INBETWEENING PARA ANIMAÇÕES 2D USANDO LINHAS-GUIAS

Leonardo de Oliveira Carvalho

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientador: Ricardo Guerra Marroquim

Rio de Janeiro
Junho de 2016

INBETWEENING PARA ANIMAÇÕES 2D USANDO LINHAS-GUIAS

Leonardo de Oliveira Carvalho

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Ricardo Guerra Marroquim, D.Sc.

Prof. Claudio Esperança, Ph.D.

Prof. Paulo Roma Cavalcanti, D.Sc.

Prof. Emilio Ashton Vital Brazil, D.Sc.

Prof. Luiz Henrique de Figueiredo, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
JUNHO DE 2016

Carvalho, Leonardo de Oliveira

Inbetweening para Animações 2D Usando Linhas-Guias/Leonardo de Oliveira Carvalho. – Rio de Janeiro: UFRJ/COPPE, 2016.

XIII, 84 p.: il.; 29, 7cm.

Orientador: Ricardo Guerra Marroquim

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2016.

Referências Bibliográficas: p. 80 – 84.

1. Inbetweening. 2. Animação 2D. I. Marroquim, Ricardo Guerra. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

A meus pais Neudson e Inêz.

Agradecimentos

Gostaria de agradecer primeiramente ao CNPq, pelo auxílio financeiro que foi fundamental para o desenvolvimento da pesquisa de doutorado.

Aos professores do Laboratório de Computação Gráfica, Ricardo Marroquim pela orientação, Claudio Esperança e Paulo Roma por participarem da banca, além dos professores Antonio de Oliveira (*in memoriam*) e Ricardo Farias.

Aos membros externos da banca: Emilio Vital Brazil, que colaborou significativamente com meu projeto, e Luiz Henrique de Figueiredo, pelos comentários relevantes para a melhoria do trabalho.

Aos animadores Diogo Viegas e Leandro Araujo, que se dispuseram a utilizar o sistema desenvolvido, nos fornecendo resultados e dicas para a melhoria do projeto.

A todos os colegas de laboratório, que compartilharam comigo muitas experiências nesses últimos anos.

Ao professor Luiz Velho pelo suporte durante minha formação. A Lance Williams pela orientação durante meu estágio na Nvidia e pelo apoio durante minha pesquisa de doutorado.

A meus familiares, meus pais Neudson e Inêz, minhas irmãs Larissa e Luciana, e minha esposa Sofia, por todo amor e incentivo aos meus estudos.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

INBETWEENING PARA ANIMAÇÕES 2D USANDO LINHAS-GUIAS

Leonardo de Oliveira Carvalho

Junho/2016

Orientador: Ricardo Guerra Marroquim

Programa: Engenharia de Sistemas e Computação

Em animação 2D tradicional diversos quadros intermediários são desenhados entre dois quadros-chaves, processo conhecido como inbetweening. Esse processo pode ser bastante demorado e tedioso para o animador. Neste trabalho, nós apresentamos um método semi-automático de inbetweening que é fortemente acoplado com a linha de produção de animações. Nós aproveitamos o fato de que tipicamente os artistas iniciam um desenho com um esboço para pré-visualizar a ilustração, e usamos esse esboço como linhas-guias para melhorar a inferência de correspondências. O método é baseado num algoritmo de inferência de correspondências em dois níveis, em que o primeiro nível encontra correspondências entre as linhas-guias, e o segundo entre a arte-final. Essa separação também ajuda o artista usando linhas-guias transformadas e desenhos de quadros anteriores para guiar a criação de novos quadros, funcionando como uma mesa de luz digital. Mostramos que o nosso método é robusto com uma variedade de exemplos, incluindo sequências de livros de animação e animadores profissionais.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

INBETWEENING FOR 2D ANIMATIONS USING GUIDELINES

Leonardo de Oliveira Carvalho

June/2016

Advisor: Ricardo Guerra Marroquim

Department: Systems Engineering and Computer Science

In traditional 2D animation several intermediate frames are drawn between two consecutive key frames. This process can be very time-consuming and tedious for the animator. In this work we present a semi-automatic inbetweening method that is tightly coupled with the animation production pipeline. We exploit the fact that artists typically start with an outline of the drawing to help previewing the illustration, and use these early sketches as guidelines to improve the correspondence inference. The method is based on a two-level matching algorithm: the first level finds correspondences between the outlines, and the second level between the final art. This separation further aids the artist by using transformed outlines and drawings from preceding frames to guide the creation of new ones, acting as a digital light table. We show the robustness of our method with a variety of animation examples, including sequences from animation books and professional animators.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Desenho é uma das mais antigas formas de expressão da humanidade. É utilizado desde a pré-história, com a arte rupestre, até os dias atuais, seja com o auxílio de dispositivos eletrônicos, ou de maneira mais tradicional com lápis e papel. Com desenhos, um artista é capaz de criar uma infinidade de objetos, animais, pessoas ou até seres e mundos extraordinários. É possível contar histórias, transmitir emoções e ideias, estimulando nossos sentimentos. Somos capazes de compreender o significado de desenhos, tanto os feitos de forma minuciosa, com muitos detalhes e precisão, quanto aqueles feitos de forma rudimentar. Assim sendo, essa é uma forma de expressão bastante relevante para a humanidade.

Os artistas podem criar sequências de desenhos com pequenas variações de um desenho para o seguinte, descrevendo assim movimentos. O resultado é conhecido como *desenho animado*, ou simplesmente *animação*. Cada desenho nessas sequências é chamado de *quadro*. É possível reproduzir uma animação pela visualização de cada quadro isoladamente por um breve instante, o que nos fornece a percepção de movimentos. Até o fim do século XIX, diversos dispositivos foram desenvolvidos com o objetivo de reproduzir animações. Um dos dispositivos mais simples é o "flip book", no qual os quadros são organizados nas páginas de um livreto, que pode ser folheado rapidamente para reproduzir a animação. A Figura1.1 ilustra um flip book.

Mais recentemente, com o progresso da tecnologia, surgiram os desenhos animados em filmes e em formato digital. Em todos os casos, o princípio é o mesmo: criar a ilusão de movimentos pela visualização de quadros em sequência.

Quando um artista desenha, ele costuma iniciar com um conjunto de *linhas-guias* que descrevem aproximadamente as formas com o mínimo de detalhes. Essas linhas-guias ajudam o artista a pré-visualizar a arte-final, e a evitar assimetrias no resultado. Além disso, em desenhos animados, uma grande quantidade de quadros é necessária para que ocorra a ilusão de movimento de maneira fluida. Como a diferença entre dois quadros sucessivos é normalmente pequena, é comum iniciar com uma separação maior entre quadros, o que fornece uma ideia sobre os movimentos a

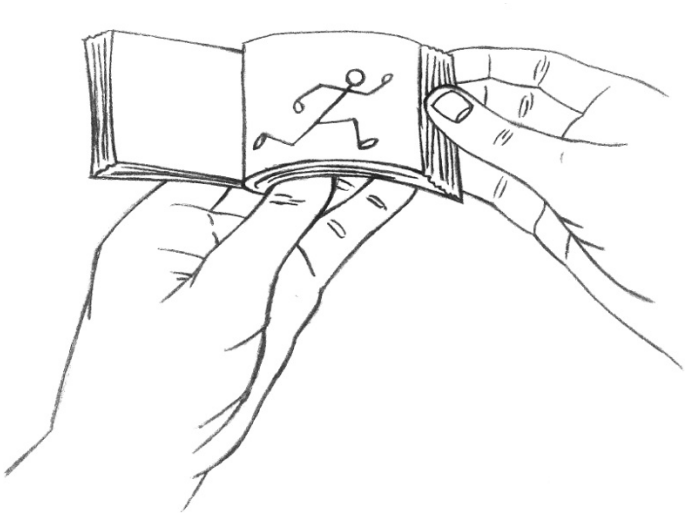


Figura 1.1: Exemplo de flip book.

serem retratados. Esses quadros são chamados de *quadros-chaves*. Posteriormente, os quadros intermediários são desenhados, processo conhecido como *inbetweening*. Todo esse processo costuma ser realizado manualmente, o que requer bastante trabalho [1].

Há certos métodos usados para pré-visualizar uma animação durante a sua criação. Uma prática comum é desenhar os quadros em papéis translúcidos, com auxílio também de mesas de luz para aumentar a translucidez dos papéis, fazendo com que o quadro anterior seja visível durante o desenho do quadro seguinte. Em sistemas de animação auxiliados por computador esse mecanismo costuma ser simulado pela exibição de alguns quadros numa sequência com opacidade variável. De qualquer forma, isso não elimina a necessidade de se criar quadros intermediários.

Para acentuar o problema, com o desenvolvimento de diversos aparelhos de mídia, a taxa de quadros está aumentando, e espera-se que atinja até 120 quadros por segundo em padrões futuros [2]. Esse aumento leva a animações mais suaves à custa da necessidade de uma maior quantidade de quadros. Esse trabalho pode ser reduzido dramaticamente com a ajuda de métodos computacionais, embora isso não seja uma tarefa trivial.

Há quase três décadas, CATMULL [3] já discutia os problemas relacionados à animação auxiliada por computador, afirmando que um dos maiores desafios nessa área é o inbetweening automático. Isso se deve ao fato que desenhos representam personagens tridimensionais projetados num espaço bidimensional, como visualizado pelo animador, o que implica numa perda de informação. Para obter uma solução automática, um programa de computador precisa conhecer um modelo que descreve os personagens da maneira idealizada pelos animadores. Catmull listou algumas abordagens que podem ser usadas para resolver esse problema:

- Inferir informação que falta a partir dos desenhos: O sistema pode buscar elementos conhecidos nos desenhos para inferir traços dos objetos sendo desenhados. Pode levar a erros caso as formas inferidas não correspondam à visão do artista.

- Especificar manualmente os dados que faltam: O animador deve informar manualmente as correspondências entre linhas, e garantir que as linhas invisíveis se tornem visíveis de maneira adequada.

- Quebrar os personagens em camadas: Diferentes partes dos personagens, como braços, pernas, e corpo, podem ser separadas em camadas para serem processadas separadamente. Isso resolve o problema de obstruções em diversos casos, como entre pés num ciclo de caminhada, mas não em casos como uma cabeça rotacionando.

- Usar esqueletos: Esqueletos podem simplificar a entrada de dados e inbetweening. Um personagem pode ser associado a um esqueleto e a animação é feita movendo-se apenas o esqueleto. Esse método funciona para movimentos limitados, não sendo adequado para rotações tridimensionais, mudanças de expressão, e figuras não-rígidas, como roupas.

- Usar esboços tridimensionais: Estendendo a abordagem de esqueletos, figuras podem ser desenhadas em diversos pontos de vista com linhas centrais, definindo um personagem pseudo-tridimensional, cuja forma final pode ser definida de acordo com o posicionamento das linhas centrais.

- Restringir a animação: O sistema pode se limitar a automatizar casos específicos, e o animador pode usar o método tradicional nos casos não-tratáveis pelo sistema.

Os desafios e as abordagens citadas por Catmull ainda são válidos hoje. Em nosso trabalho seguimos mais de uma destas orientações, usamos divisão em camadas, e uma abordagem semelhante à de esqueletos, porém com mais flexibilidade, permitindo a animação de elementos não-rígidos. Há pouca restrição em relação à animação que o sistema é capaz de automatizar, requerendo apenas alguns ajustes no resultado final nos casos não-tratáveis pelo sistema.

Na maioria dos casos em sistemas que fazem inbetweening auxiliado por computador, é necessário especificar correspondências entre curvas de pares de quadros-chaves. Encontrar essas correspondências automaticamente e com precisão é um passo crucial para os métodos de inbetweening. Esse é o problema principal que abordamos. Uma vez estabelecidas, curvas correspondentes podem ser interpoladas para gerar os quadros intermediários. Idealmente a correspondência é um-para-um,

mas na prática existem linhas escondidas, ou casos em que uma linha se divide em duas ou mais de um quadro para o outro.

As linhas-guias são informações relevantes durante a produção de animações. Além de ajudar os artistas a criar os desenhos de cada quadro-chave, elas podem fornecer também dados sobre o movimento que ocorre entre um quadro-chave e outro, o que é útil em métodos computacionais para auxílio da produção de animações. Entretanto nenhum trabalho até este momento utilizou as linhas-guias com este objetivo.

Neste trabalho apresentamos um método que utiliza linhas-guias para auxiliar o processo de criação de uma animação clássica. A Figura 1.2 ilustra um exemplo de entrada e saída de nosso método.

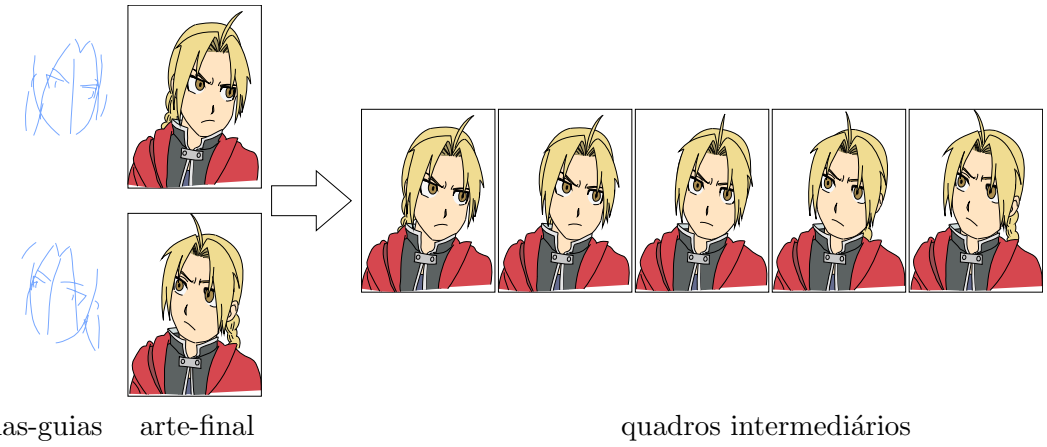


Figura 1.2: Visão geral do método proposto, ilustrando a geração de 5 novos quadros entre dois quadros-chaves. A entrada é dada pelas linhas-guias e arte-final para cada quadro-chave.

Uma vez que o artista encerra as linhas-guias de um novo quadro, o sistema encontra correspondências com os traços das linhas-guias do quadro anterior. Há duas vantagens de se encontrar essa correspondência inicial: primeiro, ela permite modificar a arte-final do quadro anterior para guiar o desenho da arte-final do novo quadro, funcionando como uma mesa de luz digital aprimorada; segundo, ela melhora bastante a inferência de correspondências entre traços da arte final, permitindo maiores movimentos e distorções entre os quadros.

Após computar as correspondências finais, quadros intermediários podem ser gerados em qualquer quantidade por meio da interpolação de traços. O artista pode também criar camadas para melhor organizar os desenhos e ajudar ainda mais a inferência de quadros intermediários.

O objetivo de nosso método não é substituir completamente o animador humano, mas servir como auxílio durante a produção da animação e reduzir drasticamente

tarefas repetitivas. O artista pode também fazer ajustes e correções para ajuste fino dos resultados.

