# *SketchSim*: Interactive Simulation of Ink Brushstrokes

Nicole Sultanum and Silvio B. Melo (orientador)
*Centro de Informática - UFPE*

## Abstract

*There is an increasing demand for new computational solutions in art and design, for digital illustrations, games, animation, among others. In order to address this need, several ideas have been developed in the artistic field, and one of them is concerned with the simulation of real artistic instruments, such as pencil, watercolor, etc. The simulation of ink and brushes, one of these instruments, is the object of study of this paper. Several techniques are discussed and implemented in an interactive artistic tool, called* SketchSim. *Artists have used this tool as part of a brief qualitative analysis, which is also discussed in this paper.*

## 1. Introduction

The computational simulation of artistic instruments (such as pencil, charcoal, pastel, watercolor, among others) is a steadily growing field in NPR [2] (or Non-Photorealistic Rendering), in face of the current needs for more sophisticated and intuitive tools for the creation of drawings. In digital illustration, the simulation of such instruments is sought by artists who wish to ease or improve their creative experience through computational solutions that offer new features, such as making slight adjustments, undoing strokes or trying out numerous colors. These functionalities are among those which, most of the times, may only be seized through digital means.

Nevertheless, the effective digital simulation of a real artistic tool is usually followed by great challenges. Here, we define "effective" solution as one which goes beyond the proper simulation of the real tool, and also:

(1) Provides intuitive means for the creation of artistic strokes;

(2) Offers resources for later modifications on the strokes, as part of the complementary experience the computational environment should provide;

(3) Renders fast enough for interactive use.

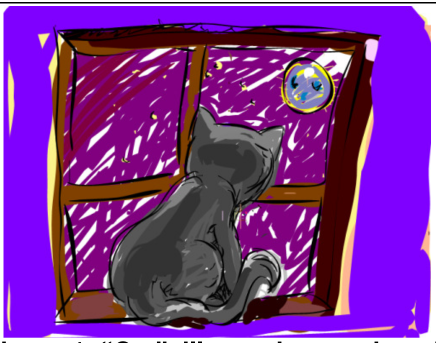

**Figure 1. "Cat", illustration produced on *SketchSim*[1]**

According to this scenario, this paper joins several techniques for the simulation and post-edition of ink brushstrokes, applied in the implementation of an interactive drawing tool called SketchSim.

### 1.1. Related Work

Many strategies have been proposed, concerning the simulation of artistic brushstrokes. Strassmann [6] presents a physically based approach for rendering strokes in *sumi-e* style, which produces realistic results, but requires too much user intervention and is too slow for interactive use. Skeletal Strokes [3] are also inappropriate for our purposes since there is a constant dependency of an image in order to reproduce different styles and is also too computationally intensive (since it is based on texture mapping strategies), not mentioning its weaknesses on resizing operations (as pointed by Su et. al. [7]). Su et. al. [7] present techniques for the creation of analytical strokes with variable width, having the disadvantage of requiring the explicit specification of control knots. Pudet [4] has also created analytical strokes with variable width, which are mapped to pressure levels acquired from a tablet device. These works usually spend little or no effort on fulfilling all of the additional requirements mentioned in the Introduction (intuitiveness, flexibility and efficiency).

[1] By Zózimo Neto.

## 2. Simulation of ink brushstrokes

Our stroke rendering process was heavily inspired on the work of Schneider [5] and Pudet [4] . A tablet is used as primary input device, since it provides pressure data, and a more intuitive interaction.

This process is composed of two main steps. The first one, the curve fitting step, creates a vector description for the user trajectory, specified through the tablet. In *SketchSim*, such as in [5], piecewise cubic Bézier curves with $G^1$ continuity were chosen to analytically represent the trajectories. The second one, the border fitting step, is responsible for simulating the appearance of an artistic stroke surrounding the vector trajectory which is resulted from the first step. Both of these steps are described on the next sections.

### 2.1. Curve Fitting

This step concerns the creation of an analytical description for the user trajectory. Initially, this trajectory is represented by a sequence of two-dimensional points (plus the corresponding pressure value, which is used in the border fitting step) provided by the tablet device, each of them captured in regular intervals of 10*ms*. This sequence goes through a few pre-processing steps (inspired by Schneider [5]), and is subsequently submitted to a least squares algorithm.

