Interactive Vector Fields for Painterly Rendering

Sven C. Olsen
Department of Computer Science
Northwestern University

Bruce A. Maxwell
Department of Engineering
Swarthmore College

Bruce Gooch
Department of Computer Science
Northwestern University

Abstract

We present techniques for generating and manipulating vector fields for use in the creation of painterly images and animations. Our aim is to enable casual users to create results evocative of expressionistic art. Rather than defining stroke alignment fields globally, we divide input images into regions using a colorspace clustering algorithm. Users interactively assign characteristic brush stroke alignment fields and stroke rendering parameters to each region. By combining vortex dynamics and semi-Lagrangian fluid simulation we are able to create stable, easily controlled vector fields. In addition to fluid simulations, users can align strokes in a given region using more conventional field models such as smoothed gradient fields and optical flow, or hybrid fields that combine the desirable features of fluid simulations and smoothed gradient information.

Key words: Non-photorealistic rendering, Fluid Simulation

1 Introduction

Creating expressionistic images of has long been a goal of nonphotorealistic rendering. Stroke based rendering systems, which create a painterly version of a source image, often cite the work of Van Gogh and other expressionist painters as a primary inspiration for their algorithms (Hays et al. [12], Hertzmann [13]).

Our work is distinguished from it's predecessors in that we have included several tools for designing the vector fields used to align the brush strokes, and that these fields are not globally defined, but rather assosiated with particular regions of the image.

Using a region based approach allows us to create output images in which the style of the brush strokes varies radically depending on the region. This is a feature sometimes seen in Van Gogh's paintings, it is particularly remarkable in landscapes such as *Starry Night* and some of his self portraits (see Fig. 3). Background regions are filled with long curling strokes arranged in turbulent patterns, while a more conservative style is used in areas of greater detail. The swirling patterns in his skies

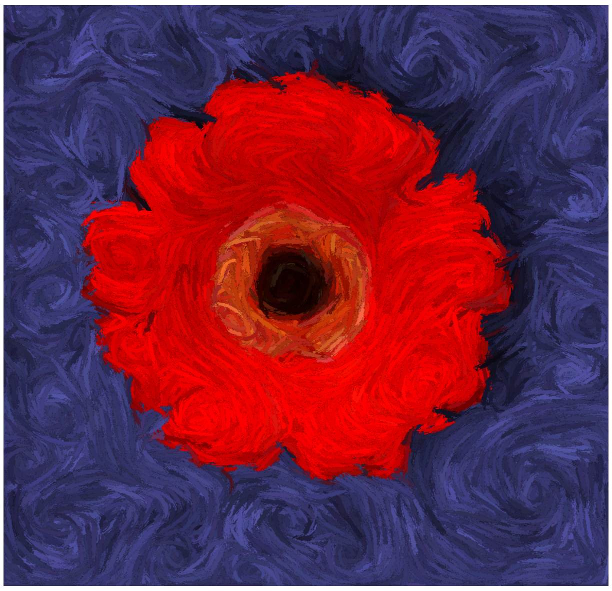


Figure 1: Our system allows a user access to a variety of vector field methods. In this rendering of a red poppy, fluid fields determine stroke alignment in the blue background region while RBF fields are used inside the flower.

and backgrounds can be one of the most striking characteristics of Von Gogh's work, but similar effects are difficult to achieve with a traditional stroke based filter, because stroke the behavior parameters must be defined globally. Thus effects which rely on a juxtaposition of different stroke styles are very difficult to achieve. Such effects may possible, given careful hand editing of the alignment fields and close user supervision of the algorithms. However our techniques are designed to make the creation of such renderings trivial - allowing casual users to easily create painterly results with the swirling backgrounds that have become one of the most recognizable signatures of expressionist art.

The purpose of this work is not to create replicas of Van Gogh paintings, nor to exactly imitate his style. Creating convincing imitations of actual paintings is beyond the scope of the textured stroke framework that we have adopted. Rather our purpose is to further empower casual users to create digital art in an expressionist style; providing tools and algorithms that give them more control than previous systems, but which are still simple to use and allow for large amounts of automation.

In order to create fields with a turbulent character we make use of a semi-Lagrangian fluid simulator, similar to the one introduced by Stam [20]. Most of the time this simulation is bounded by region definitions. User control is possible through the manipulation of vortex particles. It is also possible to hybridize the results of the fluid simulation with a gradient-based field, leading to a result which better conform to object edges, but still possesses fluid characteristics farther from boundaries.

