The 8th ACM/EG Expressive Symposium
C. Kaplan, A. Forbes, and S. DiVerdi (Editors)

Sketching and layering graffiti primitives

D. Berio[1], P. Asente[2], J. Echevarria[2], F. Fol Leymarie[1]

[1]Goldsmiths College, University of London, London, United Kingdom
[2]Adobe Research, San Jose, USA

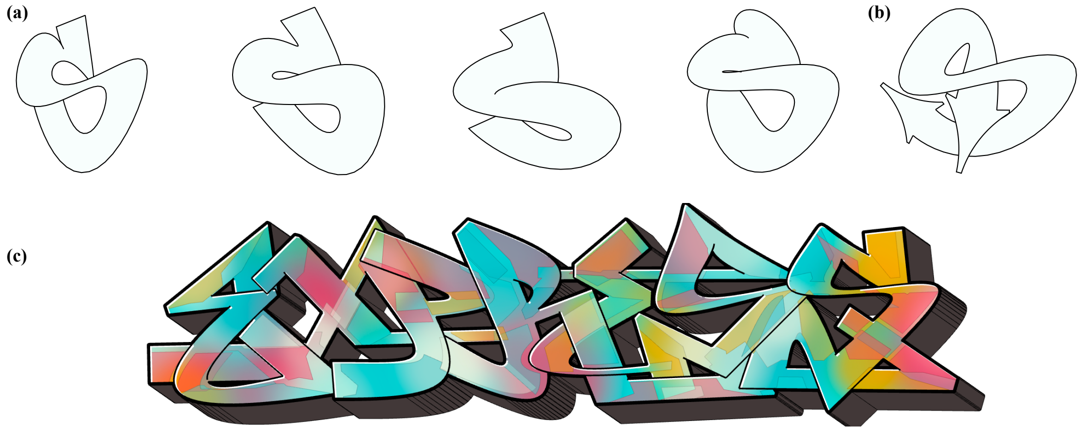


Figure 1: *Output from our system.* ***(a)****, stylistic variations of the same stroke for a letter "S".* ***(b)****, a letter "S" with non-global layering and arrows at the stroke ends.* ***(c)****, combination and layering of multiple strokes and rendering effects for the graffiti composition "EXPRESS".*

Abstract

We present a variant of the skeletal strokes algorithm aimed at mimicking the appearance of hand made graffiti art. It includes a unique fold-culling process that stylizes folds rather than eliminating them. We demonstrate how the stroke structure can be exploited to generate non-global layering and self-overlap effects like the ones that are typically seen in graffiti art and other related art forms like traditional calligraphy. The method produces vector output with no artificial artwork splits, patches or masks to render the non-global layering; each path of the vector output is part of the desired outline. The method lets users interactively generate a wide variety of stylised outputs.

CCS Concepts

• ***Computing methodologies*** $\rightarrow$ ***Parametric curve and surface models;*** • ***Applied computing*** $\rightarrow$ *Fine arts;*

1. Introduction

Like typography, calligraphy and handwriting, graffiti letters [CC84] are often conceived and created as the combination of one or more strokes. These strokes take the form of stylised building blocks that, depending on the graffiti sub-genre, are either sketched with skillful free hand motions or precisely traced in a geometric way. Strokes are often interlocked in complex ways and may have self-overlaps and loops [Fer16]. They are then fused and traced to create the outline of a highly stylised version of one or more letter forms. The resulting outlines are not limited to the boundary of the letter, but may extend to suggest where different strokes overlap or where a stroke folds over itself. This increase in visual complexity may be evocative of a 3D composition, but it does not necessarily follow the rules of projective geometry (Fig. 2).

Reproducing these kind of drawings with conventional vector drawing packages can be problematic. Many [Ado19b] assume that objects are separately layered in a back-to-front order. As a result, creating interlocking patterns and overlaps requires either manually masking hidden parts of an outline, or cutting overlapping parts and manually removing occluding parts of object outlines. Other applications [Ado19a] support planar-map decompositions, but in a way that does not maintain the continuity of the original strokes.

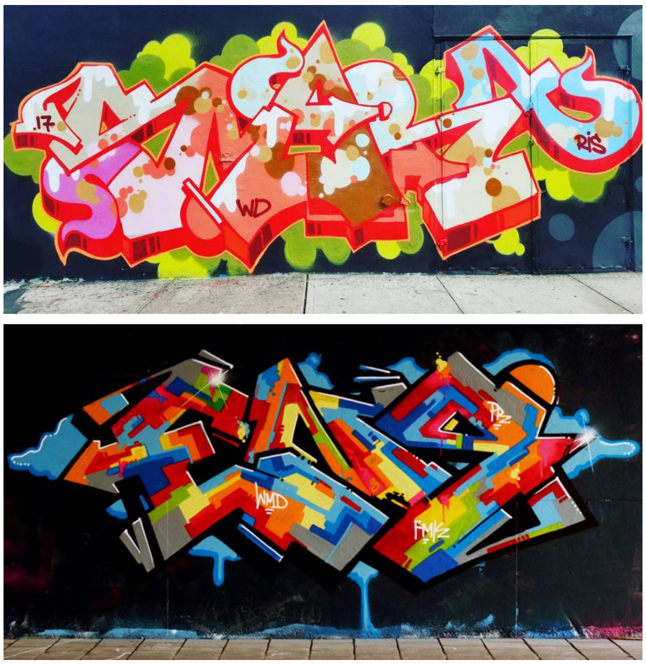


Figure 2: *Graffiti with complicated intertwined strokes, courtesy of the graffiti artists SMART (top) and ENS (bottom).*

Both approaches are time consuming and, more importantly, lose the underlying structure of the drawing. This makes it difficult to perform changes and explore variations of a drawing.

In this paper, we propose an interactive computational model of graffiti strokes (Sec. 3) and develop a method for rapidly combining these strokes into letters and other interlocking patterns (Sec. 4.3). Our stroke model relies on a variant of the popular Skeletal Strokes technique [HL94] that we extend to mimic the appearance of graffiti. We then exploit the stroke structure to develop an efficient method and interface for handling complex layering and self-overlaps. The output of our method is a set of non intersecting outlines, like the ones produced by hidden line removal methods in 3D, but relying on a fully 2D representation and interface.

