Parallelized Pixel Averaging for a Real-time Ornamental Pen

Lena Polke

Department of Mathematics, Technical University Munich, Germany; polkel@ma.tum.de

Abstract

Our goal is to implement an ornamental calligraphic pen that enriches the stroke of a pressure-sensitive digital pen with perfectly adjusted ornamental content while the stroke is drawn. For this, we propose an algorithm that calculates a local coordinate system within the stroke using the graphics processing unit. The fast calculations on the GPU make it possible that the calculation progress is made visible and that the user can observe the ornamental content flow dynamically into its intended position. The result of our algorithm is close to conformal—meaning that the map of the ornamental content to the local coordinate system within the stroke preserves angles and infinitesimal shapes.

Introduction

There are a variety of apps that enable artists, writers, students, designers, and many others to express their ideas graphically on a tablet or computer. Many applications provide a library of pens from which to select the graphical output that the pen stroke should produce. The pens range from classic pencils to pressure- and tilt-sensitive calligraphy pens, from expressive pens that scatter objects to intricate simulations of the behavior and appearance of brush hairs. Our research provides a method to enrich the stroke of a pen with ornamental content in real time such that the content is deformed in an aesthetically pleasing and mathematically sound way.

As a basis, we use the simulation of a calligraphic stroke on tablets based on the pressure sensitivity of the associated digital pen. As already pointed out in [10], we want to enrich the pen's stroke with the desired content while it is drawn. This real time aspect is crucial for our approach. We do not want to calculate the deformation of the artistic content for the finished pen stroke after pressing a key and waiting some time until the result is displayed. On the contrary, the calculations are performed in real time simultaneously with the simulation of the pen stroke itself. The main improvement with respect to [10] concerning the real time aspect is the use of the GPU which makes calculations faster.

In addition, we do not want to distribute the content along the drawn curve as if with a stamp, nor do we want to truncate or even alter the content to fit the pen stroke. Instead, we aim for an algorithm that preserves the artistic content of the given pattern in such a way that it is locally distorted as little as possible while perfectly matching the unaltered pen stroke. In this context, the main improvement in comparison to [10] is that we manage to generate a close-to-conformal map iteratively during the drawing process.

Figure 1: *The ornamental content fits the pen's stroke perfectly.*

For this, given a rectangular ornamental tile, we imagine several copies of this tile to form a strip of yet unknown length, as shown in Figure 1 on the left. In combination with the calligraphic stroke in the center, the desired result on the right contains the perfectly adjusted ornamental strip within the drawn stroke.

Before we present the algorithm providing us with digitally enriched strokes, we explain two important aspects of it: the notion of conformality, and why computations on the graphics processing unit (GPU) serve our goal of real-time computations.

Conformality: preserving artistic content

To preserve the artistic content of the ornamental tile, our algorithm is supposed to produce a close-to-conformal mapping of the given tile to the stroke. Conformal maps preserve angles everywhere on their domain and also infinitesimal shapes, which means that they are locally similarity transformations (see [7, Chapter 8.2]). We have chosen this criterion of conformality as a measure of content distortion. A different approach lies in the preservation of areas, another metric property. Whether the criterion of preserving angles or areas is better for the desired result depends largely on the purpose of the matter. A classic example are maps of the Earth: No map can preserve angles (important for ship navigation) and areas (important for land survey) simultaneously.

Our work focuses on conformal maps from an ornamental tile to a calligraphic stroke. Such a map exists due to the Riemann mapping theorem. To obtain it, we want to define a coordinate system within the stroke so that the ornamental content can be transferred to the stroke via its natural coordinate system. The standard two-dimensional coordinate system consists of an orthogonal grid of vertical and horizontal lines. When searching for an angle preserving map, this structure of the coordinate system in the ornamental tile has to be transferred to the stroke as in Figure 2.