Como já mencionado acima, os animadores costumam desenhar linhas-guias antes dos desenhos finais que compõem a animação. Nossa maior contribuição é um método que explora esse fato para:

- dar ao animador dicas visuais de como desenhar o próximo quadro-chave;
- gerar automaticamente os quadros intermediários;
- permitir grandes deslocamentos;
- fornecer controle sobre obstruções.

Capítulo 2

Revisão Bibliográfica

Há diversos trabalhos cujo objetivo é inbetweening automático (ou semi-automático) de desenhos animados em 2D. Os primeiros trabalhos relacionados surgiram há mais de quarenta anos [4], [5]. Nesta seção, serão discutidos os trabalhos mais diretamente relacionados com o trabalho que desenvolvemos.

BURTNYK e WEIN [6] utilizam esqueletos para guiar o movimento das formas finais no processo de inbetweening. Os esqueletos são formados por sequências de segmentos de retas. Os traços da arte-final são descritos em relação a sistemas de coordenadas definidos pelos esqueletos. Para gerar as posições intermediárias, os esqueletos são interpolados linearmente e depois suavizados, os quadros-chaves são então interpolados nas posições resultantes. Os esqueletos possuem configuração simples, sendo apenas linhas poligonais, e os resultados são limitados a animações simples.

O trabalho de REEVES [7] apresenta um método para inbetweening que melhora o controle sobre a interpolação. Além dos quadros-chaves, o animador especifica um conjunto de restrições chamadas de pontos móveis, que consistem em curvas no espaço e tempo restringindo a trajetória e a dinâmica de determinados pontos nos quadros-chaves. As correspondências entre curvas dos quadros-chaves são definidas de acordo com os pontos móveis, de forma que duas curvas em diferentes quadros-chaves são correspondentes caso elas contenham o mesmo ponto móvel. O animador pode inserir diversos pontos móveis para controlar o resultado. O conjunto de restrições define *patches*, que são avaliados em cada quadro intermediário usando um de três métodos de interpolação descritos pelo autor. Apesar de prover um bom controle sobre a animação, o método exige um grande trabalho manual.

Alguns requisitos para um sistema de animação 2D auxiliado por computador foram discutidos por DURAND [8]. A abordagem preferida pelo autor dentre as citadas por CATMULL [3] foi restringir a classe de animações que podem ser feitas. Os traços são vetorizados a partir de imagens matriciais. O animador identifica manualmente correspondências entre as curvas de dois quadros-chaves. As curvas

correspondentes são então interpoladas usando interpolação de splines usando um número variável de incrementos no tempo.

SEDERBERG e GREENWOOD [9] estudam como transformar suavemente duas formas bidimensionais poligonais, usando um método baseado em um modelo físico, em que as figuras são trabalhadas como se fossem formadas por arames. A primeira figura é dobrada ou esticada para ter a forma da segunda figura com o menor trabalho possível. A solução tende a associar regiões semelhantes. O usuário pode controlar atributos físicos associados ao nível de flexibilidade do arame. São adicionadas penalidades para evitar auto-interseções. Os usuários devem definir manualmente pares de pontos correspondentes entre as duas figuras. O método depende de uma distribuição razoável de pontos correspondentes. Os resultados das transformações contêm elementos indesejáveis, como o encurtamento de partes dos objetos, pois os pontos correspondentes são interpolados linearmente. A Figura 2.1 ilustra um resultado de [9], onde a silhueta de uma vaca é transformada na silhueta de um alce.

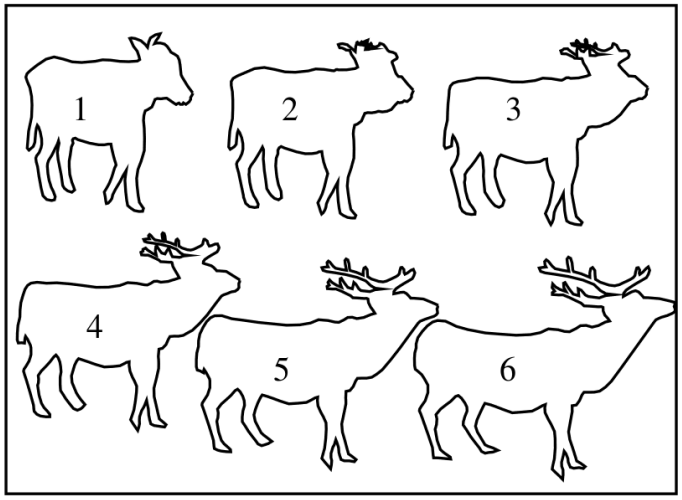


Figura 2.1: Figura de [9], mostrando a transformação da silhueta de uma vaca (1) para um alce (6). Figura reproduzida com permissão do autor.

PATTERSON [10] faz uma nova análise do problema de animação auxiliada por computador, na qual ele reforça que o maior problema nesse segmento é o inbetweening automático, dividindo-o em dois sub-problemas: como as silhuetas se modificam e como várias partes dos objetos obstruem umas às outras. Os componentes chaves indicados para se resolver esses problemas são: o método de interpolação; controle de continuidade; e um modelo de exibição hierárquico (HDM - *hierarchy display model*).

TicTacToon [11] é um sistema para animação 2D profissional sem utilizar papel, no qual desenhos são feitos diretamente no sistema, e quadros intermediários podem ser gerados automaticamente. Esse é o primeiro sistema de animação a usar desenhos vetoriais, utilizando um método próprio para obter traços a partir de canetas

com sensibilidade de pressão. No artigo os autores comentam sobre as vantagens e desvantagens de diversos elementos utilizados no processo de criação de animações 2D usando o sistema, incluindo inbetweening automático. Entretanto, os próprios autores afirmam que o processo automático é tão lento quanto fazer todos os quadros manualmente.

No trabalho de XIE [12] são usadas transformações afins para encontrar correspondência entre figuras bidimensionais, gerando então quadros intermediários. Entretanto esse método não é capaz de tratar casos em que há uma maior distorção entre os quadros, e não trata de obstruções.

REETH [13] desenvolve RUFIAS, um sistema de animação $2^1/_2$D contendo uma interface que permite a aplicação de diversos métodos de especificação de animação, incluindo uma técnica de especificação de movimento, e animação baseada em curvas de movimento. A aplicação possui também métodos para desenho de personagens, como um método baseado em esqueletos, os quais podem ser utilizados para criar animações. As animações são controladas por diversos parâmetros definidos pelos animadores pela interface do programa.

FIORE *et al.* [14] desenvolvem um método de inbetweening automático baseado em técnicas de modelagem e animação $2^1/_2$D, tratando problemas de auto-obstruções e mudanças nas silhuetas. A abordagem é dividida em quatro níveis: o nível 0 contém primitivas básicas de desenhos bidimensionais (curvas); no nível 1 há estruturas de modelagem $2^1/_2$D; no nível 2 esqueletos fornecem informação tridimensional; o nível 3 contém ferramentas de deformação em alto nível. Embora o método trate de problemas no inbetweening, o usuário precisa manipular pontos de controle pouco intuitivos usados na representação das curvas, definir manualmente correspondências, e especificar a região de influência dos esqueletos.

Uma técnica para captura e transferência de movimentos a partir de desenhos animados foi desenvolvida por BREGLER *et al.* [15]. Onde o movimento de figuras em desenhos animados é rastreado e transferido para modelos tridimensionais, desenhos bidimensionais e fotografia. Uma animação é transformada em uma representação de movimentos. Movimentos não-rígidos são rastreados usando uma combinação de transformação afim e inbetweening de formas-chaves, utilizando uma função de deformação não-linear. Essa informação de movimento é traduzida para outros formatos, gerando resultados com animações com os mesmos movimentos mas aparência diferente.

KORT [16] apresenta um algoritmo para inbetweening auxiliado por computador e sua integração em uma interface gráfica baseada em caneta digital. O algoritmo funciona com separação em camadas para reduzir o problema de ocultação. O conteúdo de cada quadro-chave é analisado e classificado em traços, cadeias de traços e relações entre eles. Determinadas regras são usadas para definir quais partes

dos desenhos são correspondentes utilizando uma função de custo. Os quadros intermediários são gerados pela interpolação de traços usando uma transformação de Coons [17]. Os artistas podem editar os resultados por meio da interface gráfica, corrigindo correspondências erradas, trajetórias e timing. Entretanto as animações eram restritas a estilos bem simples, como animações de recortes. A Figura 2.2 mostra um exemplo de inbetweening sendo editado na interface desenvolvida.

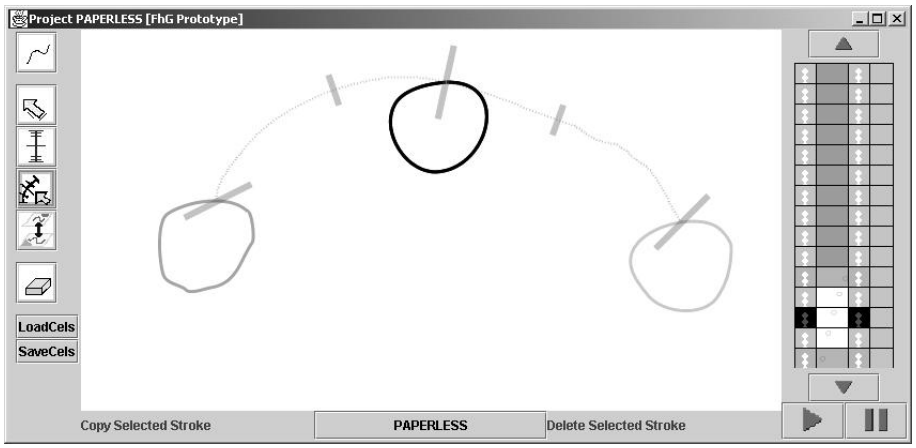


Figura 2.2: Figura de [16] ilustrando a interface desenvolvida e exemplo de inbetweening. Figura reproduzida com permissão do autor.

No trabalho de MELIKHOV *et al.* [18] foi desenvolvido um sistema que modela quadros-chaves que são interpolados em diversas camadas, preservando o estilo artístico dos desenhos feitos a mão, com um movimento suave e natural. O método utiliza esqueletos para deformar a textura ao redor de linhas desenhadas à mão. Os traços das imagens são vetorizadas por meio de linhas, que formam os polígonos que definem os esqueletos das figuras. Cada quadro é representado por um grafo, que é utilizado para encontrar as correspondências entre as figuras. A interpolação é efetuada sobre os polígonos, curvas e texturas, usando um método diferente para cada caso.

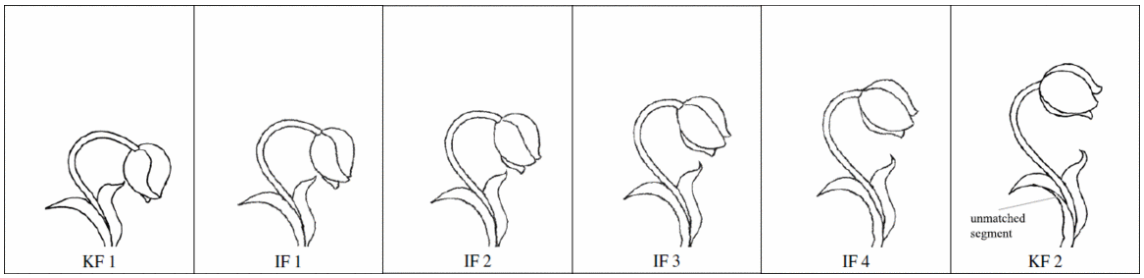


Figura 2.3: Animação de uma flor. Resultado obtido em [18]. Os quadros-chaves KF1 e KF2 são interpolados em quatro quadros intermediários. Um traço sem correspondência é destacado no quadro-chave KF2. Figura reproduzida com permissão do autor.

DE JUAN e BODENHEIMER [19] descrevem duas técnicas semi-automáticas que permitem o reuso de animações criadas previamente. Uma técnica para segmentar imagens de desenhos animados dos planos de fundo, e outra para calcular texturas e contornos usados no inbetweening pela modelagem de superfícies implícitas utilizando funções de base radial (RBF) e de um algoritmo de registro elástico não-rígido. Os personagens devem ser manualmente subdivididos em camadas. As figuras devem estar alinhadas adequadamente para que a interpolação funcione, caso contrário podem surgir artefatos na animação resultante.

SÝKORA *et al.* [20] apresentam um método para registro de imagens o mais rígido o possível útil para figuras articuladas. O método é utilizado para diversas tarefas que são comuns no processo de produção de animações, como inbetweening, pintura automática e redirecionamento de poses. O método não é adequado para animações que envolvem deformações mais complexas do que movimentos rígidos.

WHITED *et al.* [21] apresentam uma ferramenta para inbetweening estreito, i.e., com quadros-chaves semelhantes, no qual foi desenvolvido um conjunto de técnicas semi-automáticas guiadas pelo usuário, minimizando a intervenção do artista, que pode informar ao sistema dados sobre os desenhos que não tenham sido inferidos corretamente pelo processo automático. Os autores também apresentam uma técnica para interpolação de formas, na qual o movimento de traços é construído por meio de trajetórias de vértices em espirais logarítmicas.

NORIS *et al.* [22] apresenta um método para eliminar ruídos temporais de animações esboçadas, utilizando de uma combinação de extração de movimentos e técnicas de inbetweening, preservando o estilo esboçado dos quadros de entrada. A quantidade de ruídos é controlada por um valor de parâmetro contínuo. A entrada desse trabalho é uma animação completa, i.e. contendo todos os quadros, dentre os quais são escolhidos os quadros-chaves. O método funciona em duas etapas: na primeira é criada uma animação livre de ruídos a partir da sequência de entrada, isso é feito interpolando-se quadros representativos; Na segunda etapa os quadros gerados são interpolados com os quadros da animação de entrada. Ao definir o fator de interpolação usado dessa etapa, o usuário pode controlar o nível de redução de ruídos, de forma que 0% corresponde à animação de entrada (sem redução de ruídos), e 100% corresponde à animação completamente livre de ruídos.

YU *et al.* [23] trabalham em um método semi-supervisionado para encontrar correspondências entre objetos complexos em animação bidimensional. Cada ponto de um objeto possui um conjunto de dados que descrevem a geometria local na vizinhança do ponto. Esses dados são alinhados em um espaço de dimensão mais baixa, e a busca por correspondências é feita por clusterização. O usuário pode definir pontos que devem ou não ser ligados para melhorar o desempenho da busca por correspondências.

Um algoritmo de inferência de correspondências automática foi desenvolvido por SONG *et al.* [24], usando a relação entre pontos característicos nas curvas de entrada como parâmetro para as correspondências. Nesse algoritmo, a correspondência entre desenhos é formulada como um problema de rotulação muitos-para-muitos, e é construída usando um descritor local de formas, obtendo invariância a escala e rotação. Um esquema de otimização iterativa é adotado para refinar os resultados das correspondências. O método pode ser aplicado em sistemas de inbetweening e colorização automáticos.

