

Painting with Flow

Corinna Vehlouw*

VISUS, University of Stuttgart, Germany

Fabian Beck†

VISUS, University of Stuttgart, Germany

Daniel Weiskopf‡

VISUS, University of Stuttgart, Germany

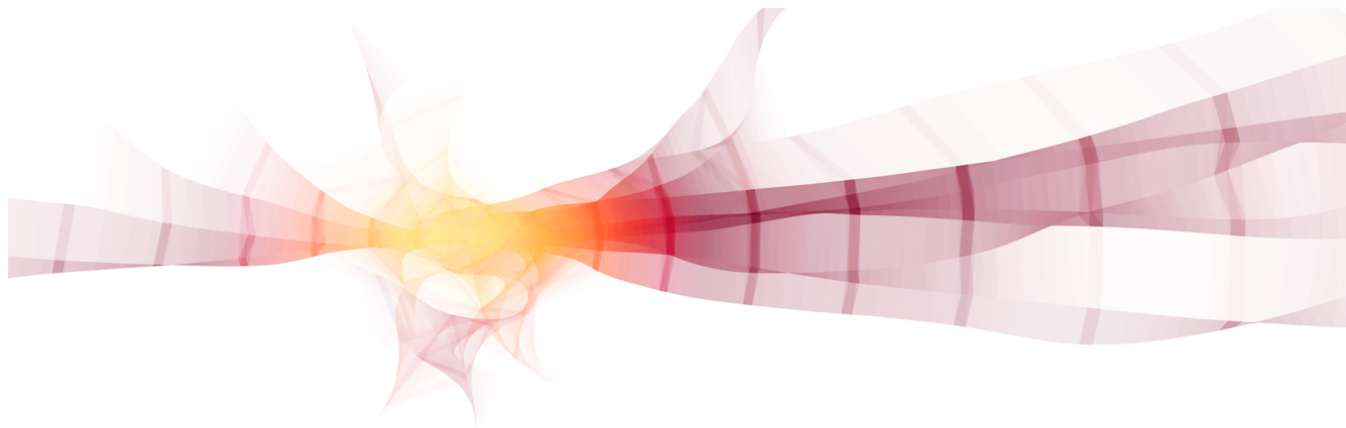


Figure 1: Painting generated by a single run of evolving ribbons.

ABSTRACT

We present an interactive algorithmic painting approach producing organic flow-like representations. Our approach is inspired by nature, in particular, the smooth patterns of flowing particles in liquid and air but also the smooth organic shapes occurring in flora. The algorithm for creating organic shapes follows mechanisms of flow visualization. It is based on the random spread of guiding points from a user-defined seeding position that grow to curves enclosing ribbons. This spreading process is initialized and can be—to some extent—influenced by the user. Rendering only requires the drawing of partly transparent polygons. Different color maps and rendering schemes are available to produce unique, non-deterministic paintings.

Index Terms: I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation; I.3.4 [Computer Graphics]: Graphics Utilities—Paint systems

1 INTRODUCTION

Organic shapes occurring in the flora have been an inspiration for art in many fields including architecture, paintings, drawings, and sculptures. In particular, curves and spirals are an inspiration for manual (hand-crafted) pieces of art as well as algorithmic art. Besides the organic shapes in nature, also the flow of air or liquids is continuous and, when visualized using flow visualization techniques, shows organic patterns and smooth curves. Organic shapes that contain only gradual changes of direction are perceived as aesthetically pleasing, according to the *biophilia hypothesis* [31].

Manually drawing or constructing organic shapes, however, is difficult. While experienced painters are to draw smooth contours

by hand, it still remains a challenge to fill these with smooth color gradients. Hence, computer support for creating these drawings is helpful and would enable less experienced artists to create organic drawings. One option is using existing flow simulations to produce the shapes. Although such simulations are certainly able to generate visually fascinating and aesthetic curves, they are difficult to control: simulation parameters only indirectly influence the vector fields and trajectories of particles. Moreover, simulations are usually computationally expensive so that long delays disturb the painting process.

To overcome these limitations, we present a drawing approach that automatically creates flow-like shapes with smooth gradients, but lets the artist control the flow more easily. The generation of shapes is based on a simple algorithm that is easy to implement and computationally inexpensive. Rendering only requires the drawing of partly transparent polygons. This allows real-time interaction and dynamic image creation similar to a usual digital painting software. The approach transfers the aesthetics of flow visualization to painting and therefore allows artists to easily create visualization-like images.

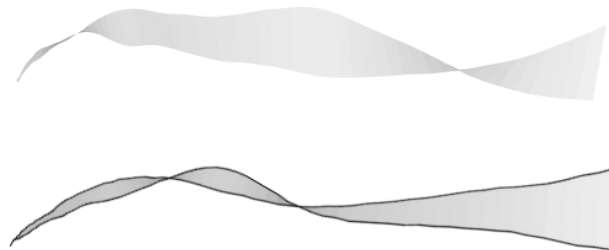


Figure 2: Ribbons bounded by two curves that intersect at the seeding point (on the very left) and two other locations. In the ribbon at the bottom, the bounding curves are rendered in black.

*e-mail: corinna.vehlouw@visus.uni-stuttgart.de

†e-mail: fabian.beck@visus.uni-stuttgart.de

‡e-mail: daniel.weiskopf@visus.uni-stuttgart.de

In particular, we provide a painting tool that generates smoothly curved, overlapping ribbons that produce appealing visual patterns. A ribbon can be understood as the area circumscribed by two curves that are built up by two guiding points moving in two-dimensional space (see Figure 2 for a single ribbon). To create smooth curves and hence ribbons, we use an acceleration model to move and spread the guiding points from their initial position and initial velocity with respect to their acceleration and hence increasing velocity. The painting process is partially controlled by the users: The seeding point for the guiding points and hence ribbons is set interactively using a mouse click. Moreover, they can drag the guiding points behind the mouse position. Finally, rendering settings, including different color maps or the explicit rendering of the curves bounding the ribbons, can be set to produce different painting styles.

2 RELATED WORK

The paintings generated with our algorithm give the impression of flowing liquids or gases. There are various flow visualization techniques that visualize the flow described by a vector field. These include different techniques based on particle tracing such as streamlines, pathlines, streak lines, and timelines [15, 14, 19], which show different properties of the two-dimensional vector field. For three-dimensional vector fields, stream lines can be extended to stream ribbons or stream surfaces. While the lines extracted from a static two-dimensional vector field do not overlap or intersect each other, in visualizations of dynamic flow [10, 11] overlap and intersections may occur. While the previously mentioned approaches were mainly developed for data analysis, the real time, interactive fluid simulation and vector visualization technique by Forbes et al. [9] was developed for incorporation in media arts projects.

Besides the organic shapes occurring in flow visualization, many more visual representations or pieces of art in general—inspired by nature—include curves. This is not only true for hand drawings or paintings but also for artworks created computationally as well as for image filters. In algorithmic or generative art [4, 8, 17, 22, 24], pieces of art, design, or architecture are generated computationally, e.g., by exploring fractals [2, 8]. Often these contain smooth shapes or curves, such as the Cortices by Henze [12], the flow art by Patel [1], or the Perlin Flow Ribbons by Mattox [18]. The screen-saver map by Esch and Rogers [7], e.g., shows elliptical curves that arise from polygonal folding. The particle tracing method “magnetic curves” by Xu and Mould [32] creates curves with constantly changing curvature and uses these to generate tree-like drawings. In contrast to their magnetic model, which is based on a constant acceleration perpendicular to the direction of motion, our model varies acceleration, which leads to random, non-deterministic irregular curves. Also in the area of information visualization, curvy representations including ribbons are used quite often, e.g., to visualize the flow of groups or attributes over time [20, 23, 25]. Further, spirals are applied to visualize serial periodic data [5, 28].

