

UNIVERSIDADE FEDERAL DE PERNAMBUCO  
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO  
CENTRO DE INFORMÁTICA

---

# **Simulação interativa de traços em tinta e pincel através do *tablet***

---

TRABALHO DE GRADUAÇÃO

Julho de 2008

# Simulação interativa de traços em tinta e pincel através do *tablet*

por

Nicole Barbosa Sultanum

Monografia apresentada ao Centro de Informática  
da Universidade Federal de Pernambuco como  
requisito parcial para obtenção do Grau de  
Bacharel em Ciência da Computação.

**Orientador:** Silvio de Barros Melo

Julho de 2008



*"In mathematics the complicated things are reduced to  
simple things. So it is in painting."  
(Thomas Eakins)*

## Agradecimentos

Agradeço aos meus familiares, pelo zelo e compreensão durante estes quatro anos e meio de muito trabalho e noites mal dormidas, em particular, nesses últimos meses de conclusão de curso e realização deste Trabalho de Graduação. Agradeço aos colegas (e amigos) Victor, Eduardo e Marconi, por terem ouvido e compartilhado as constantes queixas de que não iríamos conseguir terminar nossos TGs, além das trocas de experiências e sugestões (dadas e recebidas). Obrigada a Lucas Marinho: sem seu (excelente) tablet, nada disso seria possível. Obrigada a Silvio Melo, pela paciência, dedicação e clareza que despendeu desde o início da realização deste trabalho. Obrigada a Marcelo Walter, pelas dicas preciosas que ofereceu ao nortear e incrementar este estudo, causando profundo impacto no resultado desta pesquisa. Agradecimentos especiais também ao pessoal da Positivo, que proporcionaram agradáveis momentos de descontração nos raros encontros no Centro de Informática. E finalmente, muito obrigada a todos que participaram da avaliação do SketchSim, que produziu resultados tão incríveis: a Rebeka Maia e todos da Solver; a Lucas Menge; a Guilherme Dantas; aos designers Rafael Efrem, Antunes e Arthur; e aos excelentes artistas Fradique, Frederico, Zózimo, Ricardo e Leo Menezes.

Muito obrigada!

## Resumo

A simulação computacional de instrumentos reais de arte (como lápis, carvão, pastel, aquarela, entre outros) é uma área da Computação Gráfica que cresce constantemente, em face das necessidades atuais de ferramentas mais intuitivas e sofisticadas para criação de ilustrações. Uma aceitável reprodução digital destas ferramentas também apresenta desafios para sua criação. De frente com esta complexidade, este trabalho tem como proposta realizar um estudo sobre simulação de traços com tinta e pincéis, modalidade de arte bastante comum em ilustrações com estilo *cartum*.

Nesta direção, foi proposta uma abordagem interativa que permite a criação de traços artísticos com *tablets*, em formato analítico com espessura variável. Também é apresentada uma técnica para modificação facilitada dos traços após a criação, mantendo a espessura coerentemente após a edição. É realizada uma análise qualitativa da ferramenta criada para implementação das estratégias propostas.

**Palavras chave:** Mínimos Quadrados, *Tablet*, Simulação Artística, Vetorização, Tinta e Pincel, Traços de Pincel, Edição de Curvas de Bézier.

## Abstract

The computational simulation of real artistic tools (such as pencil, charcoal, pastel, watercolor, among others) is a steadily developing field in Computer Graphics, in face of modern needs for simpler and more powerful mechanisms for Digital Illustration. An acceptable computer simulation for those artistic tools usually faces great challenges for its implementation. Coping with this complexity, this work presents a study on simulations of ink brushstrokes, a very common artistic resource among cartoon illustrators, for instance.

In this line of work an interactive approach was proposed, which enables the creation of artistic strokes through tablet devices, keeping an analytical representation of its trajectories and variable and allowing width variability within each stroke. A technique for intuitive stroke editing is also presented, which maintains coherently the initial pressure distribution of the stroke even after modification. Finally, a qualitative analysis is made on the software created to implement the proposed strategies.

**Palavras chave:** Least Squares, Tablet, Artistic Simulation, Vectorization, Brush and Ink, Brushstrokes. Edition of Bézier Curves.

# Sumário

1.	Introdução .....	10
2.	Simulação Digital de Traços em Tinta .....	12
2.1	Abordagens <i>Raster</i> .....	12
2.1.1	<i>Hairy Brushes</i> .....	12
2.1.2	<i>Skeletal Strokes</i> .....	15
2.2	Abordagens Vetoriais .....	16
2.2.1	<i>Plass e Stone</i> .....	19
2.2.2	<i>Schneider</i> .....	24
2.2.3	<i>Thierry Pudet</i> .....	28
3.	SketchSim .....	31
3.1	Pré processamento .....	32
3.2	Encaixe de curvas .....	32
3.3	Geração de Bordas .....	34
3.4	Edição de curvas após criação.....	35
4.	Avaliação e Resultados.....	40
4.1	Avaliação de Eficiência .....	40
4.2	Justificativa para eliminação da etapa de redução de ruído .....	41
4.3	Anomalias ocasionais na geração de traços.....	42
4.4	Considerações sobre edição de curvas .....	45
4.5	Avaliação qualitativa junto a potenciais usuários.....	48
5.	Considerações finais.....	53
	Referências.....	55

## Lista de figuras

Figura 1.	Algumas ilustrações com pincel (ou caneta) e nanquim.....	10
Figura 2.	Exemplo de ilustração em estilo <i>sumi-e</i> .....	13
Figura 3.	Exemplo de variação de <i>Dip</i> .....	14
Figura 4.	“ <i>Shrimp and Leaf</i> ”, ilustração produzida digitalmente .....	14
Figura 5.	Elementos de um <i>skeletal stroke</i> , e resultado obtido.....	15
Figura 6.	Encaixe de uma cúbica a uma seqüência de pontos.....	16
Figura 7.	Representação de uma cúbica, seus pontos de controle e vetores tangentes. ....	18
Figura 8.	Resultados intermediários do método de Plass e Stone.....	19
Figura 9.	Uma curva à mão livre e uma aproximação por <i>spline linear</i> , com vértices destacados	20
Figura 10.	Exemplo de um encaixe de curva.....	22
Figura 11.	Distância do ponto $d_i$ à curva $Q$ .....	23
Figura 12.	Modificações em uma curva, no <i>Phoenix</i> .....	25
Figura 13.	Algoritmo para identificar se uma única cúbica pode ser encaixada a uma seqüência de pontos .....	27
Figura 14.	Etapas do algoritmo de encaixe de curva proposto por Pudet.....	29
Figura 15.	Exemplo de traço produzido pelo método de Pudet. ....	29
Figura 16.	Envelope do pincel $b$ , elíptico, ao longo da trajetória $s$ .....	29
Figura 17.	SketchSim .....	31
Figura 18.	Trajetória com dois pontos de descontinuidade. ....	33
Figura 19.	Obtenção de pontos das bordas .....	35
Figura 20.	Pontos discretos e resultado do encaixe de curvas para um traço de exemplo.	35
Figura 21.	Exemplo de uma possível modificação de uma curva, reajustando todos os pontos de controle.....	36
Figura 22.	Exemplo de reajuste, mantendo extremidades fixas.....	36
Figura 23.	Exemplo de deslocamento para segmentos conectados.....	37
Figura 24.	Duas curvas de Bézier conectadas pela condição $G^1$ . ....	37
Figura 25.	Projeção do ponto $p$ se estiver contida dentro (a) e fora (b) do segmento $b_0b_3$	39
Figura 26.	Estimativa do parâmetro $t_{min}$ .....	39
Figura 27.	Diferença entre curva com redução de ruído (linha tracejada) e sem redução de ruído (linha contínua).....	42
Figura 28.	Perda de detalhes da curvatura após etapa de redução de ruído. ....	42
Figura 29.	Resultados obtidos com número máximo de reparametrizações igual a 10.....	43
Figura 30.	Resultados obtidos com número máximo de reparametrizações igual a 5.....	43

Figura 31.	Mesmo traço com diferentes iterações máximas de reparametrização (10 e 5, respectivamente), e o resultado do algoritmo de encaixe de curvas.....	44
Figura 32.	Incoerência na geração de uma cúbica .....	44
Figura 33.	Ilustração antes e depois de edição de curvas. Setas indicam áreas editadas. ...	45
Figura 34.	Modificações acentuadas em curvas .....	46
Figura 35.	Passos para modificação do ombro .....	46
Figura 36.	Etapas de edição em um traço com segmento cúbico curto .....	47
Figura 37.	Etapas de edição em um traço com segmentos curtos e proeminências.....	47
Figura 38.	Problema encontrado na edição de curvas.....	48
Figura 39.	Exemplo de traço com acabamento destacado. ....	49
Figura 40.	‘Aviador’, por Fradique Filho.....	50
Figura 41.	‘Mão e Rascunho’, por Nicole Sultanum .....	50
Figura 42.	Ilustração sem título, por Frederico de Melo.....	51
Figura 43.	‘Gata’, por Zózimo Neto. ....	51
Figura 44.	‘Ilha de Lost’, por Leonardo Menezes. ....	52
Figura 45.	Ilustração sem título, por Ricardo Teixeira .....	52

## 1. Introdução

Existe uma ávida e renovável demanda por novas soluções computacionais de arte e design para uso em publicidade, ilustração digital, games, animação e indústria cinematográfica, bem como um potencial criativo quase ilimitado por parte dos profissionais inseridos nessas áreas. Para suprir esta necessidade há propostas e aplicações para diversos fins, e uma das vertentes trata da simulação digital de meios artísticos reais, como lápis, carvão, aquarela, pintura a óleo, entre outros.

A modalidade de arte a qual este trabalho está direcionado é a pintura com tinta e pincéis. A utilização deste meio artístico é particularmente comum em criação de quadrinhos de estilo cartum, como tirinhas e *mangás*<sup>1</sup>. E o principal objetivo deste estudo é oferecer uma abordagem para simulação dessas ferramentas.



Figura 1. Algumas ilustrações com pincel (ou caneta) e nanquim

Na área de ilustração digital, a simulação de instrumentos reais é buscada por artistas que desejam facilitar ou incrementar a experiência de criação através de soluções computacionais que possam oferecer novas possibilidades. Para uma ilustração real em

---

<sup>1</sup> Quadrinhos japoneses.

aquarela, por exemplo, erros são irreversíveis na grande maioria dos casos. Um artista requer anos de treino para tornar-se competente nesta técnica. Logo, a possibilidade de realizar correções como experimentar várias cores e pincéis, desfazer parte do desenho ou efetuar pequenos ajustes tornam a simulação uma alternativa poderosa para artistas, além de permitir acesso à arte para indivíduos menos habilidosos. Todavia, a funcionalidade principal – uma simulação eficaz para aquarela – oferece grandes desafios para os pesquisadores em Computação Gráfica.

A complexidade não se limita à reprodução digital de aquarela. Também existem obstáculos para a técnica abordada neste estudo, a pintura com tinta e pincéis. E eles não residem somente na correta criação de traços: é preciso oferecer uma experiência de interação intuitiva e coerente, de forma a favorecer artistas experientes e evitando a necessidade de “reaprendizado” para utilizar o software. Da mesma forma, é importante facilitar a utilização por usuários inexperientes, e melhorar seu trabalho na medida do possível.

Com estas orientações em mente, este estudo apresenta uma abordagem desenvolvida para simulação interativa de traços artísticos em pincel e tinta. A interação prevista foi a mais intuitiva possível, através do desenho direto com um *tablet* (ou mesa digitalizadora). Os traços são identificados e convertidos para formato vetorial com *splines* de curvas de Bézier, facilitando a implementação de posteriores transformações (como rotação, translação, e redimensionamento) que fazem parte da experiência complementar que somente ambientes computacionais podem oferecer. Dentre as possíveis modificações permitidas pela representação vetorial, uma delas foi estudada e implementada: a realização de pequenos ajustes em traços, através de interação *drag & drop*. Este trabalho ainda apresenta uma avaliação qualitativa da ferramenta **SketchSim**, *software* criado para implementar as técnicas propostas nesta pesquisa.

No capítulo 2 é feita uma revisão de trabalhos relacionados, e em particular, o detalhamento profundo de três pesquisas que influenciaram fortemente este estudo. No capítulo 3 é descrita a abordagem proposta neste relatório através do funcionamento do *SketchSim*. No capítulo 4 são analisados os resultados obtidos com a ferramenta, bem como a avaliação do software junto a artistas. No capítulo 5 encontram-se as conclusões do trabalho, e são discutidas possibilidade de trabalhos futuros.

## 2. Simulação Digital de Traços em Tinta

Existem inúmeros estudos que procuram simular o traçado de diferentes formatos de pintura com pincel. Cinco deles são descritos aqui, de acordo com a relevância para esta revisão de contexto. Os dois artigos descritos na seção 2.1 “Abordagens *Raster*” são dignos de menção pela sua importância na área, e para ilustrar ao leitor que tipos de estratégias variadas existem. No entanto, este trabalho baseia-se fortemente nos estudos contidos na seção 2.2 “Abordagens Vetoriais”, cujo detalhamento é bem mais cuidadoso.

### 2.1 Abordagens *Raster*

Os dois trabalhos mostrados aqui propõem diferentes estratégias de simulação de traços, que geram como saída informação rasterizada: ou seja, *pixels*. Apesar de não ser o formato ideal para os objetivos deste estudo, é importante citá-los para trazer à luz diferentes perspectivas em relação à simulação de traços, bem como justificar a não-utilização destas técnicas.

O primeiro deles é comumente conhecido como “*Hairy Brushes*” [Str86], e contém a descrição de uma simulação física bastante detalhista para desenho com pincéis espessos. O segundo é apelidado de “*Skeletal Strokes*” ([HL94] e [HLW93]), e descreve uma técnica para a criação genérica de traços de quaisquer estilos (inclusive o de pincéis e nanquim). Ambos são descritos nas subseções a seguir.

#### 2.1.1 *Hairy Brushes*

Strassmann [Str86] neste estudo busca simular o efeito de pincéis espessos e da distribuição de tinta no papel de forma realista. Em particular, ele procura criar meios para representar ilustrações em estilo *sumi-e*, uma modalidade oriental de desenho bastante similar à aquarela. Ilustrações *sumi-e* são caracterizadas por traços escassos e expressivos, possuindo diversas concentrações de tinta. Devido à sua natureza minimalista, cada detalhe é de extrema importância. O estilo também é classificado por Susan Frame [Hai98] como “impiedoso”, pois o material utilizado na técnica não oferece margem para correções, tornando o *sumi-e* um estilo difícil, que exige anos de treino.



