Procedurally Generated Script

Sam Pollard
Western Washington University

CS 580 – Fall 2015 – Geoff Matthews
December 4, 2015

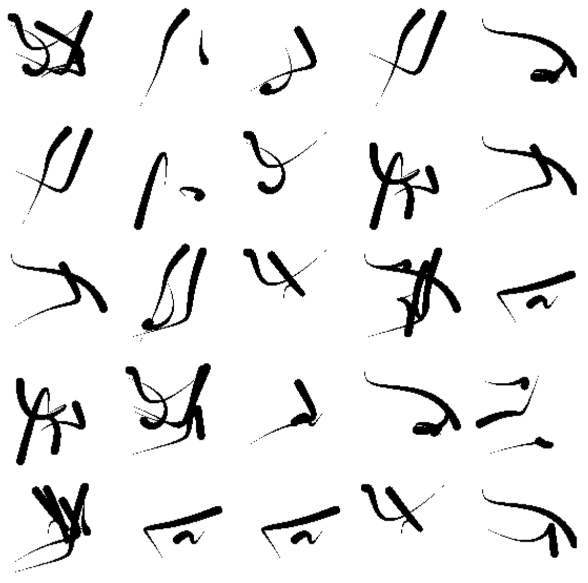

Figure 1: Random characters using $T(t) = 0.1(1 - t)$.

1 Introduction

We wish to model an arbitrary handwriting system that may be used to procedurally generate realistic looking handwriting. Once the parameters are specified this model aims to output pseudorandom sequences of characters which look realistic without any further intervention.

This model is in line with the Metafont language, which models characters as curves with finite width. This is in contrast to languages like PostScript and TrueType which describe the outline of each character.

This decision is based on the assumption the hand draws curves with thickness ϵ which may change based on physical attributes of the curve. There may be many factors which affect the

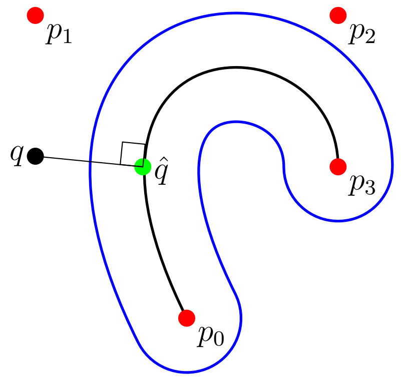

Figure 2: A sample bezier curve with a constant thickness ϵ.

thickness of a curve so in general we define a function $T : [0, 1] \to [0, \infty)$. We denote ϵ the output of this function. The T used for in Fig. 2 is a constant function.

To model each stroke (or part of a stroke), we define a Bézier curve $B(t)$ for $t \in [0, 1]$ by four control points p_0, p_1, p_2, $p_3 \in \mathbb{R}^2$. For our purposes, the control points are in $[0, 1]^2$. To clarify notation, $B(t) = \begin{bmatrix} B_u(t) & B_v(t) \end{bmatrix}^T$

To determine whether this point lies inside or outside the region, we must first express $B(t)$ implicitly. We define

$$B(t) = (1-t)^3 p_0 + 3(1-t)^2 t p_1 + 3(1-t)t^2 p_2 + t^3 p_3$$

where $p_i = (u_i, v_i)$, the coordinates of each control point.

Next, we determine if a given point $q = (u, v)$ lies inside of the stroke. That is, there exists some $\hat{q} = B(\hat{t})$ such that $||q - \hat{q}||_2 < T(\hat{t})$. One way to accomplish this is to find the nearest point to q on B, denoted $\hat{q} = (\hat{u}, \hat{v})$. Since B is a parametrized curve there exists a $\hat{t}$ such that $B(\hat{t}) = \hat{q}$. Then, q is contained in the stroke if and only if $||q - \hat{q}||_2 < \epsilon$ for some $\hat{t} \in [0, 1]$. We find one such $\hat{q}$ by observing the vector $\hat{q} - q$ is orthogonal to $\mathrm{d}B/\,\mathrm{d}t$. There may be multiple such $\hat{q}$. In this case, we simply choose the $\hat{q}$ with the smallest distance to q. There may also exist multiple $\hat{t} \in [0, 1]$ such that $B(\hat{t}) = \hat{q}$. Of these, we choose the $\hat{t}$ such that $||q - \hat{q}||_2$ is minimized. So we solve for t in