There are various approaches that transform an image into a painting- or drawing-like representation using image filters, e.g., using curves [16, 29], or into a stained-glass version of that image [21]. The line-drawing stylization approach by Wei and Mould [29] is based on particle tracing that generates almost parallel traces, i.e., the traces do not intersect. In contrast, in our painting approach interesting visual patterns occur because the paths of guiding points do intersect. Other techniques use a tessellation approach [16, 21]. In our approach, tiles (areas framed by paths) emerge by accident through intersecting paths. All these image filtering approaches are automatic and do not require interaction by the user besides for parameter adjustments. In contrast, we want to provide a painting tool that allows the user to generate paintings interactively.

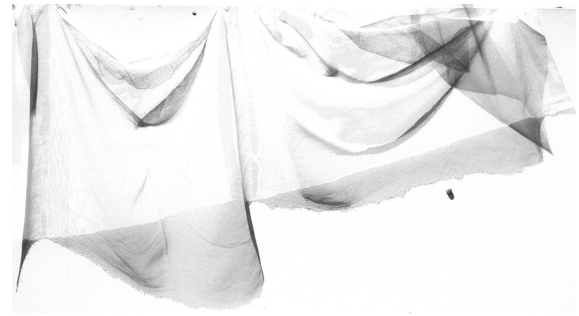


Figure 3: Photography of a hanging, overlapping fly screen fabric in front of a white wall under direct sun light.

There are several sketching and brushing approaches that are based on user interaction, e.g., via mouse or digital pen, where sketch-based interfaces are mainly used to generate the intended two-dimensional [27] or three-dimensional [13] geometric objects. Our painting approach is, to some extent, similar to brushing approaches for digital painting, such as the three-dimensional brush for pastels or oil paint [6] or wax crayons [26], where compared to those brushes our approach generates coarser patterns that are not suitable to draw detailed contours. In fact, our approach contains more random features and cannot be controlled in the same way as brushes. However, in particular the evolving random organic features make the paintings generated with our approach aesthetic.

3 GENERATING RIBBONS

The aim of our painting tool is to generate flow-like organic shapes. We draw random smoothly curved ribbons, which can be understood as the area circumscribed by two curves. These ribbons are not independent, but share the bounding curves with other ribbons, which leads to overlap of those ribbons giving the impression of irregularly folded, partially transparent fabric, like in the photography shown in Figure 3. The generation of these curves is inspired by flow, in particular, the paths of moving particles. In flow visualization the movement and hence the paths depend on the underlying vector field and the seeding. In our approach, we also let the particles move starting from their seeding position but—in contrast to particle movement defined by vector fields—the movement itself is independent from their two-dimensional position. Instead of allowing the particles to jump to random positions on the screen—which would create jagged paths—we use an acceleration model for a smooth spread of the particles, in the former referred to as *guiding points*.

The ribbons evolve by connecting every two guiding points with a line that moves together with the points, thereby dragging a tail behind them. Figure 4 (a) illustrates the movement of guiding points and the respective connecting lines by coloring them based on the time evolved (the paths of the guiding points are shown in black). By analogy to stretching fabric—where more light passes through the fabric when stretched—the longer the distance between two guiding points, the more transparent the line stretched between them. This is illustrated in Figure 4 (b), where the connecting lines are rendered in black with a transparency increasing with the length of the line. This approach has already been used before, e.g., to visualize unsteady flow [30].

In this section, we first describe the movement of guiding points based on our acceleration model and how this movement can be controlled interactively. We further discuss the rendering algorithm that generates the flow-like ribbons from the moving guiding points. To demonstrate the visual variety of our approach, we finally explore different parameters of both algorithms.

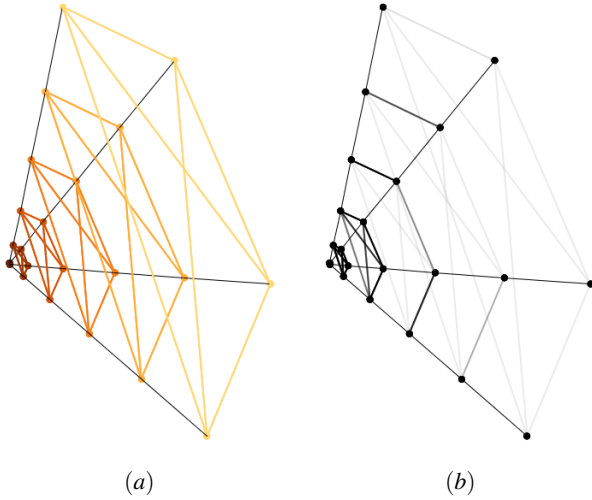


Figure 4: Schematic illustration of the movement of guiding points and the evolution of pairwise connecting lines with (a) color-coding by time and (b) increasing transparency with lengths of the lines.

3.1 Moving Guiding Points

The key aspect of our method is the combination of interactive user-based initialization of the guiding points and their random acceleration based expansion across the screen. Algorithm 1 shows the pseudo code that moves the guiding points randomly based on an acceleration model. The algorithm takes the initial seeding position $\vec{p}_{init} = (x_{init}, y_{init})$ for the points and the movement parameters δ (velocity increment), κ (volatility), and n (number of guiding points) as input. We use pixel as units for lengths; time is given by the discrete frame numbers of the animation.

Algorithm 1 Moving guiding points

```

moveGuidingPoints( $\vec{p}_{init}$ ,  $\delta$ ,  $\kappa$ ,  $n$ ):
   $\vec{p}_{init}$ ; // initial seeding position ( $x, y$ )
   $\delta$ ; // velocity increment used to accelerate the movement
   $\kappa$ ; // volatility of the change of acceleration
   $n$ ; // number of guiding points
   $\mathcal{P}$ ; // set of guiding points
   $t$ ; // current time
   $v_{scale} := v_{scale0}$ ; // current velocity scale initialized with  $v_{scale0}$ 
   $\mathcal{P} := \text{initGuidingPoints}(\vec{p}_{init}, n)$ ;
  // Iterate over time:
  while true do
     $t := t + 1$ ;
    // Iterate over guiding points:
    for all  $P_i \in \mathcal{P}$  do
       $\vec{a}_i := \text{changeAcceleration}(P_i, \kappa)$ ;
       $\vec{v}_i' := \text{changeVelocity}(P_i, v_{scale})$ ;
       $\vec{p}_i := \text{move}(P_i)$ ;
    end for
     $\text{render}(\mathcal{P}, t)$ ;
     $v_{scale} := v_{scale} + \delta$ 
  end while

```

First, a set of n guiding points $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ is initialized. Each guiding point is a triple of vectors $P_i = (\vec{p}_i, \vec{v}_i, \vec{a}_i)$, where the position is initially set to $\vec{p}_i := \vec{p}_{init}$, the velocity to $\vec{v}_i := (0, 0)$, and

acceleration to $\vec{a}_i := (0, 0)$. The movement of guiding points \mathcal{P} is then simulated in discrete time steps with a time increment $h = 1$ as part of an endless loop. At each point in time t , the acceleration and velocity of every guiding point P_i is changed and the point is moved based on the current velocity. First, the acceleration \vec{a}_i is adjusted randomly using the subroutine $\text{changeAcceleration}(P_i)$:

$$\vec{a}_i := \vec{a}_i + \kappa \begin{pmatrix} r_x \\ r_y \end{pmatrix},$$

where r_x and r_y are random values between -1 and 1. The volatility parameter κ determines the degree of random influence on the acceleration. Next, the velocity is updated using the subroutine $\text{changeVelocity}(P_i, v_{scale})$:

$$\vec{v}_i' := \vec{v}_i + h\vec{a}_i \text{ and then } \vec{v}_i := \frac{v_{scale}}{|\vec{v}_i'|} \vec{v}_i'.$$

Here, the temporary velocity \vec{v}_i' is first computed based on kinematics using Euler integration, then the new velocity \vec{v}_i is computed by multiplying the temporary velocity by the current velocity scale v_{scale} and normalizing the result with $|\vec{v}_i'|$. Finally, the current position \vec{p}_i of the guiding point P_i is updated within the subroutine $\text{move}(P_i)$ based on the Euler method:

$$\vec{p}_i = \vec{p}_i + h\vec{v}_i$$

After the transformation of each guiding point P_i , the current velocity scale v_{scale} is increased by the velocity increment δ . The increase of the current velocity scale v_{scale} together with the normalization of the velocity by $|\vec{v}_i'|$ was added to the algorithm in order to continuously increase the velocity of the movement without letting the velocity become too large. The effective movement of the points is slower, and hence the ribbons are more curved, near the initialization point. In contrast, at later time steps the effective movement is much quicker and hence the ribbons get less curvy, allowing them to fade out. This change in curvature particularly creates the floral impressions of the ribbons.