Figura 2. Exemplo de ilustração em estilo *sumi-e*

Strassman modelou o problema identificando quatro elementos independentes: Pincel, Traço, *Dip* (que pode ser mais bem traduzido como a *distribuição de tinta nas cerdas de um pincel*) e Papel. A implementação foi feita através de uma linguagem orientada a objetos (LISP), que se mostrou bastante conveniente para tal estruturação. A seguir, os quatro elementos são brevemente descritos:

- Pincel (ou *Brush*, no original): pincéis são objetos compostos por um conjunto de *Cerdas*. Cada cerda ainda possui posição relativa no pincel, e certa quantidade de tinta (especificada inicialmente pelo *Dip*, e diminuindo ao longo da construção do traço).
- Traço: composto por um conjunto de parâmetros (posição e pressão). São utilizadas *splines* para compor a trajetória, cujos pontos de controle são especificados um a um pelo *mouse*, enquanto a pressão é especificada através do teclado.
- *Dip*: descreve a distribuição de tinta em um pincel, similar à idéia do mergulho de um pincel em um recipiente de tinta. Em outras palavras, representa o estado inicial de distribuição de tinta nas cerdas de um pincel. Esta representação tão detalhada é necessária, pois a quantidade de tinta nas cerdas não necessariamente é uniforme. Abaixo segue um exemplo ilustrativo do efeito da variação de *Dip*. O mesmo traço e pincel foram utilizados, mudando apenas a distribuição de tinta.

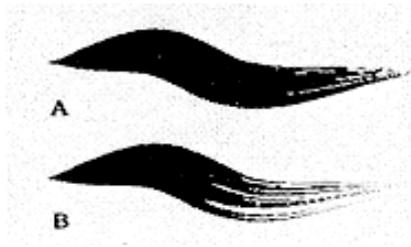


Figura 3. Exemplo de variação de *Dip*

- Papel: responsável por receber mensagens de cada cerda (informando como ela foi “pintada”) e renderizando da forma adequada: isto permite manipulações como mapeamento de texturas, ou a possibilidade de considerar fatores como graus de absorção do papel.

O funcionamento básico consiste em considerar o traçado como uma função variando ao longo do tempo (ou comprimento do traço: segundo Strassman, qualquer função monotônica crescente pode ser utilizada). Ao longo da trajetória, “posiciona-se” o pincel no instante  $t$ , estima-se a quantidade de tinta transferida ao papel naquele momento (considerando a pressão estimada naquele momento, a quantidade de tinta ainda presente em cada cerda, e propriedades do papel) e os pixels são renderizados. Ou seja, é uma aproximação de fenômenos físicos envolvidos na construção de um traço.

Os resultados são bastante satisfatórios visualmente (vide Figura 4 a seguir). Mas devido à complexidade do cenário e à simplicidade dos equipamentos de entrada utilizados (teclado e *mouse*), a construção de traços não era nada fácil. Chua e Winton [Chu88] propuseram melhoras na interação com o usuário, mas a interface ainda continuava pouco amigável. Parte disto deve-se também ao fato de a definição do caminho de um traço ser feita especificando pontos explícitos da spline, e não à mão livre.

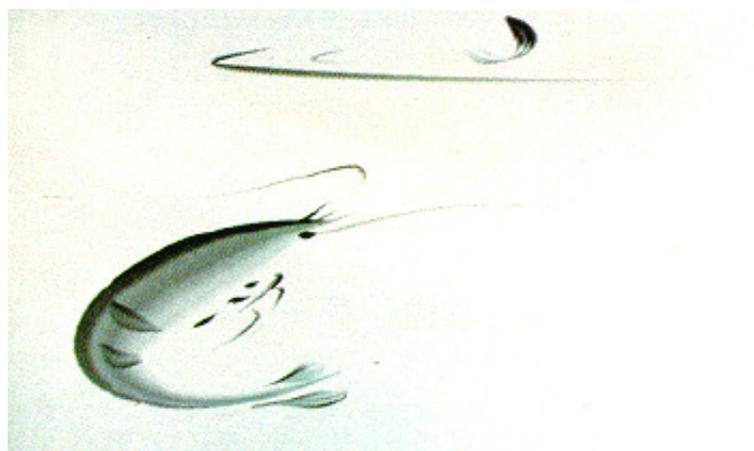


Figura 4. “*Shrimp and Leaf*”, ilustração produzida digitalmente

Outro viés da técnica diz respeito à eficiência. Como o cálculo dos traços é feito pixel a pixel, a renderização é lenta (costumava consumir 1 ou 2 minutos com o *hardware* disponível na época), inviabilizando seu uso para aplicações interativas. Isto também dificulta posteriores ajustes do traço.

Apesar das dificuldades, este é um trabalho de bastante relevância na simulação de traços artísticos, pois serviu de base e inspiração para inúmeras pesquisas. Podemos citar como exemplo o trabalho de Curtis et. al [Cur97], que realiza simulações bastante realistas de efeitos de aquarela, e Xu et. al. [Xu02], que modela o comportamento de pincéis para simulação de caligrafia de caracteres chineses. Diversas publicações também referenciam os ‘*pincéis espessos*’ de Strassmann. Só nesta monografia, temos [HLW93] [Pud94], [HL94], [DcF05] e [Su02] como exemplos.

### 2.1.2 *Skeletal Strokes*

Hsu, Lee e Wiseman publicaram em 1993 o primeiro artigo descrevendo o funcionamento dos *skeletal strokes* [HLW93]. Os autores propõem com isto uma nova forma de criar traços com estilos diversos. Um ano depois foi publicado mais um artigo relacionado [HL94], descrevendo uma ferramenta desenvolvida por Hsu e Lee para utilização de *skeletal strokes*.

Ao contrário da abordagem de Strassman, não são utilizadas abstrações para cerdas ou distribuição de tinta. A técnica consiste em *transformar uma imagem específica ao longo de uma trajetória*. Sobre uma imagem é definida uma linha, que representa o ‘esqueleto’ (ou *backbone*, no original). Ao fornecer uma trajetória qualquer, a imagem é sobreposta, de forma que seu esqueleto coincida com o caminho especificado. Para tanto, definidas operações de alongamento e achatamento, dobramento e rotação da imagem.

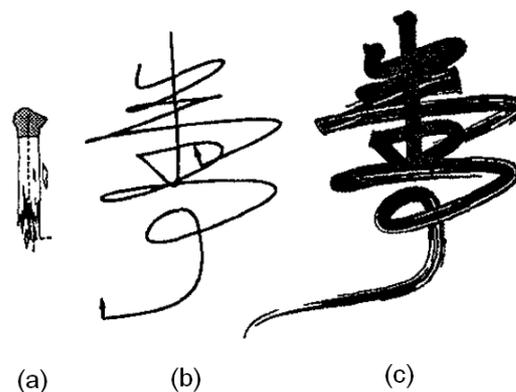


Figura 5. Elementos de um *skeletal stroke*, e resultado obtido.

- (a) Imagem de referência representando um traço de pincel, com seu respectivo *backbone*;
- (b) Trajetórias do desenho; (c) Resultado final.

Como se pode perceber, este tipo de abordagem é muito flexível, e proporciona grande liberdade artística. De fato, DeCarlo e Finkelstein [DcF05] afirmam que muitas funcionalidades de *skeletal strokes* estão presentes em pacotes padrão em programas comerciais como o *Adobe Illustrator*. No entanto, Su et.al. [Su02] citam limitações da técnica. Primeiro, há a questão da ineficiência. A cada mudança na trajetória, os pixels são totalmente remapeados da imagem, um processo computacionalmente intenso. Outra questão levantada é em relação à escalabilidade. Eles afirmam que é um desafio manter a aparência de um *skeletal stroke* para qualquer tamanho e resolução possíveis. Além disso, há sempre a dependência da existência de uma imagem para simular efeitos artísticos.

## 2.2 Abordagens Vetoriais

Há aqueles que buscam derivar uma forma analítica a partir de uma amostragem discreta seqüencial de pontos de uma curva. Os pontos em questão fazem parte de uma trajetória, a qual pode ser representada utilizando curvas de Bézier ou B-splines, por exemplo. Estas amostras podem ser extraídas por inúmeros meios, como através do traçado de um *tablet*, ou da coleta regular de pontos em uma curva rasterizada. Abaixo segue um exemplo retirado de [Sch88], que aproxima uma curva de Bézier a um conjunto de pontos de uma trajetória. Para obtenção de curvas que melhor se aproximem dos pontos fornecidos, uma das técnicas mais difundidas é utilização de *mínimos quadrados*.



Figura 6. Encaixe de uma cúbica a uma seqüência de pontos

Em termos matemáticos, podemos expressar o problema como a *procura de uma curva que minimize o somatório do quadrado das distâncias para os pontos de amostragem*. Plass e Stone [PS83] descrevem brevemente os fundamentos matemáticos desta técnica.

Considere os pontos  $P = \{(x_i, y_i), 1 \leq i \leq n\}$ . Deseja-se encontrar uma curva polinomial  $p(x_i)$  que minimize

$$\sum_{i=1}^n (p(x_i) - y_i)^2$$

Dado que  $p(x_i)$  é polinomial, podemos reescrevê-lo como

$$\sum_{j=0}^d a_j \cdot \varphi_j(x)$$

O objetivo então é encontrar o conjunto de valores  $A = \langle a_0, a_1, \dots, a_d \rangle$  tal que a função

$$S = \sum_{i=1}^n \left( \sum_{j=0}^d a_j \cdot \varphi_j(x_i) - y_i \right)^2$$

seja minimizada. Para tanto, são encontradas as derivadas de  $S$  em relação a cada um dos valores de  $A$ , e igualadas a zero. Obtém-se um conjunto de  $d+1$  equações lineares, na forma

$$\frac{\partial S}{\partial a_k} = \sum_{i=1}^n 2 \left( \sum_{j=0}^d a_j \cdot \varphi_j(x_i) - y_i \right) \cdot \varphi_k(x_i) = 0$$

e resultando em um sistema linear

$$\sum_{j=0}^d a_j \sum_{i=1}^n \varphi_j(x_i) \cdot \varphi_k(x_i) = \sum_{i=1}^n y_i \cdot \varphi_k(x_i), \quad 0 \leq k \leq d$$

Para curvas de Bézier, paramétricas, as estratégias são diferentes. Vamos supor que se deseja encontrar uma cúbica – com pontos de controle  $b_0, b_1, b_2$  e  $b_3$  – que melhor se encaixe no conjunto de pontos  $P$ . Os pontos  $b_0$  e  $b_3$  são os pontos extremos de  $P$ , ou seja,  $b_0 = (x_1, y_1)$  e  $b_3 = (x_n, y_n)$ . Como fazer então para fixar  $b_1$  e  $b_2$ ? Utiliza-se um artifício relacionado à forma de Hermite para cúbicas de Bézier. Defina  $p_1$  e  $p_2$  os pontos das extremidades (no nosso caso, eles são equivalentes a  $b_0$  e  $b_3$ ) e  $t_1$  e  $t_2$  como os vetores tangentes nos pontos  $p_1$  e  $p_2$ , respectivamente. Estes vetores tangentes definem retas pelas quais os componentes  $b_1$  e  $b_2$  podem “navegar”: basta multiplicar um certo escalar  $a_1$  e  $a_2$  a cada um dos vetores  $t_1$  e  $t_2$ , para modificar a posição de cada ponto  $b_1$  e  $b_2$ . A Figura 7 a seguir ilustra cada um destes elementos.

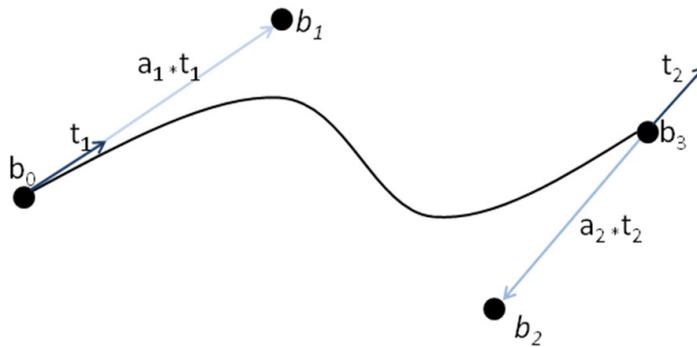


Figura 7. Representação de uma cúbica, seus pontos de controle e vetores tangentes.

Com isto em mãos, como formular o problema de mínimos quadrados? Sabe-se que os pontos  $b_0$  e  $b_3$  são fixos, restando apenas os pontos  $b_1$  e  $b_2$  como variantes. A partir de pontos de  $P$ , é possível estimar os vetores tangentes  $t_1$  e  $t_2$ . Logo, restam  $a_1$  e  $a_2$  como variáveis livres. O problema resume-se então em encontrar estes dois escalares, de forma a minimizar a distância entre a curva e os pontos de  $P$ . Esta abordagem orientada a tangentes é particularmente conveniente, principalmente quando são utilizadas *splines*, para representar trajetórias.

Segundo [Wiki], problemas de mínimos quadrados caem em duas categorias: lineares e não lineares. Para os lineares, há uma forma fechada para a solução, o que permite que ela seja encontrada em um passo. As não lineares, no entanto, exigem uma parametrização inicial e refinamento iterativo para obtenção de sucessivas aproximações. Ou seja, são necessárias várias repetições percorrendo as amostras, até atingir resultados satisfatórios.

Considerando que o conjunto de pontos usualmente contém informação redundante, é válido utilizar estratégias de pré-processamento para redução do conjunto inicial, de forma a diminuir a degradação no desempenho. Tratamento prévio no conjunto de amostragem também é realizado objetivando identificação de características e suavização de ruído.

Há inúmeros autores inspirados pela estratégia descrita, e cujo detalhamento de suas pesquisas não competem a este estudo. Caso seja de interesse do leitor, em [Sch88] é possível encontrar uma vasta revisão deles. Nesta monografia serão descritos três trabalhos, escolhidos sua influência no meio acadêmico e pela relevância de suas contribuições a esta pesquisa. O primeiro deles foi produzido por Plass e Stone [PS83] e possui como principal diferencial a utilização de programação dinâmica para encontrar pontos de junção ideais na curva traçada. O segundo trabalho, da autoria de Schneider [Sch88], consiste na descrição de um sistema interativo que permite a criação de traços e posterior modificação através de remoção e redefinição de trechos da trajetória. O terceiro

trabalho, desenvolvido por Pudet [Pud94], descreve otimizações no algoritmo de encaixe de curvas utilizado por [PS83] e [Sch88] e incorpora informações de pressão ao traço, produzindo resultados artisticamente interessantes. As subseções a seguir detalham cada um destes estudos.