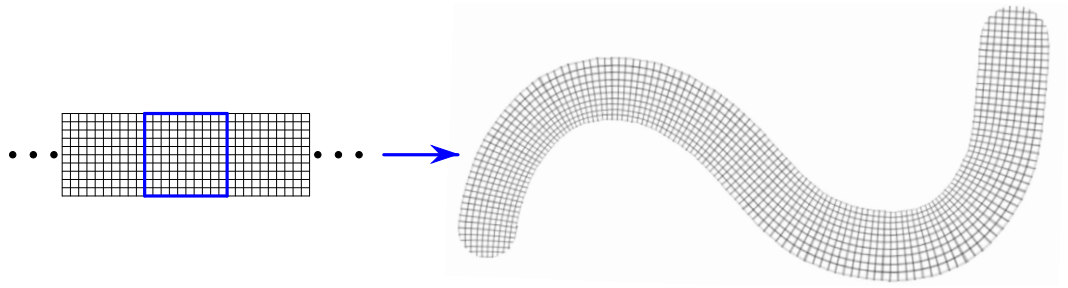

Figure 2: *Conformal map of a coordinate system from the rectangular tile to a stroke.*

Any artistic tile can then be mapped directly onto the transformed coordinate system within the stroke by reading off the content at the corresponding position within the tile and transferring it to the stroke. We will deal later with the special coordinate system our algorithm works with and how the content can be transferred.

GPU computations: parallel and fast

The other main requirement for our algorithm besides conformality is the real-time aspect. Real-time means for us that the computations are made visible in such a way that the ornament flows into its desired position while the stroke is drawn. This represents a significant difference from other algorithms that compute conformal maps for given planar domains and display the solution as soon as the calcuations are done.

The stroke as target domain of the conformal map is constantly changing. If the calculations were performed globally on the target domain, we would have to recalculate the entire mapping from the preimage to the constantly changing stroke at each point in time. So we aim for an algorithm that acts locally on both the new and the existing parts of the stroke at any given time. To make this local aspect possible and ensure the real-time aspect, we developed an algorithm that is executed on the graphics processing unit (GPU). What the GPU is and why it is interesting for us follows arguments from [5].

The GPU is one of the computational units of modern computers. Another unit, the central processing unit (CPU), is the commonly used processor for computer programs. It executes instructions given by an algorithm, for example, and sequentially tracks what it does in the computer's memory. The GPU, however, works massively in parallel which can make calculations much faster than on the CPU. Its computational results are stored to textures and the pixels of the texture are the subject of the calculations. This means that the GPU performs all computations locally, which is also consistent with our goal of computing a close-to-conformal mapping that is locally a similarity transformation. However, the purely local view of the texture's pixels restricts the number of possible commands and prohibits, for example, commands such as "*draw a circle around a point with a specified radius*." This is why the GPU is often considered "less clever" than the CPU since it has no inherent memory of what happens to geometric objects like circles or triangles on the texture for example.

A two-dimensional texture consists of a certain finite number of pixels arranged in a rectangular grid. Each pixel is assigned four values called red, green, blue and alpha channels. They are denoted by (r, g, b, a), where each entry on our target domains is usually an 8-bit integer between 0 and 255. This nomenclature is obvious if you think of a texture as an image where each pixel is assigned a color and the red, green, and blue values define the exact color of each pixel. The alpha value measures the opacity, which is interesting, for example, when overlapping textures are displayed on the computer screen. However, the $rgba$-vector of each pixel does not have to contain color information. The GPU can write arbitrary information to the textures if they are in the desired format, and it can read that information back out for every pixel. On this basis, we calculate our conformal mapping from the coordinate system of the ornamental preimage to the stroke. The GPU creates a texture that contains the conformal image of the coordinate system inside the stroke. How this is done is described in the following sections.

Pixel Averaging Algorithm

The algorithm presented in this section will form the basis for our real-time computations on the GPU. It is based on the principle of pixel averaging. Often, pixel averaging is used for blurring of images and similar effects. However, in our setting, the GPU textures won't contain color, but rather coordinate information, and one may interpret our approach as "blurring the position data." The averaging procedure is iteratively applied to all pixels simultaneously due to the parallel structure of the GPU.

Given a planar domain D, which we specify in the next section, we want to find a conformal mapping from an (ornamental) tile T to this domain. The tile T is given as a rectangular texture, each pixel of which contains color information about the artistic content. To use the GPU for calculations, the target domain D of the conformal map is also translated into a texture. In this texture, the zero vector $(r, g, b, a) = (0, 0, 0, 0)$ is assigned to each pixel outside the domain.

Given a conformal map $f : T \to D$, the goal for each pixel $q = f(p)$ within the domain D is that q's $rgba$-vector contains the coordinates of its preimage pixel p in the ornamental tile T. For this, its preimage coordinates $p = (x, y)$ are stored in the red and green channels of the $rgba$-vector and the alpha channel is set to 1 to distinguish between pixels inside and outside the stroke: $(r, g, b, a)_q = (x, y, 0, 1)$. In this way we obtain a map of the coordinate system of the tile T to the target domain D and we can read off the color of the preimage of each pixel within tile T by performing a reverse pixel lookup.