The trajectory points firstly undergo a pre-reduction step, for the high sampling frequency of the tablet device usually produces redundant point sets: points which are closer than 5 pixels are ignored. Discontinuities are also checked, represented by pointy corners, which are detected by the existence of angles sharper than 140° in the trajectory. Schneider also proposes another two pre-processing steps, which were eventually discarded. One of them, the noise removal step, was contributing negatively for the curve fitting process (Pudet [4] has also observed this phenomenon). The other step, the linear splines reduction, was also not used: it was proposed as an ultimate reduction of the sampling set, but the author himself mentioned that it could be discarded if the rendering process were to work fast enough without it (which has happened, indeed).

After pre-processing, the curve fitting process may start. Schneider [5] describes it in detail. In short, it consists of several iterations of (1) fitting of a curve and (2) evaluation of the resulting curve. The stage (2) measures the quality of the fit of the curve produced in (1), by checking the distance from the sampling points to the curve. The Newton-Raphson method is used in order to find the point on the curve which lies closest to each sampling point. If the fit doesn't reach an acceptable closeness after a predefined number of iterations, then the points sequence is subdivided on the point of greatest error, and the curve fitting procedure restarts for each of the new sequences. It results in the creation of several cubics to represent a trajectory, as illustrated by the rightmost image on Figure 2.

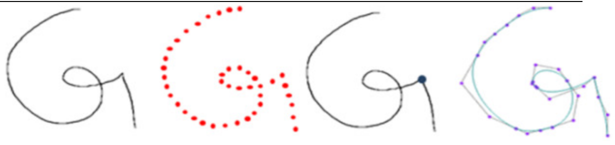
**Figure 2. Steps of the curve fitting algorithm**

### 2.2. Border Fitting

The border fitting step starts right after the curve fitting step. The technique described in this section is based in the work of Pudet[4], and may be considered as a simpler version of his border fitting method, specialized in circular brushes.

The tablet provides, for each sampling point, a value corresponding to the pressure applied on that location. The curve fitting step also computes the corresponding parametric value $t_i$ of each sampling point in the trajectory. With these data at hand, it is possible to associate pressure data directly into the analytical curve.

Take $press_i$ as the corresponding normalized pressure value of the sampling point $d_i$, and *el* as an elasticity value which limits the maximum width a stroke may have. The normalized tangent for each parametric position of samples $d_i$ (let's say, $Q(t_i)$) is computed through the De Casteljau algorithm. Then, in the direction perpendicular to the tangent at $Q(t_i)$, two points are located in this trail. These two points have distance to $Q(t_i)$ equal to $el \times press_i$, and represent points on each of the right and left borders. An example is illustrated in Figure 3.

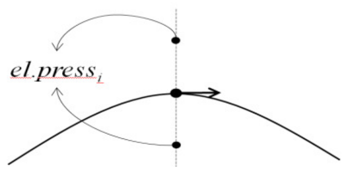

**Figure 3. Border Points**

This procedure is repeated for all samples $d_i$, resulting in two sequences: the left and right border points. Finally, these sequences are vectorized by the same curve fitting algorithm described in section 2.1.

The final result consists of two independent piecewise Bézier cubics, one for each border side.

Figure 4 shows, in the middle, the border points generated after the analytical trajectory on the left is processed. On the right side, one can see the final stroke, after applying curve fitting to the borders.

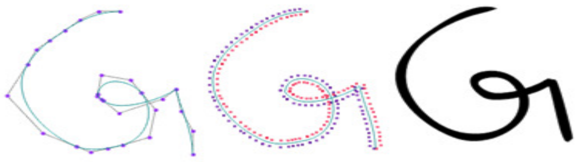
**Figure 4. Border Fitting steps**

## 3. Stroke Edition

Bartels and Beatty [1] presented a simple technique for modifying Bézier curves based on the displacement of any point in the curve, only adjusting its control points. This strategy is very convenient since it allows the implementation of drag-and-drop-based adjustments. The ability to choose which control points should be moved is another advantageous characteristic of this technique. Meanwhile, only the adjustment of isolated Bézier curves (and B-splines, which are not relevant to us) were analyzed in [1]. In this work a novel mechanism was required; one that could maintain G$^1$ continuity between adjacent segments after adjustment of any cubic in the analytic trajectory.