### 2.2.1 *Plass e Stone*

A referência mais antiga encontrada que se encaixa nos moldes descritos foi o trabalho de autoria dos pesquisadores da Xerox PARC, Michael Plass e Maureen Stone [PS83], publicado em 1983. Buscando melhorar a qualidade de fontes tipográficas em documentos, eles criaram um método para extrair uma forma analítica que representasse caracteres rasterizados. Nesta aplicação em particular, são utilizadas cúbicas paramétricas conectadas com continuidade  $G^1$  (uma versão menos restritiva da continuidade  $C^1$ ). Uma representação analítica é altamente desejável, visto que viabiliza redimensionamento dos documentos sem perda de qualidade. A abordagem adotada por eles foi tão versátil que inspirou o trabalho de muitos outros.

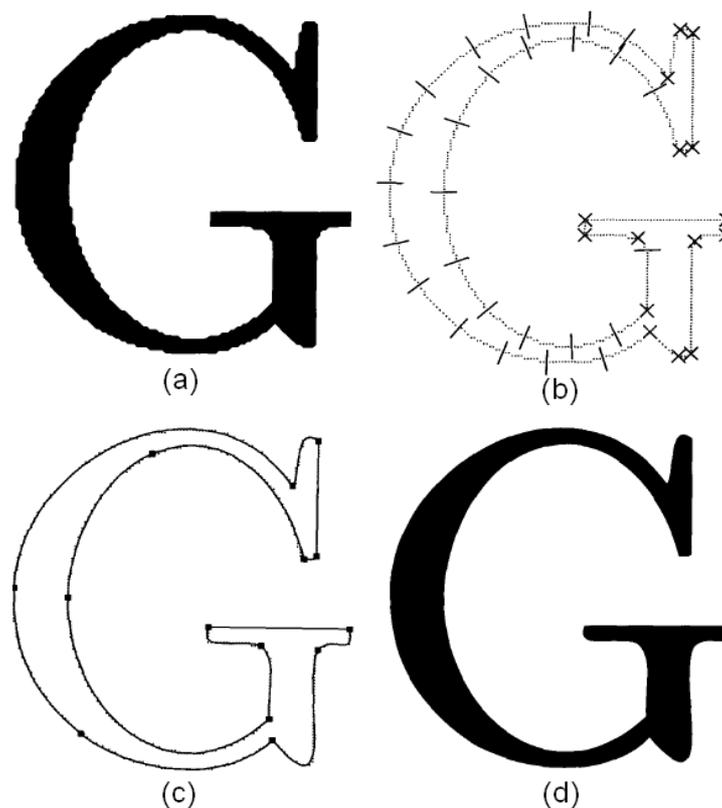


Figura 8. Resultados intermediários do método de Plass e Stone.

(a) Forma original em bitmap; (b) pontos de amostragem, proeminências (marcadas com um 'x') e nós potenciais (com suas respectivas normais); (c) resultado da identificação das curvas, com pontos de junção das splines evidenciados; (d) resultado final.

O processo resume-se em 3 grandes etapas: aquisição do conjunto de pontos de amostragem  $D = \{d_i = (x_i, y_i) \mid 1 < i \leq n\}$ , seleção e pré-processamento destes pontos e por fim, identificação de melhores segmentos através de programação dinâmica.

Primeiramente, é extraída uma seqüência de pontos de amostragem  $d_i$  para os quais se quer encontrar uma representação analítica (como em (b), na Figura 8). O meio de aquisição, segundo os autores, é de pouca relevância. Eles experimentaram curvas digitais rasterizadas, imagens escaneadas e rascunhos a mão livre, e concluíram que o que importa nestes casos é que espécie de efeitos particulares o conjunto de amostragem sofre com cada mecanismo de entrada. Eles citam exemplo de fatores de variância como remoção de ruído, proeminências bem ou mal evidenciadas, entre outros.

A seguir são escolhidos *nós potenciais*, ou seja, candidatos a extremidades dos segmentos de curva necessários para compor o desenho completo (como ilustrado em (c) na Figura 8). Proeminências são imediatamente marcadas como nós potenciais: elas são identificadas como vértices de ângulos mais agudos que  $135^\circ$ . Na Figura 8, as proeminências (ou *corners*, no original) foram marcadas em (b) com um 'x'. Os pontos entre estes nós candidatos são filtrados para reduzir os efeitos da discretização. Os outros nós potenciais são encontrados através da identificação de vértices de aproximações poligonais, também chamadas de *splines* lineares (como ilustrado na Figura 9). Existem técnicas que tornam a criação de tais aproximações uma tarefa simples e eficiente, capaz de selecionar com robustez pontos de maior representatividade no traço. Juntamente com estes pontos, são calculadas as suas respectivas tangentes.

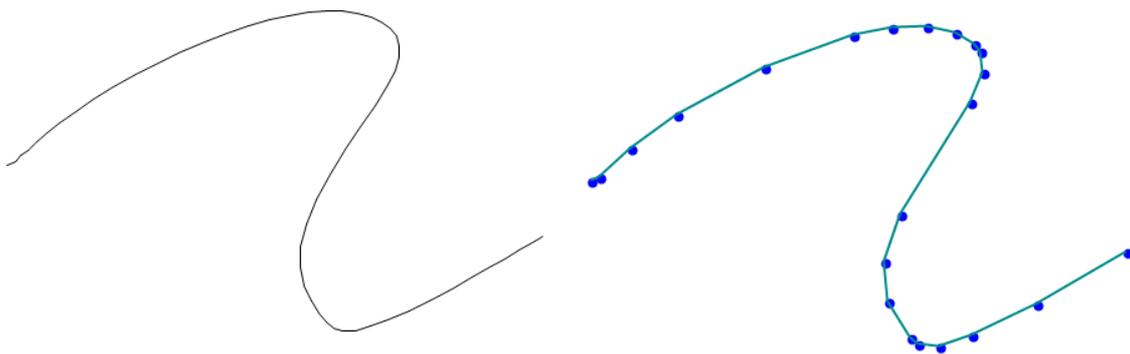


Figura 9. Uma curva à mão livre e uma aproximação por *spline linear*, com vértices destacados

Após a pré-seleção de nós, os autores utilizam programação dinâmica para selecionar os melhores candidatos. São escolhidos os nós potenciais que melhor favorecem o encaixe da curva: para Plass e Stone, os melhores são aqueles que minimizam o *erro de classificação*. Segundo eles, a definição exata do cálculo de erro não importa tanto, pois

testaram diversas alternativas com sucesso. Mas sugerem que o cálculo de erro envolva a soma do quadrado das distâncias entre os pontos  $d_i$  e a curva paramétrica  $Q$  resultante do algoritmo de encaixe de curva. A tabela de programação dinâmica armazena o erro mínimo de classificação  $e_{gj}$  entre os nós  $g$  e  $j$ , utilizando o erro mínimo de segmentos intermediários  $e_{gk}$  e  $e_{kj}$ , para  $g < k < j$ . A fórmula do erro mínimo pode ser definida como  $E_{gj} = \min(E_{gk} + e_{kj})$ ,  $g < k < j$ , sendo  $E_{gk}$  um valor pré-computado, e portanto conhecido.

Para computar o erro de classificação, é necessário invocar diversas vezes uma rotina que *identifique a curva de extremidades  $g$  e  $j$* , e ela por si só também exige várias iterações. O processo consiste em:

- (1) *Encontrar uma estimativa inicial de parâmetros para a curva.* Como citado anteriormente, para problemas de mínimos quadrados não lineares, é requerida uma parametrização inicial, e diversas iterações para refinamento. Para cada ponto  $d_i$  do conjunto de pontos  $D$  estima-se então um valor  $t_i$  que seria o parâmetro para localizar o próprio ponto  $d_i$  na cúbica  $Q$  cujos pontos de controle queremos encontrar (ou seja,  $d_i = Q(t_i)$ , no melhor caso). Uma estratégia simples para encontrar valores coerentes é a *parametrização por comprimento da corda*. Considere  $d_i = (x_i, y_i)$ . Podemos definir

$$s_k = \sum_{i=1}^{k-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

como o comprimento do segmento poligonal que vai do ponto  $d_1$  ao ponto  $d_k$ . O parâmetro  $t_k$  é então definido como  $s_k/s_n$ .

- (2) Executar o algoritmo de mínimos quadrados para encontrar a curva paramétrica. Como já foi mencionado, são utilizadas curvas de Bézier de grau 3 para compor os desenhos. Sabe-se também que uma cúbica não tem flexibilidade suficiente para permitir mais que dois pontos de inflexão, não sendo possível, portanto, representar qualquer curva desta forma. Por vezes, os segmentos ainda apresentam pontos explícitos de descontinuidade, que não devem ser tratados como pontos no meio da curva. Para contornar estas dificuldades, a utilização de *splines* é uma solução bem óbvia. São criados vários segmentos de cúbicas, conectados com continuidade  $G^1$ . Para garantir esta continuidade, só é preciso que curvas com vértices em comum possuam vetores tangentes com mesma direção e sentido. Sob este aspecto, a representação de Hermite para cúbicas torna-se interessante, ao passo que ela permite que os vetores tangentes nas extremidades da curva sejam especificados diretamente (permitindo assim que utilizemos a mesma tangente para pontos de

interseção entre curvas, garantindo assim a continuidade). Em [PS83] é possível encontrar o desenvolvimento matemático das fórmulas necessárias.

(3) *Avaliar a proximidade do resultado com os pontos de amostragem e, se necessário, criar uma nova parametrização para outra tentativa de encaixe de curva* (retornando então ao passo 2). Tendo em mãos uma curva paramétrica resultante do algoritmo de mínimos quadrados, traz-se à atenção novamente os pontos  $d_i$  e parâmetros  $t_i$ . Qual a distância dos pontos  $d_i$  para curva  $Q$  encontrada (vale ressaltar que o ponto de  $Q$  mais próximo de  $d_i$  *não necessariamente é igual a  $Q(t_i)$* , como pode ser conferido na Figura 10)? Essas distâncias representam o *erro de encaixe*, e se a distância máxima obtida ultrapassar um certo limiar, algoritmo recomeça o algoritmo de mínimos quadrados, no passo (2), com novos valores  $t_i'$  reajustados.

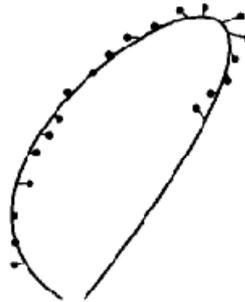


Figura 10. Exemplo de um encaixe de curva.  
Estão destacados os pontos originais  $d_i$ , e os segmentos ligando-os aos pontos  $Q(t_i)$  correspondentes.

Os novos valores  $t_i'$  são encontrados através da já mencionada distância entre o ponto  $d_i$  e a curva  $Q$ : qual o valor  $t_{i\_min}$  para o qual a distância entre  $d_i$  e  $Q(t_{i\_min})$  é mínima? De posse desta informação, fazemos  $t_i' = t_{i\_min}$ . No entanto, como encontrar  $t_{i\_min}$ ?

Pela definição de distância, o segmento que conecta os pontos  $d_i$  e  $Q(t_{i\_min})$  é *perpendicular* à tangente da curva no ponto  $Q(t_{i\_min})$ , como ilustrado na Figura 11.

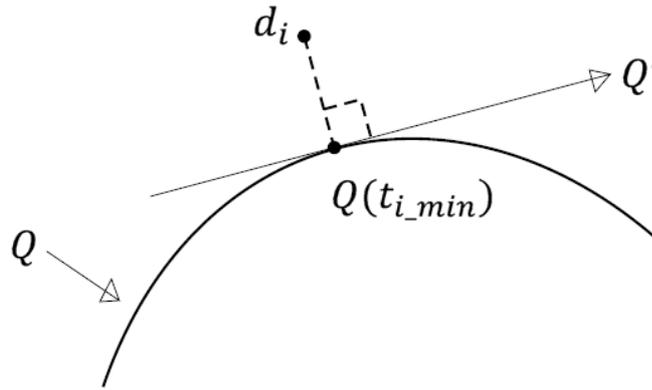


Figura 11. Distância do ponto  $d_i$  à curva  $Q$ .

Desta forma, podemos formular o problema algebricamente, como a busca de um valor  $t$  para o qual o ângulo entre o vetor  $\overrightarrow{d_i Q(t)}$  e o vetor tangente ao ponto  $Q(t)$  seja reto. Em outras palavras, procura-se o ponto  $Q(t)$  para o qual o *produto interno* entre os vetores seja zero. Ou seja,

$$[Q(t) - d_i] \circ Q'(t) = 0$$

Deparamo-nos então com a necessidade de encontrar raízes de um polinômio<sup>2</sup>. Considerando que temos uma estimativa inicial para  $t$  (que é o valor  $t_i$  atual), podemos utilizar o método de Newton-Raphson para convergir rapidamente em uma solução. A fórmula de Newton-Raphson para resolver uma equação  $f(t) = 0$  é

$$t \leftarrow t - \frac{f(t)}{f'(t)}$$

Ou seja, o valor  $t$  deve decrescer de

$$\frac{(Q(t) - d_i) \circ Q'(t)}{[(Q(t) - d_i) \circ Q'(t)]'}$$

Este é um método iterativo, ou seja, a fórmula descrita deve ser aplicada várias vezes. Como este método converge rapidamente, são necessárias poucas iterações. Quando a diferença entre valores sucessivos de  $t$  for considerada pequena o suficiente, toma-se o resultado como  $t_i'$ , o novo parâmetro do ponto  $d_i$ . O procedimento de reparametrização é repetido para todos os pontos da curva, normalizando os novos valores  $t_i$  no intervalo  $[0..1]$ . Retorna-se então ao passo 2, para outra tentativa de encaixe de curva. Há duas condições para interromper o algoritmo. A primeira delas ocorre quando a distância

<sup>2</sup> Em [Sch88] é possível encontrar o desenvolvimento matemático para a fórmula  $Q(t) - d_i$ , caso seja de interesse do leitor.

máxima entre o ponto  $t_i$  e a curva não ultrapassa o limiar pré-estabelecido. Caso contrário, o algoritmo repete os passos (2) e (3) até atingir um número máximo de iterações.

Apesar de a utilização de programação dinâmica garantir uma melhor qualidade no encaixe e uma quantidade reduzida de segmentos, tem-se como efeito colateral uma significativa degradação do desempenho. Para muitas soluções *offline*, é preferível sacrificar eficiência de forma a garantir melhores resultados. No entanto, isto inviabiliza o método de Plass e Stone para aplicações em tempo real. Naturalmente, eles também não consideram diferentes espessuras de traços: quando há um objeto extenso como ilustrado na figura 8, somente o contorno é parametrizado. Conclui-se, portanto, que o método descrito não se adequa integralmente aos propósitos deste trabalho.