2. Background

Strokes. Fonts are commonly stored as and defined by a set of outlines. However it is commonly acknowledged that letterforms are based on a set of strokes, making a stroke-based approach to font design advantageous [Noo05]. Knuth's Metafont system [Knu79] describes the strokes in a character as brush footprints swept along a set of parametric curves. Width variations along a stroke come from changes in the brush footprint, mimicking calligraphic pens. Jakubiak et al. [JPF06] observed that this approach can limit the generation of stylised fonts, and defined a stroke as a curve with parallel offsets for variable thickness and a pair of cap shapes at the stroke ends. Hu and Hertsch [HH01] developed a component-based representation of fonts, and treated each side of the outline separately to obtain more natural results in curved regions.

Skeletal Strokes [HL94] generate a stroke by deforming a vector input along a path. One common issue with them is the appearance of folds in high curvature portions of the spine. Several approaches have been proposed to adjust [LA15] or remove [HL94; Ase10] them. Contrary to those, we exploit the folding behavior to mimic artistic self-overlapping effects.

Layering. Wiley and Williams developed Druid [WW06], a system for designing interwoven drawings. The system resolves overlaps between spline curves with a local labelling of crossings. However, our experiments found their method unreliable in the presence of the folds and loops generated by the skeletal stroke algorithm, leading to edge visibility errors that can propagate around the outline. McCann and Pollard [MP09] developed an interactive system for non-globally layering transparent bitmaps based on detecting regions of overlap, but it does not handle objects overlapping themselves. Igarashi and Mitani [IM10] developed a similar method for 3D objects on a plane, which does permit self-overlaps. We follow a similar approach but operate on the outlines of 2D objects.

Asente et al. developed LivePaint [ASP07], an interactive method for editing and painting planar maps [BG89] that maintains the original underlying geometry. However, creating and modifying overlaps requires manually assigning appropriate stroke and fill attributes to edges and faces of the map. With a similar application in mind, Dalstein et al. developed Vector Graphics Complexes (VCG) [DRP14], a data structure specifically aimed at processing and editing potentially overlapping and intersecting vector art in a manner similar to planar maps, while maintaining topological and incidence relations. Our layering implementation relies on planar maps but maintains the structural information of a drawing across edits through a stroke-based representation.

3. Stroke generation

The basis for our stroke generation method is a variant of the popular skeletal strokes technique [HL94]. A skeletal stroke is defined as an input shape, called a *prototype*, that is deformed along a destination path, called the *spine*. The deformation is performed by mapping portions of the prototype to portions of the spine, and applying a deformation that depends on a variable-width profile that maps distances along the spine to a pair of widths.

Width profile. Typical skeletal stroke implementations assume that the width varies continuously along the spine. However, we observe that components of graffiti letters often have widths that change discontinuously at spine corners, resulting in an effect that evokes a 3D projection of a surface or the trace of a chiseled calligraphic pen. To facilitate this, we define a spine as a sequence of vertex pairs, where each vertex pair is connected by a *segment* and each segment has an initial and final width; see Fig. 3.

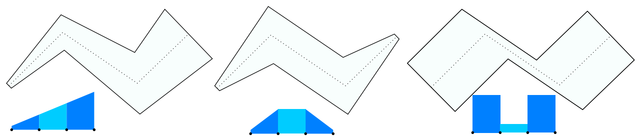


Figure 3: *Strokes with rectangular prototypes and varying width profiles (below). Each color represents a different segment.*

Our system uses only straight segments, but the method we describe is general enough to work with curved segments as well.

Prototype deformation. In the standard case of a continuous width profile, the prototype can be deformed by mapping points along its outline to points that are perpendicular to the spine. These points are given by a sequence of orthogonal *ribs* that are constructed on each side of the spine, their lengths depending on the width profile. This approach can lead to self-folds in the deformed prototype, corresponding to corners and high-curvature portions of the spine. They produce retrograde motion in the stroke outline. These folds are often considered undesirable, and the usual approach to avoid them is to adjust the orientation and length of the ribs. This can be done globally [HL94], by using the angle bisectors rather than the normals at corners and interpolating the intermediate ribs accordingly. Another approach is to perform the adjustment locally [ASP07], which avoids the potentially skewed appearance of the stroke caused by the latter method.

In our use case, we must accommodate width discontinuities at spine corners. Moreover, many graffiti styles stylize folds instead of avoiding them (Fig. 1). Our definition of ribs thus differs slightly from the one in traditional skeletal strokes. The initial and final ribs of the stroke are perpendicular to the spine. Each spine vertex gets two ribs, one for each segment, with the ribs for the segment interpolating between its starting and ending ribs (Fig. 4, bottom).

The ribs at each vertex are defined using an oblique coordinate system $[\hat{\boldsymbol{u}}_1, \hat{\boldsymbol{u}}_2]$ (Fig. 4a), centered at the vertex and defined as

$$\hat{\boldsymbol{u}}_1 = \hat{\boldsymbol{d}}_1 \mathrm{sgn}(\alpha) \quad \hat{\boldsymbol{u}}_2 = -\hat{\boldsymbol{d}}_2 \mathrm{sgn}(\alpha), \tag{1}$$

where $\hat{\boldsymbol{d}}_1$ and $\hat{\boldsymbol{d}}_2$ denote the unit tangents preceding and following the vertex, α is the angle between the two tangents, and $\mathrm{sgn}(\alpha)$ ensures that the coordinate system is always oriented towards the convex part of the stroke. The offsets with respect to this basis are then given by

$$o_1 = \frac{w_2}{\sin\alpha} \quad o_2 = \frac{w_1}{\sin\alpha}, \tag{2}$$

with w_1 and w_2 denoting the profile widths preceding and following the vertex. This construction results in a *weighted bisector* $\boldsymbol{b} = o_1\hat{\boldsymbol{u}}_1 + o_2\hat{\boldsymbol{u}}_2$, whose direction is the same as the angle bisector when the widths on each side of the vertex are equal.