Although the algorithm simulates the movement endlessly, in practice, the rendering can be soon aborted: the guiding points may leave the screen space or the distances between the points may grow so that the connections between guiding points become too transparent to be visible (see Figure 4 (b)). A threshold for the aggregated distances of the guiding points, hence, turned out to be a reasonable stop criterion.

3.2 Interactively Controlling the Movement

Our approach is intended to be a drawing tool, and hence, needs to be interactively controlled with direct visual feedback. As the ribbons are rendered at simulation time (the subroutine $\text{render}(\mathcal{P}, t)$ of Algorithm 1 will be described in Section 3.3), the users can see the spread of the ribbons and the effects of their interactions directly—a video demonstrating the interactive process is available online¹. Users may change the parameters of the drawing algorithm at runtime through keyboard shortcuts, increasing or decreasing the values by a certain factor. As described in the previous section, the movement of guiding points stops automatically if their accumulated distance becomes too large. However, the movement of guiding points can also be stopped manually by the user at any time.

The basic mouse interaction is the selection of a seeding point on the canvas (by clicking on the screen). Once the seeding position is set, the guiding points are initialized and start to move based on the method described in Section 3.1. The movement of the guiding points can be manipulated by pressing a control key (*controlGuidingPoints* option), which reinitializes the last updated

¹Video available under: <http://youtu.be/HUHU4ePpP-I>

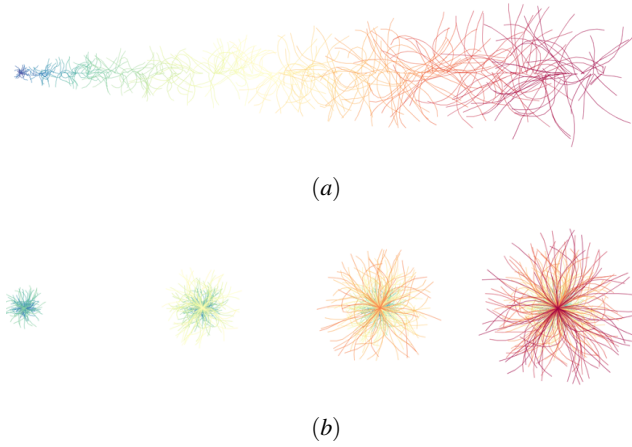



Figure 5: Schematic illustration of the paths of guiding points using the *controlGuidingPoints* option. In (a) the points are dragged behind the mouse. In contrast, in (b) the mouse was kept at the same position; here the result of every fiftieth time step t is shown.

guiding point and sets its position to the current mouse position. Hence, when moving the mouse while the guiding points spread across the screen, the user can drag the paths bounding the ribbons behind the mouse. Due to the positional reinitialization of the guiding points, the paths get interrupted, which leads to an interesting pattern that looks like a growing brush because only the position but not the acceleration or velocity of the last updated guiding point is reinitialized. Figure 5 (a) shows the paths of twenty guiding points that are initialized on the very left and then reinitialized one at a time while the mouse is slowly moved to the right. Figure 5 (b) shows the paths of twenty guiding points at time steps $t = 50$, $t = 100$, $t = 150$, and finally $t = 200$, where the mouse was kept at the same position while the *controlGuidingPoints* option was enabled. In contrast to Figure 4, in Figure 5, we show the paths of the guiding points only (black lines in Figure 4), where these are colored based on the time (starting with blue at the seeding position and ending with red at the youngest point in time .

3.3 Rendering Algorithm

As mentioned before, the rendering of the ribbons is performed at simulation time, concurrent with the movement of points. We use a frame rate of 30 fps for our animation. The goal for rendering is to connect the guiding points such that color gradients appear between the points which get darker or more opaque with decreasing distance of the points. To this end, all pairs of guiding points at the current time step are connected as already illustrated in Figure 4. To create the impression of smoothly moving lines that leave a trail, the points are, however, not connected by thin lines, but polygons that fill the space between the current pair of guiding points and the same pair from the previous time step. The transparency of the polygons increases with the distance of the pair of guiding points—varying transparency and overlap of polygons creates smooth gradients with sharp contours. Different rendering setting can be used to modify the appearance.

The specific rendering approach as implemented is described as pseudo code in Algorithm 2. For all pairwise combinations of guiding points (P_i, P_j) , where $i \neq j$, the polygon defined by the previous and current position of the two guiding points is rendered using color c and transparency α . In particular, the current average absolute velocity of both guiding points is mapped to the color using the subroutine `getColor($v, t, i, colorMap$)` to stress the outer direction of movement. The current distance between the two guiding

Algorithm 2 Rendering

render(\mathcal{P}, t):

```

 $\mathcal{P}$ ; // set of guiding points
 $t$ ; // time step

 $v$ ; // absolute velocity a guiding point
 $d$ ; // distance between the current positions of two guiding points
 $c$ ; // color of the polygon
 $\alpha$ ; // transparency of the polygon

drawSteps; // rendering option to draw steps
drawPathlines; // rendering option to draw the path of each  $P_i$ 
colorMap; // rendering option defining the color map

for all  $P_i \in \mathcal{P}$  do
  for all  $P_j \in \mathcal{P}$  do
    if  $i = j$  then
      continue;
    end if
     $v := (\text{getAbsVelocity}(P_i, t) + \text{getAbsVelocity}(P_j, t)) / 2$ ;
     $d := \text{getDistance}(P_i, P_j, t)$ ;
     $c := \text{getColor}(v, t, i, \text{colorMap})$ ;
     $\alpha := \text{getAlpha}(d, t, \text{drawSteps})$ ;
    fillPolygon( $P_i, P_j, c, \alpha, t$ );
  end for
  if drawPathlines then
    // rendering option to draw the paths of guiding points
     $v := \text{getAbsVelocity}(P_i, t)$ ;
     $c := \text{getColor}(v, t, i, \text{colorMap})$ ;
    drawLine( $P_i, c, t$ );
  end if
end for

```

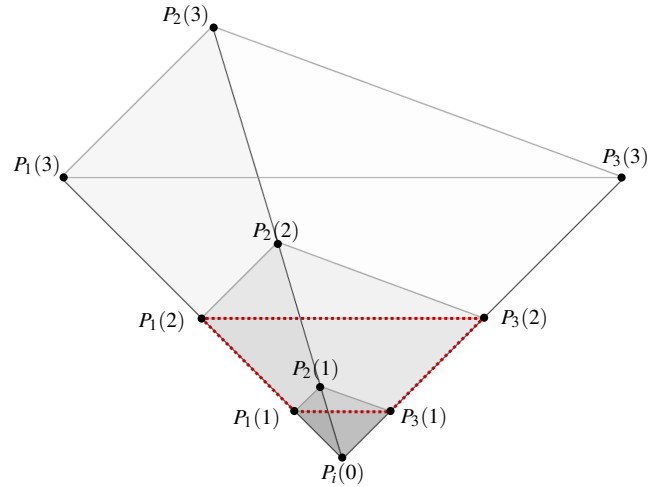


Figure 6: Schematic illustration of the rendering procedure. The rendered polygon for P_1 and P_3 at time step $t = 2$ is highlighted in red.

points P_i and P_j is mapped to the transparency using the subroutine `getAlpha($d, t, \text{drawSteps}$)`. Figure 6 illustrates the drawing procedure for three guiding points P_i over three time steps, where the position of a guiding point P_i at time t is marked by circles labeled with $P_i(t)$. To give a specific example, the polygon for P_1 and P_3 at time $t = 2$ consisting of points $P_1(1)$, $P_1(2)$, $P_3(2)$, and $P_3(1)$ is highlighted in red.

