees from the orientation given in the reference picture. The resulting slightly random orientations give the strokes a more hand-crafted look.

Figure 6: **Applying randomness to brush stroke attributes.** This image was rendered without color, orientation, or scale variation. Compare it to the images in Figure 1 which were painted with all of those attributes jittered. Note how the painterly texture of the sky and mountains is dependent on random color variations. In the haystacks, orientation and scale changes make them look less mechanical in the jittered version.

4 Results

Figures 1, 5, and 7 are images rendered using our algorithm that show a variety of different painterly looks. In Figure 1, we show frames from a Monet-style haystack animation. The still frames look like oil paintings and the brush strokes animate smoothly throughout the animation. The painterly renderer is particularly well-suited to the impressionist style because it composes a painting with many small brush strokes. In this example, we are not concerned with defining exact boundaries and instead let the overlapping brush strokes create a rhythm that unifies the composition. Large brush strokes tend to extend beyond the silhouette edge, cre-

Figure 7: **Beach ball animation frames.** In this example the beach ball is bouncing, squashing, and stretching from frame to frame. Our technique works as well for animating objects as for the haystack example where only the camera position is animating.

ating a semi-transparent look that is most apparent when surfaces are animated. We believe this adds to the painterly look. Using smaller, denser, or more opaque strokes near the edges would create a more opaque, solid look. We have used the painterly technique of abstraction to depict many of the surfaces in this animation. For example, in the sky we used the brush texture and color variation to abstractly depict sweeping clouds. The hay is captured with a brush stroke texture that shows an appropriate amount of detail. Finally because our technique uses overlapping 2d brush strokes, we have avoided the gift-wrapped look of a smooth-edged, texture-mapped surface.

In Figure 5, we show a plate of fruit rendered in four styles. The reference pictures used to create the images were the same for all four, with the exception of the orientation image for the lower right image. The different looks were achieved by varying the brush image, the amount of jittering, and the brush size. Of course even more looks could be created by changing the reference pictures, but one of the strengths of the painterly renderer is the richness of the user-selectable parameter set. For example the upper right image was brightened and desaturated by mapping colors in the color reference picture to new saturation and value ranges. Conversely, the colors in the lower right image are richer because the brush image contained some opaque black. In this painting, the brush strokes become the dominant subject of the painting.

Finally, in Figure 7, we show three frames from an animated bouncing ball sequence. Our technique works equally well for an animating, deforming object like the squashing and stretching ball in this example, as for an animated camera as shown in the haystacks example. Large brush strokes give the ball an imprecise boundary which gives the ball animation a hand-drawn look quite different from the mechanical look that a traditionally-rendered version would have.

5 Discussion and Techniques

As with any image creation process, it takes some experimentation to get the desired image. In this section, we describe some of the techniques that we've discovered. We begin with our strategies for achieving creative images and then present some technical discussion on how we achieve them.

5.1 Creative Techniques

We have discovered many techniques for rendering aesthetically pleasing images. Chief among these is separately rendering subsets of the particle set and compositing these layers into a finished image. We find that our most successful images are created using traditional painting methods such as creating a rough value underpainting with large brush strokes, adding layers of color to define the form, and then adding small brush strokes where we want more detail. Our implementation provides a skip operation that allows us to render every *n*th particle. We typically render the surface in two or three layers using image processing techniques to shrink the silhouette edge toward the center of the object. The outside layers are painted sparsely, while the inside layers are painted thickly. We also use image processing techniques to isolate highlight and shadow areas to be rendered separately. Building up layers of semi-transparent textured brush strokes, perhaps even rendering the same particle multiple times with different brush stroke characteristics, is important in achieving the painterly look. In the haystacks example, the haystacks consist of four layers: a rough dark blue underpainting, an overall orange layer, a yellow detail layer, and a sparse white highlight layer. These layers and how they contribute to the final image are shown in Figure 8.