Most stroke based painterly rendering algorithms can be set in motion, creating animations based on either still images or input videos. Our user designed vector fields are a particularly well suited to such extensions, because the fields can be made to evolve over time in a way that reflects the properties of the source image as well as the desires of the user.

2 Related Work

We follow in the tradition of stroke-based non-photorealistic rendering, which began with Haelberi [11]. Haelberi's system allowed users to generate impressionistic images by creating a number of brush strokes colored according to an input image. Unlike some more recent papers [6, 2], Haelberi did not try to simulate the characteristics of actual paint, rather his goal was to allow users to quickly create images which possessed many of the visual merits of paintings.

Much of the inspiration for this paper comes from two descendants of Haelberi's system. Litwinowicz [17] and Hertzmann [13], both of whom produced images in which large, homogeneous areas of the source images may be rendered using long flowing brush strokes. Hertzmann's method explicitly sets out to create long curved brush strokes, while in the method of Litwinowicz a more modest flowing effect arises as a result of the use of thin plate splines to interpolate gradient information. Both systems can create brush stroke patterns reminiscent of some of Van Gogh's expressionist paintings, strokes fluidly curling and turning around each other. However, both systems align their brush strokes using the color gradient of the source image; and therefore users have little control over the effect .

The painterly animation system of Litwinowicz [17] introduced several techniques which we have incorporated into our own work, most notably a stroke density control algorithm based on Delaunay triangulations, and the use of thin plate interpolations to specify stroke alignment fields.

Hertzmann [13] presented techniques through which users could enter a description of a painting style, and the system would then a version of the input image using that style. He also introduced the technique of putting down multiple layers of brush strokes, painting first at a coarse level of detail and then at progressively finer scales. This is one technique which we did not implement for our own system. Hays et al. [12] expanded on Hertzmann's work, and like Litwinowiz used radial basis functions to create smoothed gradient values. Unlike Litwinowiz, Hays et al. performed the interpolations across time as well as space, which allowed for highly temporally coherent video based animations.

Arbitrary vector fields have been incorporated into stroke based rendering in the past – Hertzmann et al. presented an interactive system in which procedural fluid fields could be used in place of optical flow fields when warping a dynamic scene [14]; Hays et al. includes a discussion of similar techniques, in which arbitrary vector fields can be used in place of optical flow to create videos based on still images [12]. Both methods are global in nature, and while the methods described can work on arbitrary fields, there is little discussion of how such fields may be manipulated by users.

Other work in NPR has made use of fluid simulations as well. Curtis et al. [6] and Baxter et al. [2] both use fluid simulations to model paint. Witting [22] explains DreamWorks's use of fluid simulations in their animated feature *The Prince of Egypt*. Unlike the previous work in stroke based rendering, Witting's artificial fields are derived from his source images; he uses the underlying colors to imply temperature forces which drive the simulation. Finally, image filtering techniques such as Cabral et al. have proven effective tools for field visualization [3]. The same techniques have also been turned to artistic purposes, warping a source image using arbitrary fields to create "Van Gogh-like" ripples and swirls.

3 Stroke Rendering

Our rendering system starts by segmenting the image into regions. For each region we define a set of characteristics governing stroke rendering and motion. Among the region's characteristics are descriptions of the fields to be used to align the strokes and advect them between frames of an animation.

3.1 Regions

We use a mean-shift colorspace clustering algorithm [5] to segment source images , and then allow users to refine that segmentation by merging any set of regions. Each region defines its own stroke alignment and advection fields, as well as stroke rendering parameters.

3.2 Rendering

Random permutations are applied to stroke intensity, color, width, length, and alignment. Different permutation parameters may be used depending on the region that the stroke is in (crossing a region boundary causes the stroke's permutation variables to be reinitialized). Strokes are clipped to region boundaries. A region may use curved brush strokes, in which case the alignment permutation is ignored, and a curved stroke is rendered by advecting points through the alignment field.

3.3 Stroke Motion

Between each frame of an animation, strokes are displaced by their region's advection field. When strokes are advected they tend to become too sparse in some areas and too dense in others. In order to combat this effect, a quality controlled Delaunay triangulation is used to determine locations where stroke centers should be inserted [19]. If the distance between any two stroke positions is below a given threshold, one is eliminated. Techniques similar to those introduced by Hays et al. [12] are used to improve temporal coherency. When new strokes are added, or old strokes reset as a result of crossing a region boundary, they are initially rendering using a small alpha value, and then fade in over time. Similarly, deleted strokes are not immediately removed, but instead fade out.