DALSTEIN *et al.* [25] criaram uma estrutura de dados para controlar animações com topologia variável no tempo. Isso permite mudanças como fusões, divisões, surgimentos e desaparecimentos de objetos utilizando informações topológicas dos quadros, sendo possível controlar essas mudanças no espaço e no tempo.

Recentemente XING *et al.* [26] desenvolveram um sistema para auto-completar animações desenhadas a mão. Esse trabalho é fortemente relacionado ao trabalho que desenvolvemos, pois nele é utilizada informação de quadros anteriores para auxiliar o artista a desenhar o quadro atual. Nosso sistema é semelhante, mas em nosso caso utilizamos a informação proveniente das linhas-guias para definir as transformações entre quadros-chaves, além disto, o artista pode organizar a animação de forma hierárquica, separando linhas-guias e arte-final, e gerenciando camadas, o que dá ao usuário um controle maior sobre os elementos da animação, podendo tratar melhor obstruções e grandes deslocamentos.

Capítulo 3

Base teórica

Um aspecto importante em sistemas de animação é a representação do desenho. Em sistemas ditos matriciais, um desenho é dado por uma imagem digital contendo uma quantidade fixa de linhas e colunas de pontos coloridos (pixels). Esse tipo de representação é útil para se trabalhar com figuras com uma grande quantidade de detalhes como fotografias, porém é pouco adequado quando se deseja modificar as formas representadas, pois é preciso extrair informações sobre a estrutura das figuras para gerar outras imagens de maneira coerente. Outra possibilidade é a representação vetorial, na qual cada imagem é dada por um conjunto de elementos definidos por meio de atributos que definem suas formas. A representação vetorial facilita a manipulação de figuras por meio de alterações de seus atributos, sendo portanto mais conveniente para ser usada em sistemas de animação 2D auxiliados por computador. Uma figura em formato vetorial normalmente é convertida para o formato matricial a fim de ser exibida em vídeo, esse processo é conhecido como rasterização. Neste trabalho, trabalhamos com imagens em representação vetorial, na qual figuras são definidas por curvas e regiões bidimensionais. As próximas seções detalham esses elementos.

3.1 Curvas planas

Seja $\mathbb{R}^2$ o conjunto de pares ordenados de números reais. Dizemos que uma aplicação contínua $\alpha : I \to \mathbb{R}^2$, com $I \subset \mathbb{R}$, é uma curva contínua em $\mathbb{R}^2$, conforme definido em [27]. Podemos entender intuitivamente uma curva plana contínua como uma figura que pode ser desenhada sem tirar o lápis do papel. Um desenho mais complexo pode ser feito por meio de um conjunto de curvas planas.

3.2 Curvas de Bézier cúbicas

Para se trabalhar com curvas no computador é necessário utilizar uma classe de curvas que possua uma representação finita e que represente com fidelidade a intenção do artista. Há diversas formas possíveis, como curvas poligonais, curvas de Bézier e NURBS [28]. Uma maneira bastante comum em sistemas de criação e edição de figuras vetoriais é por meio de curvas de Bézier cúbicas: dados os pontos de controle $c[0], \ldots, c[3] \in \mathbb{R}^2$, uma curva de Bézier cúbica é dada por:

$$c(t) = (1-t)^3 c[0] + 3(1-t)^2 t c[1] + 3(1-t)t^2 c[2] + t^3 c[3],\ t \in [0,1]. \tag{3.1}$$

A Figura 3.1 contém alguns exemplos de curvas de Bézier cúbicas.

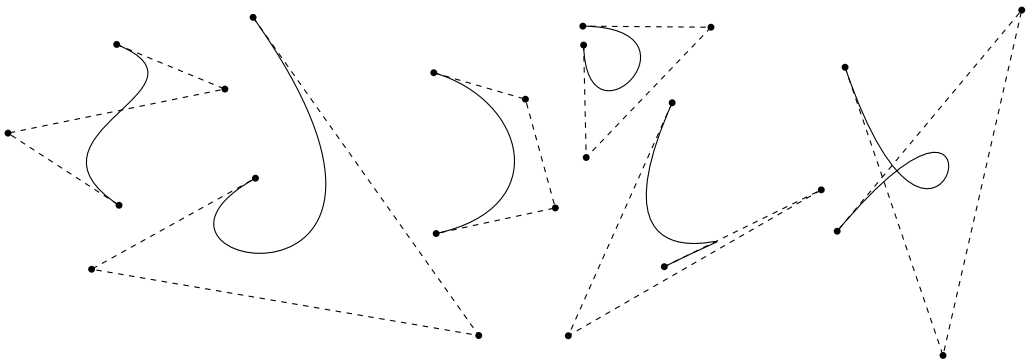


Figura 3.1: Exemplo de curvas de Bézier cúbicas. Os pontos de controle são demarcados por círculos pretos. As linhas pontilhadas são chamadas de polígonos de controle.

Este tipo de curva é conveniente para os artistas, pois permite que esses controlem as formas geradas manipulando os pontos de controle. O primeiro e último ponto de controle indicam por onde a curva passa, e os outros dois são utilizados para definir a tangente da curva nesses pontos. Isso pode ser verificado analisando-se a definição acima: é fácil observar que uma curva de Bézier cúbica interpola os pontos $c[0]$ (em $c(0)$) e $c[3]$ (em $c(1)$). A tangente da curva no ponto $c(t)$ é dada pela derivada de c em t, ou seja:

$$c'(t) = -3(1-t)^2 c[0] + 3(3t^2 - 4t + 1)c[1] + 3(2t - 3t^2)c[2] + 3t^2 c[3].$$

Assim, em $c[0]$ a tangente é dada por $c'(0) = 3(c[1] - c[0])$, e em $c[3]$ a tangente é $c'(1) = 3(c[3] - c[2])$. Logo os pontos $c[0]$ e $c[3]$ podem ser usados para se definir por onde a curva deve passar, e $c[1]$ e $c[2]$ para manipular tangentes, controlando assim o formato da curva.

3.2.1 Algoritmo de De Casteljau

Dado um parâmetro $t \in [0,1]$ é possível calcular o ponto $c(t)$ utilizando o algoritmo de De Casteljau. Nesse algoritmo, são realizadas sucessivas interpolações lineares de pontos até encontrar um ponto sobre a curva. No caso de curvas de Bézier cúbicas esse processo é definido por:

$$
\begin{aligned}
&c^{(0)}[i] := c[i] \ , \ i = 0, \ldots, 3 \\
&c^{(j)}[i] := c^{(j-1)}[i](1-t) + c^{(j-1)}[i+1]t \ , \ i = 0, \ldots, 3-j \ , \ j = 1, \ldots, 3.
\end{aligned}
$$

Fazendo isso teremos $c(t) = c^{(3)}[0]$. De fato, temos:

$$
\begin{aligned}
c^{(1)}[0] &= (1-t)c[0] + tc[1], \\
c^{(1)}[1] &= (1-t)c[1] + tc[2], \\
c^{(1)}[2] &= (1-t)c[2] + tc[3], \\
c^{(2)}[0] &= (1-t)c^{(1)}[0] + tc^{(1)}[1] \\
&= (1-t)((1-t)c[0] + tc[1]) + t((1-t)c[1] + tc[2]) \\
&= (1-t)^2 c[0] + 2(1-t)tc[1] + t^2 c[2], \\
c^{(2)}[1] &= (1-t)c^{(1)}[1] + tc^{(1)}[2] \\
&= (1-t)((1-t)c[1] + tc[2]) + t((1-t)c[2] + tc[3]) \\
&= (1-t)^2 c[1] + 2(1-t)tc[2] + t^2 c[3], \\
c^{(3)}[0] &= (1-t)c^{(2)}[0] + tc^{(2)}[1] \\
&= (1-t)((1-t)^2 c[0] + 2(1-t)tc[1] + t^2 c[2]) + t((1-t)^2 c[1] + 2(1-t)tc[2] + t^2 c[3]) \\
&= (1-t)^3 c[0] + 3(1-t)^2 tc[1] + 3(1-t)t^2 c[2] + t^3 c[3] \\
&= c(t).
\end{aligned}
$$

3.2.2 Transformações

Dada uma curva de Bézier cúbica c, podemos obter uma nova curva Tc aplicando-se uma transformação $T : \mathbb{R}^2 \to \mathbb{R}^2$ ao conjunto imagem de c. Dependendo da transformação T, o resultado Tc pode ou não ser descrito como uma curva de Bézier cúbica.

Em particular, caso T seja um operador linear, i.e., $T(a+b) = T(a) + T(b)$ e

$T(\lambda a) = \lambda T(a)$, onde $\lambda \in \mathbb{R}$, e $a, b \in \mathbb{R}^2$, teremos então:

$$\begin{aligned} Tc(t) &= T((1-t)^3 c[0] + 3(1-t)^2 t c[1] + 3(1-t)t^2 c[2] + t^3 c[3]) \\ &= T((1-t)^3 c[0]) + T(3(1-t)^2 t c[1]) + T(3(1-t)t^2 c[2]) + T(t^3 c[3]) \\ &= (1-t)^3 T(c[0]) + 3(1-t)^2 t T(c[1]) + 3(1-t)t^2 T(c[2]) + t^3 T(c[3]). \end{aligned}$$

Ou seja, Tc é a curva de Bézier cúbica cujos pontos de controle são obtidos pela aplicação de T aos pontos de controle de c.

O mesmo vale quando T é um operador afim, ou seja, quando esse pode ser escrito como $T(a) = L(a) + d$, onde L é um operador linear e $d \in \mathbb{R}^2$. De fato, teremos:

$$\begin{aligned} Tc(t) =& L((1-t)^3 c[0] + 3(1-t)^2 t c[1] + 3(1-t)t^2 c[2] + t^3 c[3]) + d \\ =& (1-t)^3 L(c[0]) + 3(1-t)^2 t L(c[1]) + 3(1-t)t^2 L(c[2]) + t^3 L(c[3]) + d \\ =& (1-t)^3 L(c[0]) + (1-t)^3 d + 3(1-t)^2 t L(c[1]) + (1-t)^2 t d \\ &+ 3(1-t)t^2 L(c[2]) + (1-t)t^2 d + t^3 L(c[3]) + t^3 d \\ =& (1-t)^3 (L(c[0]) + d) + 3(1-t)^2 t (L(c[1]) + d) \\ &+ 3(1-t)t^2 (L(c[2]) + d) + t^3 (L(c[3]) + d) \\ =& (1-t)^3 T(c[0]) + 3(1-t)^2 t T(c[1]) + 3(1-t)t^2 T(c[2]) + t^3 T(c[3]). \end{aligned}$$

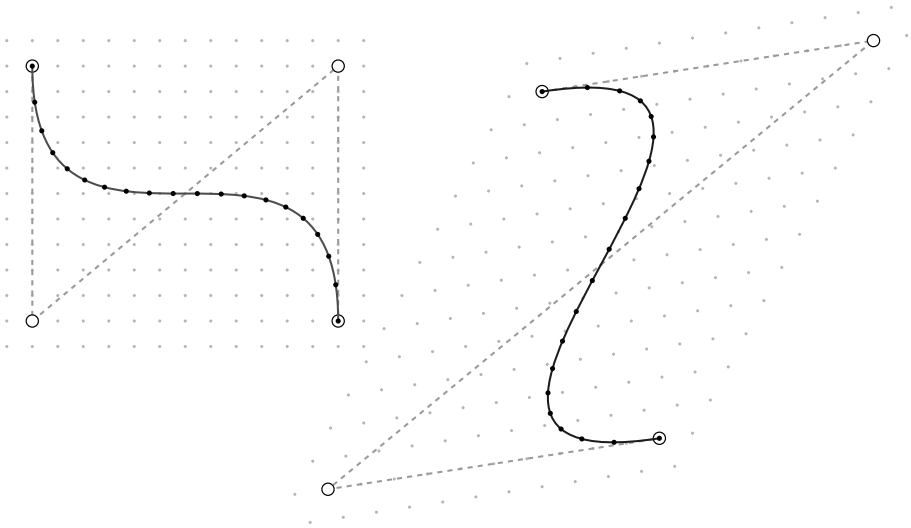


Figura 3.2: Transformação afim aplicada a uma curva de Bézier cúbica

A Figura 3.2 ilustra um exemplo de transformação afim aplicada a uma curva de Bézier cúbica. Uma transformação afim T foi aplicada aos pontos de controle de uma curva de Bézier cúbica à esquerda, gerando a curva à direita. A transformação T também foi aplicada a pontos sobre a curva original (pontos pretos), onde o

resultado são pontos sobre a curva à direita. Ao fundo podem ser vistos pontos menores distribuídos numa grade regular, a transformação T também foi aplicada a esses pontos, servindo como referência para a transformação.

Entretanto, em geral não há garantia de que se aplicarmos uma transformação qualquer T aos pontos de controle de uma curva de Bézier teremos o mesmo resultado da aplicação de T a pontos da curva original. A Figura 3.3 ilustra a aplicação de uma transformação não-linear aos pontos de controle e a pontos sobre a mesma curva da figura anterior.

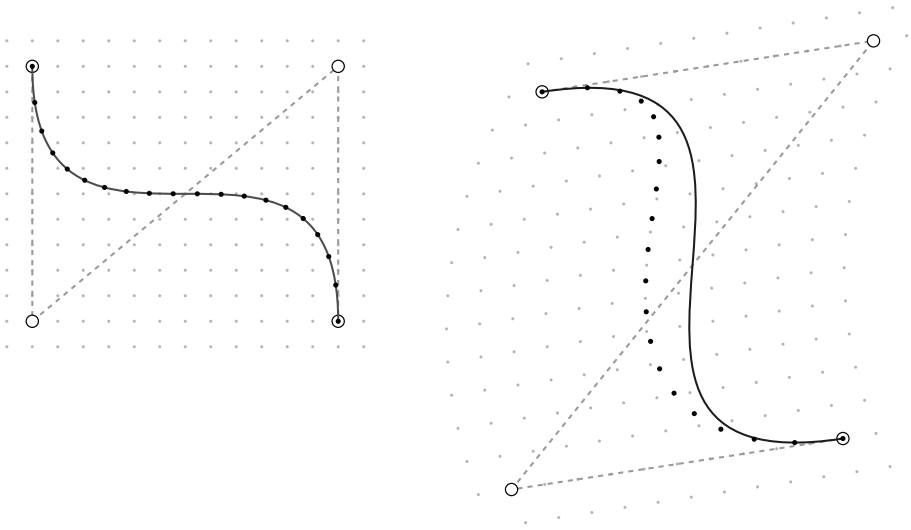


Figura 3.3: Transformação não-linear aplicada a uma curva de Bézier cúbica.