$$\begin{aligned} 0 &= \hat{q} \cdot \frac{\mathrm{d}B}{\mathrm{d}t} \\ &= B_u(t)\frac{\mathrm{d}B_u}{\mathrm{d}t}(t) + B_v(t)\frac{\mathrm{d}B_v}{\mathrm{d}t}(t) \end{aligned} \tag{1}$$

which amounts to finding the real roots of a fifth-order polynomial. We know fifth order polynomials have at least one real root and at most five real roots. Therefore, the problem of nonunique $\hat{t}$ and $\hat{q}$ will require at most finding these five roots and computing the minimimum distance for each $\hat{t}$. For the exact formula of this polynomial see the attached Python code.

2 The `Handwriting` Texture

We define a texture as a function which takes as input a (u, v) coordinate and returns either another texture (which can in turn be evaluated at (u, v)) or an RGB color value. In the case of handwriting, we return one texture if (u, v) is inside the curve and a second texture otherwise. The T function described in Section 1 is known as the brush in the `Handwriting` class. This implementation defines T as a function only of the Bézier curve's parameter t.

A given `Handwriting` instance has a list of characters defined by zero or more Bézier curves. Each character has all of its Bézier curves (also called strokes) contained in $[0, 1]^2$. Specifically, for any curve, all of its control points are in $[0, 1]^2$. For each (u, v) texture coordinate, we determine if $\hat{q}$ is contained in any of the strokes in the nearby characters.

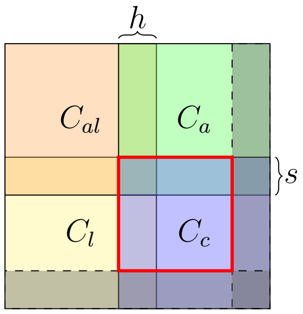

Figure 3: A point inside of the red box may be inside any of C_{al}, C_c, C_a, or C_l.

To make the writing look more natural, we wish to have characters overlap. This may be specified with h and s. If $h, s \in [0, 0.5)$ then there will never be more than four characters present in any given point. Consider Fig. 3. Given that each colored box is $[0, 1]^2$ and we stack each character so that the overlap horizontally is h and the overlap vertically is s, for each character C we must also check the characters to the left, above, and diagonally above and to the left of C. We do not need to check the other surrounding characters because these will be checked if (u, v) is in the first $[1 - h] \times [1 - s]$ box of the next character. For the center character C_c this box is outlined with thick, red lines in Fig. 3.

Then, given some coordinate (u, v) we transform the coordinate so we can determine which character box the point is contained in as well as where inside the box. This is achieved with the functions

$$g_u(u) = u + h \left\lfloor \frac{u}{1-h} \right\rfloor \tag{2}$$

$$g_v(v) = u + s \left\lfloor \frac{v}{1-s} \right\rfloor . \tag{3}$$

$$f_u(u) = u + h \left\lfloor \frac{u-h}{1-h} \right\rfloor \tag{4}$$

$$f_v(v) = v + s \left\lfloor \frac{v-s}{1-s} \right\rfloor \tag{5}$$

Equations (2) and (3) transform from a (u, v) coordinate to the position in the overlapped characters, where the ingetral parts of $(g_u(u), g_v(v))$ determine the character and the decimal parts determine the coordinate in that character. However, this overlapping lattice may have multiple characters for a given point. Thus for each point u, v we must check $(g_u(u), g_v(v))$, $(g_u(u), f_v(v))$, $(f_u(u), g_v(v))$, and $(f_u(u), f_v(v))$. These determine the coordinates inside of C_c, C_a, C_l, and C_{al} from Fig. 3, respectively. The plots of (2) and (4) are shown in Fig. 4. Notice this is only along one axis and there is overlap of these functions on the domain $n + [h, 1 - h]$ for $n \in \{0, 1, 2, \dots\}$.