Depending on the user-defined setting *drawPathlines*, the path of each guiding point P_i (see black straight lines in Figure 6) is

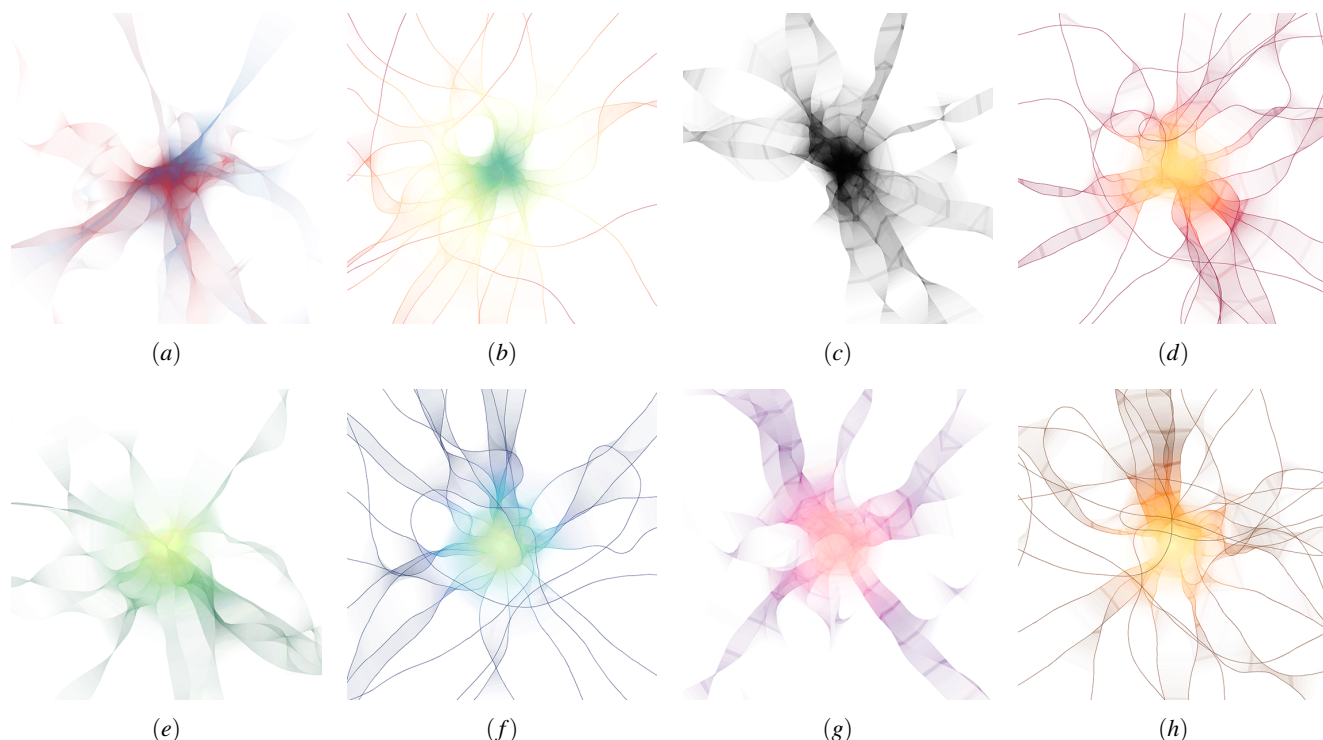


Figure 7: Overview of rendering settings ($\kappa = 4.0$, $\delta = 0.2$, and $n = 20$): different color map applied for each of the figures (a)–(h); (b) and (f) rendered using the *drawPathlines* option; (c) and (g) rendered using the *drawSteps* option; (d) and (h) rendered using both options (*drawPathlines* and *drawSteps*).

rendered. This is done by drawing a line from the previous to the current position of the guiding point. Similar to the polygons, the color depends on the current absolute velocity.

Besides the pathlines of the guiding points, another rendering setting that has an interesting visual effect on the painting results is the *drawSteps* option. If this option is enabled, the polygons for every tenth time step are rendered less transparent—we chose ten as parameter because it creates steps that are separable even for small velocities and velocity increments. This rendering option will be evaluated in the subroutine *getAlpha*($d, t, drawSteps$). The *drawSteps* option generates interesting step artefacts that remind of spider webs (see Figure 7 (c)).

Finally, different color mapping options are an essential feature of our painting approach. The velocity v , time t , or index i are mapped to color using the subroutine *getColor*($v, t, i, colorMap$). Many color maps could be applied here; we integrated a set of sequential color maps for v and a diverging color map for t —created with ColorBrewer [3]—that create aesthetically pleasing paintings into our painting tool. These include color maps that map the velocity v to shades of gray, red, green, blue, purple, brown, or the time t to a spectrum of different hues. Our painting tool also includes a color mapping option that is independent from the velocity: a guiding-point-based color mapping. This mapping applies two hues by alternating between them for the successive guiding points (see Figure 15), where the color c of the polygon for P_i and P_j depends on the index i of the first guiding point. The user can choose among those color mappings interactively before or during the painting procedure using keyboard shortcuts.

The settings of the rendering method are illustrated in Figure 7. It shows a summary of the available sequential and diverging color mappings of the velocity (Figure 7(b-h)) as well as the color map alternating between two hues (blue and red) based on the guiding

points indices (Figure 7(a)). Figure 7 also illustrates the rendering options *drawPathlines* (second column) and *drawSteps* (third column) individually as well as in combination (last column).

3.4 Parameter Space

In this section, we want to demonstrate how the different parameters of our acceleration model as well as the different rendering settings affect the appearance of the ribbons.

The first parameter that comes into play is the number of guiding points. This parameter affects the number of resulting ribbons (see Figure 8). The simple effect is that, with an increasing number of guiding points, the ribbon structure becomes denser. Figure 8 was generated with the gray scale sequential color map and using the *drawPathlines* option to highlight the paths of guiding points.

The first parameter that influences the acceleration model is the volatility κ . As demonstrated in Figure 9, the volatility κ controls the possible variation of the random change of acceleration. It therefore indirectly affects how much the guiding points change their direction—from smooth to curly lines.

The velocity increment δ first of all controls the velocity increase, hence the movement of a guiding point from one time step to the other and the size of the polygons that are rendered. Figure 10 illustrates this by alternating between light gray and black after every second time step t without mapping the velocity, where the distance is mapped to the transparency as described before. The resulting ring-like pattern that becomes stronger with increasing δ shows that the velocity changes more uniformly with increasing δ . Figure 10 shows that δ indirectly also affects the curvature of the ribbons: with increasing δ , the ribbons become less curved but more regular. Hence, κ should be increased together with δ to keep the degree of curvature about constant.