However, searching for the preimage pixel within the target domain corresponds to the inverse of the conformal mapping $f : T \to D$ that we are actually looking for. To find this inverse map, we apply pixel averaging. We basically follow the same steps as the pixel averaging algorithm presented in [11], but their algorithm is applied to predefined areas and seems to be computed on the CPU. The authors of [4] use the algorithm of [11] to map the hyperbolic plane to arbitrary simply-connected shapes with long, narrow regions. Especially, our treatment of the boundary conditions differs from the methods presented in these articles.

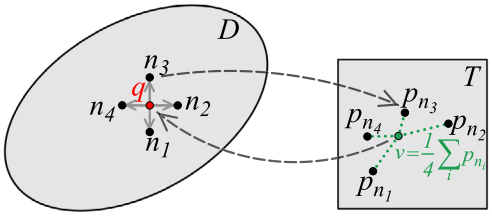

Figure 3: *Pixel averaging algorithm for interior pixels of the domain.*

For every pixel q within the target domain D, we consider its four neighbors n_i for $i \in \{1, 2, 3, 4\}$. We distinguish two cases. First, the pixel q and all its neighbors n_i lie inside the domain D as in Figure 3 on the left. Then, the $rgba$-vectors of the neighbors $(r, g, b, a)_{n_i} = p_{n_i}$ can be read off and their average $v = \frac{1}{4}\sum_i p_{n_i}$ is calculated and assigned to the pixel q: $(r, g, b, a)_q = v$.

In the second case, one or more neighbors of the pixel q are outside the domain and we say that pixel q is in the boundary area of the target domain. Averaging cannot be applied immediately because external pixels do not contain coordinate information but only zero. For this reason, we adjust the averaging procedure for the boundary pixels, inspired by the Schwarz Reflection Principle as it is stated in [6, Chapter 5.XI.5]. The Schwarz Reflection Principle tells us that a conformal map from the tile T to the target domain D can be analytically extended to the boundary of the target domain and beyond. For a domain with line or circle segments as boundary components, the extension works by reflection at the respective boundary segment.

This means for us that if a neighbor pixel n lies outside the target domain D, we would want to calculate its reflection r_n along the boundary which would be inside the domain and contain coordinate information. The corresponding point p_{r_n} within the ornamental tile T could then be reflected back at the boundary of T and the resulting coordinates p_n could be used for the averaging procedure. However, we do not yet know what the boundary of the pen stroke will be and how the reflection will be adjusted appropriately. We need to determine the shape of the boundary before we concretize our averaging procedure for the pixels in the boundary area.

Additionally, we need initial values for all pixels in the target domain before we can apply the averaging algorithm. How they are selected also depends on the particular target domain, which we will discuss below.

The Pen Stroke, our Target Domain

Given the entire target domain, i.e. the whole pen stroke, we can consider its definition in the continuous world. This definition is based on a planar curve $\gamma : I \to \mathbb{R}^2$ with $I \subset \mathbb{R}$ a closed interval. For each $t \in I$, a continuous function $r(t)$ assigns a real value to the curve point $\gamma(t)$. The stroke s is defined as the union of all disks $C(t)$ with radius $r(t)$ drawn around $\gamma(t)$ as in Figure 4:

$$s = \bigcup_{t \in I} C(t) \quad \text{with} \quad C(t) = \{x \in \mathbb{R}^2 : \|x - \gamma(t)\|^2 \leq r(t)^2\}.$$

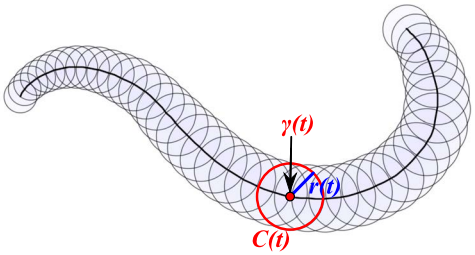

Figure 4: *Nomenclature of the stroke.*

The radius function $r(t)$ is supposed to satisfy the condition

$$r'(t)^2 < \|\gamma'(t)\|^2 ,$$

which guarantees that two infinitesimally close circles intersect at two different real intersection points. This is important, for example, to exclude the possibility of a large circle covering the rest of the stroke, as it would be the case for $\gamma(t) = (t, 0)$ and $r(t) = \sin(t)$ with $t \in [0, 1]$. This corresponds to a case where the user starts drawing and increases the pressure much faster than the pen is moved.