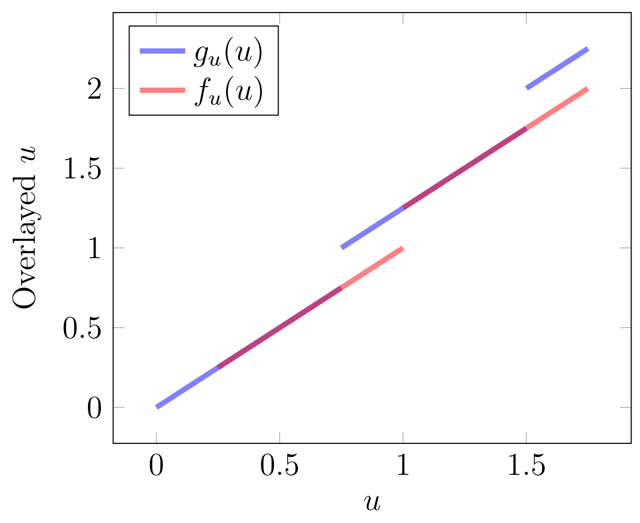

Figure 4: Plots of equations (2) and (4) with $h = 0.25$.

3 Lessons Learned

This approach is rather complicated and limited. Using Equations (2)–(5), characters must all have the same spacing between them. Furthermore, performing other improvements is difficult because those equations are required anywhere to transform from the character space to the texture space.

The open-endedness of this project makes it exciting but also slows progress progress. It is difficult to know which design choices will lead to favorable results and which are nebulous or do not produce anything useful.

4 Future Work

The ideas outlined here are ordered by decreasing importance. Generalizing how characters are positioned on the surface is the most important because without it further improvements become increasingly difficult.

4.1 Generalizing Character Positioning

A more general approach to positioning the characters would be to use traditional computer graphics methods (see [2]) and with each character associate a 3×3 transformation matrix. In general, this makes determining which characters to check at every point more difficult but a reasonable assumption with handwriting is the characters are roughly evenly-spaced. Thus, one could split up the image into a rectangular grid and each time a transformation is set, keep track of which boxes that character fits inside. This would be a good starting point to make future changes easier. This method is known as a grid acceleration structure and described in Chapter 4 of [2].

Another generalization would be to join multiple Bézier curves. In characters such as "z," there are sharp curves but the letter itself is still ultimately one stroke. This requires continuity but not smoothness of the first derivative. Some characters require first derivative continuity such as the character "s." This should also be supported. Lastly, we would also like T to be continuous across an entire stroke instead of just one Bézier curve. This models one "stroke" as multiple Bézier curves and represents one unbroken line drawn by some writing utensil.

4.2 Adding Tunable Parameters

Other methods exist for generating human-looking handwriting from a character sequence, such as the company Bond (`http://bond.co`) and generation of handwriting using Recurrent Neural Networks ([1]). There has been some work done procedurally generating handwriting (`http://bwiklund.github.io/roboglyphics/`). However, there is little apart from designing an entire typeface which can allow for a completely automated, arbitrarily precise script system. The ultimate goal is to provide artists with a tool with many tunable parameters to give a large space of possible handwriting styles. Whether these are to emulate an alien language or human language is undetermined.

To accomplish this, more parameters should be implemented. The first would be the ability to connect characters together. This could better emulate cursive writing. For example, one could specify the mean, variance, minimum, and maximum word lengths to get a probability density of word lengths, which could in turn determine which characters to connect.