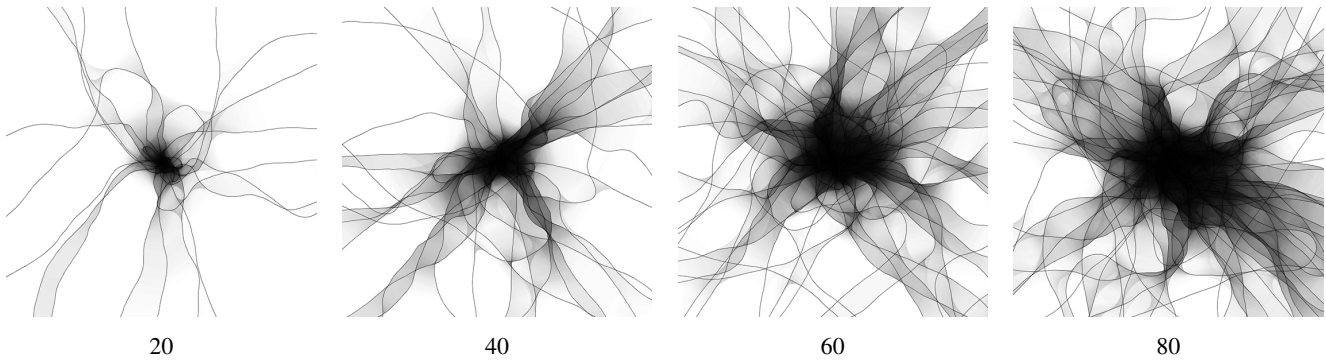


Figure 8: Effect of changing the number of guiding points n , where $\delta = 0.1$ and $\kappa = 2.0$.

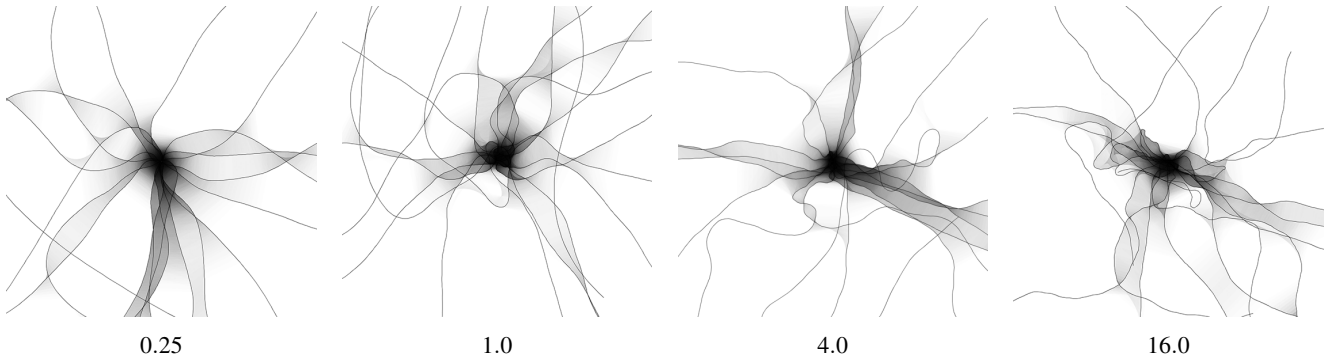


Figure 9: Effect of changing the volatility κ , where $\delta = 0.1$ and $n = 20$.

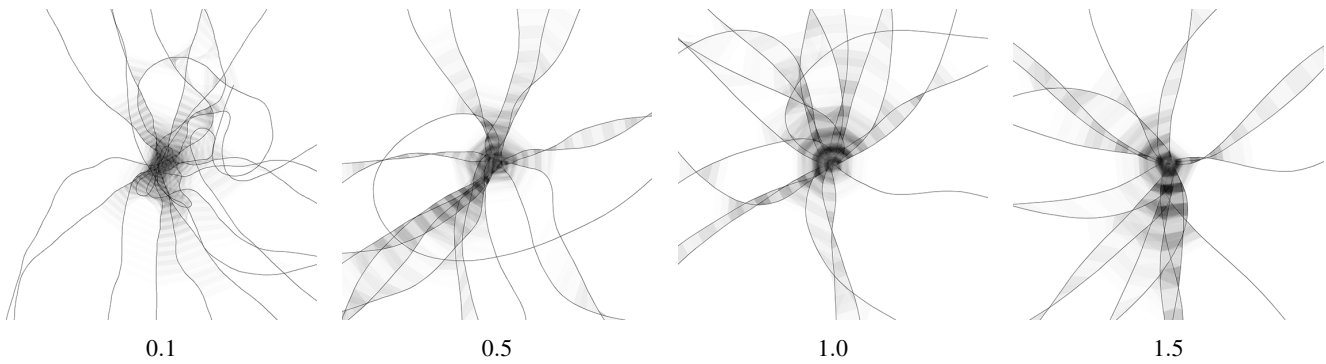


Figure 10: Effect of changing the velocity increment δ , where $\kappa = 2.0$ and $n = 20$.

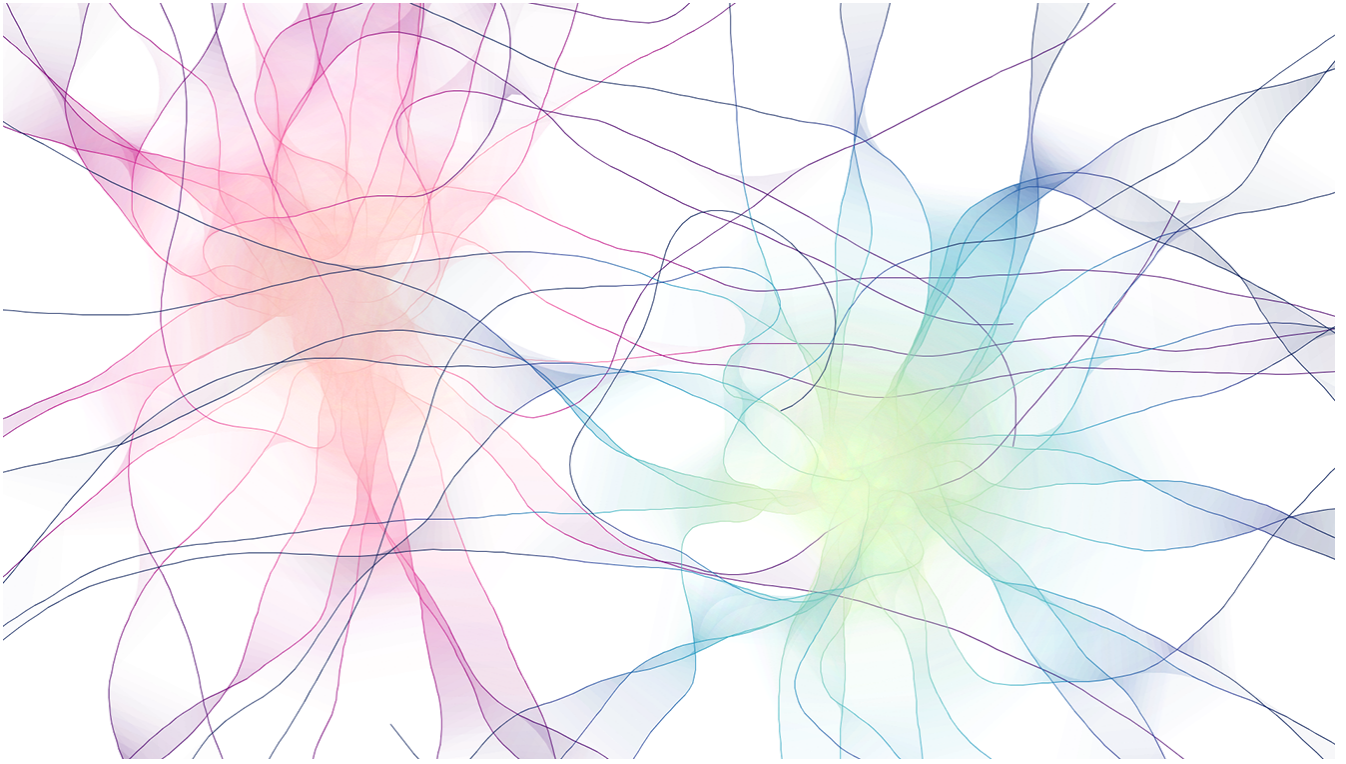


Figure 11: “Flowers” generated using a higher number of guiding points ($n = 40$); $\kappa = 4.0$ and $\delta = 0.1$.