This leads us to our work, where the stroke is created by user input on a tablet with a pressure-sensitive digital pen. The drawn curve is given by a collection of discrete data points $\gamma(t_d)$ for each registered time step t_d and is not continuous. The data also contains the pressure exerted at each t_d, which is translated into the

circle radii $r(t_d)$ at the corresponding curve points. Whether we should interpolate the discrete data points to obtain continuous functions $\gamma(t)$ and $r(t)$, and whether this would effectively improve our approach, is a question we are currently working on. Nevertheless, to get close to continuity, we linearly refine the sample of data points such that the distance between two registered points $\gamma(t_d)$ and $\gamma(t_{d+1})$ is two pixels within the target texture.

For computational reasons, we need to restrict the strokes that our algorithm works with. We are not yet able to handle singularities of the stroke's boundary or self-intersecting strokes as in Figure 10. For now, we assume that the curve γ is free of singularities or self-intersections. How we plan to address these problems in the future is described in the outlook section.

Concrete Computations

The pixel averaging algorithm is used to map the coordinate system of an ornamental tile to the stroke. For this, we consider one copy T of the rectangular tile and identify its left and right edges to get an infinitely tiled pattern. This reduces the size of the coordinates that need to be stored in the texture's pixel values compared to an ornamental strip of undetermined length as shown in Figure 1 on the left. All calculations concerning the coordinate values are then performed modulo the width of the ornamental tile.

The target domain is the stroke that appears circle by circle. For each new circle $C(t_d)$ at time t_d, we assign initial values to the pixels so that the averaging algorithm can begin its work. If good initial values are chosen, the algorithm will converge faster to the desired close-to-conformal mapping. For this, we consider the vector $v(t_d) = \gamma(t_d) - \gamma(t_{d-1})$ for every new circle $C(t_d)$ and we set the vector of the very first circle to $v(t_0) = \gamma(t_1) - \gamma(t_0)$ as soon as the second circle appears. We define a map $\tau : C(t_d) \to T$ by considering the square around $C(t_d)$ mapped to a particular position within our cylindrical T as in Figure 5. The square is oriented so that two of its edges are parallel to the vector $v(t_d)$, which are mapped to the top and bottom edges of T under consideration of the drawing direction. The circle center $\gamma(t_d)$ is mapped to the middle horizontal line of T. The exact position $\tau(\gamma(t_d))$ on this line is calculated so that the position of the preimage circle in T advances in proportion to the progress of $\gamma(t_d)$ in the drawing direction and the radius $r(t_d)$. In particular, the distance of $\gamma(t_d)$ to its predecessor is always 2 pixels, and the corresponding progress of $\tau(\gamma(t_d))$ in the tile is calculated by

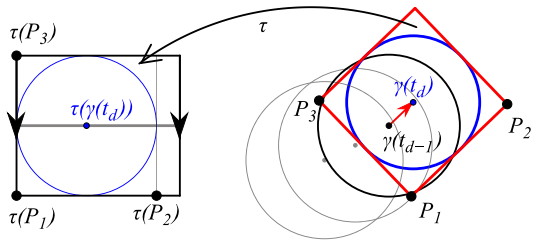

Figure 5: *Inverse coordinate map.*

$$|\tau(\gamma(t_d)) - \tau(\gamma(t_{d-1}))| = \frac{2 \cdot w}{2 \cdot r(t_d)} ,$$

where $w = |\tau(P_2) - \tau(P_1)| = 32$ is the pixel width of the square around the preimage circle in T.

If a pixel q inside the circle $C(t_d)$ becomes part of the stroke at time t_d, its preimage coordinates $\tau(q) = (x, y)$ are stored in the $rgba$-vector of q. More precisely, x and y are split and stored in two textures, since the GPU only provides 8 bits per pixel entry and this precision would not be sufficient for our purpose of precise pixel averaging. Concretely, the decimal digits of x and y are stored separately from the integer parts and for the averaging step both are brought together again.