We also usually render the objects in a scene as separate layers. In the haystack example, we painted the sky, mountains, field, each haystack, and each haystack shadow as a separate layer. This allowed us to use very large brush strokes on the sky and not worry about them creeping too far into the mountains. By rendering these layers separately, we were better able to use the painting parameters most appropriate for each layer. The fruit images in Figure 5 were rendered in three layers: the wall, the table, and the plate of fruit. In this case, because the brush stroke characteristics of each fruit were similar, we wanted the brushes strokes to interact as much as possible to enhance the painterly look.

We typically use only one light source to maintain focus in the composition. We use exaggerated hue as well as value variations to distinguish light and shadow areas. For example, the sunset light on the haystacks is emphasized through exaggerated use of orange and blue. Shadows may be rendered by compositing a shadow element onto the color reference picture and rendering the surface and the shadow at the same time, or shadows may be painted as a separate layer and composited, giving the user more creative control.

We use many traditional painting techniques such as using background color in shadow areas to help them recede and juxtaposing complementary colors, such as the orange and blue of the haystacks, to create a shimmering light effect. We repeat brush stroke color, size, and texture in different areas of the scene, as shown in the fruit example, to marry the various elements into a unified composition. Users of the painterly renderer are encouraged to examine the numerous existing texts on traditional painting techniques for more possibilities.

5.2 Technical Considerations

If reference pictures are used, it is often helpful to "grow" the reference image outward using image processing techniques, so that when we look up particular screen locations we don't fall off the edge of the surface onto anti-aliased or unrendered parts of the image. This is applicable only if we are rendering layers separately and then compositing them afterwards. To ensure that individual