4 Fields

4.1 RBF Fields

First used in stroke-based rendering by Litwinowicz [17], radial basis functions (RBFs) continue to be used in more modern systems [12]. In two dimensions, we may define an interpolation $f(x, y)$ of the control points (x_i, y_i) having values v_i as:

$$\begin{aligned} f(x,y) &= a_1 + a_2 x + a_3 y + \\ &\sum_{i=1} w_i U(||(x_i, y_i) - (x, y)||). \end{aligned}$$

The linear system implied by setting all $f(x_i, y_i) = v_i$ is underspecified, so the following additional constraints are applied.

$$\sum_{i=1} w_i = \sum_{i=1} w_i x_i = \sum_{i=1} w_i y_i = 0.$$

Though the exact application of RBFs to stroke-based rendering varies from one stroke-based rendering system to another, the general goal is always the same: creating a smooth stroke alignment field based on the color gradient of the image. By aligning strokes perpendicular to such

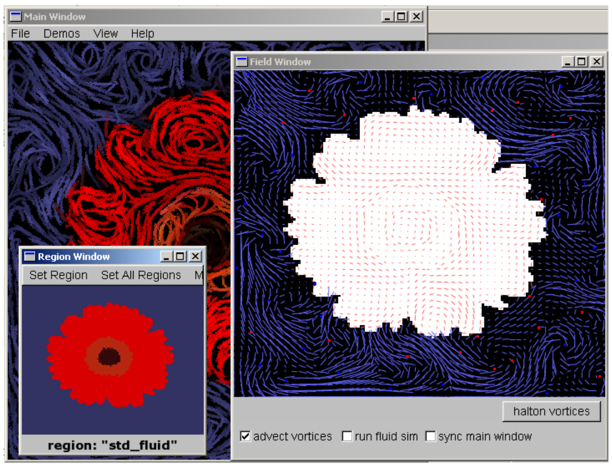


Figure 2: Our application, with both field and region editing windows open.

a field, it is possible to create renderings with long brush strokes which are aligned to region boundaries. We use a thin plate basis function, $U(r) = r^2 \log(r)$, which has desirable smoothness characteristics [7].

In our system, a set of pseudo-gradient values are calculated on the borders of each region by treating all pixels in the region as intensity 1, all pixels outside the region as intensity 0, and then applying Sobel convolution matrices. Interpolating those gradients over the region would require inverting a $(p + 3)$x$(p + 3)$ matrix, where p is the number of pseudo-gradient points. In order to speed up the solver, we divide the image into an MxM grid and average all values in each grid box to create a single vector. We vary the value of M used in each region in order to yield control point sets in a preferred size range. Vectors perpendicular to the averaged values are then used as control points for two RBFs; which together define a field over the region.

4.2 Fluid Fields

We simulate a fluid velocity field using a semi-Lagrangian solver as in Stam [20] and vorticity confinement as in Fedkiw et al. [8]. Any region may be treated as a barrier to fluid flow, using the Von Neumann no-penetration boundary conditions.

Vortex Dynamics

Vortex particle simulations can be used to create fast, stable, and visually appealing fluid effects [10]. A turbulent two dimensional fluid velocity field can be simulated using a set of vortex points with strengths Γ_i and positions (x_i, y_i). The velocity field at any point is determined by the sum:

$$U_{vort}(x,y) = \sum_i (\Gamma_i \frac{y-y_i}{r2}, \Gamma_i \frac{x-x_i}{r2}).$$

where $r = ||(x,y)-(x_i,y_i)||$ [4].

This formalism is attractive because changing particle positions and Γ values is a simple and intuitive means of controlling the velocity field. Unfortunately, for small numbers of vortex points, velocity fields created using this formalism lack many of characteristics that attracted us to fluid simulations. Therefore we have adopted a hybrid approach. We use a small number of vortex particles to apply a force at all grid points in the semi-Lagrangian simulation equal to $\epsilon \cdot u_{vort}(x,y)$. The result is a dynamic vector field that is both easily edited and reasonably "fluid looking". In order to balance out the velocity constantly being added into the system by the vortex particles, we use a fairly high viscosity value. Properly, vortex points should always be advected by the fluid velocity field, but we find that sometimes it is useful to keep the vortex points fixed. When advecting the particles, we use simple r^{-n} repulsive forces to prevent the particles from coming too close to each other or drifting through region boundaries.