On the convex side of a vertex, we test the outline angle at $\boldsymbol{b}$ and generate a miter if it is too acute, as is done in other stroking algorithms. The ribs at the vertex end either at $\boldsymbol{b}$ or at the miter intersections. On the concave side, the ribs could end at the tip of the vector $-\boldsymbol{b}$, removing folds (Fig. 4a) in a manner similar to the bisector-based method proposed in the original skeletal strokes implementation [HL94]. However, for our application we exploit the folds in order to render overlapping effects. To do so, we end the ribs at the tips of the vectors $\breve{w}\hat{\boldsymbol{u}}_1 - \boldsymbol{b}$ and $\breve{w}\hat{\boldsymbol{u}}_2 - \boldsymbol{b}$ (Fig. 4b). Here $\breve{w}$ is the minimum of o_1 and o_2 scaled by an angle fall-off function

$$1 - \exp\left(-\frac{\alpha^2}{\sigma^2}\right) \tag{3}$$

that decreases the amount of folding proportionally to the angle between spine segments, according to a user configurable parameter σ that we set to $\pi/4$ (Fig. 5). This avoids excessive folding

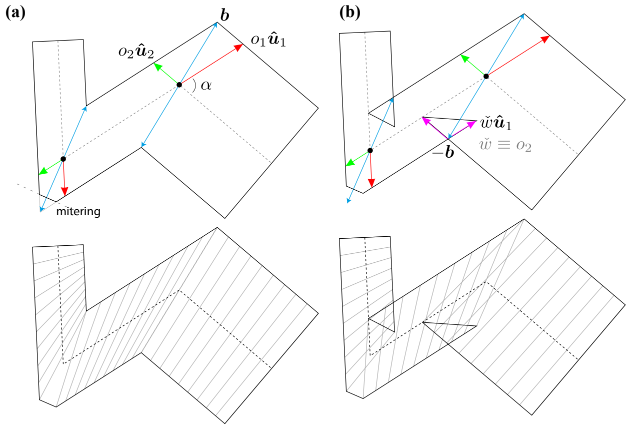


Figure 4: *Corner rib adjustment according to the oblique coordinate system* $[\hat{\boldsymbol{u}}_1, \hat{\boldsymbol{u}}_2]$ *and corner mitering.* ***(a)*** *Unfolded construction similar to the one proposed by Hsu et al. [HL94].* ***(b)*** *Folded construction. Below, the ribs generated by each construction.*

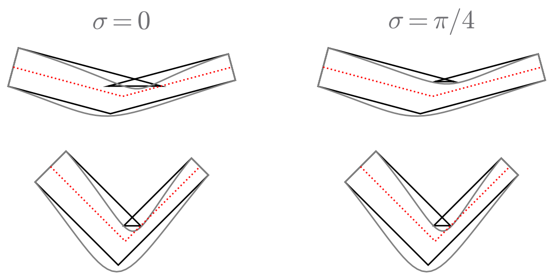


Figure 5: *Effect of the angle fall-off parameter* σ*. When* $\sigma = 0$ *(left) the parameter has no effect. A larger value of* $\sigma = \pi/4$ *(right) reduces the amount of folding proportionally to the angle between spine segments* α*. This affects the resulting smoothed trajectory, shown in grey. Note that the fall-off function does not affect folds with acute angles (second row).*

for obtuse angles and, for our use case, improves the visual quality of the smoothing discussed in the subsequent section. Note that the segment-end ribs for a vertex do not actually pass through the vertex, but since our prototypes are always rectangles, only the rib endpoints matter.

The folds generated by this method remain through the stroke-smoothing step described in the rest of this section, and are resolved in the layering method described in Sec. 4.

3.1. Smooth strokes

In addition to applying strokes to polygonal and curved spines, we would like to smooth strokes applied to polygonal spines to achieve certain graffiti styles. One approach would be to smooth the spine before applying the stroke, but we note that the result often looks rather mechanical and not hand-drawn (Fig. 6b). Instead, we apply the stroke to the original spine and then smooth the resulting outline (Fig. 6c).

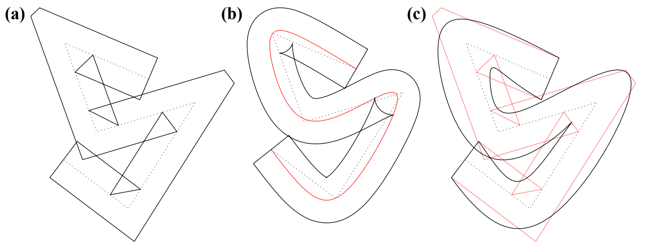


Figure 6: *Variations of strokes for the same spine and width.* ***(a)*** *Polygonal stroke.* ***(b)*** *Curved stroke from the smoothed spine (in red).* ***(c)*** *Smoothed outline from the polygonal stroke (in red).*

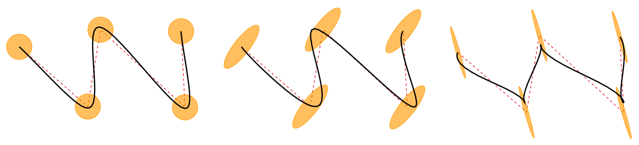


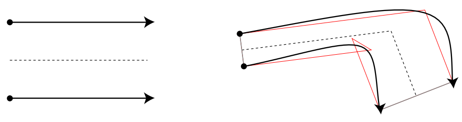
Figure 7: *Examples of trajectories generated with different combinations of tied Gaussians.*

We begin with a polygonal spine defined by a sequence of vertices and a simple rectangular prototype. We then define variably smooth strokes by considering the vertices of the deformed prototype as a *motor plan* [SHKF04] for a synthetic motion that follows their configuration. Different types of strokes and stylisations can be produced by varying the kinematics of the motion as well as the shape of the prototype used to generate the stroke. This allows for a large range of stylistic variations of a stroke. They resemble graffiti art visually and also mimic the process typically followed when constructing graffiti letters. Furthermore, spines usually consist of a small number of vertices, making them easy to author interactively or procedurally.