Porém é possível encontrar uma curva de Bézier cúbica $\hat{c}$ cuja imagem seja uma aproximação da imagem de Tc. Sejam $0 \leq t_1 < t_2 < \cdots < t_N \leq 1$, para um determinado número de amostras N, podemos calcular os pontos de controle de $\hat{c}$ de forma a minimizar a seguinte soma: $S = \sum_{i=1}^{N} \|\hat{c}(t_i) - T(c(t_i))\|^2$, isto é, encontrar uma solução em termos de mínimos quadrados. Vamos fixar os pontos $\hat{c}[0]$ e $\hat{c}[3]$ como $T(c[0])$ e $T(c[3])$, respectivamente (isso será conveniente quando trabalharmos com curvas de Bézier compostas, na Seção 3.2.5), e calcular $\hat{c}[1]$ e $\hat{c}[2]$ minimizando S. O termo $\|\hat{c}(t_i) - T(c(t_i))\|$ pode ser escrito como $\|A_i x - b_i\|$, onde

$$A_i = \begin{bmatrix} 3(1-t_i)^2 t_i & 0 & 3(1-t_i)^2 t_i & 0 \\ 0 & 3(1-t_i)^2 t_i & 0 & 3(1-t_i)^2 t_i \end{bmatrix}, \qquad x = \begin{bmatrix} \hat{c}[1] \\ \hat{c}[2] \end{bmatrix},$$

$$b_i = \Big[T(c(t_i)) - (1-t_i)^3 \hat{c}[0] - t_i^3 \hat{c}[3] \Big].$$

Onde as coordenadas de cada ponto de $\mathbb{R}^2$ são dispostas na forma de uma matriz coluna. Ou seja x é uma matriz de dimensões 4×1 e b_i uma matriz de dimensões 2×1.

Podemos escrever então: $S = \|Ax - b\|^2$, onde

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_N \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}.$$

A solução do problema de mínimos quadrados é dada por $x = (A^\top A)^{-1} A^\top b$. Como a matriz A depende apenas dos valores t_i, ela pode ser reaproveitada para outras curvas, desde que esses valores sejam mantidos. Por exemplo é possível armazenar apenas o valor de $(A^\top A)^{-1} A^\top$, e em cada caso calculamos apenas o produto desta matriz por b, reduzindo assim o custo de sua computação.

Também é possível obter a solução por meio de decomposição em valores singulares [29], na qual matrizes obtidas pela decomposição de A são utilizadas para resolver o problema de mínimos quadrados. Neste caso podemos decompor A apenas uma vez e utilizar o resultado em múltiplos casos.

Em nosso sistema utilizamos sempre 10 valores para t_i, variando de 0 a 1. Assim sendo, foi possível pré-processar a matriz A para obtermos os dados necessários para a resolução do problema de mínimos quadrados. Testamos a decomposição em valores singulares e o produto $(A^\top A)^{-1} A$, não havendo nenhuma mudança significativa entre essas abordagens no resultado final.

A Figura 3.4 mostra o resultado dessa minimização para a curva e transformação usadas na Figura 3.3.

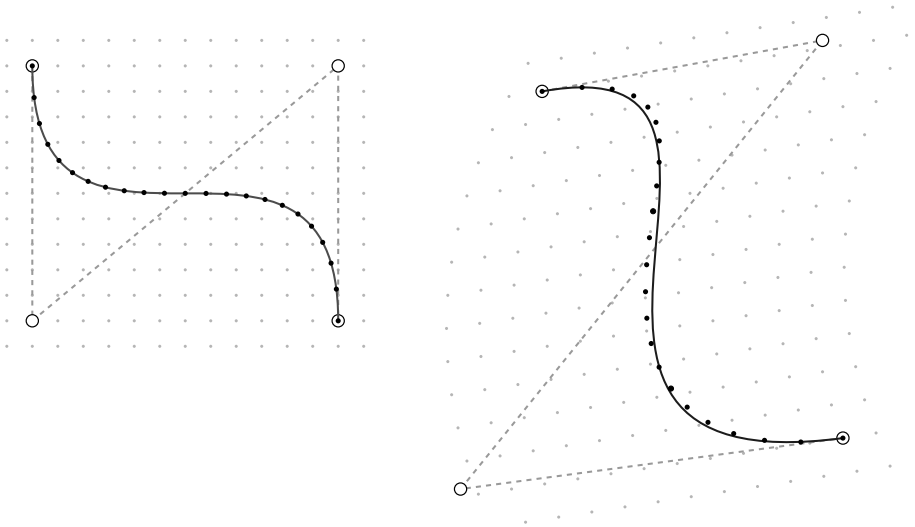


Figura 3.4: Curva de Bézier aproximando pontos após transformação não-linear.

É possível aperfeiçoar esse procedimento de forma que a curva resultante seja ainda mais próxima do resultado Tc, como será descrito na Seção 3.2.9.

3.2.3 Corte de curvas

Dado um parâmetro $t_a \in (0,1)$, uma curva de Bézier cúbica c pode ser recortada de forma a gerar duas curvas c_{0a} e c_{a1}, onde a imagem de c_{0a} é equivalente ao subconjunto da imagem de c ao se utilizar parâmetros entre 0 e t_a, e a imagem de c_{a1} equivale ao subconjunto de c com parâmetros entre t_a e 1. As curvas c_{0a} e c_{a1} também podem ser definidas por curvas de Bézier cúbicas. Nesta seção, veremos como encontrar os pontos de controle que definem estas curvas a partir dos pontos de controle de c.

Sejam $c_{0a}[0], c_{0a}[1], c_{0a}[2], c_{0a}[3] \in \mathbb{R}^2$ os pontos de controle que definem a curva c_{0a}, que é equivalente à curva c com parâmetro $t \leq t_a$. Esses pontos podem ser obtidos pelo algoritmo de De Casteljau (Seção 3.2.1) utilizando o parâmetro t_a, bastando fazer $c_{0a}[i] = c^{(i)}[0]$, para $i = 0, \ldots, 3$.

Fazendo isso teremos $c_{0a}(t) = c(t \cdot t_a)$. De fato:

$$\begin{aligned}
c_{0a}(t) =& (1-t)^3 c_{0a}[0] + 3(1-t)^2 t c_{0a}[1] + 3(1-t)t^2 c_{0a}[2] + t^3 c_{0a}[3] \\
=& (1-t)^3 c[0] \\
&+ 3(1-t)^2 t((1-t_a)c[0] + t_a c[1]) \\
&+ 3(1-t)t^2((1-t_a)^2 c[0] + 2(1-t_a)t_a c[1] + t_a^2 c[2]) \\
&+ t^3((1-t_a)^3 c[0] + 3(1-t_a)^2 t_a c[1] + 3(1-t_a)t_a^2 c[2] + t_a^3 c[3]) \\
=& \alpha_0 c[0] + \alpha_1 c[1] + \alpha_2 c[2] + \alpha_3 c[3],
\end{aligned}$$

onde

$$\begin{aligned}
\alpha_0 &= (1-t)^3 + 3(1-t)^2(1-t_a)t + 3(1-t)t^2(1-t_a)^2 + t^3(1-t_a)^3 \\
&= ((1-t) + t(1-t_a))^3 \\
&= (1-tt_a)^3, \\
\alpha_1 &= 3(1-t)^2 t t_a + 3(1-t)t^2\,(2t_a(1-t_a)) + 3t^3(1-t_a)^2 t_a \\
&= 3tt_a\left((1-t)^2 + 2(1-t)t(1-t_a) + t^2(1-t_a)^2\right) \\
&= 3tt_a(1-tt_a)^2, \\
\alpha_2 &= 3(1-t)t^2 t_a^2 + 3t^3(1-t_a)t_a^2 \\
&= 3(tt_a)^2(1-t+t(1-t_a)), \\
&= 3(tt_a)^2(1-tt_a), \\
\alpha_3 &= (tt_a)^3.
\end{aligned}$$

Logo,

$$\begin{aligned} c_{0a}(t) &= (1-tt_a)^3 c[0] + 3tt_a(1-tt_a)^2 c[1] + 3(tt_a)^2(1-tt_a)c[2] + (tt_a)^3 c[3] \\ &= c(tt_a). \end{aligned}$$

Fazendo t variar no intervalo $[0,1]$, o valor de $t \cdot t_a$ varia no intervalo $[0,t_a]$, e portanto a curva c_{0a} equivale à porção da curva c com parâmetro menor do que ou igual a t_a.

Os pontos de controle da curva c_{a1} podem ser facilmente obtidos aplicando-se o mesmo procedimento para encontrar o corte da curva inversa $\tilde{c}$ com parâmetros de 0 até $1-t_a$ utilizando-se um cálculo semelhante ao utilizado para calcular c_{0a}, obtendo $c_{a1}[i] = c^{(3-i)}[i]$, para $i = 0, \ldots, 3$. De fato, efetuando o algoritmo de De Casteljau com os pontos de controle em ordem reversa $(c[3], c[2], c[1], c[0])$, com parâmetro $1-t_a$ e revertendo os pontos no final para retornar à orientação original, teremos:

$$\begin{aligned} c_{a1}[3] &= c[3] = c^{(0)}[3], \\ c_{a1}[2] &= t_a c[3] + (1-t_a)c[2] = c^{(1)}[2], \\ c_{a1}[1] &= t_a^2 c[3] + 2t_a(1-t_a)c[2] + (1-t_a)^2 c[1] = c^{(2)}[1], \\ c_{a1}[0] &= t_a^3 c[3] + 3t_a^2(1-t_a)c[2] + 3t_a(1-t_a)^2 c[1] + (1-t_a)^3 c[0] = c^{(3)}[0]. \end{aligned}$$

Podemos ainda combinar esses resultados e encontrar a curva c_{ab} equivalente à curva c entre os parâmetros t_a e t_b, onde $0 \leq t_a < t_b \leq 1$. Para isso seja c_{0b} o corte da curva c entre 0 e t_b. Como $c_{0b}(t) = c(t \cdot t_b)$, teremos $c_{0b}(t_{a'}) = c(t_a)$ quando $t_{a'} = t_a/t_b$. A curva c_{ab} pode ser obtida então pelo corte de c_{0b} entre os parâmetros $t_{a'}$ e 1. A Figura 3.5 mostra exemplos de cortes de uma curva de Bézier cúbica.

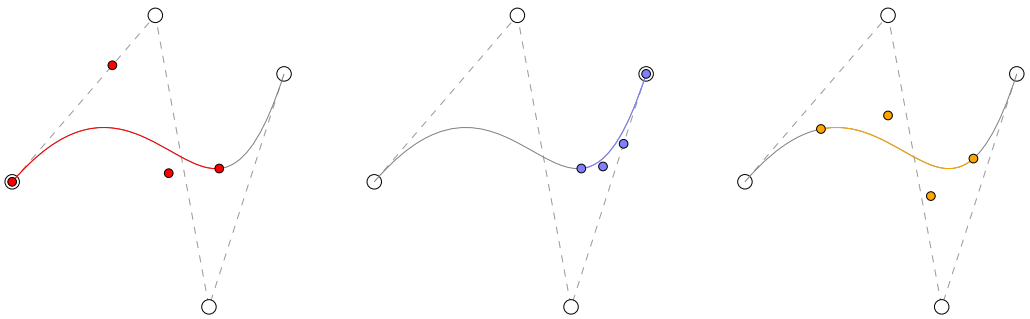


Figura 3.5: Corte de uma curva de Bézier cúbica (na cor cinza, com pontos de controle representados por círculos grandes, polígono de controle com linhas tracejadas). À esquerda, em vermelho: corte da curva até o parâmetro $t_a = 0.7$. Ao centro, em azul: corte da curva a partir do parâmetro $t_a = 0.7$. À direita, em laranja: corte da curva entre os parâmetros $t_a = 0.2$ e $t_b = 0.8$.

3.2.4 Comprimento de arco

O comprimento de arco de uma curva de Bézier cúbica c é dado por

$$l = \int_0^1 \|c'(t)\| dt,$$

Entretanto, não há solução fechada para essa integral, mas podemos recorrer a algum método numérico para calculá-la. Em [30] foi descrito um algoritmo para calcular uma aproximação do comprimento de arco utilizando-se apenas o comprimento do polígono de controle: $L_p = \|c[1]-c[0]\|+\|c[2]-c[1]\|+\|c[3]-c[2]\|$, e o comprimento da corda: $L_c = \|c[3] - c[0]\|$. O valor de l é aproximado por $\frac{1}{2}(L_p + L_c)$. Pode-se utilizar o valor $L_p - L_c$ como medida de erro. Caso o erro seja maior do que uma determinada tolerância, divide-se a curva em duas partes, aplicando-se a operação de corte (conforme visto na seção anterior) com parâmetro $t_a = 0,5$, e repete-se o cálculo para cada parte.

3.2.5 Curvas compostas

Formas mais complexas podem ser obtidas interligando-se duas ou mais curvas de Bézier cúbicas. Uma curva c pode ser criada pela junção de n segmentos formados por curvas de Bézier, utilizando $3n + 1$ pontos de controle $c[0], \ldots, c[3n]$. Seja $b \in [0, \ldots, n-1]$, diz-se que c^b é o b-ésimo segmento de Bézier, dado pela curva de Bézier cúbica definida pelos pontos de controle: $(c[3b], c[3b+1], c[3b+2], c[3b+3])$. Chamaremos esse tipo de curva de *curva composta*. A Figura 3.6 ilustra um exemplo para $n = 2$.

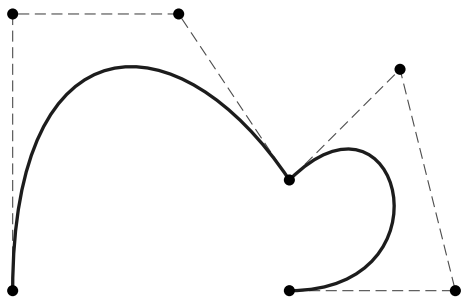


Figura 3.6: Exemplo de uma curva composta formada por dois segmentos de Bézier cúbicos. Os pontos de controle são demarcados por círculos. A linha tracejada é o polígono de controle.

Esse tipo de curvas é bastante utilizado em sistemas para criação imagens vetoriais no computador, devido à sua facilidade de representação e manipulação. Há vários formatos de arquivos capazes de armazenar curvas definidas dessa maneira. Por exemplo, a curva da Figura 3.6 é definida pelos pontos de con-

trole $(10, 10), (10, 60), (40, 60), (60, 30), (80, 50), (90, 10), (60, 10)$ e pode ser escrita em PostScript© [31] por meio dos comandos:

```
10 10 moveto
10 60 40 60 60 30 curveto
80 50 90 10 60 10 curveto
stroke
```

Em SVG (Scalable Vector Graphics) [32], curvas desse tipo podem ser definidas usando a tag `path`, cujo atributo `d` contém as coordenadas dos pontos de controle. A curva mostrada no exemplo acima pode ser obtida por meio dos comandos:

```
<g transform="matrix(1,0, 0,-1, 0,70)">
<path
    d = "M 10,10 C 10,60 40,60 60,30 80,50 90,10 60,10"
    style = "fill:none;stroke:black"
/>
</g>
```

Foi aplicada uma transformação inicial apenas para alterar a direção do eixo y, pois no formato SVG esse eixo é voltado para baixo. Tornando assim possível utilizar os mesmos valores para as coordenadas dos pontos de controle.

Desenhos mais complexos, como o da Figura 3.7, podem ser feitos utilizando um conjunto de curvas desse tipo.