Nevertheless, before adjusting the trajectory, it is required to detect which of its cubics must be modified, and also to identify which point of this curve should be displaced (more specifically, the parametric *t* value which represents a point in a Bézier curve), taking the user specified start point $p_s = (x,y)$ as the sole input (the beginning of the drag-and-drop operation). The problem is then reduced to the computation of the distance between a point $p_s$ and a Bézier curve. We used a technique based on subdivision, which divides the curve into smaller pieces until they are close enough to line segments. When such condition is reached, the projection of the point *p* is computed on the line segment defined by the endpoints of this sub curve. If the projection is placed between the endpoints, an interpolation is made between the endpoint parameter values (which are known) to estimate the *t* parameter of the projection. Several candidate points on the curve may be found through this process. We choose the one which has the smallest squared distance from $p_s$.

After determining *t*, it is possible to perform the displacement of the control points. Let's define that an arbitrary trajectory is composed of *n* adjacent cubics $c_0$ to $c_{n-1}$. Supposing that $c_k$, $0 < k < n-1$, was identified as the cubic to be displaced, then its control points $b_{0\_k}$, $b_{1\_k}$, $b_{2\_k}$ and $b_{3\_k}$ will be modified to the new points $\hat{b}_{0\_k}$, $\hat{b}_{1\_k}$, $\hat{b}_{2\_k}$ and $\hat{b}_{3\_k}$, such that the corresponding curve $\hat{c}_k$ passes through the new point $p_e$ (the endpoint of the drag-and-drop operation). After that, the curves $c_{k-1}$ and $c_{k+1}$ must be readjusted, in order to assure that the points $\hat{b}_{2\_k}$, $\hat{b}_{3\_k}$ (which is equal to $\hat{b}_{0\_k+1}$) and $\hat{b}_{1\_k+1}$ are collinear (as shown in Figure 5). The same condition must be verified on the points $\hat{b}_{1\_k}$, $\hat{b}_{0\_k}$ (equal to $\hat{b}_{3\_k-1}$) and $\hat{b}_{2\_k-1}$.

Let's rename the points $b_{2\_k}$, $b_{3\_k}$ e $b_{1\_k+1}$ to *a*, *b* and *c*, as illustrated on Figure 5. After the adjustment of the curve $c_k$, *a* and *b* are modified to $\hat{a}$ and $\hat{b}$, respectively. The point $\hat{c}$ is defined as a point lying on the line defined by $\hat{a}$ and $\hat{b}$ which keeps the same proportion between the segments $ab/bc$ (composed by the earlier points *a*, *b* and *c*) and $\hat{a}\hat{b}/\hat{b}\hat{c}$.

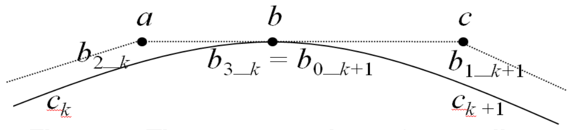

**Figure 5. The control points of two adjacent curves in a trajectory**

Not always are all the four control points adjusted on each modification. As a matter of fact, for the first cubic on a trajectory, its control point $b_{0\_0}$ is locked, and only $b_{1\_0}$, $b_{2\_0}$, and $b_{3\_0}$ are displaced. This behavior was adopted since it was taken as more intuitive and predictable.

Nonetheless, if the user tries to displace a point close enough to $b_{0\_0}$, it is expected that one wishes to modify the endpoint of the curve, and then $b_{0\_0}$ is also moved (as suggested by [1]).

This whole adjustment process is applied to the *core trajectory* of a stroke (the zero-width curves generated after the curve fitting step). After this edition process, the borders must be recalculated for the new trajectory, as described in the Section 2.2.

## 4. Results and Discussion

As discussed in the introduction of this paper, this work was focused on the creation of an ink brushstrokes simulator which, beyond properly renderizing the strokes, could also satisfy other requirements: provide efficiency, complementary resources and intuitive means of user interaction.