While we experimented with different strategies for automatically generating fields by adding forces based on the image, none of them worked well for more than a handful of cases. However, with the hybrid vortex approach, much more robust automatic field generation becomes possible. For example, using a pseudorandom sequence to generate a collection of vortex points consistently leads to appealing turbulent fields.

4.3 Hybrid Fields

It is sometimes useful to control strokes with hybrid fields defined as a weighted sum of the fluid and RBF fields. Given velocity field v and RBF field t we define a hybrid field f as:

$$f = \begin{cases} \frac{d-w}{w}t + \frac{d}{w}v & \text{if } d < w \\ t & d \geq w, ||v|| < r \\ v & \text{otherwise} \end{cases}$$

w indicates the "width" of the hybridization, and r controls a replacement threshold. Thus we create a field that is both aligned with region boundaries, and which exhibits fluid-like behavior farther from the region edges.

4.4 Optical Flow Fields

While Litwinowicz used Bergen's hierarchical algorithm to calculate the optical flow fields, we found that in the case of our videos, Bergen's algorithm tended to produce troublesomely inaccurate fields. Instead, we found that Horn and Schunck's optical flow algorithm created much nicer, smoother fields for video sequences [15]. This observation is in keeping with a comparative analysis that suggests that, despite its age, Horn and Schunck is one of the most robust optical flow algorithms available [9].

5 Results

We developed our application on an Athlon 1.4 ghz machine with a GeForce2MX video card. We use multiple windows to allow the user to interact with fluid fields, vortex particles, and region definitions in real time. Users select from one of several configuration files to determine the stroke rendering parameters and field properties in each region. Those files themselves may be edited in order to fine tune stroke behaviors within a given region. The region configuration files, in turn, are designed to allow human editing.

The slowest component of the algorithm is stroke rendering. The rendering time for our demo images, which use around 6000 strokes, was typically 5 seconds per frame. The next most expensive operation is stroke density control, which takes about 2 seconds per frame. Total time to process a single frame is around 8 seconds.

Hybrid fields have proven to be incredibly useful. They have all of the advantages of both fluid and RBF fields, and none of the disadvantages. Even when neighboring regions are used as boundaries to fluid flow, strokes aligned according to fluid fields may not appear to match neatly to region boundaries. Pure RBFs, on the other had, can have poor temporal coherency. Hybrid fields have neither of these problems.

Using fixed vortex points leads to fields with a very different character than using advected points. The fixed case tends to create smooth steady-state flow, and the only noticeable vortices are those explicitly specified. The advected case creates an endlessly evolving turbulent field, with lots of small scale effects outside of those implied by the vortex particles. We have found that the first method is well suited to users who desire large amounts of control, while the second is an effective means of quickly creating interesting dynamic fields.

6 Conclusion and Discussion

We have presented a painterly rendering system in which a range of user designed vector fields may be used in the generation of brush stroke rendering. The user's control over the system takes the form of a few simple high level choices.

A fully automatic system could be implemented by assigning each region rendering characteristics based on region properties such as size or texture. However, it would likely be hard to find region specification rules that would

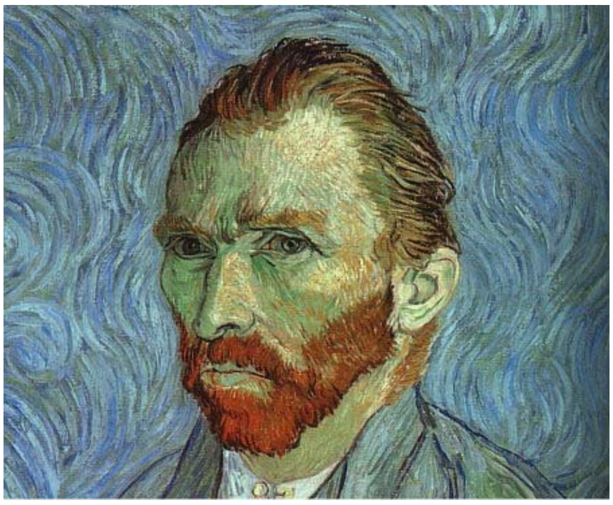


Figure 3: Detail of Self Portrait, *Vincent Van Gogh, 1889. Brush strokes in the background region are long, curved, and arranged in a fluid-like pattern, while the face itself is painted with much smaller, more detailed strokes.*

work well in a wide range of cases.