### 2.2.2 Schneider

Cinco anos depois da publicação de Plass e Stone, Philip J. Schneider [Sch88] concluiu sua tese de mestrado, detalhando o funcionamento de um sistema interativo chamado *Phoenix*, que permitia a criação e alteração de curvas paramétricas em tempo real, a partir de traços provenientes do *tablet*. Deve-se dar uma ênfase especial em relação ao termo '*alteração*', da frase anterior: o sistema permite que a curva seja apagada e redesenhada em qualquer trecho, enquanto ele continua mantendo uma representação analítica após este processo. Este tipo de funcionalidade foi utilizado por se aproximar bastante do ato de desenhar, sendo portanto uma abordagem intuitiva para criação de curvas. A figura 12 ilustra o processo de alteração.

O estudo de Schneider foi fortemente inspirado no trabalho de Plass e Stone. Porém, há alguns pontos cruciais de dissimilaridade, que serão descritos juntamente ao resto do processo.

Primeiramente serão descritos os passos de pré-processamento. Como o *mouse* é o exclusivo mecanismo de entrada, foram utilizadas técnicas bem específicas para tratamento dos pontos de amostragem. Primeiramente, há uma pré-redução de pontos muito próximos. A taxa de amostragem do *mouse* é alta, e dependendo da velocidade de desenho, muita informação redundante é gerada. Este processo também auxilia na redução de ruído produzido pela mão humana, e na identificação de proeminências, que é o próximo passo após a pré-redução. O processo de identificação de pontos de descontinuidade (*corners*) é análogo ao utilizado por Plass e Stone [PS83]: vértices de ângulos mais agudos que um determinado limiar  $\Phi$  são marcados como proeminências.

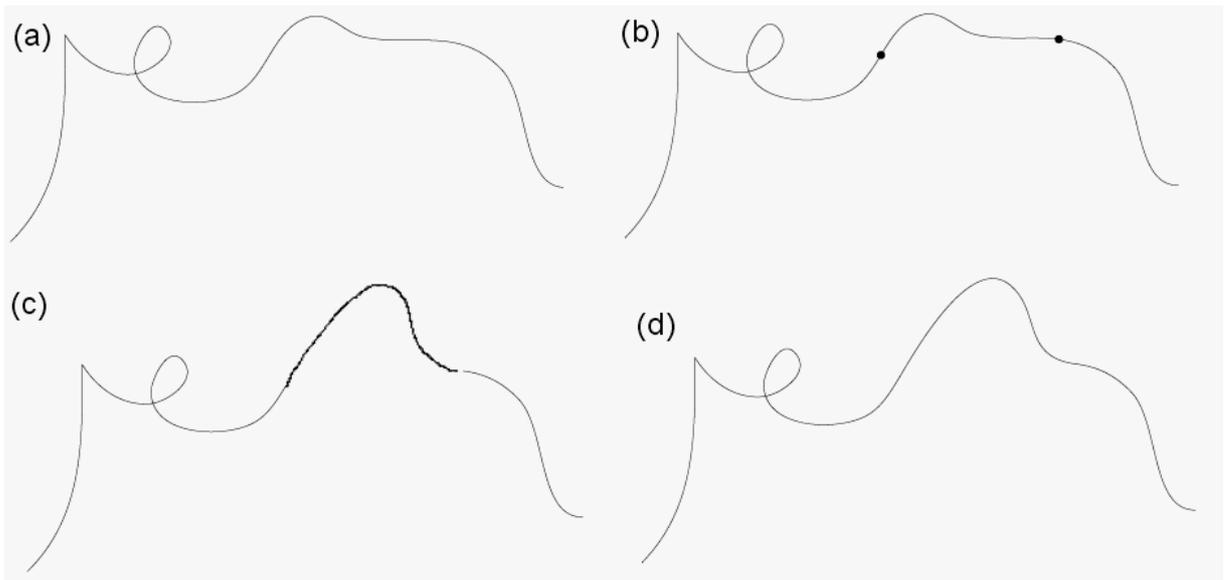


Figura 12. Modificações em uma curva, no *Phoenix*.

Em um desenho formado por curvas paramétricas(a), são informados os dois pontos entre os quais haverá um novo esboço, respectivamente como em (b) e (c). Finalmente, em (d), o sistema incorpora o novo segmento à curva paramétrica.

O passo seguinte é a redução de ruído entre extremidades e *corners*, estratégia também adotada por Plass e Stone. Schneider aplica diversas vezes (no máximo dez iterações) uma aproximação de um filtro Gaussiano para suavizar variações irregulares, segundo as fórmulas:

$$x'_i = (0.5)x_i + (0.25)x_{i-1} + (0.25)x_{i+1}$$

$$y'_i = (0.5)y_i + (0.25)y_{i-1} + (0.25)y_{i+1}$$

nas quais,  $x'_i$  e  $y'_i$  representam os valores  $(x, y)$  do ponto  $d_i$  filtrado, que é composto em parte pelos pontos vizinhos  $d_{i-1}$  e  $d_{i+1}$ . E por fim, é realizada a redução final de pontos através de *splines* lineares. O procedimento implementado por Schneider para encontrar os vértices da *spline* é recursivo. Traça-se uma reta entre duas extremidades  $d_{i_0}$  e  $d_{i_n}$ , e calcula-se a distância da reta e dos pontos  $d_k$  compreendidos entre os pontos  $d_{i_0}$  e  $d_{i_n}$  (ou seja,  $i_0 < k < i_n$ ). Se alguma das distâncias ultrapassar um limiar máximo pré-determinado, toma-se o ponto  $d_{k_{max}}$  como um vértice, e o algoritmo continua com retas traçadas entre  $d_{i_0}$  e  $d_{k_{max}}$ , e  $d_{k_{max}}$  e  $d_{i_n}$ .

Em seguida, iniciam-se os passos de *identificação da curva*, que é bastante similar ao de Plass e Stone. Há :

(1) A parametrização inicial  $t_i$  para os pontos  $d_i$  resultantes da fase de pré-processamento, também utilizando *parametrização por comprimento da corda*, como [PS83];

(2) Uma aplicação do algoritmo de mínimos quadrados para encontrar uma estimativa da curva;

(3) A verificação da proximidade dos pontos com a curva estimada, possivelmente resultando em reparametrização dos pontos e retorno ao passo (2).

Neste último passo, existem três situações dignas de menção. Uma delas ocorre após a *primeira tentativa* de encaixe de curva: *caso a distância máxima de classificação entre os pontos  $d_i$  e a curva  $Q$  não for pequena o suficiente, a operação é interrompida*. Isto é um sinal de que dificilmente será possível encaixar uma cúbica naquele trecho, mesmo com reajuste de parâmetros. Ao leitor atento isto pode parecer uma decisão um tanto arbitrária. Mas Schneider afirma que os casos de encaixe que não podem ser representados por cúbicas (como curvas com mais de dois pontos de inflexão, mais de dois *loops*, entre outros) costumam apresentar valores excepcionalmente altos para a soma dos quadrados das distâncias; e também mostra estatisticamente que existe uma forte correlação entre esta soma e a distância máxima entre os pontos  $d_i$  e  $Q(t_i)$ . Logo, a verificação da distância máxima é considerada uma heurística relativamente confiável para evitar tentativas de encaixe impossíveis.

A segunda situação ocorre quando, *até um número máximo pré-estabelecido de iterações, o algoritmo atinge uma distância aceitável entre os pontos de amostragem e a curva estimada*. Neste caso, o procedimento é encerrado, e o resultado obtido é considerado satisfatório.

Caso contrário, chegamos à terceira situação: *se após todas as possíveis iterações, a distância máxima ainda não atingir valores ideais, conclui-se que não deve possível encaixar ali uma cúbica, dado que reparametrizações adicionais não proporcionam melhoras significativas*. Na primeira e terceira situações descritas, utiliza-se uma estratégia similar à da criação de splines lineares. A seqüência de pontos  $d_i$  é dividida no ponto  $d_{\max}$  de maior distância para a curva, e o algoritmo de encaixe de curva é aplicado sobre os dois novos segmentos.

A seguir, na Figura 13, segue um pseudocódigo fornecido por Schneider que resume o funcionamento do algoritmo de encaixe de curva.

**ENCAIXAR-CUBICA***/\*Encaixar uma única cúbica a um conjunto de pontos\*/**/\*Retorna TRUE se uma única cúbica pode ser encaixada dentro da tolerância\*/*Encontre  $\hat{t}_1$  e  $\hat{t}_2$ , as tangentes nas extremidades da seqüência de pontos;Compute uma estimativa inicial  $u_i$  para cada ponto  $d_i$ , utilizando *parametrização por comprimento da corda*.Encontre uma curva que se encaixe aos pontos  $d_i$ , utilizando mínimos quadrados.Encontre a distância máxima  $dist_{max}$  entre os pontos  $d_i$  e a curva.**IF**  $dist_{max}$  for menor que um limiar  $L$  de distância máxima determinado pelo usuário  
**THEN****RETURN TRUE****ENDIF****IF**  $dist_{max}$  é pequeno o suficiente que indique que mais iterações podem ajudar **THEN****WHILE**  $dist_{max}$  exceder o limiar  $L$  **AND** o número máximo de tentativas (iteraões) não tiver sido atingido:    Compute novos parâmetros  $u_i$  utilizando o método de Newton-Raphson.    Encaixe uma cúbica, utilizando os novos parâmetros  $u_i$ .    **IF**  $dist_{max}$  não exceder limiar  $L$  **THEN**        **RETURN TRUE**    **ENDIF****ENDWHILE****RETURN FALSE**, pois o erro não ficou inferior à  $L$  em nenhum momento.**ELSE****RETURN FALSE**, pois futuras iteraões não devem ser de grande valia**ENDIF****END**

Figura 13. Algoritmo para identificar se uma única cúbica pode ser encaixada a uma seqüência de pontos

Caso este procedimento retorne FALSE, ou seja, quando ele não encontra uma curva que se aproxime dos pontos de forma satisfatória, uma estratégia similar à utilizada em splines lineares é empregada: a seqüência de pontos  $d_i$  é dividida no ponto  $d_{max}$  que apresentou maior distância para o ponto correspondente em  $Q$ , e os passos (1) a (3) do algoritmo de *identificação da curva* são executados recursivamente para os dois segmentos. Para os pontos que estão na divisão entre os dois novos segmentos, o mesmo vetor tangente é compartilhado.

O estudo de Schneider, confrontado com os objetivos deste trabalho, evoluiu em relação ao trabalho de Plass e Stone quanto à eficiência. Ele consegue bons resultados, apesar de não necessariamente ótimos, e computa curvas em tempo real. No entanto, ele tem limitações em relação ao aspecto estético do traço: são criadas curvas de largura zero, e não se consideram informações de pressão. Outra limitação, mencionada pelo próprio autor, é a inconveniência de realizar pequenos ajustes na curva: é necessário apagar o trecho indesejado e redesenhar para efetuar quaisquer alterações.

### 2.2.3 *Thierry Pudet*

Com o objetivo de simular traços artísticos, Pudet [Pud94] propôs um algoritmo para a criação de curvas com espessura variável, através do rastreamento de traçados com pressão enviados por *tablets*. Ele se baseou, entre outras referências, no trabalho de Plass e Stone [PS83] e Schneider [Sch88], e conseguiu enxergar otimizações e melhorias para o processo de encaixe de curvas.

Discutindo as técnicas utilizadas por Schneider, Pudet indica que o principal gargalo é a constante reavaliação da distância de erro que é utilizada para decidir quando se deve parar o encaixe. E destaca também a inconveniência da utilização de cúbicas, pois devido à baixa flexibilidade destas curvas, geralmente são necessários muitos segmentos para compor um único traçado.

Pudet propôs então algumas modificações. Primeiramente, a mudança de curvas de grau 3 (cúbicas) para grau 5, pois elas permitem maior liberdade que cúbicas, e geram aproximações melhores. Isto tem o efeito de não somente reduzir o erro de classificação, como também permitir que segmentos maiores sejam utilizados, minimizando o número de segmentos de curva necessários para compor o desenho. Entretanto, a computação destas curvas é consideravelmente mais custosa. E para equilibrar tal situação, Pudet exclui a etapa de reparametrização: ele afirma que o aumento de grau na curva causa efeito equivalente.

Outra otimização proposta envolve a avaliação da distância de erro entre a curva e os pontos  $d_i$ . Ao invés dos pontos  $Q(t_i)$ , computados em [PS83] e [Sch88] através da fórmula de Bézier na forma de Bernstein, são utilizadas interpolações entre pontos de uma aproximação poligonal da curva, além de uma verificação diferenciada. Seu método de avaliação de distância, apelidado por ele de “preguiçoso”, não garante encontrar o ponto de erro máximo. Logo, se for identificada a necessidade de subdividir a curva em duas para recalcular novas aproximações, o ponto de separação  $d_i$  escolhido é o *ponto central* da seqüência.

Adiante segue uma série de passos que resumem o funcionamento do algoritmo de encaixe de curva utilizado por Pudet. Perceba que não há nenhuma etapa de reparametrização neste procedimento.

1. Compute parametrização por comprimento da corda, para os pontos da trajetória digitalizada.
2. Estime a direção dos vetores tangentes nos pontos de extremidade.
3. Encaixe uma curva de Bézier de grau 5 à trajetória digitalizada.
4. Avalie a qualidade do encaixe utilizando o método de avaliação “preguiçoso”. Se a aproximação não foi boa o suficiente, divida a trajetória em duas (pelo ponto médio) e execute recursivamente desde o passo 1, para cada metade.

Figura 14. Etapas do algoritmo de encaixe de curva proposto por Pudet.

A contribuição mais interessante de Pudet, no entanto, não está relacionado ao encaixe da curva, e sim, ao “encaixe do traço”. Em outros termos, estamos nos referindo à *espessura* do traço, ou em outras palavras, ao tratamento que é dado às bordas da curva para que ela possua aparência mais próxima de uma pincelada. Isto exige a existência de um terceiro parâmetro nos pontos  $d_i$  além das coordenadas  $x$  e  $y$ : a *pressão* da caneta naquele ponto. A Figura 15 adiante ilustra um exemplo de traçado com a pressão variável em sua extensão. A curva do meio representa a trajetória, e as outras curvas delimitam as bordas do traço.

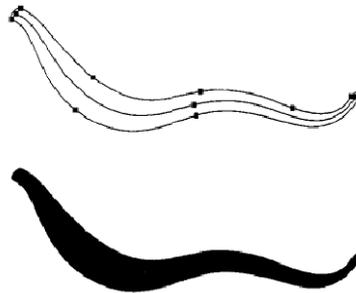


Figura 15. Exemplo de traço produzido pelo método de Pudet.

O processo pode ser resumido como a extração do envelope da convolução discreta do pincel ao longo da curva, como ilustrado na Figura 16.