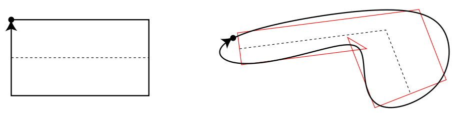
Curve generation. The proposed method can be implemented with a variety of curve generation and smoothing methods depending on the application domain of choice. Because of the shared goal of achieving graffiti aesthetics, we use the curve model from Berio et al. [BCFL17]. It drives the evolution of a dynamical system with a controller that tracks the spatial layout of a series of control points. The tracking precision is defined by pairing each control point with a covariance matrix, which results in a sequence of multivariate Gaussians. A higher variance produces a smoother trajectory. Forcing all the covariance ellipses to share the same orientation allows to produce different stylisations of the trajectory with a small number of parameters (Fig. 7).

Smooth stroke types. We define three kinds of smooth strokes:

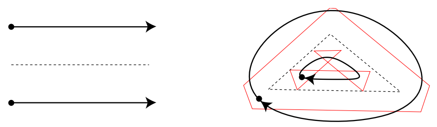
- *Squared end strokes* are defined with a prototype made by two parallel and similarly oriented lines. The stroke is then produced by tracing each side of the stroke with two separate motions, and connecting the trajectory ends with two straight line segments.



- *Rounded strokes* are defined with a rectangular prototype in our approach. The stroke is then produced with a single looping motion that follows all the vertices of the resulting skeletal stroke and returns to the beginning. To generate this motion we append three copies of the n vertices of the skeletal stroke and duplicate them in order, creating a sequence of $3n$ points. These points are the input to the motion plan described before, producing an open trajectory that goes around the outline three times. We then consider the subset of the generated trajectory corresponding to the middle n vertices. Because the endpoints of the middle segment may not meet (Fig. 8a), we join them with a line segment and smooth the area around the join (Fig. 8b) by linearly blending a small window of vertices in the corresponding region. The roundness at the stroke ends can be parametrically controlled by adjusting the covariance matrices of the control points at the ends of the stroke (Fig. 8c).



- *Closed strokes* are defined with a prototype made by two parallel lines and a closed polygonal spine. A closed stroke is then produced with two looping motions that follow each side of the deformed prototype The looping motions are generated with the same approach used for rounded strokes.



Some graffiti styles alternate smooth parts of a stroke with polygonal ones. To do so, we generate smooth trajectories for subsets of the envelope and the connect them into a single stroke (Fig. 9).

4. Apparent layering and overlaps

Graffiti often contains intricate overlaid and intertwined parts with non-global layering (Fig. 2). These compositions can be evocative of a 3D projection, but rarely follow the rules of projective geometry, representing an abstraction or caricature of such rules. An analysis of the geometry of "pictorial space" [Koe12] in graffiti is beyond the scope of this paper. However, we can exploit the stroke-based structure to develop a 2D interface that allows self-overlaps (Sec. 4.2) and non-global layering (Sec. 4.3).

4.1. Partitions

Our layering and self-overlap rendering method relies upon subdividing strokes into a series of *partitions*. Each partition corresponds to a contiguous portion of the spine and represents a potential layer with a depth value. Not all depth assignments are possible; for example in Fig. 10 partition 1 cannot come between partitions 3 and 4 in depth order. Sec. 4.3 discusses how we prevent impossible assignments. Partitions are also assigned integer indices, in order, from the beginning of the spine to the end.

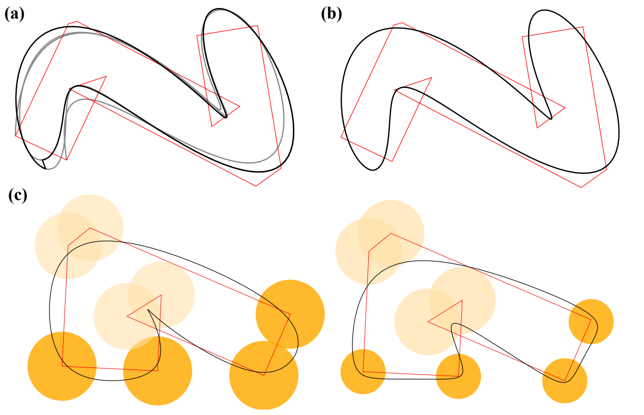


Figure 8: *Rounded strokes.* ***(a)*** *The curve generated for* $3n$ *skeletal stroke vertices. The curve ends do not meet perfectly at the join (at the left end of the stroke), so we perform smoothing* ***(b)****.* ***(c)*** *The orange circles depict the covariance ellipses for the vertices of the stroke, with the emphasised ellipses denoting the covariances at the stroke ends. Decreasing the variance of these (right) decreases the roundness at the stroke ends.*

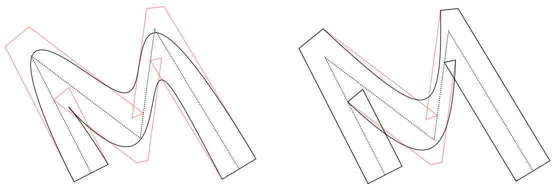


Figure 9: *A smooth stroke with squared ends (left) and a piece-wise smooth version of it (right)*

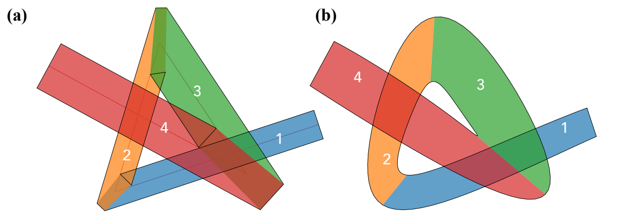


Figure 10: *Partition shapes (color coded) for a* ***(a)*** *polygonal and* ***(b)*** *smooth stroke.*

The skeletal stroke algorithm allows a straightforward mapping from points on the spine to corresponding points on the stroke outline. These portions of the outline form the outside edges of *partition shapes* (Fig. 10). For outside corners with bevels, we assign the bevel edge to the outline of both adjacent partition shapes, and for inside corners with retrograde segments, we assign the loop area to both, as shown in Fig. 10a. This ensures that the partition shapes fully cover the area of the stroke, sometimes producing small regions where adjacent partition shapes overlap.