With these initial values, the pixel averaging algorithm comes into play, and for the pixels in the stroke's boundary area we apply the special handling that we address now. The structure of the stroke's boundary results from its definition as the union of all circles $C(t)$ along the drawn curve $\gamma(t)$. In the continuous case, and with our restriction to boundaries free of singularities and self-intersection, the boundary is defined as in Theorem 2 of [1]. It mainly consists of the intersection points of infinitely close circles $C(t)$ and $C(t + \epsilon)$,

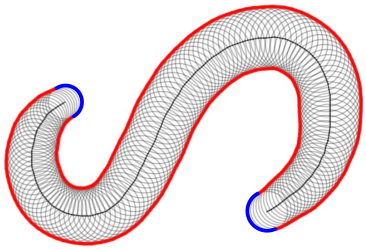

Figure 6: *The stroke's boundary.*

resulting in two curves called the envelope of the family of all circles $C(t)$ (see the two red curves in Figure 6). To complete the boundary, the two circle segments of the first and last circle of the stroke are added which connect the envelope curves (see the two blue circle segments in Figure 6). In our nearly continuous case, the boundary consists of the circle segments of the first and last circles, as well as the intersections of adjacent circles and the line segments between them.

For the averaging process this means that for pixels near the first and last circle segments of the boundary we can actually use circular reflection as in the Schwarz Reflection Principle to find corresponding coordinates for neighboring pixels outside the stroke. We reflect neighbors outside the stroke once at the circle segment of the first or last circle of the stroke and a second time at the preimage circles within tile T before averaging the coordinates.

For pixels near the two enveloping curves we do not know the exact shape of the boundary. Transferring the Reflection Principle is difficult and, at the moment, we only consider those neighboring pixels n_i of pixels q near the envelope that lie within the stroke. For these pixels n_i we read off their preimage coordinates p_{n_i} within the ornamental tile T. If, let's say, the neighboring pixel n_3 opposite to n_1 lies outside the stroke as in Figure 7 on the left, we consider p_{n_1} and its reflection $refl(p_{n_1}) = p_{n_3}$ at the edge of the rectangular tile T in the averaging step as in Figure 7 on the right. The result is then encoded in the $rgba$-vector of the pixel q.

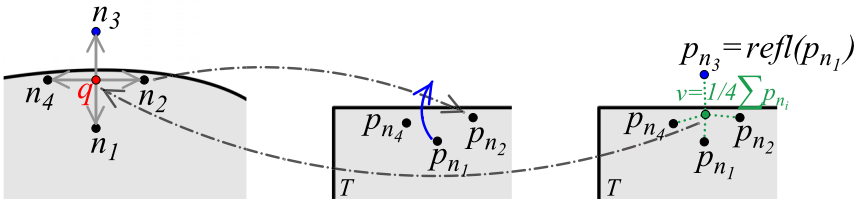

Figure 7: *Averaging procedure for pixels in the boundary area.*

The effect of each averaging iteration becomes visible as soon as the coordinates within the target domain D is translated into an image texture using reverse pixel lookup. Doing so, you can watch the target image flow into its final position, as seen in the screen capture [8]. Figure 8 shows some image examples drawn with our algorithm. How good our algorithm approximates a conformal map will be examined in the following section.

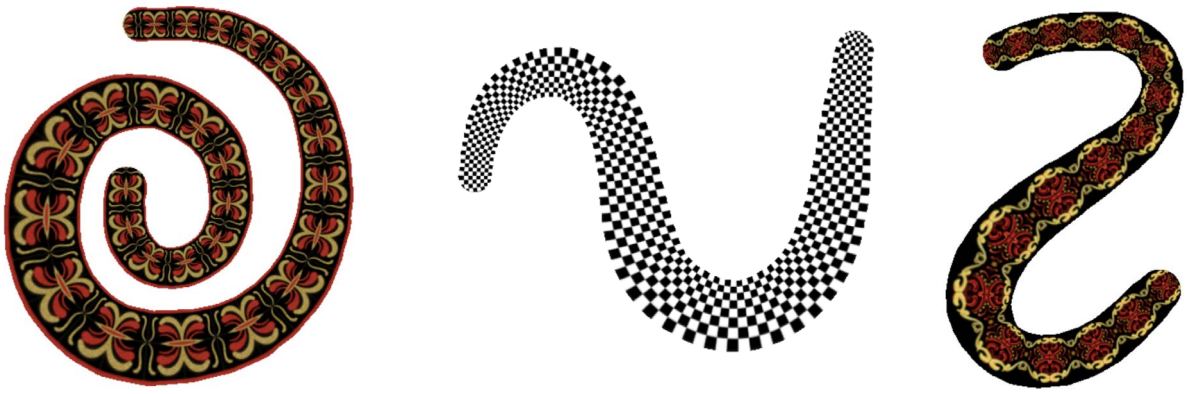

Figure 8: *Strokes drawn using our algorithm.*

Test for Conformality

In order to test our algorithm, it is important to find a reference measure for the actual distortion of the coordinate system in the target domain. We measure the distortion by conformality, i.e. we check to what extent right angles and infinitesimal shapes are preserved.