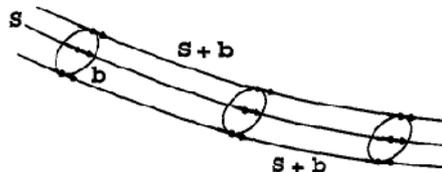


Figura 16. Envelope do pincel  $b$ , elíptico, ao longo da trajetória  $s$

Obtém-se primeiramente uma aproximação poligonal da trajetória e do pincel. Em seguida, são computadas aproximações discretas das bordas esquerda e direita do

envelope. Para tanto, o centro do pincel “é posicionado” em cada ponto da trajetória discretizada, e os pontos do pincel que estiverem mais distantes da direção da trajetória são adicionados às bordas discretas. Após isto, os pontos de cada borda são submetidos ao algoritmo de encaixe de curvas, e o resultado é conectado às extremidades. A primeira ilustração na Figura 15 destaca bem estes três elementos: *trajetória central*, *bordas* (esquerda e direita) e *extremidades*.

Mudanças de pressão têm efeito na expansão variável das dimensões do pincel e são controladas por um fator de elasticidade que regula o impacto da pressão nas bordas do traçado. Considerando que os valores de pressão são normalizados no intervalo [0..1], a elasticidade pode ser vista como o valor máximo pelo qual um determinado pincel pode dilatar, em relação ao seu tamanho natural. Ou seja, um pincel circular de diâmetro  $p$  pode ser dilatado para um círculo de diâmetro  $ep$  caso a pressão seja máxima.

A solução de Pudet traz para o conjunto de estudos analisados o grande diferencial do tratamento de pressão para a geração interativa de traços artísticos. Além disto, propõe a utilização de quárticas ao invés de cúbicas, e oferece uma nova estratégia para cálculo de distâncias. No entanto, ele não prevê em nenhum momento a edição de traços com espessura.

### 3. SketchSim

Neste capítulo, será detalhada a solução proposta para a simulação de traços em tinta e pincel, permitindo a posterior modificação dos traços. Foi criada em C# .NET 2.0 uma ferramenta – apelidada de **SketchSim** – para implementar, testar e visualizar as técnicas descritas nas seções a seguir.

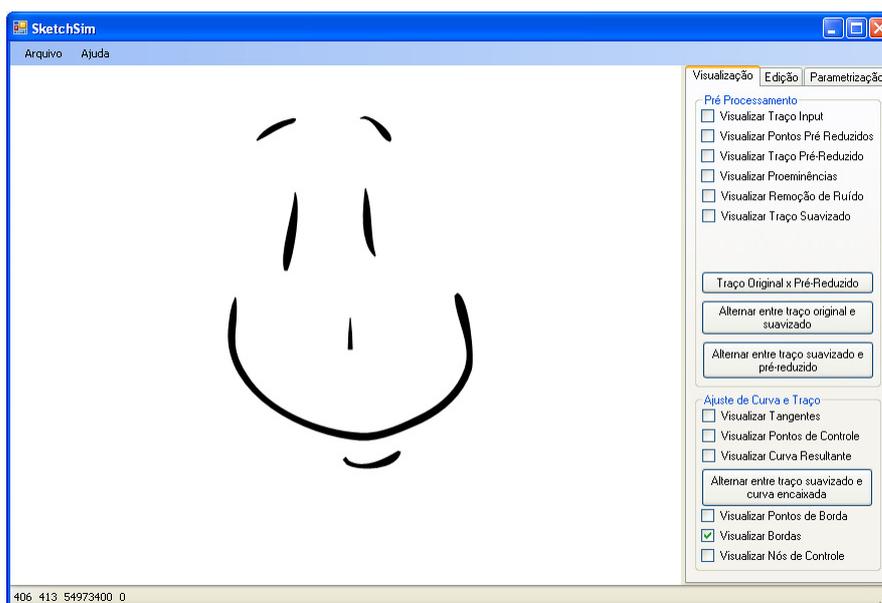


Figura 17. SketchSim

A interface com o usuário é bastante simples. Basta iniciar um traço dentro do espaço em branco. Durante o desenho, é mostrado ao usuário uma estimativa das bordas do traço final. Isto acontece porque somente ao término do traçado que o algoritmo de geração de traços é iniciado.

O SketchSim possui duas principais funcionalidades:

- (1) Geração de novos traços;
  - a. Pré Processamento
  - b. Encaixe de Curva
  - c. Geração de Bordas
- (2) Edição de curvas após criação;

A seção 3.1 descreve as etapas de pré-processamento; a seção 3.2 cita pontos específicos do algoritmo de encaixe de curvas (pois muito do que foi utilizado já está detalhado em seções anteriores); a seção 3.3 detalha como se dá a finalização dos traços com espessura; e por fim, a seção 3.4 explica como é realizada a edição de curvas.

### 3.1 Pré processamento

A etapa de pré-processamento foi fortemente inspirada no trabalho de Schneider [Sch88], pois existem diversos pontos de similaridade com este estudo. Como já foi descrito anteriormente, ele adota quatro passos (detalhados anteriormente, na seção 2.2.2) para tratamento dos pontos de amostragem: (1) Remoção de pontos muito próximos, (2) Identificação de proeminências, (3) Atenuação de ruído e (4) Redução do conjunto de pontos.

Destas 4 etapas, apenas as etapas 1 e 2 foram utilizadas. O valor padrão para a taxa de amostragem foi estabelecido em 10 milissegundos, e o ângulo para detecção de proeminências em 140°. A etapa 3 foi eliminada, pois se observou que ela estava contribuindo negativamente para o encaixe de curvas. Schneider [Sch88] e Plass e Stone [PS83] (estes últimos também utilizam a redução de ruído) justificam seu uso para atenuação de conseqüências da rasterização em imagens. No entanto, a própria etapa 1 já reduz os efeitos de *aliasing* no traço original, e o próprio *tablet* realiza filtragem espacial, segundo Pudet [Pud94]. Foi observado também que a similaridade do traço final para o traço original foi prejudicada: ou seja, as curvas que utilizaram este tratamento não estavam tão próximas das trajetórias originais quanto as que foram geradas sem redução. Pudet [Pud94] também cita brevemente que experimentou estratégias de remoção de ruído, mas os artistas que utilizaram o sistema alegaram que os traços resultantes não eram “fiéis” o suficiente a seus desenhos. Na seção 4.1 são mostrados alguns resultados que ilustram este fenômeno.

A etapa 4 também não foi utilizada, pois Schneider afirma que ela foi adotada como uma otimização facultativa, podendo ser eliminada caso o algoritmo rodasse em máquinas mais potentes (ou mais modernas, considerando que seu trabalho foi publicado em 1988). E de fato, experimentos mostraram que o algoritmo de encaixe de curvas ainda funciona em tempo real, mesmo com um conjunto maior de pontos.

### 3.2 Encaixe de curvas

O algoritmo de encaixe de curvas utilizado neste estudo baseou-se essencialmente no algoritmo utilizado por Schneider [Sch88], por duas razões. Primeiramente, a solução dele é altamente aderente aos objetivos deste estudo. Além disto, seu trabalho é mais bem documentado. O que é natural, visto que uma tese de mestrado quando confrontada com artigos científicos acaba dispondo de mais espaço para detalhamento, o que facilita a compreensão.

Em suma, esta etapa tem como objetivo criar curvas de Bézier de grau 3, conectadas por condição  $G^1$ , que possam representar adequadamente uma trajetória definida por uma seqüência de pontos bidimensionais.

Como já foi ilustrado no Capítulo 2, tal operação se dá através de sucessivos encaixes e reparametrizações, estas últimas utilizando o método de Newton-Raphson em seu cálculo. Uma seqüência de passos que resume o funcionamento do algoritmo segue abaixo:

1. Uma parametrização inicial, por comprimento da corda, é criada.
2. Encaixa-se uma curva aos pontos, utilizando a parametrização mais recente.
3. Estima-se a qualidade do encaixe, verificando a distância dos pontos da trajetória original, e da curva criada. Para encontrar esta distância, emprega-se o método de *Newton-Raphson* (descrito detalhadamente no capítulo 2).
4. Se a distância máxima não ultrapassar um limiar pré-estabelecido, considera-se o encaixe satisfatório e o algoritmo termina. Caso contrário, uma nova parametrização é utilizada (baseando-se nos pontos da curva atual que são mais próximos de cada ponto) e o processo recomeça a partir do passo 2. Caso o limite de iterações de reparametrização tenha sido atingido, a seqüência de pontos é dividida no ponto de maior erro, e o algoritmo retorna recursivamente ao passo 1, para cada uma das metades da seqüência particionada.

Os passos descritos acima já foram parcialmente detalhados em seções anteriores. No entanto, há dois pontos a ser esclarecidos.

O primeiro ponto refere-se ao tratamento de discontinuidades. Sabe-se que não se deve tentar encaixar curvas entre proeminências. Portanto, o algoritmo de encaixe é utilizado *entre* estes pontos. Para a vetorização da trajetória ilustrada na Figura 18, o algoritmo será invocado três vezes, para cada segmento de trajetória compreendido entre extremidades ou pontos de discontinuidade.

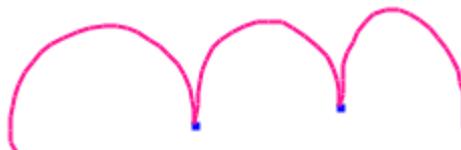


Figura 18. Trajetória com dois pontos de descontinuidade.

O segundo ponto refere-se à estimativa de tangentes para as extremidades de uma curva. Plass e Stone[PS83] utilizam uma técnica que produz uma aproximação a partir de um pequeno conjunto de pontos próximos ao ponto de tangente desconhecida. Neste

estudo, porém, utilizou-se uma técnica muito mais simples, e que produziu resultados satisfatórios. A tangente no ponto  $p_i$ , com sentido de  $p_i$  para  $p_{i+1}$  é definida pelo vetor,  $\overrightarrow{p_i p_{i+1}}$ , ou encontrado pela fórmula  $p_{i+1} - p_i$ . Similarmente, a tangente de  $p_i$  para  $p_{i-1}$  é dada por  $\overrightarrow{p_i p_{i-1}}$ .

### 3.3 Geração de Bordas

Como foi descrito no capítulo anterior, Pudet [Pud94] constrói traços com espessura variável ao longo de uma trajetória a partir de informações de pressão e de um formato de “pincel” convexo. São encontradas estimativas poligonais da(s) borda(s) esquerda e direita do traço, e o algoritmo de encaixe de curva é novamente utilizado para cada representação discreta de borda. Por fim, estas bordas são conectadas às extremidades, e o contorno do traço é obtido.

Neste estudo, foi utilizada uma técnica similar, mais simples, mais rápida, e que não exige a definição de um formato de pincel. Assim como o método de Pudet, ela também utiliza aproximações poligonais da curva e estima a trajetória discretizada das bordas esquerda e direita, para posterior utilização do algoritmo de encaixe de curvas. Ela pode ser vista como uma especialização da técnica de Pudet, utilizando pincéis circulares. Os pontos das bordas são computados simplesmente obtendo os pontos perpendiculares à trajetória, com distância variando proporcionalmente à pressão.

Para cada ponto da trajetória discretizada, são computadas as *tangentes normalizadas* através do algoritmo de De Casteljaou. Pela fórmula da curva de Bézier de grau  $n$  na forma recursiva [Far97], temos:

$$\frac{d}{dt} b^n(t) = n[b_1^{n-1}(t) - b_0^{n-1}(t)]$$

Para o caso cúbico, temos então que a tangente de  $b_0^3(t)$  pode ser definida como  $3[b_1^2(t) - b_0^2(t)]$ . Como este vetor é posteriormente normalizado, utiliza-se a simplificação  $[b_1^2(t) - b_0^2(t)]$ .

Com este vetor  $v$  normalizado em mãos, a pressão  $p$  (também normalizada) é empregada como fator redimensionador da tangente. Assim como Pudet, é utilizado um fator de elasticidade  $e$  (que controla o impacto da pressão na espessura), multiplicando o vetor por  $ep$ . Esta tangente, já redimensionada, é também rotacionada de  $90^\circ$  (os pontos da borda esquerda) e  $270^\circ$  (pontos da borda direita). Para  $90^\circ$ , o vetor resultante é:

$$\begin{bmatrix} \cos 90^\circ & -\text{sen } 90^\circ \\ \text{sen } 90^\circ & \cos 90^\circ \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -y \\ x \end{bmatrix}$$

Para rotacionar o vetor tangente em  $270^\circ$ , basta inverter o sinal do vetor rotacionado de  $90^\circ$ , obtendo então  $\begin{bmatrix} y \\ -x \end{bmatrix}$ . Somam-se estes vetores ao ponto correspondente na trajetória, obtendo os pontos das bordas esquerda e direita. A Figura 19 adiante ilustra estas operações.

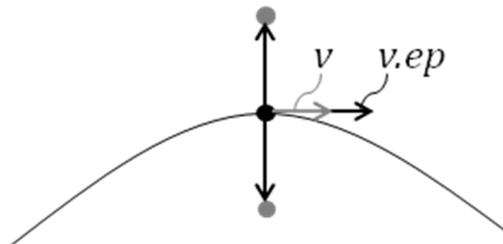


Figura 19. Obtenção de pontos das bordas

Finalmente, os pontos discretos de cada borda são submetidos ao algoritmo de encaixe de curvas, para representar o traço de forma completamente vetorial.

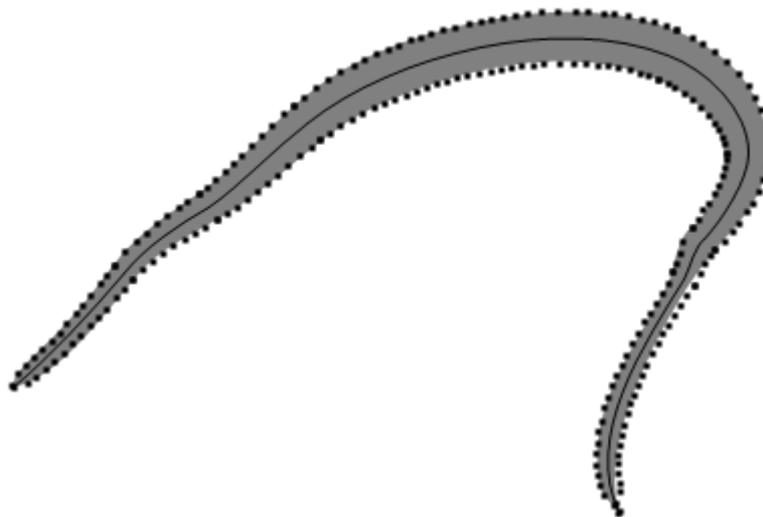


Figura 20. Pontos discretos e resultado do encaixe de curvas para um traço de exemplo.