Figura 3.7: Desenho composto por diversas curvas. À esquerda: visualização apenas das curvas de Bézier. À direita: visualização do polígono de controle (linhas tracejadas) sobreposto às curvas de Bézier (em cinza).

3.2.6 Parametrização

Cada segmento de Bézier que compõe uma curva possui uma parametrização do intervalo $[0, 1]$ no plano $\mathbb{R}^2$, conforme definido pela Equação 3.1. Há diversas formas de se criar uma parametrização global para a curva composta. Uma possibilidade é pela divisão do intervalo $[0, 1]$ em n partes iguais $I_b = [b/n, (b + 1)/n]$, para $b \in [0, \ldots, n - 1]$. A curva c é parametrizada então fazendo-se $c(t) = c^b(nt - b)$, onde b é tal que $t \in I_b$. Consequentemente a tangente da curva é dada por $c'(t) = n \cdot (c^b)'(nt - b)$ (a tangente pode não ser contínua em junções, ver Seção 3.2.11).

Podemos generalizar essa parametrização por meio de uma divisão do intervalo $[0, 1]$ em n intervalos $I_b = [w_b, w_{b+1}]$, $b \in [0, \ldots, n - 1]$, onde $0 = w_0 < w_1 < \cdots < w_{n-1} < w_n = 1$. A *parametrização ponderada* da curva c é dada por

$$c(t) = c^b((t - w_b)/(w_{b+1} - w_b)),$$

onde b é tal que $t \in I_b$. Nesse caso, a tangente é dada por

$$c'(t) = \frac{1}{w_{b+1} - w_b} \cdot (c^b)'((t - w_b)/(w_{b+1} - w_b)).$$

Dado $t \in [0, 1]$ é necessário encontrar o valor de b adequado para calcular $c(t)$. Na primeira parametrização, temos $w_b = b/n$, o valor de b pode ser encontrado por um cálculo simples: $b = \lfloor t \cdot n \rfloor$. No caso geral, é necessário fazer uma busca para encontrar um intervalo que contenha t. Como os valores w_b são ordenados de forma crescente, podemos encontrar b por intermédio de uma busca binária, que é um algoritmo de ordem $O(\log n)$, sendo portanto computacionalmente mais caro do que a primeira parametrização (onde o cálculo de b é $O(1)$). Porém como a quantidade de segmentos de Bézier cúbicos costuma ser baixa, a complexidade da busca não é um empecilho à utilização dessa parametrização. Além disso, a parametrização ponderada é útil quando se deseja tratar cada segmento de Bézier cúbico de forma diferente, como veremos nas próximas seções.

Seja l_b o comprimento de arco de c^b, uma possível escolha para os valores w_b é:

$$w_b = \frac{\sum_{k=0}^{b-1} l_k}{\sum_{k=0}^{n-1} l_k}.$$

A parametrização ponderada que utiliza esses valores para w_b será chamada aqui de *parametrização ponderada por comprimentos de arco.*

Seja B um segmento de Bézier cúbico definido pelos pontos de controle $B[0], B[1], B[2], B[3]$, e $\tilde{B}$ o segmento de Bézier cúbico definido pelos pontos de controle de B em ordem reversa $B[3], B[2], B[1], B[0]$. Pela equação 3.1 é fácil ver que $B(t) = \tilde{B}(1 - t)$. Esse comportamento se mantém para curvas compostas, isto

é, seja $\tilde{c}$ a curva definida pelos pontos de controle de c em ordem reversa, ou seja $\tilde{c}[k] = c[3n-k]$, e $\tilde{w}_b = 1 - w_{n-b}$. Os traços de c e $\tilde{c}$ são equivalentes, isto é, seus conjuntos imagens possuem o mesmo conjunto de pontos, porém em sentidos opostos, ou seja $c(t) = \tilde{c}(1-t)$. Para verificar essa relação, observa-se primeiramente que c^b é equivalente a $\tilde{c}^{(n-1-b)}$, já que os segmentos de Bézier cúbicos estão em ordem contrária de uma curva para a outra, assim tem-se $c^b(t) = \tilde{c}^{(n-1-b)}(1-t)$. Se $t \in I_b$, temos

$$\begin{gathered} w_b \leq t \leq w_{b+1} \\ 1 - \tilde{w}_{n-b} \leq t \leq 1 - \tilde{w}_{n-b-1} \\ -(1 - \tilde{w}_{n-b-1}) \leq -t \leq -(1 - \tilde{w}_{n-b}) \\ 1 - (1 - \tilde{w}_{n-b-1}) \leq 1 - t \leq 1 - (1 - \tilde{w}_{n-b}) \\ \tilde{w}_{n-b-1} \leq 1 - t \leq \tilde{w}_{n-b}, \end{gathered}$$

ou seja, $(1-t) \in \tilde{I}_{n-b-1}$, e então:

$$\begin{aligned} \tilde{c}(1-t) &= \tilde{c}^{(n-b-1)}(((1-t) - \tilde{w}_{n-b-1})/(\tilde{w}_{n-b} - \tilde{w}_{n-b-1})) \\ &= \tilde{c}^{(n-b-1)}(((1-t) - (1 - w_{b+1}))/((1-w_b) - (1-w_{b+1})) \\ &= \tilde{c}^{(n-b-1)}((-t + w_{b+1})/(w_{b+1} - w_b)) \\ &= \tilde{c}^{(n-b-1)}((-t + l_b + w_b)/(w_{b+1} - w_b)) \\ &= \tilde{c}^{(n-b-1)}(1 - (t - w_b)/(w_{b+1} - w_b)) \\ &= c^b((t - w_b)/(w_{b+1} - w_b)) \\ &= c(t), \end{aligned}$$

como queríamos demonstrar.

Quando é utilizada uma parametrização ponderada por comprimentos de arco teremos naturalmente $\tilde{w}_b = 1 - w_{n-b}$. De fato, como são equivalentes, os comprimentos de arco de c^b e $\tilde{c}^{(n-1-b)}$ são iguais, ou seja $l_b = \tilde{l}_{(n-1-b)}$, e consequentemente os comprimentos totais de cada curva são iguais: $\sum_{k=0}^{n-1} l_k = \sum_{k=0}^{n-1} \tilde{l}_k$.

Temos então

$$\begin{aligned} w_b &= \frac{\sum_{k=0}^{b-1} l_k}{\sum_{k=0}^{n-1} l_k} \\ &= \frac{\sum_{k=0}^{b-1} \tilde{l}_{n-1-k}}{\sum_{k=0}^{n-1} \tilde{l}_k} \\ &= \frac{\sum_{k=n-b}^{n-1} \tilde{l}_k}{\sum_{k=0}^{n-1} \tilde{l}_k} \\ &= \frac{\left(\sum_{k=0}^{n-1} \tilde{l}_k - \sum_{k=0}^{n-b-1} \tilde{l}_k\right)}{\sum_{k=0}^{n-1} \tilde{l}_k} \\ &= 1 - \frac{\sum_{k=0}^{n-b-1} \tilde{l}_k}{\sum_{k=0}^{n-1} \tilde{l}_k} \\ &= 1 - \tilde{w}_{n-b}. \end{aligned}$$

Embora as curvas c e $\tilde{c}$ sejam visualmente equivalentes, em determinadas aplicações é mais apropriado utilizar uma representação do que a outra, como veremos nas seções seguintes.

3.2.7 Distância paramétrica entre curvas

Para analisar a semelhança entre dois desenhos, nosso algoritmo de inbetween requer uma medida de distância entre duas curvas. Nesta seção, descrevemos o método que utilizamos para o cálculo dessa distância.

A distância paramétrica entre duas curvas c_i e c_j é definida como a média entre as distâncias entre amostras das duas curvas:

$$d(c_i, c_j) = \begin{cases} \frac{1}{n} \sum_{k \in I} d(p_i[k], p_j[k]), & \text{se } fw < bw, \\ \frac{1}{n} \sum_{k \in I} d(p_i[k], p_j[n-1-k]), & \text{caso contrário,} \end{cases} \tag{3.2}$$

onde cada curva é amostrada utilizando n pontos, $I = [0, \ldots, n-1]$, $p_i[k] = c_i(k/(n-1))$, $\bar{p}_i = \frac{1}{n} \sum_{k \in [0,\ldots,n-1]} p_i[k]$ (de forma semelhante para a curva c_j), e $d(u, v)$ é a distância Euclideana entre os pontos u e v. Utilizamos $n = 5 \cdot \max(n_{c_i}, n_{c_j})$, o que fornece uma boa aproximação da forma das curvas, pois 5 pontos costumam descrever a forma de um segmento cúbico razoavelmente bem. Temos ainda $fw = \text{dfw}(c_i, c_j)$ e $bw = \text{dbw}(c_i, c_j)$, onde:

$$\text{dfw}(c_i, c_j) = \sum_{k \in I} d(p_i[k] - \bar{p}_i, p_j[k] - \bar{p}_j),$$

$$\text{dbw}(c_i, c_j) = \sum_{k \in I} d(p_i[k] - \bar{p}_i, p_j[n-1-k] - \bar{p}_j).$$

As funções dfw e dbw indicam qual é a orientação mais adequada para ser utilizada no cálculo da distância paramétrica entre as curvas. A Figura 3.8 mostra uma comparação entre dois pares de curvas, no primeiro par $fw < bw$ e no segundo par $fw > bw$.

Figura 3.8: Diferentes orientações em pares de curvas. A cor indica a orientação de cada curva: preto corresponde ao início da curva e cinza corresponde ao final da curva.

Também tentamos utilizar outras funções de distância como a distância de Hausdorff, porém obtemos resultados melhores usando a Equação 3.2, pois foi mais frequente encontrarmos correspondências erradas ao utilizarmos a distância de Hausdorff. Essas correspondências erradas acontecem porque a distância de Hausdorff é dada pela maior distância mínima entre pares de pontos dos dois conjuntos, e portanto porções semelhantes das duas curvas não são levadas em consideração, ou seja mesmo se houver pontos de uma curva próximos a pontos da outra curva, a distância de Hausdorff dependerá apenas da maior distância. Consequentemente uma curva c pode ter a mesma distância de Hausdorff para um par de outras curvas, mesmo que uma dessas possua pontos mais próximos de c do que a outra. Por sua vez, a distância paramétrica fornece valores que levam em conta pontos próximos. Um exemplo disso pode ser observado na Figura 3.9.

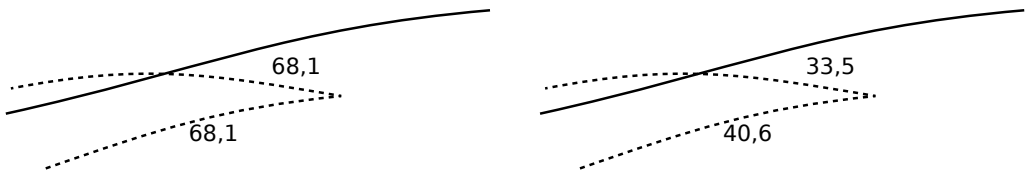


Figura 3.9: Comparativo entre a distância de Hausdorff e a distância paramétrica. Os valores indicam as distâncias entre a curva de cima (linhas contínuas) com cada curva de baixo (linhas tracejas). À esquerda: utilizando a distância de Hausdorff. À direita: utilizando a distância paramétrica.

Além disto, a informação fornecida pelas funções dfw e dbw são úteis para se

determinar a orientação quando as curvas são interpoladas. Assim a distância paramétrica fornece informações mais apropriadas para a inferência de curvas correspondentes.

3.2.8 Corte de curvas compostas

Podemos estender a operação de corte de curvas vista na Seção 3.2.3 para curvas compostas. Dado um valor $t_a \in [0,1]$, e b tal que $t_a \in I_b$ a curva equivalente a c (utilizando uma parametrização ponderada), entre os parâmetros 0 e t_a pode ser obtida pela junção de todos os segmentos anteriores ao segmento c^b com o corte de c^b entre os parâmetros 0 e $t'_a = (t_a - w_b)/(w_{b+1} - w_b)$. De forma semelhante, a curva equivalente a c após o parâmetro t_a pode ser obtida pela junção do corte de c^b entre os parâmetros t'_a e 1 com todos os segmentos posteriores a c^b. Estas junções são feitas criando-se uma nova curva composta por segmentos de Bézier cúbicos cujos pontos de controle são definidos como cópias dos pontos de controle de cada segmento de Bézier utilizado.

Da mesma forma, podemos obter também um corte de c entre dois parâmetros $0 \leq t_a < t_e \leq 1$. Seja b' tal que $t_a \in I_{b'}$, b'' tal que $t_e \in I_{b''}$, $t'_a = (t_a - w_{b'})/(w_{b'+1} - w_{b'})$ e $t'_e = (t_e - w_{b''})/(w_{b''+1} - w_{b''})$. O corte é dado pela junção do corte de $c^{b'}$ entre os parâmetros t'_a e 1, com os segmentos entre b' e b'' e juntamente com o corte de $c^{b''}$ entre os parâmetros 0 e t'_e. A Figura 3.10 mostra um exemplo de corte de uma curva composta.

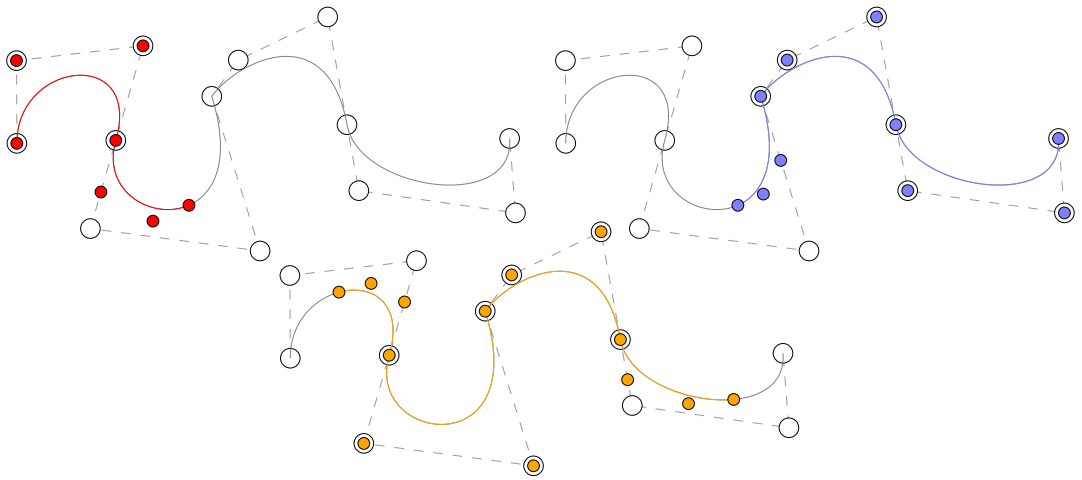


Figura 3.10: Corte de uma curva composta. Na parte superior esquerda, em vermelho: corte da curva até o parâmetro $t_a = 0.4$. Na parte superior direita, em azul: corte da curva a partir do parâmetro $t_a = 0.4$. Na parte inferior, em laranja: corte da curva entre os parâmetros $t_a = 0.1$ e $t_b = 0.9$.