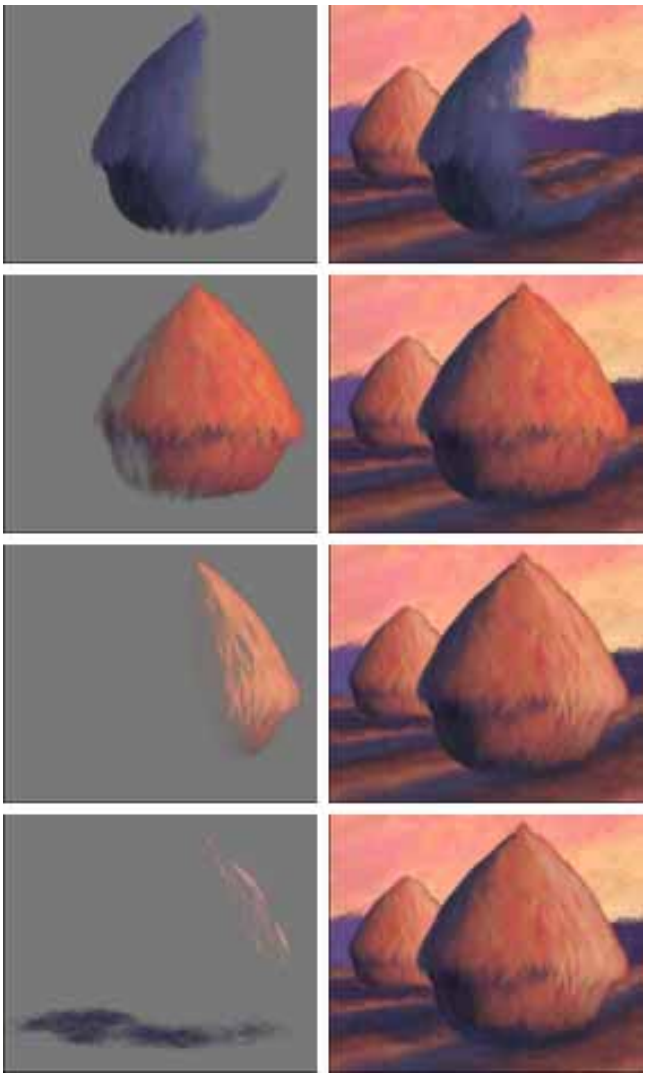

Figure 8: **Compositing a haystack from several layers.** Each layer of the haystack is shown by itself on the left while its contribution to the composited image is shown on the right. We used image processing techniques on the color reference picture to isolate the shadow and highlight areas to be painted separately. Following traditional painting techniques, we created a dark blue underpainting of the shadow areas as shown in the top row. The next layer provides most of the color and texture of the haystack, but allows some of the blue underpainting to show through. The bottom two rows show two separate detailed highlight layers and a final shadow layer that helps integrate the haystack with the field. For each layer, we changed the brush size and the amount of color variation.

brush strokes do not jitter in size and orientation slightly with every frame, it is also useful to blur the orientation and size images slightly. Perfect particle placement and sub-pixel sampling would eliminate the need for these steps, but we have found that these techniques work well in practice.

The simple surface normal shader that we described previously provides surface normal information based on a particular orientation of the surface after it has been through a camera transformation. But as a surface animates, so does its orientation with respect to the camera. This gives a particular look, but we prefer to have brush strokes oriented with respect to the surface and not change as the surface animates. To achieve this, we have specified our desired orientations with respect to the *(u, v)* surface parameters in texture maps. A special shader looks up values in the maps and then applies the camera transformation to them to obtain the screen space orientation that is output to the reference picture.

Brush stroke attributes may be stored with the particles or encoded in reference pictures. An advantage to storing attributes with the particles is that we avoid aliasing errors looking up values in reference pictures. An advantage to using reference pictures is that they are usually quickly rendered and thus easily changed and can encode more complex lighting information. Storing attributes with particles is better for those that are unlikely to change because rerunning the particle placement or simulation may be costly. In practice, a mixture of the two methods works well.

At first glance, one might suggest we not render back-facing particles, but this is very important in animation since particles will pop on and off as they become visible and invisible if we cull the back-facing ones. If we always render them, however, they will be revealed gradually as they become visible. Front-Facing brush strokes must be dense enough to obscure back-facing particles, unless a translucent effect is desired. In practice, we find that we do not always want to render a completely opaque object if we are building up textured layers of paint, but we also do not want to see through to the back-facing brush strokes if they are animating. In this case, we do cull back-facing particles, letting them fade in as they get close to front-facing to eliminate the popping effect as particles come into view. This was necessary in the haystack example so that as the view animates, we do not see the back side of each sparse layer through the front.

One should also note that because particles are sorted by distance from the viewpoint at each frame, there will be some popping of brush strokes in front or behind one another as particles animate, but with some attention to brush stroke size and translucency, this effect is not visually problematic and can add to the painterly effect.

6 Future Directions

Although our use of the renderer thus far has been to create images that are entirely painted, we can imagine incorporating this look with traditional rendering methods. For example, when artists depict foliage, they don't paint every leaf. Instead, they use brush strokes to abstractly represent the leaves. Certainly particle rendering methods have been used for this purpose before [10], but we believe our technique can eliminate complex modeling issues such as generating realistic tree models made of particles, and that the level of control we provide for achieving different looks will prove to be a more powerful but easier-to-use tool. We foresee using this method to render surfaces that are difficult to model and render using traditional geometry and texture maps. This class of objects, which includes many of those found in nature, must be abstracted when rendered because of their high complexity.

Our renderer does not handle changing object sizes in an automated way. We can address this issue with staging or by animating the brush size reference picture; however, it would be helpful if the renderer could automate brush stroke size based on the screen surface covered, and then change the size smoothly as the object changes size using multi-resolution techniques such as those used by [1].

We would like to use a better particle placement method that covers both the geometric surface and screen space more evenly. While we can address this situation with the layer rendering technique described above, this is not always satisfactory and also requires active intervention by the user. Metric tensor techniques [18] could be used to specify particle density for surfaces that do not radically change their orientation with respect to the viewpoint within

an animation, but other multi-resolution methods might be required for those surfaces that do change orientation.