### 3.4 Edição de curvas após criação

Um dos requisitos deste estudo é permitir a posterior edição de traços criados. Como já foi mencionado, dentre os artigos analisados, somente o Schneider [Sch88] propõe alguma espécie de edição pós-processamento, e somente para curvas com espessura zero. Com um pouco de imaginação do leitor, é possível observar que tentar reproduzir sua forma de interação com curvas de largura variável é no mínimo complexo, considerando que seria necessário haver um mecanismo de transição de espessura bastante inteligente para enxertar uma nova curva no meio de outra.

Diante desta dificuldade, utilizou-se uma abordagem distinta proposta por Bartels e Beatty [BB89], que fornece mecanismos para permitir o reajuste de uma curva de Bézier ou B-Spline, especificando o deslocamento de um ponto qualquer contido dentro delas. Ou seja, é possível escolher qualquer ponto dentro da curva, arrastá-lo para outro local, e a curva é recalculada de forma a passar pelo novo local com a mesma parametrização.

Para que isto seja feito, é efetuado um simples deslocamento dos pontos de controle. Para uma cúbica de Bézier  $Q$  qualquer, se considerarmos que um determinado ponto  $Q(t)$  sofreu um deslocamento de  $\Delta x$  no eixo  $x$  e de  $\Delta y$  no eixo  $y$  e foi movido para o ponto  $Q(t)'$ , então cada ponto de controle  $b_i, i = 0,1,2,3$  sofre também um deslocamento  $(\Delta x_i, \Delta y_i)$  de:

$$\begin{aligned}\Delta x_i &= \Delta x \cdot w_i \\ \Delta y_i &= \Delta y \cdot w_i \\ w_i &= \frac{B_i}{B_0^2 + B_1^2 + B_2^2 + B_3^2}, \quad B_i = B_i^3(t)\end{aligned}$$

Este cálculo desloca os 4 pontos de controle de  $Q$ , e o resultado final é uma curva  $Q'$  que passa pelo novo ponto  $Q(t)'$ .



Figura 21. Exemplo de uma possível modificação de uma curva, reajustando todos os pontos de controle

No entanto, pode ser desejável fixar as extremidades, e dar liberdade de movimento somente aos pontos  $b_1$  e  $b_2$ . Neste caso,  $w_i$  é definido como:

$$w_i = \frac{B_i}{B_1^2 + B_2^2}$$



Figura 22. Exemplo de reajuste, mantendo extremidades fixas.

No SketchSim, foi adotada uma convenção sugerida pelos próprios Bartels e Beatty. São fixadas as extremidades do traço para quase todos os pontos da curva, utilizando então  $w_i = B_i / (B_1^2 + B_2^2)$ . No entanto, aqueles muito próximos das pontas (por uma distância pequena e pré-determinada) têm a possibilidade de mover os respectivos pontos de

controle:  $w_i = B_i / (B_1^2 + B_2^2 + B_3^2)$  ou  $w_i = B_i / (B_0^2 + B_1^2 + B_2^2)$ , dependendo do ponto de controle que vai ser movido.

Isto funciona perfeitamente para curvas isoladas. No entanto, é necessário considerar o que acontece quando se utiliza *splines*. Dado que este trabalho utiliza segmentos cúbicos conectados por continuidade  $G^1$ , como fazer para manter a continuidade entre curvas adjacentes?

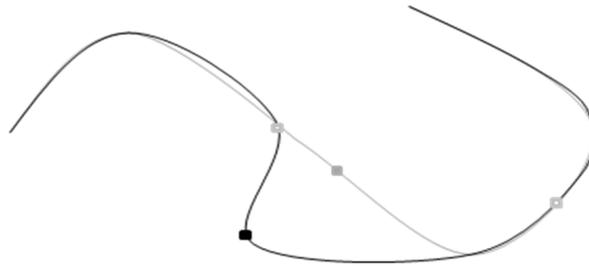


Figura 23. Exemplo de deslocamento para segmentos conectados.

Para manter a continuidade, a solução é imediata: é necessário propagar o deslocamento, no mínimo, para os segmentos vizinhos. Considere a curva  $c_i$  e a curva  $c_{i+1}$ , adjacentes, compartilhando uma extremidade como ilustrado na Figura 24 a seguir.

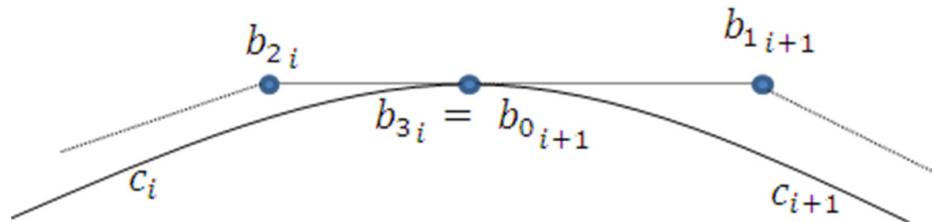


Figura 24. Duas curvas de Bézier conectadas pela condição  $G^1$ .

Vamos supor que a curva  $c_i$  sofreu deslocamento nos seus quatro pontos de controle, e alterou a posição do ponto  $b_{2i}$  para  $b_{2i}'$ . Neste caso, a linha definida pelos pontos  $b_{2i}'$  e  $b_{3i}'$  da curva  $c_i$  está em discordância com a linha definida pelos pontos  $b_{0i+1}$  e  $b_{1i+1}$  da curva  $c_{i+1}$ , e a continuidade no ponto de junção entre as curvas é perdida. Para manter a condição  $G^1$ , é necessário reajustar a posição do ponto  $b_{1i+1}$ .

Sabe-se que este novo ponto  $b_{1i+1}'$  precisa estar posicionado na linha definida por  $b_{2i}'$  e  $b_{3i}'$ . Sua posição dentro desta linha é computada através da proporção definida pelas dimensões dos antigos vetores  $\overrightarrow{b_{3i}b_{2i}}$  e  $\overrightarrow{b_{1i+1}b_{0i+1}}$ .

Para o reajuste da curva, parte-se do pressuposto de que já se sabe, dentre os diversos segmentos cúbicos que compõem um traço, qual deles será modificado. Além, é claro, do valor 't' que localiza o ponto deslocado nesta curva. Uma das formas mais intuitivas de se especificar isto é simplesmente clicando e arrastando em uma determinada posição  $pos = (x, y)$  próxima o suficiente do traço. No entanto, determinar a proximidade de  $pos$  para a curva não é uma tarefa muito simples, quicá encontrar o ponto da curva mais próximo de  $pos$ .

O leitor mais atento pode perceber que um cenário similar já foi abordado neste estudo, ainda na etapa de encaixe de curvas. Quando era necessário avaliar a qualidade do encaixe, encontrava-se a o ponto  $Q(t_{min})$  da curva que fosse mais próximo de cada ponto de amostragem  $d_i$ , para finalmente computar a distância entre  $d_i$  e a curva. No entanto, naquele momento, havia já uma estimativa  $t$  prévia para  $t_{min}$ , que viabilizava a utilização do método de Newton-Raphson. Nesta situação, no entanto, não existem estimativas confiáveis, e é necessário encontrar outro caminho.

Schneider [Sch88] utiliza outro método de minimização de distâncias ponto-curva além do Newton-Raphson, um que não exige estimativas prévias. Ele consegue transformar a equação  $[Q(t) - d_i] \circ Q'(t) = 0$  (com ilustrações na seção 2.2.1, Figura 11) em outra curva de Bézier (de grau 5) na forma de Bernstein, e então identificar quantas vezes a curva cruza o eixo horizontal. Para computar aproximações de forma eficiente, ele utiliza propriedades específicas das curvas de Bézier e subdivisão nos trechos candidatos. Quando o trecho subdividido que cruza o eixo horizontal está próximo o suficiente de uma reta, é computada uma aproximação da raiz pela interseção do eixo horizontal com o segmento formado pelos pontos inicial e final da curva. Ma e Hewit [MH03] utilizam subdivisão diretamente na curva. Com as subcurvas são realizados testes rápidos com produto interno entre certos vetores, que servem para identificar se o ponto  $pos$  se localiza em áreas que sinalizem a possibilidade de aquela curva conter pontos próximos a ele.

Neste estudo, foi utilizada uma técnica própria, que importa características de ambas as aplicações. Ela consiste, basicamente, em efetuar subdivisões recursivas na curva, utilizando o algoritmo de De Casteljau, até encontrar subcurvas que sejam próximas o suficiente de retas. Em tal situação, é possível então definir um segmento  $\overline{b_0 b_3}$  entre os pontos extremos desta curva, e verificar se a projeção do ponto  $pos$  está contida dentro dele.

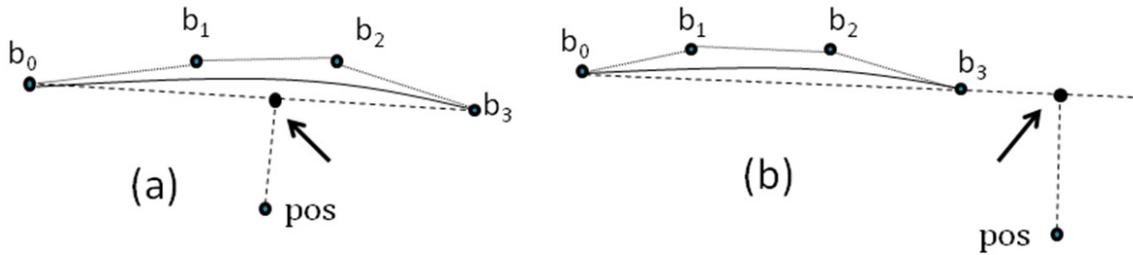


Figura 25. Projeção do ponto  $pos$  estiver contida dentro (a) e fora (b) do segmento  $\overline{b_0b_3}$

Se não há (Figura 25 (b)), o ponto não possui candidato mais próximo naquela subcurva, e o algoritmo retorna sem resultados. Caso contrário (Figura 25 (a)), computa-se então uma estimativa do parâmetro  $t_{min}$ , a partir da razão entre os parâmetros das extremidades  $b_0$  e  $b_3$  na curva completa, e a posição da projeção de  $pos$  no segmento  $\overline{b_0b_3}$  (como ilustrado na Figura 26).

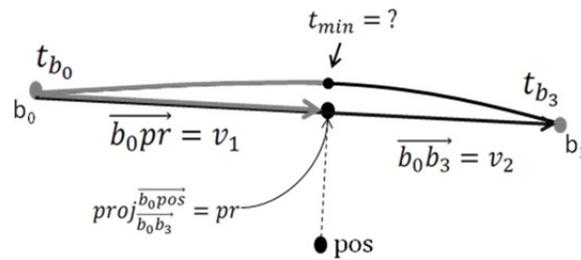


Figura 26. Estimativa do parâmetro  $t_{min}$

Ou seja:

$$\frac{t_{min} - t_{b_0}}{t_{b_3} - t_{b_0}} = \frac{\|v_1\|}{\|v_2\|}$$

$$t_{min} = \frac{\|v_1\|}{\|v_2\|} \cdot (t_{b_3} - t_{b_0}) + t_{b_0}$$

Este algoritmo pode chegar a encontrar vários candidatos por curva. No final, é utilizado o ponto que possui menor distância para  $pos$ , e que for inferior a um limiar pré-definido de distância máxima para a curva.

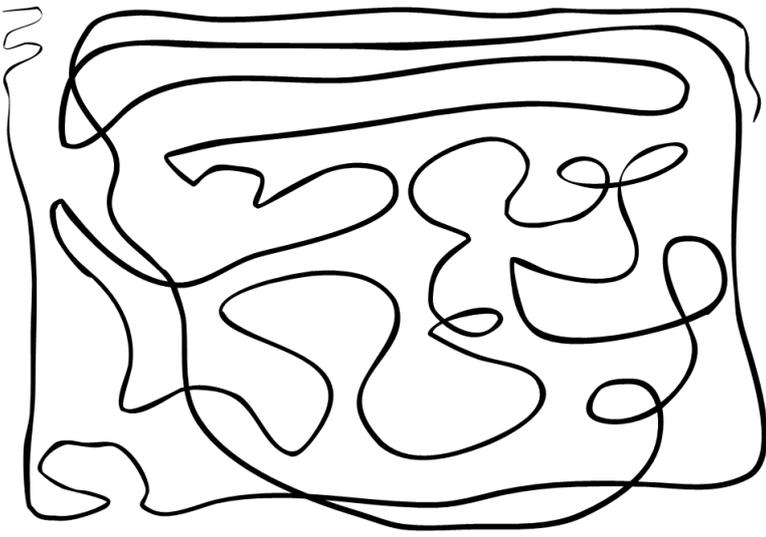
Para concluir esta seção, é preciso relembrar que os traços editados possuem espessura variável, com conjuntos independentes de curvas de Bézier representando cada uma das bordas. Então, para uniformizar o efeito do deslocamento, é *modificada apenas a curva central*. Após isto, os pontos das bordas são re-estimados, e são então submetidos ao processo de encaixe de curvas. Para permitir esta renovação, é necessário armazenar dados de pressão, e associá-los a posições específicas na curva.

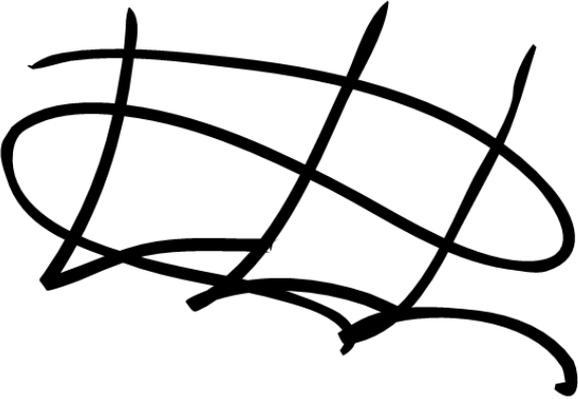
## 4. Avaliação e Resultados

### 4.1 Avaliação de Eficiência

Um dos objetivos deste trabalho é realizar a completa criação do traço com o mínimo de atraso possível, e de forma a viabilizar seu uso em uma ferramenta interativa. Experiências mostraram que este requisito foi preenchido satisfatoriamente. Para traços de tamanho pequeno e médio, o período de processamento da curva é praticamente imperceptível. Ele se torna significativo com traços muito longos, porém, mesmo nestes casos não costuma ultrapassar dois ou três segundos para o cálculo e pintura.

O SketchSim foi desenvolvido e testado em um computador equipado com Intel® Core 2 Duo 1,66GHz e 1GB de memória RAM. Adiante, é listado o tempo de execução de quatro ilustrações de exemplo. Estes números representam o tempo compreendido entre o início do pré-processamento e a finalização do traço, não incluindo o período necessário para a exibição do resultado na tela.