The form of interaction is quite simple and straightforward, since the artist may work with a tablet device almost as if it was a regular pen (which is a very familiar tool). During the drawing of a stroke, an approximate outline of the final result is presented to the user. When it is finished, the trajectory is computed and fully renderized. Experiments have shown that the whole process operates quickly enough, allowing its use on interactive systems.

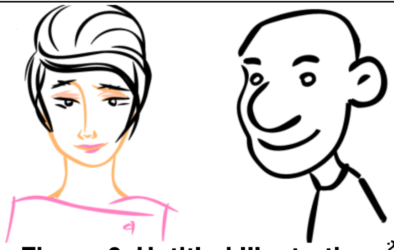
Figure 6. Untitled Illustrations[2]

In order to evaluate the user interaction and the quality of the strokes, *SketchSim* was used and analyzed by eight end users, including designers and digital artists. Figures 1, 6 and 7 represent some of the resulting illustrations. They evaluated positively the tool, particularly emphasizing the intuitiveness of the stroke creation process in comparison to the quality of the strokes created. They also observed that the generated curves were coherent and adherent to what they were trying to draw.

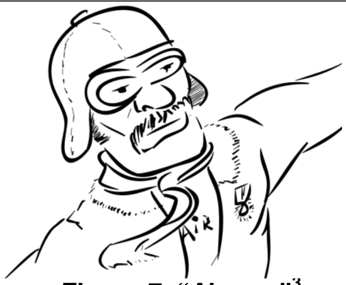

Figure 7. "Airman"[3]

This study has also encountered some problems concerning the stroke rendering process, consequence of the instability of the Newton-Raphson method. This technique, used in the curve fitting algorithm, eventually computes inadequate values which result in anomalous strokes.

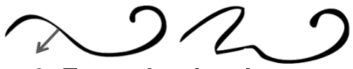
Figure 8. Example of an inappropriately local adjustment

A small issue was also observed on the stroke edition technique, due to the fact that the maintenance of continuity considers only the immediately adjacent segments. When the cubics generated by the curve fitting algorithm are too short, the edition effects are also reflected in a very small area of the stroke, resulting in an uneven appearance, as illustrated in Figure 8.

[2] By Frederico de Melo (left) and Antunes Neto (right).
[3] By Fradique Filho.

## 5. Conclusions and Future Work

An artistic ink brushstroke simulator was developed, which uses pressure data provided by a tablet device to represent width variations on a stroke. The strokes are represented by analytical curves which provide flexibility on further readjustments while maintaining resolution quality. A stroke edition technique was developed, based on intuitive 'drag and drop' operations. The simulator was also evaluated by artists, validating its artistic potential.

One possible future work that could increase this potential is the simulation of ink dilution effects (such as in [7]). Another idea would be to explore more advanced types of curve modification beyond position readjustment. Width edition along the strokes, for example, would be quite helpful for the artists. Another possible example of stroke edition is the redraw, as proposed by Schneider [5] for zero width curves, but in this case considering the continuity aspects of the borders on the endpoints of each edited portion of a curve.

## 6. References

[1] R. H Bartels and J. C Beatty, "A Technique for the Direct Manipulation of Spline Curves". *Proceedings of Graphics Interface '89*, p.33-39, 1989.

[2] B. Gooch and A. Gooch, *Non-Photorealistic Rendering*, A. K. Peters, 2001.

[3] S. C. Hsu and I. H. H. Lee, "Drawing and animation using skeletal strokes". *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM Press, pp. 109-118, 1994.

[4] T. Pudet, "Real Time Fitting of Hand-Sketched Pressure Brushstrokes". *Computer Graphics Forum*, 13(3), pp. 205-220, 1994.

[5] P. J. Schneider, *Phoenix: An interactive curve design system based on the automatic fitting of hand-sketched curves*, Master's Thesis, University of Washington, 1988.

[6] S. Strassmann, "Hairy Brushes", *Computer Graphics (SIGGRAPH '86 Proceedings)*, 20(4), pp. 225-232, 1986.

[7] S.L Su et. al., "Simulating Artistic Brushstrokes Using Interval Splines". *The 5th International Conference on Computer Graphics and Imaging*, pp. 85—90, 2002.