Finally, although we are unlimited in brush stroke shape, we find a rectangular or oval shape works best to show changes in orientation, but these shapes stick out along the edges of curved surfaces. We would like to implement longer, deformable brushes than can follow curves on a surface.

7 Conclusions

We have presented a new technique for rendering animations in a painterly style. Our work has brought together two previous rendering methods: using reference pictures to define 2d brush stroke attributes and using particles to define the locations where brush strokes will be rendered. Our algorithm solves the two major problems of rendering animations with previous painterly techniques. First, images created by our renderer are coherent over time and do not exhibit random frame-by-frame changes. Second, brush strokes stick to animating surfaces, not to the viewplane, thus eliminating the "shower door" effect. We have illustrated our algorithm with images that have painterly qualities such as exaggerated use of light, broken silhouette edges that create rhythm, brush stroke textures and sizes that describe surface qualities, and abstracting the subject to strengthen and unify the composition.

8 Acknowledgments

Many thanks to Ken Hahn, Scott Johnston, Jason Herschaft, and Craig Thayer for turning the painterly renderer prototype into a production program, contributing many new ideas and features along the way. Ken Hahn also wrote the particle placer and went beyond the call of duty to make many last minute bug fixes. Thanks to Dave Mullins and Andrea Losch for modeling and rendering support, Craig Thayer and Scott Johnston for valuable comments on early drafts of the paper, and Nancy Smith for video production support. We are grateful to Al Barr and Scott Fraser of Caltech and to Hammerhead Productions for providing production facilities. Finally, many thanks to David Laidlaw for technical discussions about the painterly renderer, extensive paper reviews, diagrams, many hours of paper production support, and help coping with my pregnancy madness.

References

[1] Deborah F. Berman, Jason T. Bartell, and David H. Salesin. Multiresolution painting and compositing. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 85–90. ACM SIGGRAPH, ACM Press, July 1994.

[2] Kurt W. Fleischer, David H. Laidlaw, Bena L. Currin, and Alan H. Barr. Cellular texture generation. In *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 239–248. ACM SIGGRAPH, Addison Wesley, August 1995.

[3] Fractal Design Corporation. *Fractal Design Sketcher*. Aptos, California, 1993.

[4] Paul E. Haeberli. Paint by numbers: Abstract image representations. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 207–214, August 1990.

[5] Carole Katchen. *Creative Painting with Pastel.* North Light Books, 1990.

[6] Gregg Kreutz. *Problem Solving for Oil Painters*. Watson-Guptill Publications, 1986.

[7] Martin E. Newell, R. G. Newell, and T. L. Sancha. A solution to the hidden surface problem. In *Proc. ACM Nat. Mtg.* 1972.

[8] Parallax Software Limited. *Matador Paint System.* London, 1995.

[9] W. T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graphics*, 2:91–108, April 1983.

[10] William T. Reeves and Ricki Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 313–322, July 1985.

[11] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-D shapes. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 197–206, August 1990.

[12] Michael P. Salisbury, Sean E. Anderson, Ronen Barzel, and David H. Salesin. Interactive pen–and–ink illustration. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 101–108. ACM SIGGRAPH, ACM Press, July 1994.

[13] S. Allyn Schaeffer. *The Big Book of Painting Nature in Oil.* Watson-Guptill Publications, 1991.

[14] Steve Strassmann. Hairy brushes. In *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 225–232, August 1986.

[15] Richard Szeliski and David Tonnesen. Surface modeling with oriented particle systems. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 185–194, July 1992.

[16] Frank Thomas and Ollie Johnston. *Disney Animation–The Illusion of Life*. Abbeville Press, 1981.

[17] Georges Winkenbach and David H. Salesin. Computer–generated pen–and–ink illustration. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 91–100. ACM SIGGRAPH, ACM Press, July 1994.

[18] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 269–278. ACM SIGGRAPH, ACM Press, July 1994.