	<b>Número traços: 1</b> <b>Tempo: 31 ms</b> <b>Subcurvas: 7</b> <b>Pontos: 46</b>
	<b>Número traços: 1</b> <b>Tempo: 859 ms</b> <b>Subcurvas: 143</b> <b>Pontos: 1456</b>

	<p><b>Número traços:</b> 1  <b>Tempo:</b> 343 ms  <b>Subcurvas:</b> 74  <b>Pontos:</b> 376</p>
	<p><b>Número de traços:</b> 4  <b>Tempo Médio:</b> 50 ms  <ul style="list-style-type: none"> <li>• <b>Máximo:</b> 140 ms</li> <li>• <b>Mínimo:</b> 15 ms</li> </ul> <b>Subcurvas:</b> 30  <b>Pontos:</b> 233</p>

#### 4.2 Justificativa para eliminação da etapa de redução de ruído

Como já foi mencionado na seção 3.1 deste relatório, umas das técnicas propostas por Schneider como etapa de pré-processamento foi eliminada, por prejudicar a aderência aos traçados do usuário. A Figura 27 adiante mostra a diferença entre uma curva que passou pelo processo de redução de ruído (em linha tracejada) com 10 iterações de suavização (como implementada por Schneider[Sch88]) e que não passou (linha contínua).

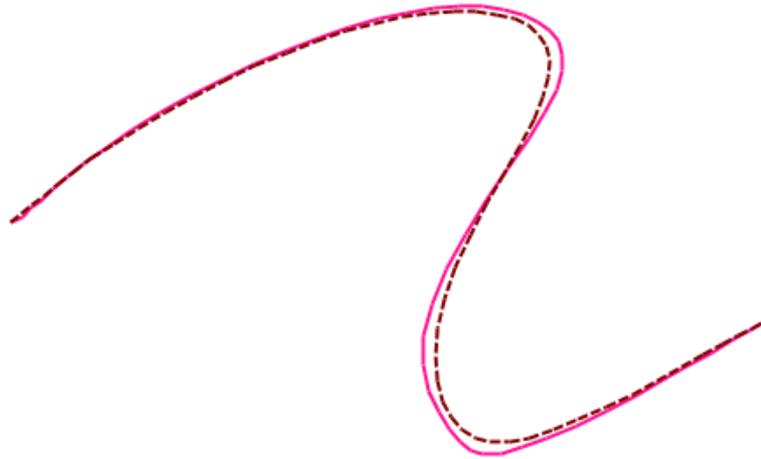


Figura 27. Diferença entre curva com redução de ruído (linha tracejada) e sem redução de ruído (linha contínua).

Como se pode notar, a diferença é perceptível a olho nu. Esta degradação de detalhes é particularmente inconveniente quando os traços se tornam menores e mais sutis. A seguir, na Figura 28, é mostrado um exemplo com curvas como estas que perderam porção significativa de sua curvatura e quase degeneraram para retas. Isto é ruim para o artista, pois perde-se controle e precisão, e a ação de desenhar torna-se menos natural.



Figura 28. Perda de detalhes da curvatura após etapa de redução de ruído.

As curvas da esquerda representam o input do usuário após redução de pontos muito próximos (etapa 1). No meio, observa-se as mesmas curvas, juntamente o resultado da etapa de redução de ruído. À direita, é mostrado o resultado final após encaixe de curva e criação de bordas. A diferença entre o traço de entrada e a curva gerada é considerável, e finalmente optou-se pela eliminação da etapa de redução de ruído.

#### 4.3 Anomalias ocasionais na geração de traços

Em raras ocasiões, é possível encontrar certos traços que não apresentam aparência contínua (como ilustrado na Figura 29 a seguir).



Figura 29. Resultados obtidos com numero máximo de reparametrizações igual a 10.

Investigando a raiz deste fenômeno, descobriu-se nestes casos o problema reside na etapa de encaixe de curvas. Isto foi identificado ao modificar a quantidade de iterações máximas para re-encaixe antes da divisão da curva no ponto de maior erro. A Figura 30 adiante mostra o resultado obtido quando a quantidade máxima de reparametrizações foi reduzida de 10 para 5.



Figura 30. Resultados obtidos com numero máximo de reparametrizações igual a 5.

A figura 31 a seguir ilustra bem a razão da deformidade. Abaixo de alguns dos mesmos traçados ilustrados anteriormente está a curva resultante do algoritmo de encaixe de curvas juntamente com os pontos de controle de cada cúbica da trajetória.

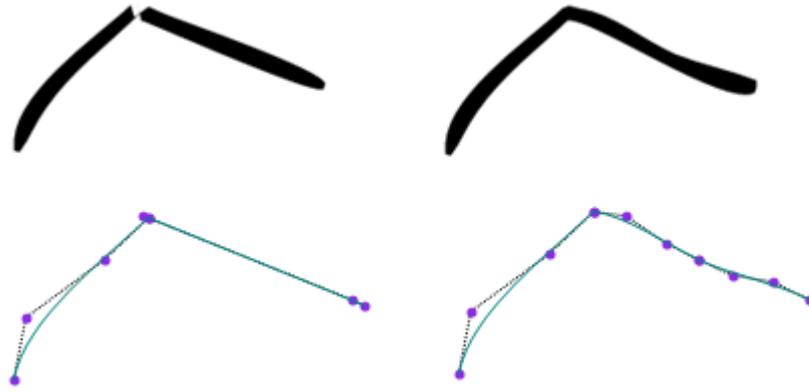


Figura 31. Mesmo traço com diferentes iterações máximas de reparametrização (10 e 5, respectivamente), e o resultado do algoritmo de encaixe de curvas.

O que provoca esta anomalia? A segunda cúbica do traçado defeituoso contém uma estrutura curiosa, ilustrada na Figura 32. O ponto  $b_1$  se localiza não na frente, mas *atrás* do ponto  $b_0$ . Por isto, ocorrem inversões no cálculo dos pontos das bordas da esquerda e da direita

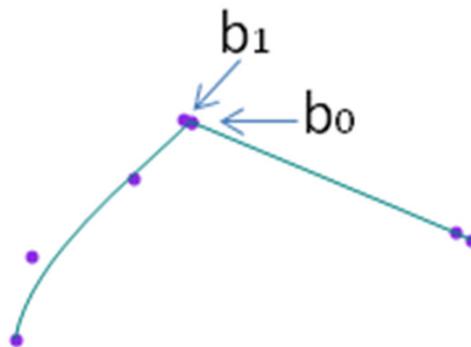


Figura 32. Incoerência na geração de uma cúbica

Isto é conseqüência da instabilidade do método de Newton-Raphson, problema também enfrentado por Ma e Hewitt [MH03]. Observando sucessivas iterações do mesmo, notou-se que ele por vezes retorna resultados incoerentes e muito distantes da estimativa inicial. Ocasionalmente, ele também gera valores que extrapolam o intervalo  $[0..1]$ . Com os traços da figura 31, provavelmente houve, até 10 iterações, convergência do método para valores incorretos e a posterior geração de uma cúbica anômala. Até 5 iterações não se conseguiu gerar um segmento próximo o suficiente dos pontos de amostragem, o que provocou uma cisão do conjunto de pontos naquela área e a posterior geração de duas cúbicas (felizmente, sem anomalias). Visto que o método de Newton-Raphson envolve conceitos matemáticos mais avançados, investigar o motivo da instabilidade e tentar

mitigar seus os efeitos iria requerer uma pesquisa mais aprofundada que extrapola o escopo deste estudo.

#### 4.4 Considerações sobre edição de curvas

A edição de curvas é realizada através de interação de clique e arrasto (*drag & drop*) do mouse. Em uma lista, seleciona-se o traço a ser modificado, e com um clique do usuário, é identificado o ponto de deslocamento na curva. Ao “arrastar” a curva, é iniciado seu o reajuste a intervalos regulares de 300 ms.

A edição de curvas mostrou resultados satisfatórios para cúbicas uniformes e pequenas alterações. A figura adiante mostra duas versões de uma ilustração: a primeira representa a ilustração imediatamente após a criação com o *tablet*, e a segunda após várias modificações em lugares distintos. Não foram removidos ou adicionados novos traços ao desenho.

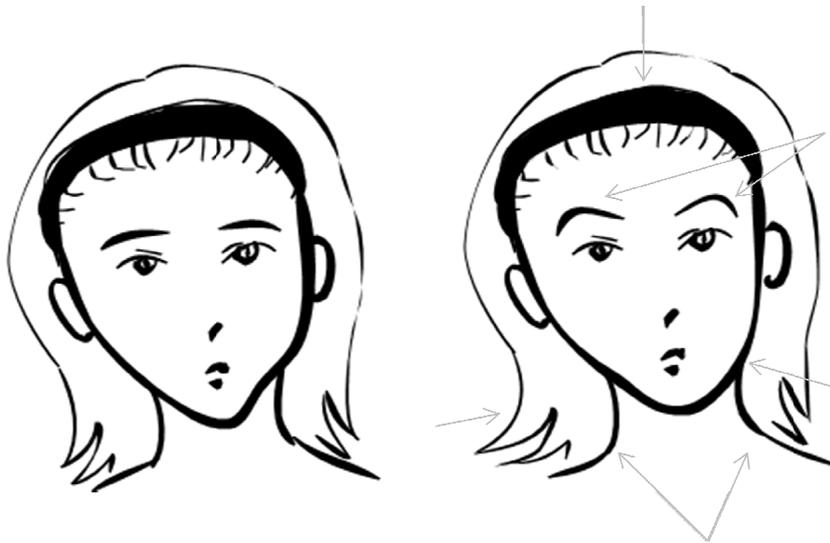


Figura 33. Ilustração antes e depois de edição de curvas. Setas indicam áreas editadas.

É possível também realizar deslocamentos mais acentuados:



Figura 34. Modificações acentuadas em curvas

Observe os traços dos ombros, na Figura 34. Para realizar esta modificação são necessários 2 passos de edição: um que desloque a extremidade dos traços para baixo, e outro que aumente a curvatura (como ilustrado na Figura 35).



Figura 35. Passos para modificação do ombro

No entanto, isto não funciona bem em todos os casos. Pelo fato de a modificação ser bastante local (modificar no máximo 3 segmentos adjacentes), para segmentos muito pequenos a edição produz resultados estranhos. Na figura 36 a seguir, o segmento mostrado em (a) sofreu um deslocamento como indicado pela seta em (b). Como o segmento é pequeno, somente uma parte limitada da curva é modificada, produzindo uma proeminência pouco natural, como sinalizado pela seta em (d).

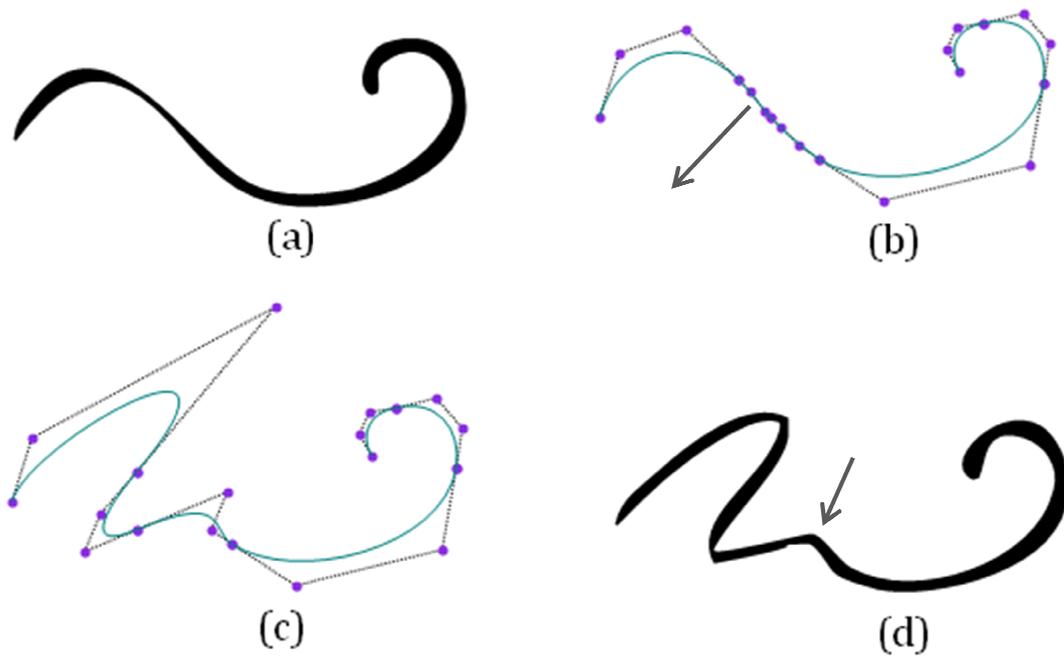


Figura 36. Etapas de edição em um traço com segmento cúbico curto

A combinação de proeminências e segmentos pequenos também não cria cenários favoráveis. Segmentos cujos extremos correspondam a proeminências no traço não propagam o deslocamento para segmentos vizinhos, pois não há continuidade a ser mantida nas extremidades. No entanto, em algumas situações como a ilustrada na Figura 37, proeminências acabam surgindo devido à forte angulação do *loop* (destacados na curva (b)). E a edição que deveria atuar uniformemente na espiral acaba produzindo deformidades, como ilustrado em (d).

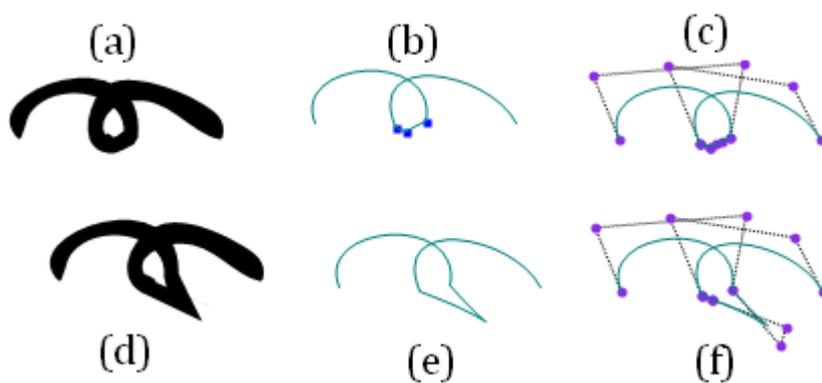


Figura 37. Etapas de edição em um traço com segmentos curtos e proeminências.

É fato também que, por vezes, a edição gera resultados incorretos. Isto ocorre principalmente com curvas anômalas (como ilustrado na figura a seguir), mas

ocasionalmente aparece com curvas normais também. Uma deficiência reportada pelos próprios autores da técnica, Bartels e Beatty [BB89], faz com que o deslocamento de pontos próximos de extremidades fixas produzam proeminências bastante acentuadas: e isto também influencia negativamente na robustez da edição.