For polygonal strokes and spines the subdivision into partitions is trivial: each partition corresponds to a spine segment and the partition shapes are given by the outline segments mapped to a given spine segment. For a polygonal spine with a smoothed stroke the partition shapes are given by mapping polygonal spine vertices to corresponding vertices in the smooth stroke outline. The smoothing method we use [BCFL17] allows this mapping because the trajectory generation produces an equal number of time steps for each control point. Each segment of the outline can track its time step and thus its control point. This can be extended to other curve generation methods as long as a mapping is possible between the control points and the resulting curve. Finally, for an arbitrarily curved spine the partitioning is based on an estimate of curvature extrema and corners along the spine.

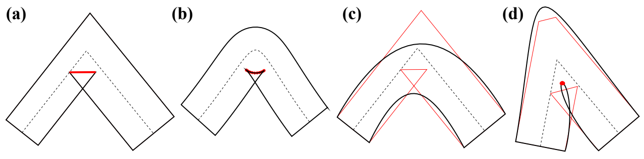


Figure 11: *Different kinds of fold cases.* ***(a)****, corner in a polygonal stroke.* ***(b)****, curvature extrema in a stroke with a curved spine.* ***(c)****, flattened fold in a smooth stroke.* ***(d)****, fold in a smooth stroke. The retrograde portions are marked in red.*

4.2. Fold culling

Deforming a prototype according to the proposed method often results in a shape that contains self-folds. Most traditional implementation of skeletal strokes consider this an issue and suggest methods to overcome it [Ase10; HL94; LA15]. In our application, we exploit this property to generate soft strokes (as discussed above) and to achieve stylised folding/overlap effects that are often seen in graffiti art, as well as comic drawings.

We identify folds with a procedure similar to the method proposed by Asente [Ase10] and find portions of the stroke outline that present retrograde motion. For a polygonal spine, these are trivially given by outline segments that are part of a concave portion of the outline and connect two vertices belonging to two different partitions (Fig. 11a). For a curved spine, these are given by the outline points that map to points of the spine with radius of curvature less than the corresponding stroke half-width (Fig. 11b). For the case of a smoothed envelope, we first identify a series of potentially retrograde portions by finding the smoothed points that map to retrograde portions of the polygonal envelope. However, our smoothing technique is sufficiently free that it does not always maintain the retrograde segments, so there may not be a fold anymore (Fig. 11c). To determine whether there is a fold, we check if the midpoint of a potentially retrograde portion is contained in both adjacent partition shapes, in which case the portion is considered retrograde (Fig. 11d).

Once all retrograde portions have been identified, we traverse the outline on each side until we reach a common point of intersection. We then cull the retrograde portion of the side with lower depth. The remaining side is marked as partially visible according to a user configurable parameter $\in [0,1]$ that interpolates the visibility of the side relative to its length (Fig. 12).

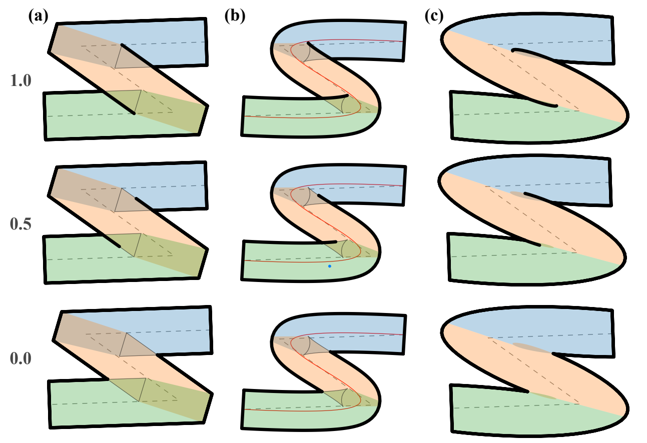


Figure 12: *Stylised folds, showing the effect of the fold-rendering parameter.*

4.3. Layering

The partitioning scheme allows us to compute a layering of one or more strokes analogously to Igarashi and Mitani's method for 3D shapes on a plane [IM10]. We compute a planar map from the combination of all partition shapes, where each partition corresponds to a layer. Each edge of the resulting planar map is assigned to an edge of a partition shape and thus can be assigned the corresponding partition index. For each interior face of the planar map we then compute a *partition list* P that indicates which partitions overlap the region defined by the face. This can be easily done by choosing a point inside the face and testing which partition shapes contain the point. We then sort the partition list according to an ascending depth order, and iterate over each face edge. An edge is marked as visible if the partition it belongs to is the same as the higher one in the depth-sorted partition list.

Resolving impossible layer orders. The procedure above is efficient and can handle many types of complicated layering structures. At the same time, there can be combinations of partitions and depth values that have no consistent layering solutions, especially in the neighbourhood of spine vertices (Fig. 10a). To resolve these cases, we use a *list graph* structure [IM10; MP09], which has a vertex for each internal face of the planar map and an edge for each pair of faces that are adjacent and share a common partition.

Impossible overlaps can be detected by examining the connected components of the list graph and checking for inconsistencies in the layer ordering across the corresponding faces. For each connected component we compute a list of partitions assigned to it and sort it by increasing depth. By construction, two partitions with indices p_i and p_j are adjacent in the stroke if $|p_i - p_j| = 1$. A connected component of the list graph contains an impossible overlap if any adjacent pair in the list is not contiguous in the depth sorted list. If an impossible order is detected, we proceed in a manner similar to [IM10] and compute the maximum area covered by each partition and consider all permutations that do not contain impossible orders. We then choose the permutation with the lowest number layer of swaps, weighted by the area of each layer.

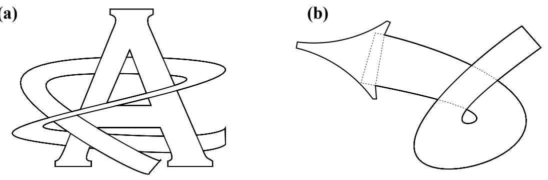


Figure 13: *Additional layering effects.* ***(a)*** *A stroke is combined with the outline of a letter "A". The letter is assigned a single partition and depth value.* ***(b)*** *A union operation is used to add an arrow head to a stroke.*

Mixing strokes and arbitrary vector inputs. This layering method relies on a partitioning of the input given by our stroke representation. However, we can also combine strokes with arbitrary shapes (Fig. 13a) as long as each is treated as a single partition with a unique depth value, in which case the method operates as a vector counterpart of the one proposed by McCann and Pollard [MP09] for bitmap inputs.