Our system cannot run in real-time given high stroke densities. However, we suspect that a GPU-based implementation could. All stroke information could be kept in texture memory, and stroke polygons generated using fragment programs. It is, however, not clear to the authors how a Delaunay triangulation algorithm could be efficiently implemented on a GPU. Thus, to prevent density control from becoming a bottleneck, a GPU-based system would require the development of new density control techniques.

There are a number of ways in which our techniques could be integrated with other recent work. Our segmentation algorithms only allow for relatively crude forms of user control, a modern user-assisted region segmentation system would be preferable, such as Rother et al. [18] or Li et al. [16]. Working with video input would be much easier if we had a robust user-guided algorithm that was able to track the motion of regions in the source video – Agarwala et al. [1] would probably work well. It could also be interesting to compare the results of our RBF solver to that of other fast approximate solvers, such as those discussed in Donato et al. [7], or the iterative energy minimization strategy introduced in Terzopoulos [21].

References

[1] Aseem Agarwala, Aaron Hertzmann, David H. Salesin, and Steven M. Seitz. Keyframe-based tracking for rotoscoping and animation. *ACM Trans. Graph.*, 23(3):584–591, 2004.

[2] William V. Baxter, Jeremy Wendt, and Ming C. Lin. IMPaSTo: A realistic model for paint. In *Proceedings of the 3rd International Symposium on Non-Photorealistic Animation and Rendering*, pages 45–56, June 2004.

[3] Brian Cabral and Leith Casey Leedom. Imaging vector fields using line integral convolution. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 263–270. ACM Press, 1993.

[4] Alexandre J. Chorin. *Vorticity and Turbulence*. Springer, New York, 1994.

[5] Dorin Comaniciu and Peter Meer. Robust analysis of feature spaces: Color image segmentation. In *1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, pages 750–. IEEE Computer Society, 1997.

[6] Cassidy J. Curtis, Sean E. Anderson, Joshua E. Seims, Kurt W. Fleischer, and David H. Salesin. Computer-generated watercolor. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 421–430. ACM Press/Addison-Wesley Publishing Co., 1997.

[7] Gianluca Donato and Serge Belongie. Approximate thin plate spline mappings. In *ECCV (3)*, volume 2352 of *Lecture Notes in Computer Science*, pages 21–31. Springer, 2002.

[8] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM Press, 2001.

[9] Ben Galvin, Brendan McCane, Kevin Novins, David Mason, and Steven Mills. Recovering motion fields: An evaluation of eight optical flow algorithms. In *Proceedings of the British Machine Vision Converence (BMVC)*, 1998.

[10] Manuel Noronha Gamito, Pedro Faria Lopes, and Mario Rui Gomes. Two-dimensional simulation of gaseous phenomena using vortex particles. In *Proceedings of the 6th Eurographics Workshop on Comput. Anim. and Sim.*, pages 2–15. Springer-Verlag, 1995.

[11] Paul Haeberli. Paint by numbers: abstract image representations. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 207–214. ACM Press, 1990.

[12] James Hays and Irfan Essa. Image and video based painterly animation. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 113–120. ACM Press, 2004.

[13] Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 453–460. ACM Press, 1998.

[14] Aaron Hertzmann and Ken Perlin. Painterly rendering for video and interaction. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pages 7–12. ACM Press, 2000.

[15] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1-3):185–203, August 1981.

[16] Yin Li, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum. Lazy snapping. *ACM Trans. Graph.*, 23(3):303–308, 2004.

[17] Peter Litwinowicz. Processing images and video for an impressionist effect. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 407–414. ACM Press/Addison-Wesley Publishing Co., 1997.

[18] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. ”grabcut”: interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, 2004.

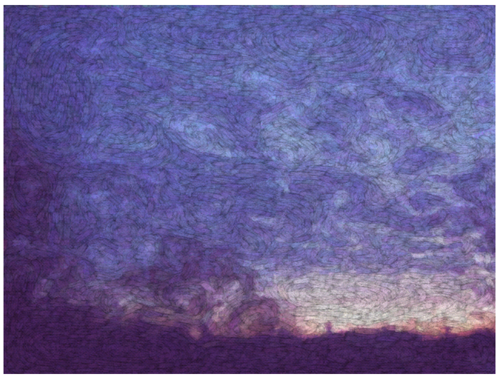
[19] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.