Figura 38. Problema encontrado na edição de curvas

#### 4.5 Avaliação qualitativa junto a potenciais usuários

A ferramenta SketchSim foi submetida à avaliação de alguns artistas e designers, que produziram ilustrações em diversos estilos, e opinaram sobre aspectos artísticos do traço, abordando deficiências e qualidades. Este trabalho também foi importante para trazer à luz outras possibilidades artísticas não identificadas pela autora.

Em geral, os experimentadores gostaram da interação com a ferramenta. Acharam fácil e rápido de usar. A maioria deles já possuía experiência com *tablets*, mas mesmo os que não possuíam adaptaram-se rapidamente, devido ao formato intuitivo de criação de traços. Observou-se também que a aderência das curvas geradas ao traçado original do usuário foi considerada satisfatória.

Entre as sugestões fornecidas, um deles mencionou que oferecer variedade de “pincéis” simulando outros meios artísticos (como lápis, por exemplo) seria uma boa funcionalidade. A maioria gostou do estilo do traço, mas um dos experimentadores questionou que por vezes o traço perdia em naturalidade, por causa do acabamento nas proeminências (como apontado na figura adiante).



Figura 39. Exemplo de traço com acabamento destacado.

A seguir, são apresentadas algumas ilustrações produzidas através da ferramenta, por autores diversos.



Figura 40. 'Aviador', por Fradique Filho.



Figura 41. 'Mão e Rascunho', por Nicole Sultanum

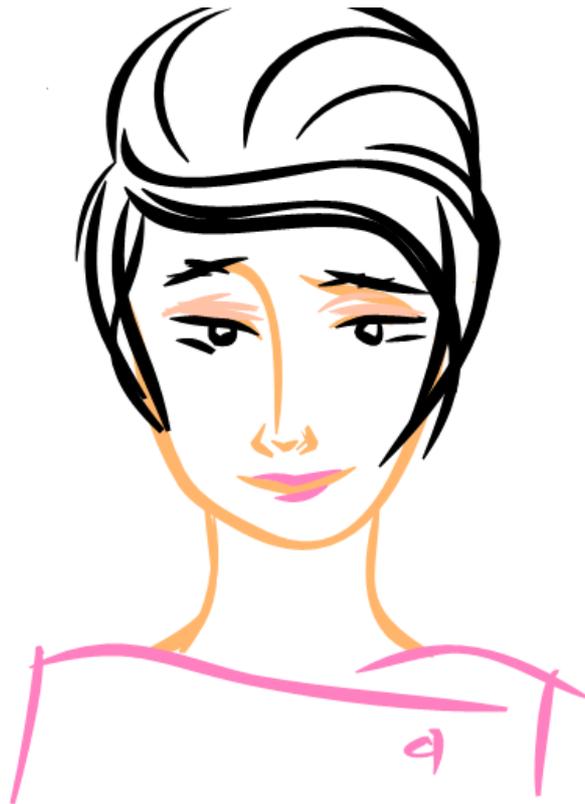


Figura 42. Ilustração sem título, por Frederico de Melo



Figura 43. 'Gata', por Zózimo Neto.

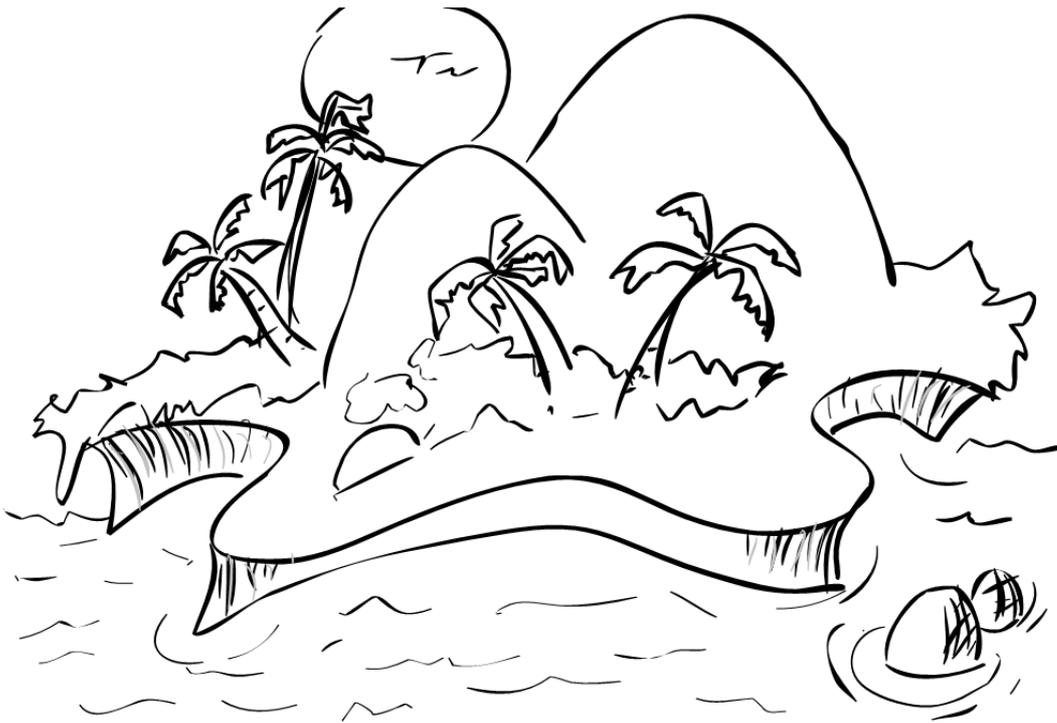


Figura 44. 'Ilha de Lost', por Leonardo Menezes.



Figura 45. Ilustração sem título, por Ricardo Teixeira

## 5. Considerações finais

Pode-se dizer que os objetivos traçados para este trabalho foram atingidos satisfatoriamente. Foi proposta e implementada uma simulação interativa para criação e edição de traços em tinta e pincel. Houve também a preocupação de proporcionar qualidade de interação para criação e edição dos traços, facilitando a atividade do usuário. Produzindo resultados coerentes, as técnicas foram devidamente validadas por artistas profissionais e amadores.

Há bastante espaço para melhora, entretanto. As anomalias nos traços, por exemplo, apesar de decorrerem de certa forma por efeitos colaterais de técnicas previamente utilizadas, representam uma falha grave a ser atacada. É preciso elaborar uma forma mais robusta de lidar com a instabilidade do método de Newton-Raphson.

São diversas as estratégias que reduziriam a ocorrência de deformidades nos traços. Uma delas é a melhoria do cálculo de estimativa de tangentes, que poderia diminuir a quantidade de segmentos gerados e a quantidade de reparametrizações, proporcionando encaixes de melhor qualidade. Outra possível abordagem é o aumento do grau das curvas, de 3 para 5 (como sugerido por Pudet [Pud94]). No entanto, é necessário analisar o impacto desta decisão na eficiência, para evitar que a inegável perda de desempenho não afete de forma significativa a interação com a ferramenta.

Outro ponto que merece atenção é a fragilidade da edição de curvas, tanto na sua dependência da representação da curva quanto na forma ineficiente de reajuste das bordas. É fato que com uma melhoria na qualidade do encaixe, as ocorrências de anomalias na edição diminuiriam bastante. No entanto, seria mais interessante adotar uma estratégia distinta, que resolveria as duas deficiências citadas neste parágrafo: modificar a estrutura analítica das curvas. O ideal seria utilizar uma forma que pudesse por si só representar a trajetória central e as trajetórias das bordas do traço. Su et. al.[Su02] e Seah et. al.[Seah05] apresentam *splines* modificadas que incorporam informações de espessura nos próprios pontos de controle. Isto facilitaria enormemente a edição de curvas, e reduziria as deformações geradas atualmente pelo fato das bordas possuírem existência independente (pois cada uma possui seu próprio conjunto de cúbricas de Bézier, completamente independentes e alheios à borda oposta).

Para aumentar o poder do usuário incrementando ainda mais a edição de curvas, existem duas possíveis vertentes. Uma delas é a adaptação da edição proposta por Schneider [Sch88] (que permite que trechos de traços sejam apagados e redesenhados

livremente) de forma a considerar informações de pressão coerentemente. Outra possibilidade é a edição da espessura ao longo do traço: permitir que determinado trecho fique mais largo ou mais fino. Neste caso, o desafio não é somente implementar mecanismos de mudança da espessura, mas também oferecer uma possibilidade simples de interação que também não reduza a liberdade do usuário.

Quanto a aspectos artísticos, existem várias melhorias interessantes. A possibilidade de escolher diferentes acabamentos de traço (afiados, quadrados ou arredondados, por exemplo) é um diferencial importante que conta muito para quem trabalha com arte e ilustração. E finalmente, uma das funcionalidades mais interessantes é a simulação da difusão de tinta, permitindo que os pigmentos interajam com água e diferentes tipos de papel, gerando então graus variados de concentração. Su et. al.[Su02] e Yu et. al.[Yu03] abordam estratégias relacionadas a esta vertente de estudo.

## Referências

- [BB89] Bartels, R. H; Beatty, J. C. **A Technique for the Direct Manipulation of Spline Curves.** Proceedings of Graphics Interface '89, p.33-39, 1989.
- [Chu88] Chua, Y.S; Winton, C. N. A. **User Interface for Simulating Calligraphic Pens and Brushes.** Proceedings of the 1988 ACM sixteenth annual conference on Computer science, P. 408 – 413, 1988.
- [Cur97] Curtis, C.J. et al. **Computer-Generated Watercolor.** Proceedings of the ACM SIGGRAPH 97, P. 421-430, ACM Press, 1997.
- [DcF05] DeCarlo, D., Finkelstein, A. **Line Drawings from 3D Models.** SIGGRAPH 2005 Course 7, 281 p., 2005.
- [Far97] Farin, G. **Curves and Surfaces for Computer-Aided Geometric Design.** Quarta edição, Academic Press, 1997.
- [Hai98] HAIGA Online. **The Art of Sumi-e Painting .** Entrevista com Susan Frame. HAIGA Online, Novembro de 1998. Disponível em <<http://members.aol.com/Jemrich/artofsumie.html>>. Último acesso em 25 de Maio de 2008.
- [HLW93] Hsu, S.C.; Lee I.H.H. ; Wiseman N.E. **Skeletal Strokes.** Proceedings of the 6th annual ACM symposium on User interface software and technology, P. 197 - 206, 1993.
- [HL94] 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, P 109-118, 1994.
- [MH03] Ma,Y.L; Hewitt, W.T. **Point Inversion and Projection for NURBS Curve and Surface: Control Polygon Approach.** Computer Aided Geometric Design, Vol.20, Num 2, P. 79-99, 2003.
- [PS83] Plass, M.; Stone, M. **Curve-fitting piecewise parametric cubics.** Computer Graphics (SIGGRAPH '83 Proceedings), Vol. 17, Num. 3, P. 229-239, 1983.
- [Pud94] Pudet, T. **Real Time Fitting of Hand-Sketched Pressure Brushstrokes.** Computer Graphics Forum, Vol. 13, Num. 3, P. 205-220, 1994.
- [Sch88] Schneider, P. J. **Phoenix: An interactive curve design system based on the automatic-fitting of hand-sketched curves.** Tese de Mestrado, University of Washington, 1988.
- [Seah05] Seah, H.S. et al. **Artistic brushstroke Representation And Animation With Disk B-Spline Curve.** Advances in Computer Entertainment Technology, P. 88-93, 2005.
- [Str86] Strassmann, S. **Hairy Brushes.** Computer Graphics (SIGGRAPH '86 Proceedings), Vol. 20, Num. 4, P. 225-232, 1986.
- [Su02] Su,S.L et. al. **Simulating Artistic Brushstrokes Using Interval Splines.** The 5th International Conference on Computer Graphics and Imaging, P. 85—90, 2002.

[Wiki] Wikipedia. **Least Squares**. Disponível em <[http://en.wikipedia.org/wiki/Least\\_squares](http://en.wikipedia.org/wiki/Least_squares)>. Último acesso em 23 de Maio de 2008.

[Xu02] Xu, S. et. al. **A Solid Model Based Virtual Hairy Brush**. Computer Graphics Forum, Vol. 21, Num. 3, P. 299-308, 2002.

[Yu03] Yu, Y.J. et. al. **Interactive Rendering Technique for Realistic Oriental Painting**. Journal of WSCG, Vol. 11, 2003.

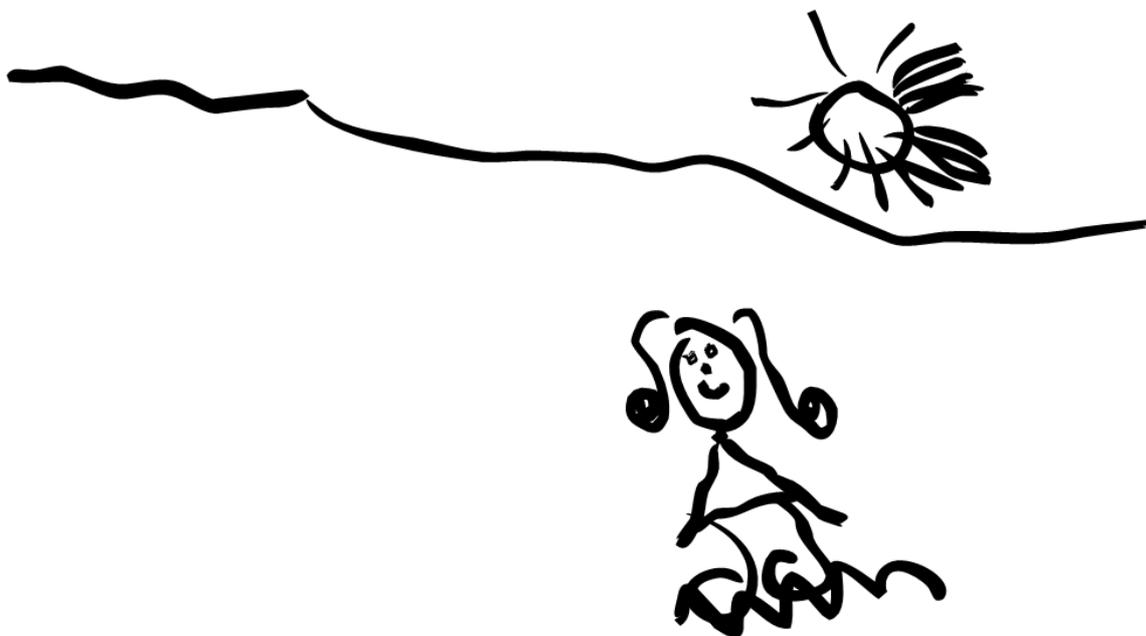


Ilustração por Júlia Pereira, 5 anos.