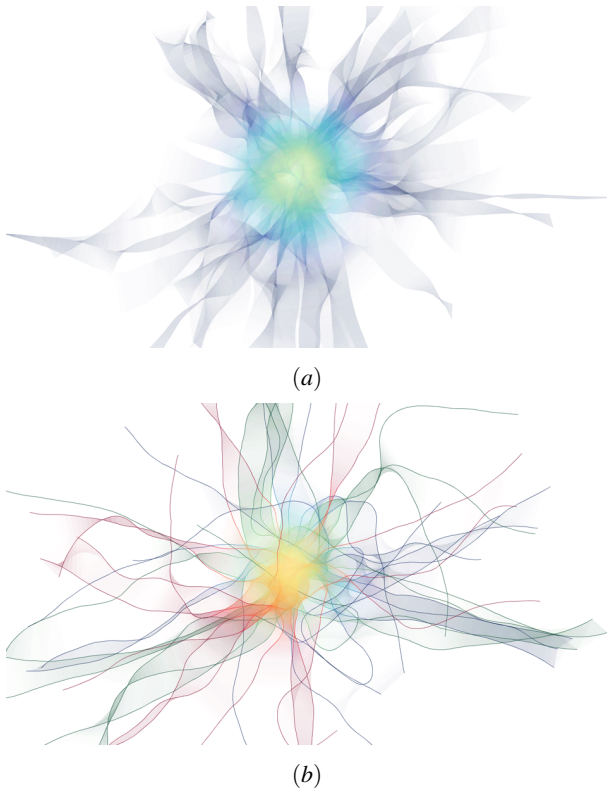


Figure 12: Reuse of the same initial seeding position over (a) six and (b) three runs; $\kappa = 2.0$, $\delta = 0.2$, and $n = 20$.

4 EXAMPLES

In this section, we show how our painting approach can be used to create different paintings and how the parameters and interactions can be altered to achieve different painting styles.

We used our painting approach to generate floral representations applying different drawing mechanisms. The representations in Figure 7 show organic smoothly curved ribbons starting in one point (the seeding position) and running into different directions, thereby partially overlapping each other. These ribbons look, to some extent, like the petals of a flower. However, although these representations show flower-like characteristics, the “flowers” are still very sparse with respect to their “petals”. To make the representations more flower-like, we can either increase the number n of guiding points (see Figure 11) or use the same seeding point for several runs (see Figure 12). To avoid extreme—with respect to the shape of “flowers” unnatural—curvatures of the ribbons, we set the volatility to $\kappa = 0.2$. For Figure 11, we applied a higher number of guiding points ($n = 40$) to generate two flowers, where the *drawPathlines* option was enabled to produce the impression of an abstract stained glass painting. The “flowers” in Figure 12 were generated by clicking the same point on the display over and over again, where a new set of guiding points was only initialized once the movement of the previous set stopped. In Figure 12 (a), the same color map is used for six runs, whereas in Figure 12 (b), three different color maps are used and the paths of guiding points are drawn to highlight the edges of the “petals” of our abstract flowers.

Also the *controlGuidingPoints* option can be used to generate “flowers”. By enabling this option and keeping the mouse at its position, the guiding points are reinitialized at that same position over and over again, which leads to star- or hedgehog-like patterns (see Figure 13). As the color map used for the velocity maps high velocities to red, the “flowers” become completely red af-

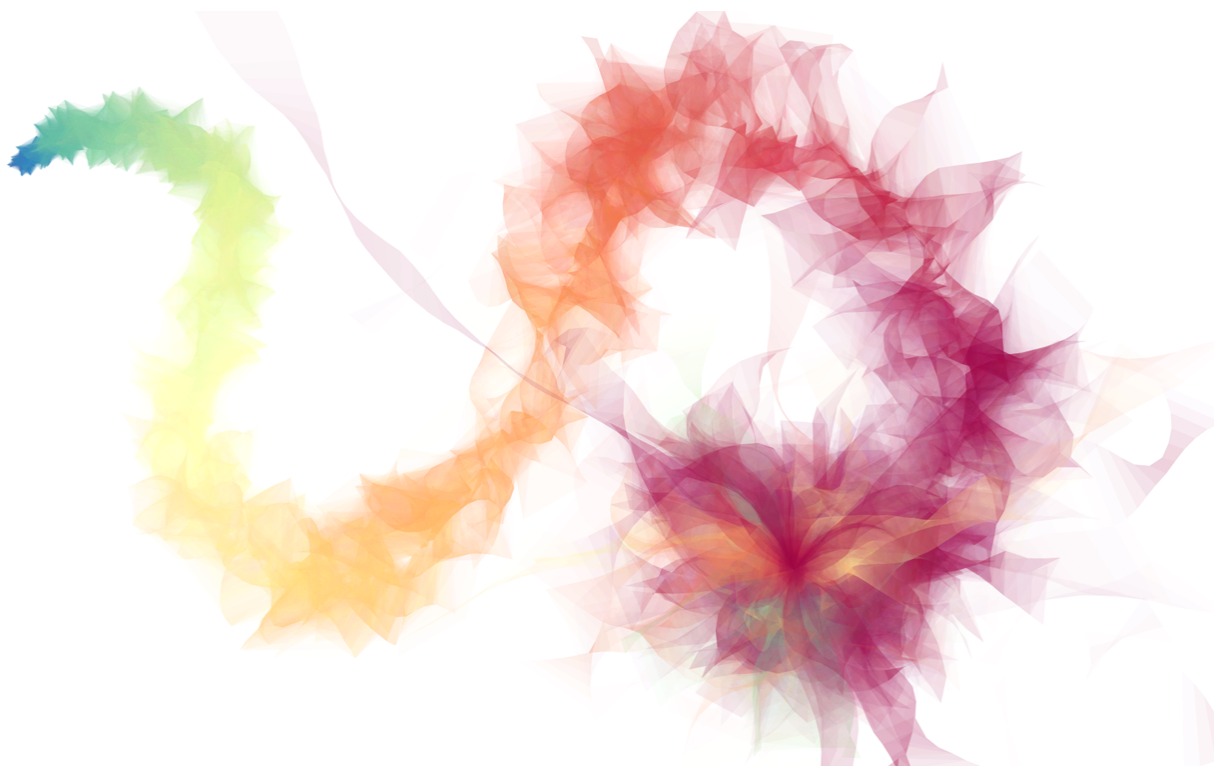


Figure 14: “Floral loop” generated using the *controlGuidingPoints* option and the diverging color map for the velocity; $\kappa = 16.0$, $\delta = 0.1$, and $n = 20$.

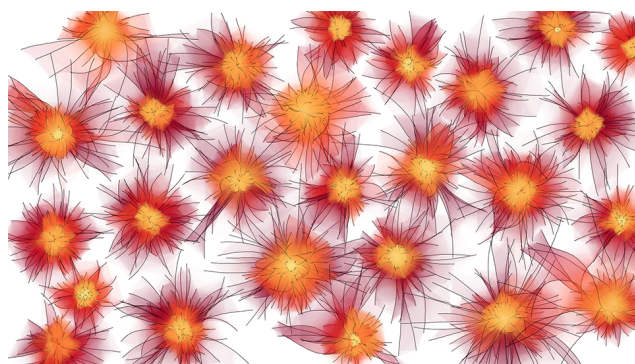


Figure 13: “Flowers” generated using mouse interaction and the *controlGuidingPoints* options; $\kappa = 0.5$, $\delta = 0.2$, and $n = 20$.

ter a while. The yellow inner part of the “flowers” can be drawn by starting the *moveGuidingPoints* subroutine again and using the *controlGuidingPoints* option, but stopping it early (before the velocity gets too high).