A "curviness" paramter would be useful in determining the appearance of the langauge. A value of zero would be completely straight lines, while increasing this would tend the middle control points p_1 and p_2 away from the line between p_0 and p_3.

Each handwritten character looks different. To add realism, a parameter could be specified which indicates how much each character is perturbed from its original specification. We wish to perturb the character such that it looks different but is still recognizable. This could consist of a weighted combination of the original character's control points with random control points. Other perturbations could be slight rotations, translations, shearing, and scaling. These could all be specified using the translation matrices described in Section 4.1.

4.3 Writing Utensil Modeling

For a fountain pen, the thickness of the stroke depends on the minimum and maximum widths of the nib, the angle of the nib to the paper, and tangent of the stroke with the angle of the nib. For brushes, the amount of ink on the bristles and the force with which the brush is pressed also have an effect. These could be modeled by a given T function. Noise could also be added so thickness would vary randomly along the curve.

4.4 Rendering Speed Improvements

This method relies on finding solutions to the polynomial (1). This gives arbitrary precision when sampling, but requires solving a fifth order polynomial at each pixel. This takes about 600 seconds to render a 512×512 image with an average of 4 Bézier curves per character. However, since we have the implicit equation for B we may sample B at arbitrary points to "draw" the curve much more efficiently; it is inexpensive to compute $B(t)$ and a normal to B. These provide enough information along with the value of T to fill in the area around a given curve. Furthermore, this gives the benefit of providing the drawing of the curve as it would actually be drawn by hand. One could specify some increasing sequence of points $\{t_i\}_{i=1}^{N}$ where $t_i \in [0, 1]$, and the line between the two borders could be drawn for each $B(t_i)$. This could also provide an interesting effect by serializing the drawing so the process of writing could be simulated.

Another way the rendering speed could be increased is with OpenGL. A uniform buffer could be used to store every Bézier curve. Another uniform buffer could be used to store the grid acceleration structure. The fragment shader would get these buffers as well as the texture coordinate and could check a limited number of curves to determine what to display.

4.5 Custom Characters

While this process would generate random characters, an artist may desire a more direct control over the look of the characters. This would also allow specifying existing languages. Then, this process could create random sequences of characters and perturb the characters in the methods described above.

5 Other Graphics

Fig. 5 was generated using parameters

```
Handwriting(scale=4, brush=lambda t: 0.06, min_s=2, max_s=4, seed=5,
            h_border=0.1, v_border=0.35, h_stack=0.45, v_stack=0.4)
```

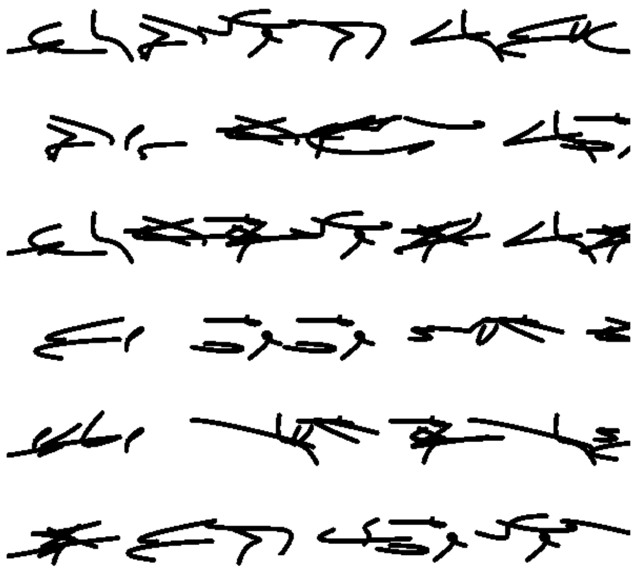
Figure 5: Different parameters to the `Handwriting` class

References

[1] Graves, A. Generating sequences with recurrent neural networks. *arXiv.org* (2013).

[2] Pharr, M., and Humphreys, G. *Physically Based Rendering: From Theory To Implementation*, 2nd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, 2010.