3.2.9 Aperfeiçoando aproximações

Na Seção 3.2.2 vimos um método de aproximação de curvas de Bézier cúbicas dada uma transformação qualquer T. Podemos utilizar os conhecimentos sobre curvas compostas para aprimorar estas aproximações.

Dada uma curva de Bézier cúbica c (ou um dos segmentos de Bézier que compõem uma curva composta), podemos aplicar o método descrito na Seção 3.2.2 para obter $\hat{c}$ que aproxima Tc. Vimos que $\hat{c}$ minimiza a soma $S = \sum_{i=1}^{N} \|\hat{c}(t_i) - T(c(t_i))\|^2$, caso o valor de S esteja acima de um determinado patamar, podemos aprimorar o resultado aplicando um corte na curva c, e utilizando o mesmo método de aproximação para cada porção do corte, juntando os resultados numa única curva composta. Por exemplo, a curva c pode ser recortada no ponto $t_x = \frac{1}{2}$, formando duas porções c_{0x} e c_{x1}, a partir das quais se encontram $\hat{c}_{0x}$ e $\hat{c}_{x1}$, que aproximam Tc_{0x} e Tc_{x1}, respectivamente. Uma curva composta pode ser criada a partir dos pontos de controle de $\hat{c}_{0x}$ e $\hat{c}_{x1}$. Esse procedimento pode ser aplicado recursivamente às porções de curvas até que o valor de S de cada cálculo seja inferior ao patamar determinado ou caso a quantidade de refinamentos supere um nível máximo (para evitar subdivisões infinitas caso a transformação T seja descontínua).

A Figura 3.11 mostra um exemplo dessa operação. Foi aplicada uma transformação não-linear T a pontos (círculos pretos) da curva c (à esquerda), gerando os pontos vistos à direita. A curva composta à direita é o resultado da aproximação de Tc. Círculos brancos correspondem a pontos de controles das curvas.

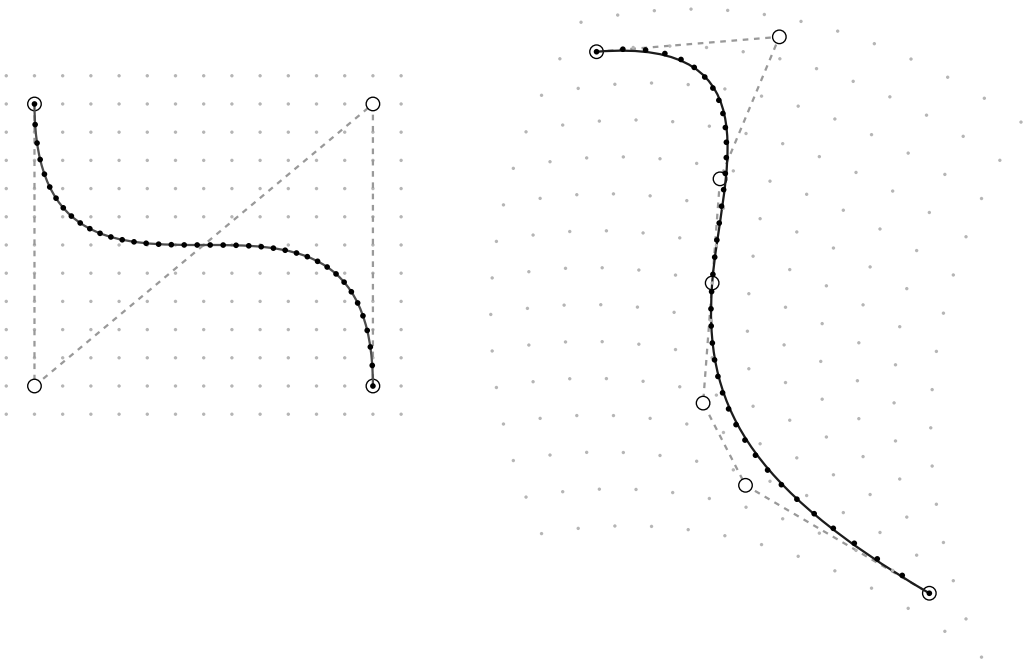


Figura 3.11: Aproximação aprimorada de uma curva de Bézier cúbica.

3.2.10 Amostragem

O conjunto imagem de uma curva é composto por um número infinito de pontos. Porém para se trabalhar com curvas no computador, é necessário que haja uma quantidade finita de pontos, uma vez que a memória do computador possui capacidade limitada. Há diversas formas possíveis de se reduzir o número de pontos de uma curva, em geral a ideia é escolher um conjunto (finito) de pontos que descrevam suficientemente bem as características da curva que forem necessárias, dependendo da aplicação. Chamamos esse processo de redução do número de pontos de *amostragem*.

Neste trabalho precisamos fazer amostragem em diversas ocasiões, para: o cálculo da distância entre curvas (seção 3.2.7); o ajuste de curvas após a aplicação de transformações não-lineares (seções 3.2.2 e 3.2.9); o cálculo de transformações entre quadros (seção 6.1.2); e a equalização do número de segmentos para interpolação entre duas curvas (seção 6.1.4).

Uma maneira de se fazer uma amostragem de uma curva formada por segmentos de Bézier é utilizando apenas os próprios pontos de controle dessa curva. Essa é uma forma bastante simples de amostragem, que não requer cálculos adicionais, porém fornece poucos detalhes sobre a geometria da curva, sendo útil em situações em que se deseja apenas uma aproximação grosseira da curva.

Uma outra forma de amostragem consiste na seleção de pontos pertencentes à imagem da curva, isto é, dada uma sequência de N valores reais $t_0 < t_1 < \cdots < t_{N-1}$, a curva c é amostrada pelo conjunto de pontos $c(t_0), c(t_1), \ldots, c(t_{N-1})$. A amostragem depende então do número N de pontos, da escolha dos valores $t_k, k \in [0, \ldots N-1]$, e da parametrização utilizada para a curva c.

Quanto mais alto é o valor de N mais detalhada será a amostragem da curva, porém o processamento de uma grande quantidade de pontos pode ser uma operação computacionalmente cara. O valor de N deve ser então um valor baixo, mas o suficiente para descrever a curva com os detalhes necessários de acordo com a aplicação.

A escolha dos parâmetros $t_0, t_1, \ldots, t_{N-1}$ deve ser feita de uma forma que os pontos amostrados forneçam uma boa descrição de toda a curva. Uma possível escolha para os parâmetros é fazer $t_k = \frac{k}{N-1}$. Porém, dependendo da parametrização, os pontos da amostragem podem se concentrar em determinadas partes da curva.

Utilizando a primeira parametrização vista na Seção 3.2.6, na qual o intervalo de parâmetros $[0, 1]$ é dividido igualmente para cada segmento de Bézier, haverá a mesma quantidade de pontos amostrados em cada segmento, o que normalmente não é desejável quando há segmentos com comprimentos consideravelmente distintos. Se uma curva é formada por dois segmentos de Bézier com o primeiro muito menor do que o segundo, haverá uma concentração de pontos amostrados na parte da

curva correspondente ao segmento menor, enquanto o segmento maior terá pontos bastante espalhados. Por outro lado, utilizando-se a parametrização ponderada por comprimentos de arco, a amostragem terá pontos melhor distribuídos ao longo da curva. Utilizando-se essa parametrização e um valor de N proporcional ao número de segmentos, tem-se uma amostragem com uma boa quantidade de pontos bem distribuídos.

Na Figura 3.12 são ilustradas as amostragens de uma mesma curva pelas duas formas de parametrização vistas. Para esse exemplo foi utilizado $N = 50$, e a curva é composta por 4 segmentos de Bézier cúbicos, havendo assim uma média de 12,5 pontos por segmento. Na primeira amostragem o número real de pontos por segmento é bastante próximo desta média, então segmentos curtos possuem aproximadamente o mesmo número de pontos de segmentos longos, causando assim um desbalanceamento na distribuição dos pontos. Entretanto na segunda amostragem observa-se uma melhor distribuição dos pontos, devido à parametrização utilizada.

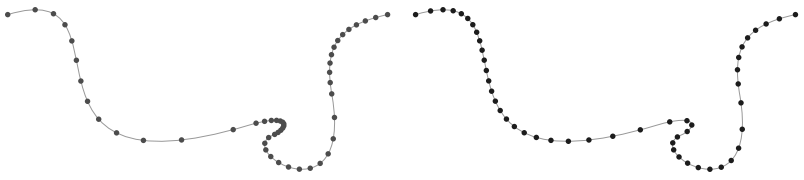


Figura 3.12: Amostragens diferentes para uma mesma curva. À esquerda: amostragem usando a parametrização com divisão uniforme do espaço de parâmetros. À direita: amostragem usando a parametrização ponderada por comprimentos de arco.

Podemos observar que, apesar de apresentar uma melhor distribuição de pontos, a parametrização ponderada por comprimentos de arco não gera uma amostragem perfeitamente uniforme, i.e., há uma variação nas distâncias entre dois pontos consecutivos na amostragem, isso acontece pois a parametrização de cada segmento de Bézier não é feita por comprimento de arco. Para nossa aplicação, não é necessário haver uma perfeita distribuição de pontos, logo as formas apresentadas são suficientes para nossos objetivos. Ainda assim é possível aprimorar as amostragens utilizando algum método que aproxime a parametrização por comprimento de arco, como o método descrito por WALTER e FOURNIER [33].

3.2.11 Continuidade

Dados dois segmentos de Bézier interligados, podemos categorizar a curva de acordo com sua continuidade no ponto de junção. Se, no ponto de junção, os vetores tangentes dos dois segmentos forem paralelos diz-se que a curva é G1-contínua nesse ponto. Se as direções juntamente com as magnitudes dos vetores tangentes forem

iguais, diz-se então que a curva é C1-contínua nesse ponto. A Figura 3.13 ilustra exemplos de curvas C1-contínua, G1-contínua e G1-descontínua.

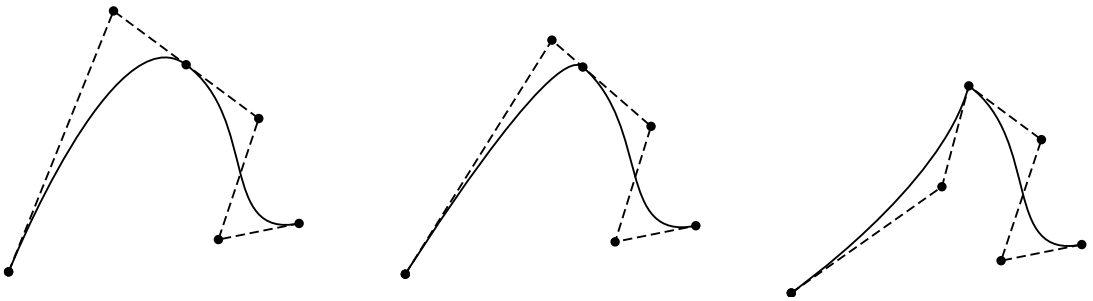


Figura 3.13: Níveis de continuidade. À esquerda: curva C1-contínua na junção entre segmentos de Bézier; Ao centro: curva G1-contínua na junção; À direita: curva G1-descontínua.

Em determinadas aplicações é necessário que curvas sejam C1-contínuas, para que seja possível fazer cálculos que dependam dessa característica. Entretanto, em certas circunstâncias é suficiente trabalhar com curvas G1-contínuas, cujo resultado pode ser visualmente aceitável. Quando uma curva não é G1-contínua (e consequentemente também não é C1-contínua), percebe-se uma mudança brusca na direção do traço, porém isso pode ser desejável, dependendo do que se deseja representar com a curva.

Portanto é vantajoso que sistemas de criação de curvas incluam controles de continuidade. Por exemplo, pode haver uma ferramenta para impor que uma curva seja G1-contínua em um determinado ponto de controle. A Figura 3.14 mostra um exemplo dessa operação.

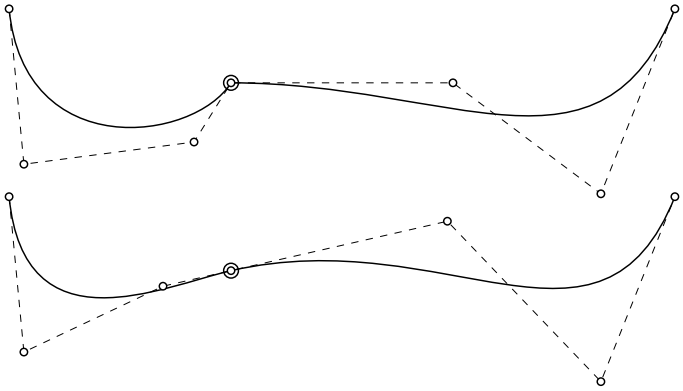


Figura 3.14: Continuidade G1 imposta em um ponto de controle. No topo: uma curva com descontinuidade G1 no ponto marcado com um círculo grande. Na parte inferior: curva resultante após imposição de continuidade G1.

Seja $c[k]$ o ponto de junção entre dois segmentos de Bézier, pelo cálculo dos vetores tangentes, a curva é G1-contínua em $c[k]$ se e somente se os pontos de controle

$c[k-1], c[k]$ e $c[k+1]$ forem colineares. Uma maneira de impor que isso ocorra é pelo deslocamento do ponto $c[k-1]$ para a posição $c[k] - \frac{\|u\|}{\|d\|}d$, e do ponto $c[k+1]$ para $c[k] + \frac{\|v\|}{\|d\|}d$, onde $u = c[k-1] - c[k]$, $v = c[k+1] - c[k]$ e $d = c[k+1] - c[k-1]$. Fazendo isto, a curva se torna G1-contínua sem haver grandes modificações em seu formato.

3.3 Regiões

Seja uma sequência de curvas $c_0, c_1, c_2, \ldots, c_n$, essa sequência define as bordas de uma região R no plano. Se o último ponto de controle de uma curva c_k não coincide com o primeiro ponto de controle da curva seguinte c_{k+1}, podemos adicionar uma linha reta ligando esses pontos. A última curva da sequência é ligada à primeira curva, formando assim uma região fechada no plano, A Figura 6.15 ilustra algumas regiões.

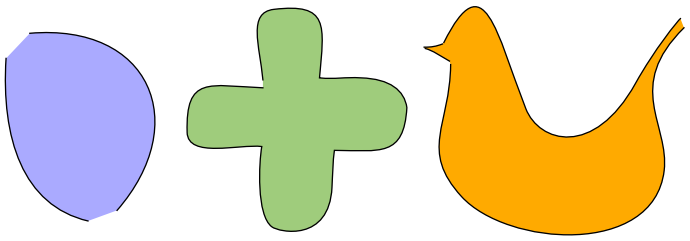


Figura 3.15: Exemplo de regiões. As curvas que definem as regiões estão pintadas com traços pretos. Linhas retas são adicionadas entre curvas para completar as regiões.