To generate abstract floral representations that go beyond the blossom, we move the mouse while enabling the *controlGuidingPoints* option. Figure 14 shows an example of a floral loop colored based on the time. First, the loop itself was drawn by moving the mouse in a curve while keeping the *controlGuidingPoints* option enabled. Then, a new set of seeding points was initialized at the end of that loop, where the *controlGuidingPoints* option was activated only every few time

steps for the same seeding position. This way, the ribbons could move freely but not too much outside of the desired radius.

Besides floral representations, we also used our approach to create paintings that give the impression of folded fabric. The ribbons in Figure 1 were created using a stronger acceleration in x-direction for the majority of ribbons. To generate relatively straight ribbons in the horizontal direction that give the impression of a bow, we use a low volatility of $\kappa = 0.5$. Finally, the *drawSteps* option was used to generate the striped texture pattern.

Also the representation in Figure 15 gives the impression of folded fabric. Here, we use the coloring of ribbons by guiding point index to generate ribbons of two different colors—red and blue—without a change of color along the ribbon. This creates ribbons that—to some extent—look like silk scarfs that are twirled together at a point and moving away from that center (here we see four such scarf nodes). Again, to generate a texture-like pattern on the fabric, we used the *drawSteps* rendering option.

5 APPLICATION TO FLOW DATA

There are various flow visualization approaches that show the movement of particles in 2D or 3D space. These depend on the vector field, i.e., the movement of a particle depends on the vector at its current position. In contrast, we let particles move into random directions from a user-defined initial position, where the particle movement is independent from the current particle position. Although the random movement of particles is an essential aspect of our painting approach, the rendering algorithm can also be applied to the movement of particles described by pathlines. Figures 16 (a) and 16 (d) shows the result of applying our rendering approach to pathlines of the Kármán flow data set. The pathlines were derived for different time steps and intervals as well as slightly different seedings. As these pathlines depend on the data, the painting system becomes more rigid and loses its stochastic nature.

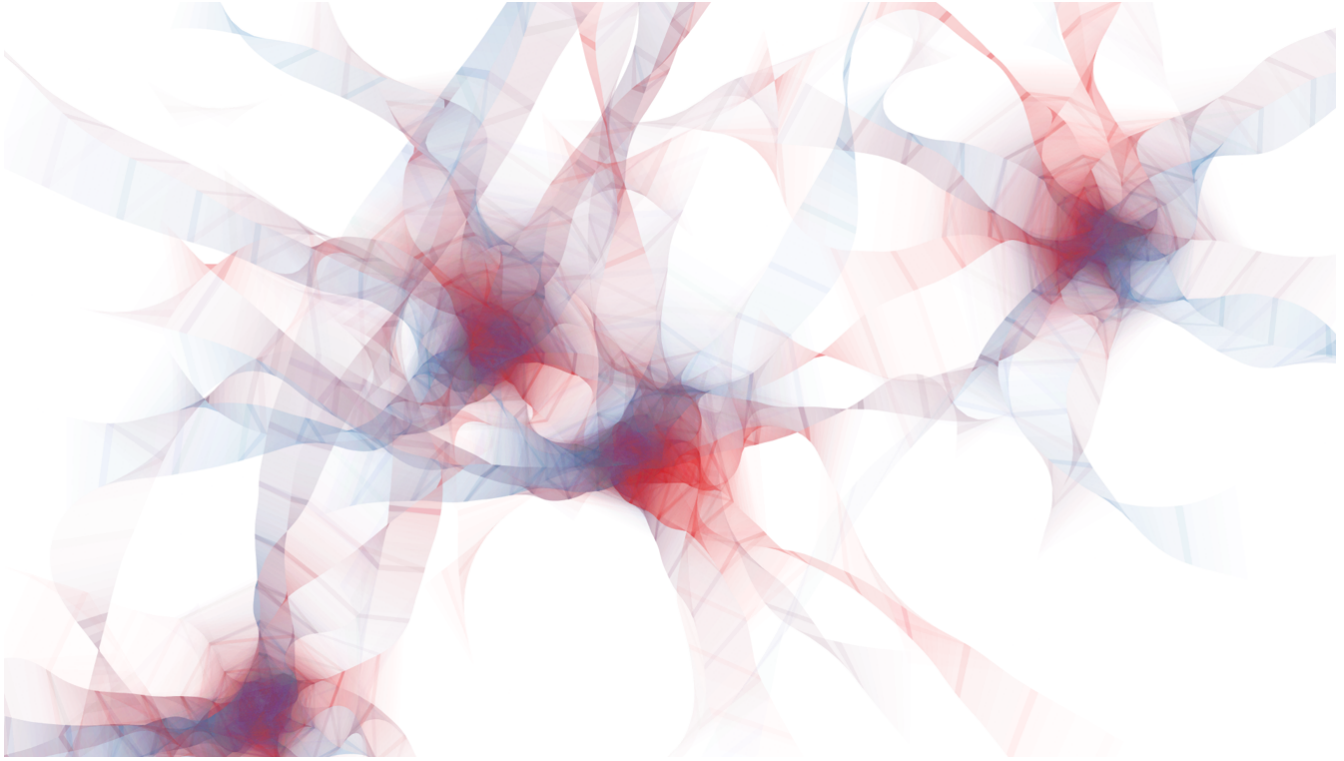


Figure 15: “Folded silk scarfs” generating using the guiding-point-based coloring and the draw steps option; $\kappa = 1.0$, $\delta = 0.1$, and $n = 25$.

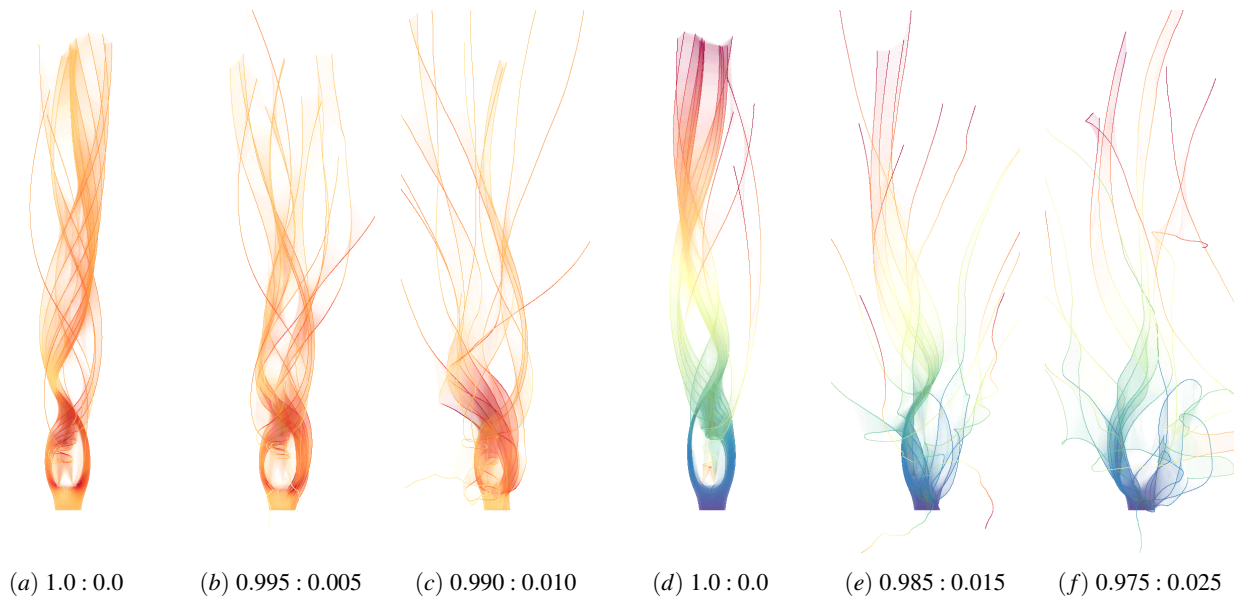


Figure 16: Application to flow data: pathlines from the Kármán data set derived using two different seedings and visualized using our rendering technique. We used two different color mappings, one for each seeding: (a)–(c) a sequential color map for the velocity or (d)–(f) a diverging color map for time. While (a) and (d) are only based on the data, the other figures were generated with a certain random influence described by the ratio *data : random*.