Unions. In addition to the depth ordering, we can also easily handle unions between one or more layers. To do so we define a set of union pairs $\{p_i, p_j\}$ between partitions, and cull an edge if any pair of partitions assigned to it are assigned to a union. As an example application of unions, we can add arrowheads to a stroke (Fig. 13b) by simply generating an arrowhead shape and then specifying an union between the arrow head and the partition corresponding to the end of a stroke. The same approach can be used to append arbitrary caps to the strokes with an effect similar to the one proposed by Jakubiak et al. [JPF06].

5. Results and applications

The combination of the parametric stroke model and the proposed folding and layering methods lets us easily render intertwined strokes in a way that would be difficult to achieve with traditional vector graphics methods. The stroke representation can be constructed and edited with a simple interface and is well suited for the rapid generation of compositions and renderings that mimic the appearance of graffiti art.

Performance and interaction. The stroke generation and layering procedures can be used interactively and let a user quickly produce and explore variations of graffiti compositions. To test the performance of the method we generated patterns of increasing complexity, similarly the one shown in Fig. 18. On a commodity laptop, we achieve frame rates suitable for interactive editing as long as the number of curve samples is fewer than 1000 (Fig. 14). For example the letters in Fig. 16 have about 400 points each and take less than 30 milliseconds for layering and rendering. The main bottleneck of the system is currently the curve generation method [BCFL17]. When generating closed curves the performance hit is even higher, since we repeat curve points 3 times. The pattern in Fig. 18 is generated with a single closed stroke, has 7488 vertices and takes 10 seconds for curve generation and 0.2 seconds for layering. For interaction and preview purposes we can limit the number of curve samples, allowing for interactive editing of complex patterns like the one shown in Fig. 1c, which was produced interactively with our user interface.

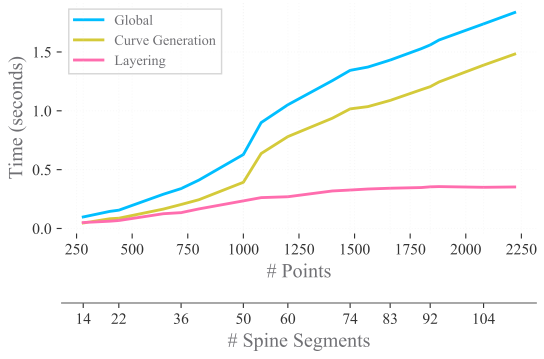


Figure 14: *Performance of the method for increasing number of curve samples and spine segments.*

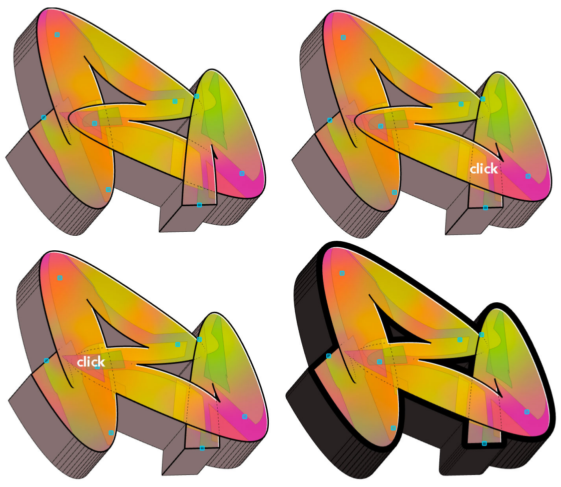


Figure 15: *Layering interactions: the user can swap depth ordering (top right) or create unions (bottom left) with a click. The figures also show extrusions and an additional thick outline around the merged strokes (bottom right), two common effects in graffiti.*

The interface to our method is simple: the user creates a stroke by clicking to define a sparse sequence of spine vertices. The user can then vary the shape of a stroke by adjusting stroke parameters such as the amount of smoothing. The width of a stroke can be adjusted globally with a set of sliders, or locally by dragging perpendicularly to a spine edge. The layering interface lets a user perform layer swaps in a manner similar to the one described by Igarashi and Mitani [IM10]. Clicking on an overlap area brings the bottommost partition to the top. Unions can also be created similarly, by clicking on an overlap area with a different tool. (Fig. 15)

Fills and rendering effects. We generate colorful compositions by exploiting the faces of the planar map generated during the layering process. Randomly offsetting the faces and assigning each face a color from a user-specified palette gives results similar to those often seen in graffiti art (Fig. 17d). We can use the same palette

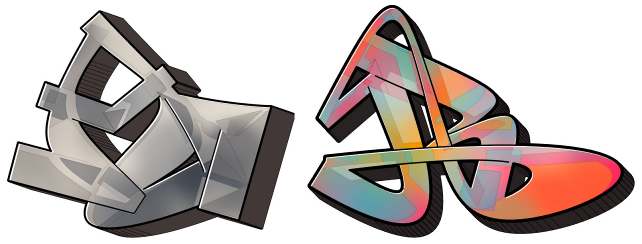


Figure 16: *Graffiti letters ("A" and "R") generated and rendered with our method.*

to smoothly fill areas in ways that mimic the diffused use of spray paint. This can be simply done by using the union of all outlines to mask a raster fill. In the examples given here we generate the fill by randomly alpha-blending smooth gradient bitmaps over the the interior of the outlines (Fig. 17c). To increase the realism of the rendering we can add highlights to parts of the outline that are approximately perpendicular to a given light direction. This, combined with an extrusion effect captures a visual effect that is often seen in conventional instances of graffiti art. Fig. 16 shows results that combine all these effects.