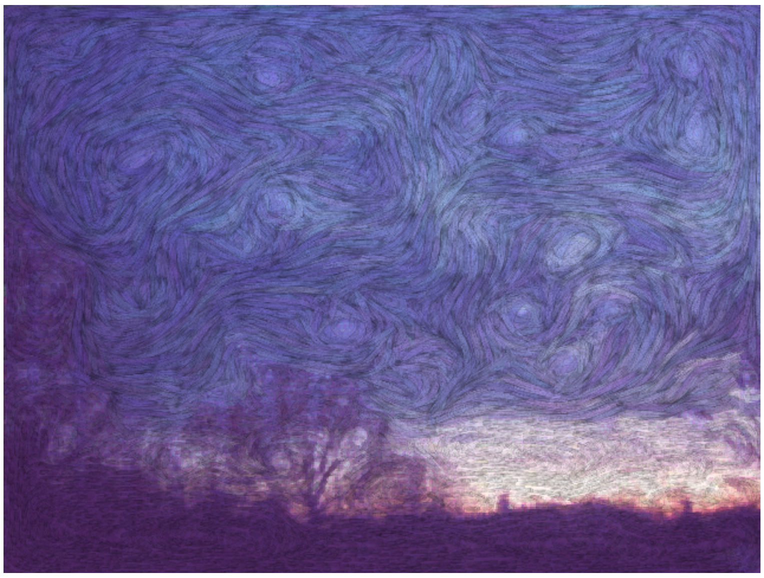
[20] Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128. ACM Press/Addison-Wesley Publishing Co., 1999.

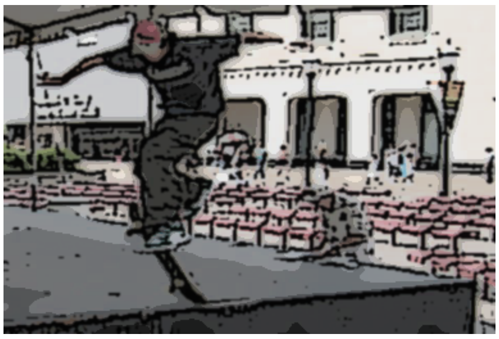
[21] Demetri Terzopoulos. The computation of visible-surface representations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(4):417–438, 1988.

[22] Patrick Witting. Computational fluid dynamics in a traditional animation environment. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 129–136. ACM Press/Addison-Wesley Publishing Co., 1999.

Source image.


“Traditional” rendering using a single brush type and no fluid fields.


Rendering using fluid fields and larger brush strokes in the background region.


Source Image, a cartoon rendered photograph.

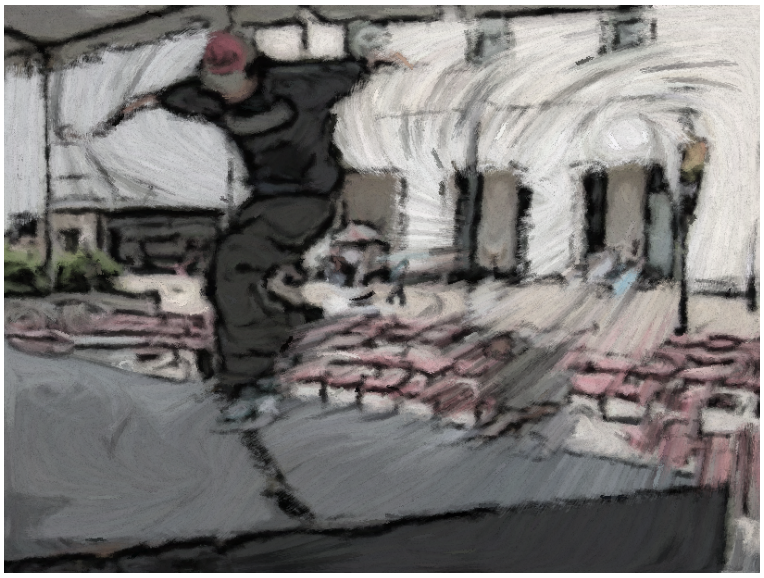

Rendering with opaque strokes and unblocked hybrid fields.

Figure 4: Results: The sunset demo was generated very quickly (in less than a minute) using a pseudorandom vortex point initialization. For the larger skateboarder image more time was spent hand editing the vortex particle strengths and positions.