Dado um ponto $p \in \mathbb{R}^2$, dizemos que p está no interior de R se, para toda direção $d \in \mathbb{R}^2$, com $\|d\| > 0$, é ímpar o número de interseções entre a semi-reta $p + \alpha d, \alpha > 0$ e a borda de R (formada pelas curvas $c_0, c_1, c_2, \ldots, c_n$ e pelos segmentos de retas entre pares de curvas). Esse procedimento é conhecida como *regra par-ímpar*, e é comum em formatos de imagens vetoriais, como PostScript© [31], e implementada em diversos sistemas de criação/edição de figuras em formato vetorial.

É possível que a sequência de curvas forme uma região com auto-interseção. Nesses casos, de acordo com a regra par-ímpar, a parte onde essa se auto-intersecta pode ou não ser considerada interior da região, a depender da quantidade de interseções, como pode ser observado na Figura 3.16.

Uma curva pode ser utilizada em mais de uma região simultaneamente, porém também é possível que o sentido dessa curva em uma região seja oposto ao sentido utilizado para essa curva em outra região. Para resolver esse conflito, podemos aperfeiçoar a representação de uma região: para cada curva c_k de uma região, incluímos uma informação booleana de sentido fw_k. Caso fw_k seja verdadeiro, a curva c_k é

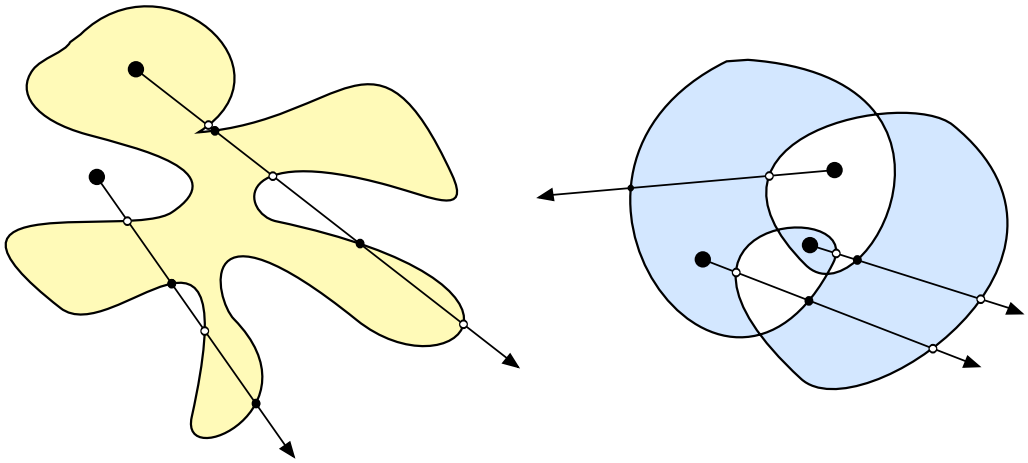


Figura 3.16: Regra par-ímpar ilustrada para alguns pontos (círculos maiores). Seguindo pelas direções indicadas pelas setas, os pontos interiores possuem uma quantidade ímpar de interseções com as curvas que definem as bordas das regiões. Para facilitar a visualização, as cores dos pontos de interseção se alternam entre preto e branco, um ponto pertence ao interior de uma região se o último ponto de interseção está pintado de branco.

utilizada no mesmo sentido em que está definida, caso contrário ela é utilizada no sentido oposto ao definido, i.e. como se os pontos de controle da curva estivessem em ordem reversa. Dessa maneira a mesma curva pode ser utilizada em duas regiões com sentidos distintos.

Dada uma sequência de curvas, é possível definir o sentido de cada curva de forma a minimizar o comprimento do contorno da região resultante, o que simplifica o formato da região. Por exemplo, a Figura 3.17 apresenta duas curvas formando uma região, ao alterar o sentido de uma destas curvas verificamos a simplificação da região, eliminando-se o cruzamento de linhas de contorno.

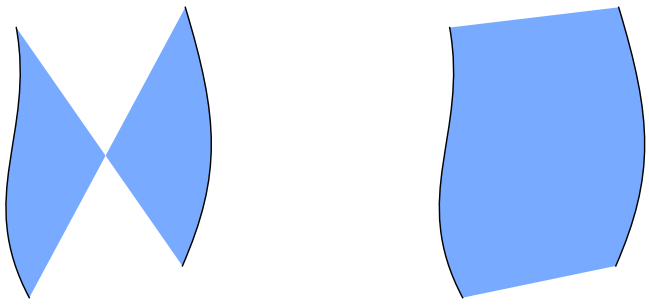


Figura 3.17: Ajuste de sentido de curvas numa região. À esquerda, há uma região definida por duas curvas. À direita, é possível ver o resultado após uma alteração de sentido de uma das curvas.

Isso pode ser feito automaticamente por intermédio de um processo de relaxamento, onde as curvas de contorno são percorridas, e o sentido de uma curva é

alterado sempre que for constatado que o comprimento do contorno diminui ao se fazer essa alteração. O processo é repetido até que nenhuma alteração seja feita, quando se atinge um comprimento de contorno mínimo. O Algoritmo 1 descreve em detalhes essa operação de relaxamento:

```
flipped = True;
while flipped do
    flipped = False;
    for i ∈ 0, ··· n do
        p ← c_{i−1}(1);
        q_a ← c_i(0);
        q_b ← c_i(1);
        r ← c_{i+1}(0);
        if dist(q_a, p) + dist(q_b, r) > dist(q_a, r) + dist(q_b, p) then
            fw_i ← notfw_i;
            flipped = True;
        end
    end
end
```

Algoritmo 1: Inferência do melhor sentido para cada curva numa região. Se fw_i é verdadeiro, $c_i(0)$ e $c_i(1)$ retornam respectivamente o primeiro e o último ponto de controle da curva c_i. Caso contrário $c_i(0)$ e $c_i(1)$ retornam respectivamente o último e o primeiro ponto de controle da curva c_i. Considere $c_{-1} = c_n$ e $c_{n+1} = c_0$. A função $dist(a, b)$ retorna a distância euclidiana entre os pontos a e b.

3.4 Grafos

No método de inbetweening desenvolvido, utilizamos conceitos e algoritmos de grafos, nesta seção detalharemos esses conceitos.

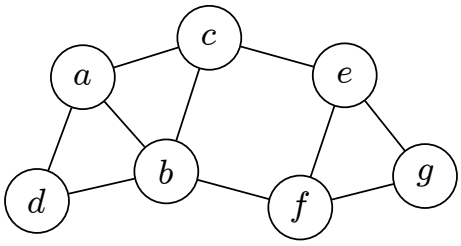


Figura 3.18: Exemplo de grafo, onde $E = \{a, b, c, d, e, f, g\}$ e $V = \{(a, b), (a, c), (a, d), (b, c), (b, d), (b, f), (c, e), (e, f), (e, g), (f, g)\}$.

Um *grafo* é um par ordenado $G = (V, E)$, onde V é um conjunto de vértices, e E é um conjunto de arestas, cada aresta é um par de vértices. A Figura 3.18 ilustra um grafo.

Um *grafo bipartido* é um grafo $G = (V = (X \cup Y), E)$, onde X e Y são conjuntos disjuntos, e cada aresta de E conecta um vértice de X com um vértice de Y. Um exemplo de grafo bipartido está ilustrado na Figura 3.19.

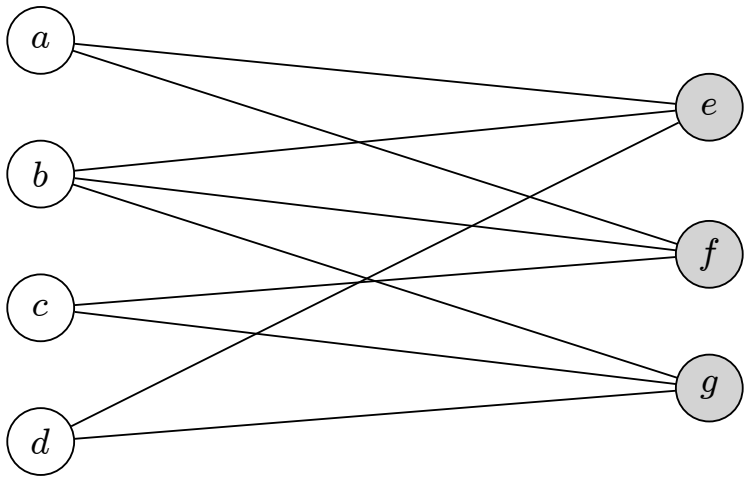


Figura 3.19: Exemplo de grafo bipartido, onde $X = \{a, b, c, d\}$ (vértices brancos) e $Y = \{e, f, g\}$ (vértices cinzas).

Um *casamento* é um conjunto de arestas sem vértices em comum. A quantidade de arestas de um casamento M é chamada de *cardinalidade* de M, denotada por $|M|$. Um *casamento de máxima cardinalidade* é um casamento que utiliza o maior número possível de arestas. Para encontrar tal casamento, podemos utilizar um algoritmo de *caminhos de aumento*, descrito a seguir.

Dado um casamento M de um grafo G, um *vértice livre* é um vértice que não pertence a nenhuma aresta de M. Um *caminho de aumento* P é uma sequência de vértices $v_0, v_1, \ldots, v_n \in V$, onde: o primeiro e o último vértice são vértices livres; cada par de vértices consecutivos (v_i, v_{i+1}) é uma aresta de G; as arestas se alternam entre arestas em M e arestas que não pertencem a M. A partir de um caminho de aumento P é possível obter um novo casamento M' onde $|M'| = |M| + 1$. Para isto, M' pode ser obtido por meio de $M' = M \oplus P = (M \setminus P) \cup (P \setminus M)$. A Figura 3.20 ilustra um caminho de aumento P e o resultado da operação $M \oplus P$.

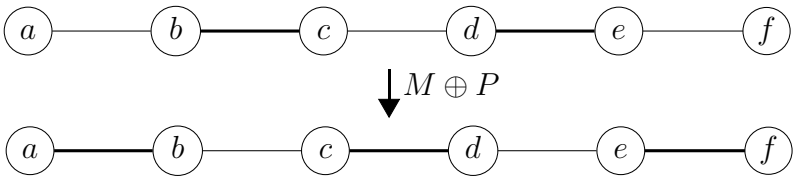


Figura 3.20: Exemplo de caminho de aumento. Linhas grossas representam arestas no casamento atual. Acima vemos o caminho antes de aplicarmos a operação $M \oplus P$. Abaixo vemos o caminho após essa operação.

É possível encontrar um casamento de máxima cardinalidade pela busca por caminhos de aumento, enquanto houver um caminho de aumento será possível encontrar um casamento com maior cardinalidade. Quando não houver mais caminhos de aumento, teremos um casamento de cardinalidade máxima.

Dizemos que um grafo é *ponderado* caso cada aresta esteja associada a um valor, chamado de *peso da aresta*. Um *casamento de peso máximo* é um casamento cuja soma dos pesos das arestas seja a maior possível.

O problema que estamos interessados em resolver é encontrar um casamento M de um grafo bipartido G que use o maior número possível de arestas de G e cuja soma dos pesos das arestas seja a maior possível, ou seja, um casamento de cardinalidade máxima e peso máximo. Isso pode ser feito por uma varição do algoritmo de buscas por caminhos de aumento, no qual a cada etapa encontramos o caminho de aumento que cause o maior acréscimo ao peso total do casamento. Ao final (quando não houver mais caminho de aumento), teremos o casamento de máxima cardinalidade e peso máximo.

A Figura 3.21 ilustra um grafo bipartido ponderado e o casamento de cardinalidade máxima e peso máximo encontrado por esse algoritmo.

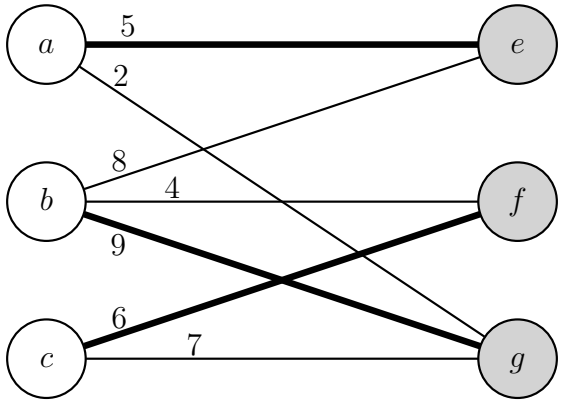


Figura 3.21: Casamento máximo (arestas grossas) em grafo bipartido. Os pesos das arestas estão descritos próximos a estas.

Capítulo 4

Animações 2D

Uma animação 2D é formada por um conjunto de figuras bidimensionais que se modificam ao longo do tempo. Numa animação em quadros o tempo é amostrado, havendo um quadro a cada amostra de tempo. Quanto mais densa for essa amostragem, mais suave é o movimento percebido pelo sistema visual humano. Animações são tradicionalmente exibidas numa taxa de 24 a 30 quadros por segundo. Produzir todos os quadros nessa taxa é um processo custoso. Os estúdios de animação costumam usar artifícios para minimizar esse custo, como o reuso de animações, ou ainda a exibição de uma mesma imagem em dois ou mais quadros seguidos quando há pouco movimento [10, 34]. Um sistema de produção de animações auxiliado por computador pode ajudar a reduzir custos automatizando processos que envolvam repetição de trabalho, como inbetweening, que é o objetivo deste trabalho. Neste capítulo, vamos especificar quais são os elementos e estruturas de dados utilizados no sistema proposto.

4.1 Quadros

Uma animação consiste de uma sequência de quadros, em que cada quadro contém um conjunto de figuras definidas por curvas e regiões. A ilusão de movimento ocorre quando há pequenas mudanças nas figuras em uma sequência de quadros. A Figura 4.1 ilustra uma sequência de quadros que formam uma animação.

4.1.1 Quadros-chaves

Para facilitar a criação de animações, normalmente os quadros são criados de forma não sequencial, ou seja, criando-se quadros espaçados representando posições chaves nos movimentos. Isso permite que os animadores tenham uma prévia do resultado sem a necessidade de criarem todos os quadros numa sequência. Os quadros-chaves devem conter informação suficiente para o preenchimento dos quadros inter-

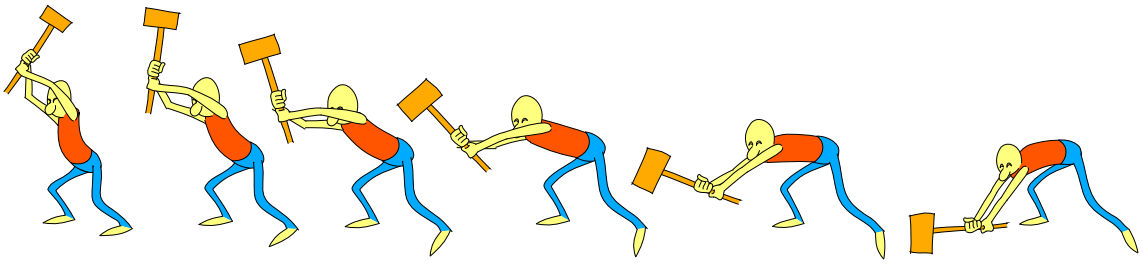


Figura 4.1: Animação formada por uma sequência de quadros.

mediários. Na Figura 4.1 o primeiro e último quadro são quadros-chaves, os quadros intermediários foram criados a partir desses quadros-chaves, isso foi feito utilizando o método desenvolvido neste trabalho, que será detalhado no próximo capítulo.