Extrusion. One effect that is often seen in graffiti art is a simple oblique isometric extrusion of the composition as a whole; see Fig. 2, top. Our method to construct these is as follows:

1. Rotate the entire composition so that the extrusion can be done directly downward, in the negative y direction.
2. Create a planar map from the rotated composition and extract the edges.
3. Split each edge at corners and at extrema in the x direction. The result is a set of straight and curved segments that intersect each other only at their endpoints. Each segment has x coordinates that monotonically increase or decrease.
4. Perform a topological sort of the segments, with the ordering function being that segment s_1 is greater than segment s_2 if some point on s_1 and some point on s_2 have the same x coordinate, different y coordinates, and the point on s_1 has a larger y coordinate. This formalizes the idea that s_1 is greater than s_2 if s_1 is higher than s_2 in the y direction.
5. Construct a total order from the partial order produced by the topological sort.
6. For each segment, construct an extrusion face by offsetting the segment vertically by the extrusion depth and connecting the ends of the original and offset segments. Stack these faces with the last – the one from the edge with the smallest y coordinate – on the top of the stacking order.
7. Place the extrusion faces below the rotated composition, and rotate everything back to the original orientation.

The extrusion faces can then be stroked and filled as desired. Figures 1, 15, 16, and 17 show our results. The extrusion can be modified by choosing areas to subdivide more finely, creating an effect similar to that in Fig. 2, top.

Fig. 15 also includes a thicker outline around the union of all the strokes, another common effect in graffiti art (Fig. 2, bottom).

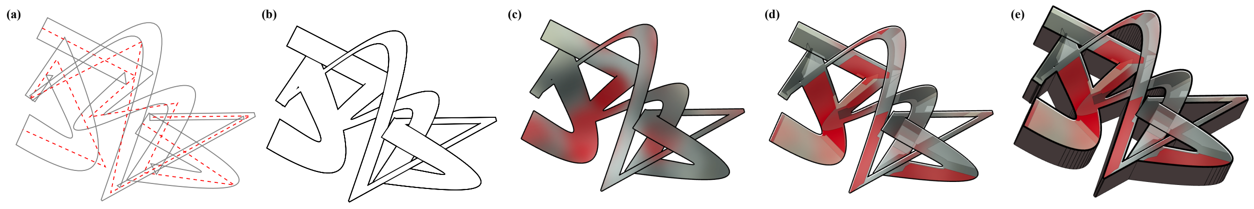


Figure 17: *Interactive construction of a graffiti letter "R". **(a)**, stroke outlines and spine polygons. **(b)**, layered outlines. **(c)**, fill in gradients. **(d)**, geometric effects using planar map faces and highlights. **(e)**, extrusion.*

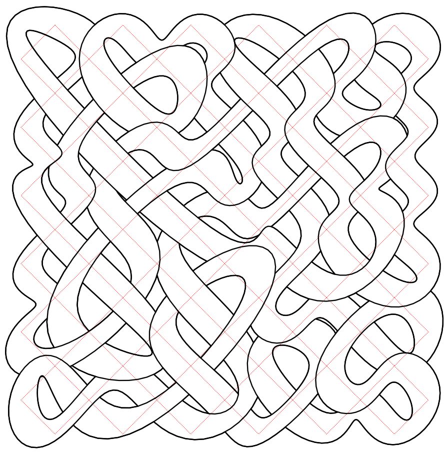


Figure 18: *Weaving pattern generated by constructing an Eulerian path over the planar graph in dashed red.*

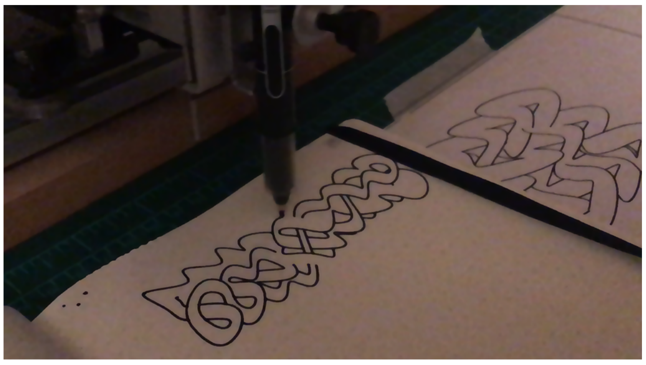


Figure 19: *A plotter drawing a pattern generated by our system on a notebook.*

Generative applications Specifying a stroke with a sparse sequence of control points is a simple user interaction procedure. The same sparse representation is also convenient in a procedural modelling applications, in which a procedural system can operate at a high level by specifying sparse sequences of control vertices and then various stylisations of the output can be explored parametrically. For example, a simple procedure can generate stylized knots or weave patterns. We first generate a 2D lattice (Fig. 18, dashed red) and compute an Eulerian path or cycle along the lattice. We then construct a single stroke along the path, making sure that the depth values of crossings are interleaved. The results are evocative of more abstract forms of graffiti art and sketchy renditions of weaving patterns.

Figure 20: Left*: Recreated version of the logo for the 1970s band ELOY showing text stylized in a similar way to graffiti.* Right*: Graffiti-inspired fashion. Photo by Antonio de Moraes Barros Filho/WireImage/Getty Images.*

Machine drawings

The output of our method is suitable for being realized with a drawing robot or plotter. Once the primary printing tool in the early days of computing, plotters have today regained popularity as a creative tool for computer graphics because of their affordability and their ability to create vector drawings using a variety of physical drawing mediums. The output of our method is suitable for constructing tool paths for such machines. Furthermore, since we maintain the path ordering defined at the stroke level, the motions of the machine are visually consistent and often evoke the sequence of movements that would be followed by a human when producing a drawing (Fig. 19). This same property could be used to generate stroke animations from the output of our system.

6. Limitations and future work

We presented a system and interface that permit the generation of convincing synthetic graffiti with simple, flexible stroke representation. Since its beginnings in the 60s, cross-pollination between graffiti and other areas in graphic design and illustration has been a constant [Ans15]. As an example, typographic styles of the 60s and 70s had, and still has, a big influence on graffiti letter design (Fig. 20, left). In turn, current instances of graffiti art can be seen on record covers, in advertisements and in fashion (Fig. 20, right). The computational recreation of graffiti's main stylistic features is a valuable addition to the toolset of digital creatives, and makes our system a useful tool across the 2D design space.