Since the coordinate system itself was mapped on the level of the pixels of the textures, we also test conformality from a discrete viewpoint. The pixels within the target texture are arranged rectangularly. For each pixel q inside the stroke, we take its four neighbors n_1 through n_4 and read their encoded coordinate values p_{n_i}. Suppose that n_1 and n_3 as well as n_2 and n_4 are opposite with respect to q in the rectangular pixel grid. Then we check whether the vectors $v_1 = p_{n_1} - p_{n_3}$ and $v_2 = p_{n_2} - p_{n_4}$ within the ornamental tile are perpendicular too. If this is the case, right angles are preserved by our map, and one criterion of being close-to-conformal is fulfilled. We also check that local length ratios are maintained in accordance with the definition of discrete conformal maps on polyhedral surfaces (see [2]). Amongst other ratios, we test the deviation of the ratio $\frac{|p_{n_1}-p_{n_2}|\cdot|p_{n_3}-p_{n_4}|}{|p_{n_2}-p_{n_3}|\cdot|p_{n_4}-p_{n_1}|}$ from the value $1 = \frac{|n_1-n_2|\cdot|n_3-n_4|}{|n_2-n_3|\cdot|n_4-n_1|}$ associated with the square formed by the four neighbors n_1 through n_4 around the considered pixel q.

To make the results visible, we encode them by colors. When a pixel is displayed in green, right angles and length ratios are preserved. The deviation from these criteria is gradually indicated by a color change, with red indicating that the map is not close-to-conformal in this area. As can be seen in Figure 9 and in the screen capture [9], our algorithm satisfies the conformality requirements quite well, at least for the type of strokes that fit our assumptions. The red and blue colors disappear during drawing because the ongoing pixel averaging brings the coordinates into a close-to-conformal position.

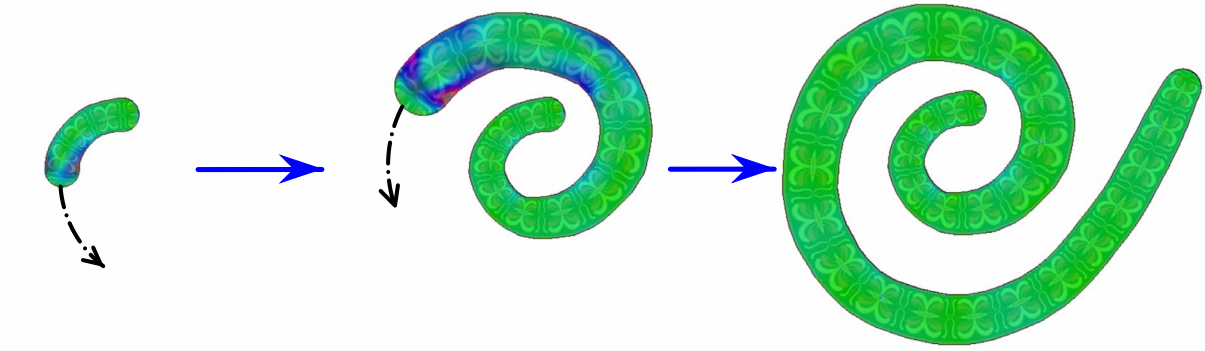

Figure 9: *Test for conformality while the stroke is drawn.*

Summary and Outlook on Future Work

We have presented an algorithm based on pixel averaging on the GPU that allows us to dynamically compute a close-to-conformal map of a rectangular ornamental tile to a simulated calligraphic stroke in real time during the drawing process. The image of the artistic content is preserved in the sense that it is neither truncated nor altered to fit perfectly into the target domain.

So far, our strokes had to be free of self-intersections, and we only considered strokes whose boundaries were free of singularities. However, it is desirable to remove these assumptions, as one would like to use a simulated calligraphic pen for writing or drawing without stroke restrictions. Figure 10 shows two examples of strokes that we plan to work on in the future.