At least, the deterministic behavior can be dissolved to some extent by influencing the data-based particle movement by a random movement of points starting at the same seeding positions. We initialize a set \mathcal{P} of guiding points, one P_i for each seeding point and hence pathline, and let those move as described in Section 3.1. Finally, the effective movement of each particle is defined by the weighted sum of data- and random-based positions at similar time. The relative influence of both is described by the ratio *data : random*. Figures 16 (b), (c), (e), and (f) show the results of randomizing the pathlines of (a) and (d) with different random influences. The latter two show that, with increasing time, the randomly moved particles have a larger influence as their velocity increases with time. This randomization produces more art-like versions of the original pathline visualizations, which are of course not suitable for any data analysis.

6 CONCLUSION

We presented a painting tool to draw overlapping, semi-transparent ribbons. To create organic, aesthetic shapes, users of the tool do not have to draw the shapes manually, but an algorithm randomly creates these ribbons. Nevertheless, the algorithm can be controlled by the users through interactively setting seeding points and changing the movement parameters and rendering options. By systematically varying these settings, we showed how our approach can be used to generate different visual patterns similar to patterns occurring in nature. We demonstrated that the approach is capable of producing visually appealing images that share the aesthetics and characteristics of floral representations but also of flow visualizations. With respect to the latter, we refer to pathlines of a time-dependent vector field, which do overlap and—depending on the rendering technique—can create similar visual patterns as our technique. In contrast to a flow visualization approach—which is based on data and can only be changed significantly by changing the data and seeding—our approach has more fine-grained interactions. While the approach is only able to produce a limited similarity to flow visualizations, an open question still is how to transfer other styles and features of flow visualizations into a painting tool like ours. Moreover, our approach does not aim at drawing detailed objects as it would be possible with brushes and pens. Instead, our painting approach—although influenceable—has a relatively high random component that creates a rather coarse brush when used with mouse movement. In the future, we want to explore further interaction techniques for our painting tool including multi-dimensional interactions by several users, e.g., on a touch table.

ACKNOWLEDGEMENTS

This work was supported by DFG within SFB 716/D.5.

REFERENCES

- [1] Flow art <http://www.saatchiart.com/pigaro>.
- [2] Stunning examples of fractal art <http://www.webdesignmash.com/art/stunning-examples-of-fractal-art/>.
- [3] ColorBrewer <http://colorbrewer2.org/>, May 2011.
- [4] M. A. Boden and E. A. Edmonds. What is generative art? *Digital Creativity*, 20(1-2):21–46, 2009.
- [5] J. V. Carlis and J. A. Konstan. Interactive visualization of serial periodic data. In *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology*, pages 29–38. ACM, 1998.
- [6] N. Chu, W. Baxter, L.-Y. Wei, and N. Govindaraju. Detail-preserving paint modeling for 3D brushes. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*, pages 27–34. ACM, 2010.
- [7] J. Esch and T. D. Rogers. The Screensaver Map: Dynamics on elliptic curves arising from polygonal folding. *Discrete & Computational Geometry*, 25(3):477–502, 2001.
- [8] G. W. Flake. *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation*. MIT Press, 1998.
- [9] A. G. Forbes, T. Höllerer, and G. Legrady. Generative fluid profiles for interactive media arts projects. In *Proceedings of the Symposium on Computational Aesthetics*, pages 37–43. ACM, 2013.
- [10] C. Garth, S. Barakat, and X. Tricoche. Interactive computation and rendering of finite-time Lyapunov exponent fields. *IEEE Transactions on Visualization and Computer Graphics*, 18(8):1368–1380, 2012.
- [11] C. Garth, G.-S. Li, X. Tricoche, C. Hansen, and H. Hagen. Visualization of coherent structures in transient 2D flows. In H.-C. Hege, K. Polthier, and G. Scheuermann, editors, *Topology-Based Methods in Visualization II*, Mathematics and Visualization, pages 1–13. Springer Berlin Heidelberg, 2009.
- [12] E. Henze. Cortices <http://enohenze.de/cortices/>, 2007.
- [13] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A sketching interface for 3D freeform design. In *ACM SIGGRAPH Courses*. ACM, 2007.
- [14] R. S. Laramée, H. Hauser, H. Doleisch, B. Vrolijk, F. H. Post, and D. Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum*, 23(2):203–222, 2004.
- [15] R. S. Laramée, H. Hauser, L. Zhao, and F. H. Post. Topology-based flow visualization, the state of the art. In *Topology-based Methods in Visualization*, Mathematics and Visualization, pages 1–19. Springer Berlin Heidelberg, 2007.
- [16] H. Li and D. Mould. Artistic tessellations by growing curves. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*, pages 125–134. ACM, 2011.
- [17] J. Maeda and P. Antonelli. *Design by Numbers*. MIT Press, 2001.
- [18] A. Mattox. Perlin flow ribbons <http://archive.anthonymattox.com/2007-2011-blog/work/ribbons.html>.
- [19] W. Merzkirch. *Flow Visualization*. Academic Press, 2nd edition, 1987.
- [20] C. Minard. Carte figurative des pertes successives en hommes de l'Armée Française dans campagne de Russie, 1869.
- [21] D. Mould. A stained glass image filter. In *Proceedings of the 14th Eurographics Workshop on Rendering*, pages 20–25. Eurographics Association, 2003.
- [22] M. Pearson. *Generative Art: a Practical Guide Using Processing*. Manning, 2011.
- [23] D. Phan, L. Xiao, R. Yeh, P. Hanrahan, and T. Winograd. Flow Map Layout. In *Proceedings of IEEE Symposium on Information Visualization*, pages 219–224. IEEE, 2005.
- [24] C. Reas, C. McWilliams, and J. Barendse. *Form and Code in Design, Art, and Architecture*, volume 21. New York: Princeton Architectural Press, 2010.
- [25] P. Riehmann, M. Hanfler, and B. Froehlich. Interactive Sankey Diagrams. In *Proceeding of the IEEE Symposium on Information Visualization*, pages 233–240. IEEE, 2005.
- [26] D. Rudolf, D. Mould, and E. Neufeld. Simulating wax crayons. In *Proceedings of 11th Pacific Conference on Computer Graphics and Applications*, pages 163–172. ACM, 2003.
- [27] T. M. Sezgin, T. Stahovich, and R. Davis. Sketch based interfaces: Early processing for sketch understanding. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06. ACM, 2006.
- [28] M. Weber, M. Alexa, and W. Müller. Visualizing time-series on spirals. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 7–13, 2001.
- [29] C. Wei and D. Mould. Coordinated particle systems for image stylization. In *Proceedings of the 2014 Graphics Interface Conference*, pages 225–233. Canadian Information Processing Society, 2014.
- [30] D. Weiskopf, R. P. Botchen, and T. Ertl. Interactive visualization of divergence in unsteady flow by level-set dye advection. In *Proceedings of SimVis*, pages 221–232, 2005.
- [31] E. O. Wilson. *Biophilia*. Harvard University Press, 1984.
- [32] L. Xu and D. Mould. Magnetic Curves: Curvature-controlled aesthetic curves using magnetic fields. In *Proceedings of the Fifth Eurographics conference on Computational Aesthetics in Graphics, Visualization and Imaging*, pages 1–8, 2009.