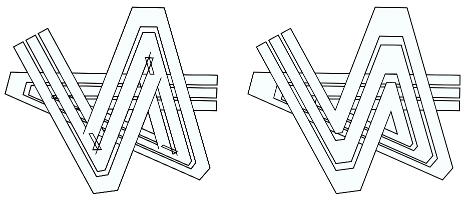


Figure 21: *Layering of strokes with a more complex stroke prototype. On the left, our self-overlap procedure fails because folds occur within the stroke, and not just along the boundary. On the right, we shrink the ribs along the bisector, as in Hsu et al. [HL94] and as discussed in Sec. 3, resulting in a visually consistent output.*

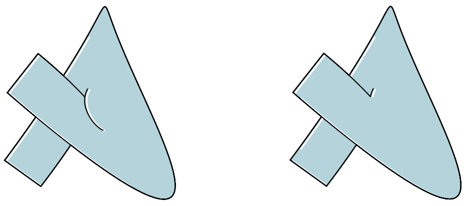


Figure 22: *Overlaps with self-folds. We currently support this effect only through user interaction. The effect is prohibited by the impossible layer order resolution procedure (right).*

Our method generates strokes with self-overlap effects that are typical of this art form. However, our current approach builds on the assumption of a rectangular skeletal stroke prototype. Extending this approach to arbitrary vector inputs is an interesting extension for future studies, but doing so is not trivial. With complex prototypes (Fig. 21, left) folded areas can occur within the stroke, and not just along the boundary. One possible way to handle these cases could be to estimate the spine through the computation of a symmetry axes [BN78]. Symmetry axes also have the potential to extend our layering procedure to arbitrary shapes. While our current approach relies on the partitioning of the input given by the skeletal stroke spine, the same partitioning could be computed automatically from the skeleton of the input.

In our description of the layering procedure, we have focused on the generation of outlines. In future work we plan to handle the layering of fill patterns and gradients defined along a stroke as well. This can be achieved by exploiting the planar map and partition list generated during our method.

The layering procedure combined with fold culling can automatically and rapidly handle many different configurations of one or more strokes with self overlaps. However, our current implementation of impossible layer resolution [IM10] does not permit certain configurations that visually make sense. As an example, it may be desirable to render a partition of a stroke that passes below a fold produced by the adjacent partition (Fig. 22, left), but this effect is discarded by the resolution process (Fig. 22, right). In order to handle these cases, we currently allow the user to disable the resolution step and create this effect by performing layer swaps with a few clicks in the regions of interest. In future iterations of this work we plan to handle these cases automatically.

References

[Ado19a] Adobe. *Adobe Animate: User Guide*. 2019. URL: https://helpx.adobe.com/animate/user-guide.html 1.

[Ado19b] Adobe. *Illustrator: User Guide*. 2019. URL: https://helpx.adobe.com/illustrator/user-guide.html 1.

[Ans15] Anssi, A. *Forms of Rockin': Graffiti Letters and Popular Culture*. Dokument Press, 2015. ISBN: 9185639745 8.

[Ase10] Asente, P. "Folding avoidance in skeletal strokes". *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*. Eurographics Association. 2010, 33–40 2, 5.

[ASP07] Asente, P., Schuster, M., and Pettit, T. "Dynamic planar map illustration". *ACM Transactions on Graphics (TOG)*. Vol. 26. 3. ACM. 2007, 30 2, 3.

[BCFL17] Berio, D., Calinon, S., and Fol Leymarie, F. "Dynamic Graffiti Stylisation with Stochastic Optimal Control". *ACM Proceedings of the 4th International Conference on Movement and Computing*. London, UK, 2017 4–6.

[BG89] Baudelaire, P. and Gangnet, M. "Planar maps: an interaction paradigm for graphic design". *ACM SIGCHI Bulletin*. Vol. 20. SI. ACM. 1989, 313–318 2.

[BN78] Blum, H. and Nagel, R N. *Shape description using weighted symmetric axis features*. 1978. DOI: 10.1016/0031-3203(78)90025-0 9.

[CC84] Cooper, M. and Chalfant, H. *Subway Art*. Holt, Rinehart and Winston, 1984. ISBN: 0030719631 1.

[DRP14] Dalstein, B., Ronfard, R., and Van de Panne, M. "Vector graphics complexes". *ACM Transactions on Graphics (TOG)* 33.4 (2014), 133 2.

[Fer16] Ferri, A. *Teoria del writing, La ricerca dello stile*. Professional Dreamers, 2016 1.

[HH01] Hu, C. and Hersch, R. D. "Parameterizable fonts based on shape components". *Computer Graphics and Applications, IEEE* 21.3 (2001), 70–85 2.

[HL94] Hsu, S. C. and Lee, I. H. H. "Drawing and animation using skeletal strokes". *Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH '94* (1994), 109–118 2, 3, 5, 9.

[IM10] Igarashi, T. and Mitani, J. "Apparent layer operations for the manipulation of deformable objects". *ACM Transactions on Graphics (TOG)*. Vol. 29. 4. ACM. 2010, 110 2, 6, 7, 9.

[JPF06] Jakubiak, E. J., Perry, R. N., and Frisken, S. F. "An improved representation for stroke-based fonts". *Proceedings of ACM SIGGRAPH*. Vol. 4. 2006 2, 6.

[Knu79] Knuth, D. "Mathematical typography". *Bulletin of the American Mathematical Society* 1.2 (1979) 2.

[Koe12] Koenderink, J. "Geometry of imaginary spaces". *Journal of Physiology-Paris* 106.5 (2012), 173–182 4.

[LA15] Lang, K. and Alexa, M. "The Markov pen: online synthesis of free-hand drawing styles". *Proceedings of the workshop on Non-Photorealistic Animation and Rendering*. Eurographics Association. 2015, 203–215 2, 5.

[MP09] McCann, J. and Pollard, N. "Local layering". *ACM Transactions on Graphics (TOG)*. Vol. 28. 3. ACM. 2009, 84 2, 6.

[Noo05] Noordzij, G. *The stroke*. Hyphen, 2005 2.

[SHKF04] Sosnik, R., Hauptmann, B., Karni, A., and Flash, T. "When practice leads to co-articulation: the evolution of geometrically defined movement primitives". *Experimental Brain Research* 156.4 (2004), 422–438 4.

[WW06] Wiley, K. and Williams, L. R. "Representation of interwoven surfaces in 2 1/2 D drawing". *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2006, 65–74 2.