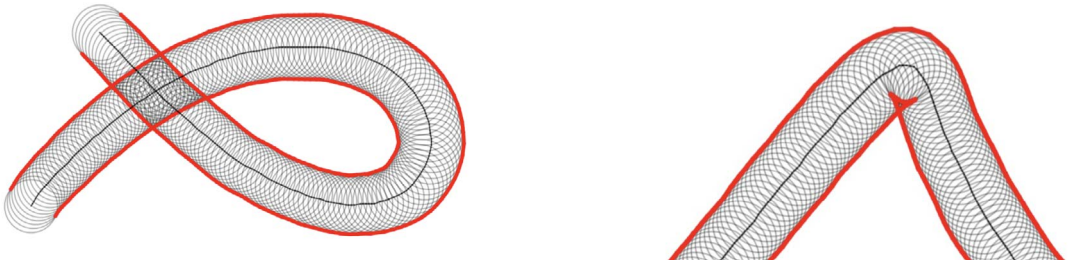

Figure 10: *Excluded cases of strokes due to self-intersection (left) and singularities of the boundary (right).*

For this, the theory of stroke envelopes becomes important. As mentioned before, envelopes are defined as the curves consisting of the intersections of infinitesimally close circles of the family in the continuous case. However, there are equivalent definitions for the same envelopes in terms of differential geometry (see again [1] or [3]). We want to understand the theory behind the envelopes in order to predict and handle singularities.

In addition, self-intersections of the stroke should be possible. For this, it might be a good approach to split the stroke at a suitable point into two branches, each of which is stored in its own texture. Pixel averaging can be applied to each of these textures, but the seams between them need to be handled properly.

Another open question is whether the boundary would be handled better in the pixel averaging algorithm if specific interpolating curves were computed from the discrete data points for $\gamma(t_d)$ and the radius function $r(t_d)$. Then, if $\gamma(t)$ and $r(t)$ are given continuously, one could even compute continuous boundary curves, i.e., the envelopes of the family of circles defined by $\gamma(t)$ and $r(t)$. This will hopefully make it possible to fully apply an adaption of Schwarz's Reflection Principle to achieve a better handling of pixels in the stroke's boundary area in the averaging algorithm.

References

[1] K. Bickel, P. Gorkin, and T. Tran. "Applications of Envelopes." *Complex Analysis and its Synergies*, vol. 6, no. 1, 2020, pp. 1–14.

[2] A. I. Bobenko, S. Sechelmann, and B. Springborn. "Discrete Conformal Maps: Boundary Value Problems, Circle Domains, Fuchsian and Schottky Uniformization." *Advances in Discrete Differential Geometry*, Springer, Berlin, Heidelberg, 2016, pp. 1–56.

[3] J. W. Bruce and P. J. Giblin. *Curves and Singularities: A Geometrical Introduction to Singularity Theory.* Cambridge University Press, vol. 12, no. 1, 1992.

[4] E. Kopczyński and D. Celińska-Kopczyńska. "Conformal Mappings of the Hyperbolic Plane to Arbitrary Shapes." *Bridges Conference Proceedings*, Linz, Austria, July 16–20, 2019, pp. 91–98. https://archive.bridgesmathart.org/2019/bridges2019-91.pdf.

[5] A. Montag. *Domain Parallel Machines.* Technische Universität München, 2020.

[6] T. Needham. *Visual Complex Analysis.* Oxford University Press, 1997.

[7] R. Penrose. *The Road to Reality.* Random House, 2006.

[8] L. Polke. *StrokeRecording*. YouTube. 2023. https://youtu.be/8go2SzSr2vA.

[9] L. Polke. *ConformalityRecording*. YouTube. 2023. https://youtu.be/I80_VwhFCkA.

[10] L. Polke and J. Richter-Gebert. "Real-time Ornamental Calligraphic Pens." *Bridges Conference Proceedings*, online, Aug. 01–03, 2021, pp. 141–148. https://archive.bridgesmathart.org/2021/bridges2021-141.pdf.

[11] D. Swart. "Warping Pictures Nicely." *Bridges Conference Proceedings*, Coimbra, Portugal, July 27–31, 2011, pp. 303–310. https://archive.bridgesmathart.org/2011/bridges2011-303.pdf.