4.1.2 Camadas

A sobreposição de diferentes partes de um desenho podem dificultar a criação de uma animação. Para reduzir esse problema é possível organizar as figuras de cada quadro em camadas. Cada camada contém curvas e regiões, compondo uma parte de um quadro. Cada quadro possui uma sequência de camadas, as camadas são desenhadas na ordem em que são definidas nessa sequência. Assim, as primeiras camadas da sequência aparecem por trás das camadas seguintes, o que é útil para o gerenciamento de formas que se sobrepõem.

A Figura 4.2 ilustra um quadro separado em camadas.

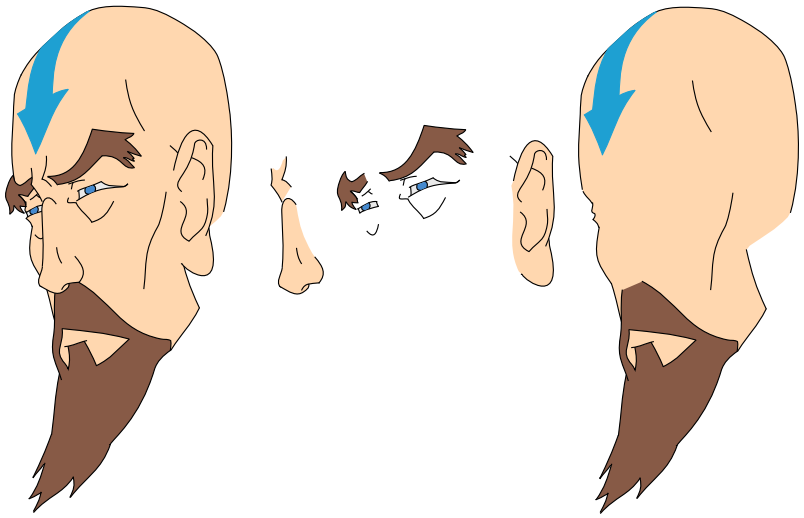


Figura 4.2: Desenho formado por camadas. À esquerda vemos a figura completa. À direita cada camada foi separada para melhor visualização.

Uma camada pode ainda ser dividida em sub-camadas, o que permite uma melhor organização dos desenhos. Por exemplo, o desenho de uma pessoa pode ser divido em

3 camadas: cabeça, tronco e membros. A cabeça pode ser dividida nas sub-camadas: crânio, olhos, nariz, boca, orelha, cabelo. Os membros podem ser divididos em: braço esquerdo, braço direito, perna esquerda e perna direita. Essa divisão ainda pode continuar enquanto for conveniente para a edição, por exemplo dividindo cada perna em pé, canela e coxa. Esse procedimento gera uma estrutura hierárquica de camadas. Um quadro completo é então definido por uma camada raiz, que pode ser dividida em sub-camadas. As camadas da Figura 4.2 podem ser organizadas na hierarquia ilustrada na Figura 4.3:

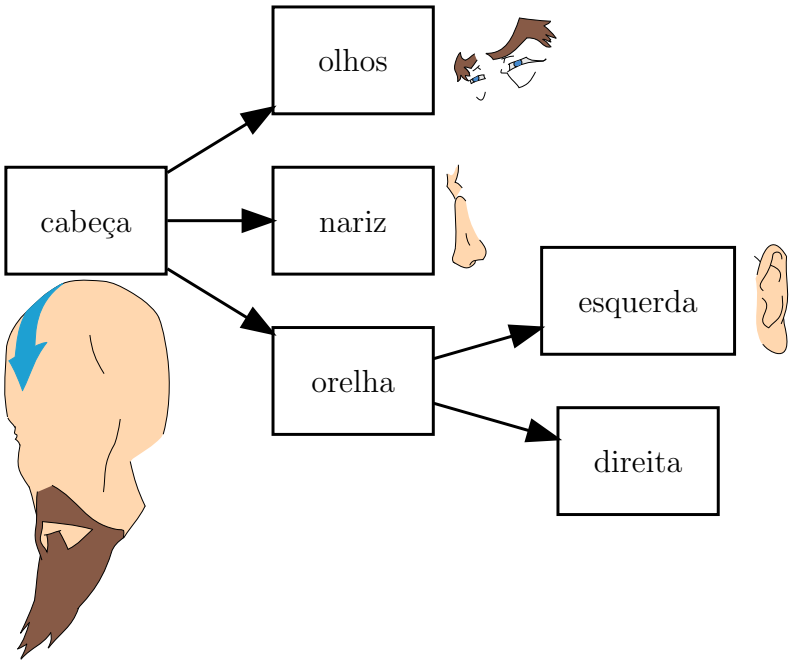


Figura 4.3: Hierarquia de camadas. Cada camada está associada a um desenho (mostrados próximos aos rótulos), e sub-camadas. A camada correspondente à orelha direita não é visível no quadro ilustrado, portanto não há desenho associado.

O artista é livre para criar a hierarquia de camadas. A orelha direita não é visível no quadro descrito pela Figura 4.2, por isso o desenho dessa orelha não aparece na Figura 4.3, mas é possível que essa orelha seja visível em outros quadros da animação, por esse motivo foi criada a camada correspondente. Em outras palavras, o artista deve planejar a hierarquia de camadas incluindo todos os elementos que podem aparecer durante a animação. Quanto melhor for a distribuição dos elementos na hierarquia, mais fácil será para o próprio artista gerenciar todas as figuras, além de ajudar os algoritmos utilizados no inbetweening, pois o espaço de busca por curvas e regiões correspondentes se reduz a elementos de uma mesma camada.

A ordem em que as camadas são desenhadas é importante na composição do quadro. Por exemplo, se um braço aparece na frente do corpo de um personagem, é importante que o corpo seja desenhado antes do braço, garantindo assim que o

desenho do braço se sobreponha ao do corpo. Porém, durante a animação é possível que o braço se mova para trás do corpo desse personagem. Portanto a ordem em que as camadas são desenhadas pode variar entre os quadros de uma animação. Para controlar a ordem de desenho das camadas, o artista pode indicar o nível de profundidade de cada camada em cada quadro, de forma que camadas mais profundas são desenhadas antes de camadas menos profundas. Durante o inbetweening o nível de profundidade de cada quadro intermediário pode ser calculado interpolando-se os níveis de profundidade dos quadros-chaves utilizados.

4.1.3 Linhas-guias

Um artista pode fazer um esboço de um desenho antes de criá-lo com todos os detalhes. Para isso, ele utiliza *linhas-guias*, que ajudam a ter uma prévia do resultado final sem grandes esforços, além de auxiliar o artista a manter proporções e simetrias. Após desenhar as linhas-guias, o artista cria os traços que estarão presentes na versão final do desenho. Neste trabalho, essa versão final é chamada de *arte-final*, e é formada por curvas e regiões conforme visto nas seções anteriores. As linhas-guias são uma versão simplificada da arte-final, contendo apenas curvas. Como essas linhas são vistas apenas pelos artistas durante a produção da animação, há menos rigor em sua confecção, o que dá aos artistas uma maior liberdade em relação à arte-final. A Figura 4.4 ilustra linhas-guias usadas na criação de um quadro.

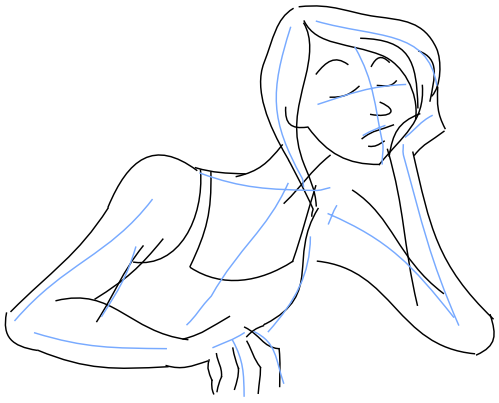


Figura 4.4: Linhas-guias (em azul) utilizadas para auxiliar a criação do desenho (em preto).

Sistemas de criação de animação auxiliados por computador normalmente processam apenas informações da arte-final. Entretanto, neste trabalho as linhas-guias também são processadas, como forma de utilizar mais informação para auxiliar o animador durante a confecção das animações.

4.2 Modos de criar animações

Uma animação pode ser produzida em três formas distintas: de modo direto; de pose em pose; ou uma combinação do modo direto com o modo de pose em pose [1].

No modo direto, o animador desenha os quadros na sequência em que eles serão exibidos. Esse método possui algumas vantagens, como a obtenção de ações naturalmente fluidas, e é uma forma do animador explorar sua criatividade. Porém esse modo é custoso, exige bastante trabalho para ajustar e finalizar o resultado, não fornece uma prévia da animação final, e é difícil manter o tamanho de personagens (os desenhos tendem a crescer ou encolher indesejadamente ao longo dos quadros).

No modo de pose em pose, a animação é planejada antes de ser produzida. Os animadores criam inicialmente os quadros mais importantes, contendo momentos chaves na animação. Em seguida são definidos quais são os próximos quadros mais importantes, esses quadros são chamados de extremos. Após essa etapa, os animadores trabalham em como criar as transições entre os extremos. Isso é feito adicionando-se alguns quadros entre cada par de quadros extremos. Finalmente a animação é completada adicionando-se os quadros intermediários que ainda não tenham sido desenhados. Esse modo oferece um controle maior sobre a animação durante a produção dessa, é um método estruturado, lógico, fácil de perceber qual será o resultado final. Porém por esse método diminui-se o controle sobre a fluidez das ações, que podem perder a naturalidade.

O terceiro modo aproveita as vantagens dos modos anteriores, por meio de uma combinação deles. O artista inicia novamente com os quadros mais importantes, com posições chaves. Esses quadros são usados então como guias para a produção dos quadros extremos, que são produzidos pelo modo direto, porém gerando apenas os quadros necessários para essa etapa. São feitos então vários passos pelo modo direto em partes diferentes, iniciando pelas partes mais importantes, encerrando com ajustes e correções no resultado final.

A criação de uma animação pelo método de pose em pose segue naturalmente uma hierarquia, começando em um nível alto de abstração, com o planejamento das cenas, e encerra com a criação de quadros intermediários. Os níveis mais altos de abstração dificilmente podem ser automatizados, uma vez que para criar os desenhos é necessário haver conhecimento sobre diversas características dos elementos desenhados, como a estrutura, forma, dinâmica e consistência. Porém essas características não são bem definidas por figuras formadas apenas por curvas e regiões bidimensionais, pois uma grande parte desses traços são perdidos ao se projetar as formas de um mundo tridimensional para um bidimensional. Por outro lado, processos num nível baixo de abstração são mais propensos a serem automatizados, uma vez que as tarefas necessárias nesse nível consistem em grande parte de combinações

simples de figuras semelhantes.

A proposta deste trabalho é auxiliar o animador em estágios de alto nível de abstração, facilitando a criação de novos quadros, e ainda provê uma forma de automatização para estágios de baixo nível de abstração (inbetweening). Há situações em que não é possível automatizar o processo de criação da animação, devido à simplificação causada pela projeção de formas tridimensionais em um espaço bidimensional. O artista possui liberdade de utilizar o método quando for conveniente, podendo usar a forma tradicional nos casos que não podem ser tratados automaticamente.

Capítulo 5

DiLight

Utilizando os conceitos estudados neste trabalho, desenvolvemos um programa para criação e edição de animações baseadas em quadros-chaves. Esse programa foi nomeado DiLight, pois emula uma mesa de luz digital (digital light table) para a criação de animações.

DiLight permite a criação e edição de curvas, regiões, quadros e camadas, e auxilia o artista durante o processo de criação de novos quadros-chaves e na geração de quadros intermediários. A Figura 5.1 mostra a interface do programa, nas seções seguintes discutiremos os elementos utilizados e as funcionalidades desse sistema.

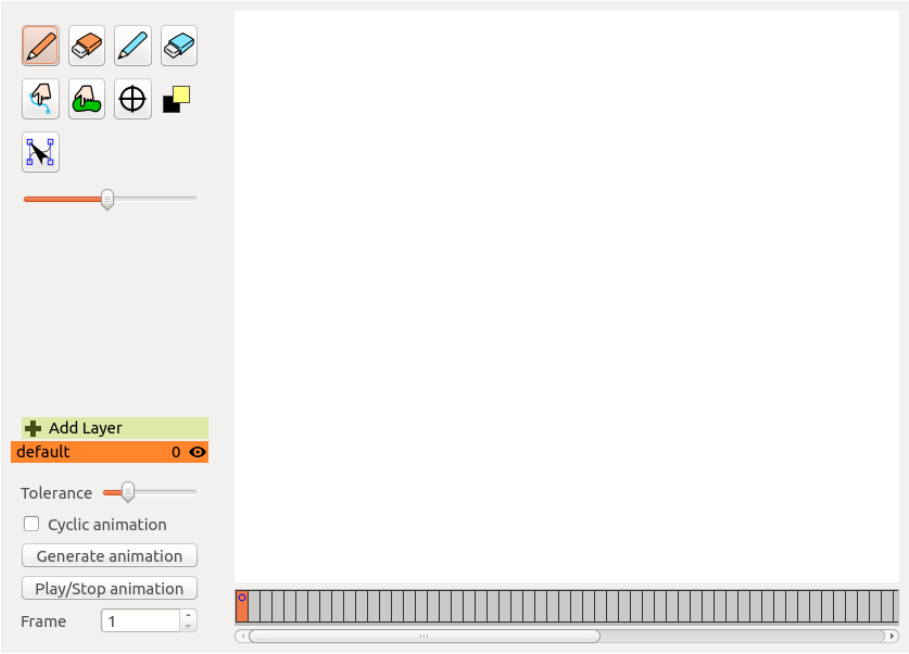


Figura 5.1: Interface do DiLight.

5.1 Curvas

No DiLight, o usuário pode criar curvas à mão livre em uma tela virtual, podendo utilizar para isso qualquer dispositivo de entrada de pontos, como um mouse ou uma caneta digital. Todas as curvas são formadas por segmentos de Bézier cúbicos (Seção 3.2), para isso utilizamos o algoritmo definido por FRISKEN [35], que encontra segmentos de Bézier que aproximam a curva de entrada (dada por uma sequência de pontos digitalizados) de acordo com uma determinada tolerância, de forma que o resultado é mais próximo à entrada ao se usar baixa tolerância. Na interface, o usuário pode manipular um parâmetro que define essa tolerância, controlando assim o nível de precisão e suavidade das curvas desenhadas. O usuário pode ainda ampliar/reduzir a tela virtual utilizando a roda do mouse para criar desenhos em diferentes níveis de detalhes.

O usuário pode criar curvas com duas ferramentas: para linhas-guias, e para a arte-final. Há ainda um controle deslizante para definir o nível de suavidade das curvas desenhadas, de forma que valores pequenos levam a curvas menos suaves, pois são mais próximas às curvas poligonais desenhadas, e valores maiores levam à formação de curvas mais suaves. A Figura 5.2 mostra uma captura de tela do DiLight com um curva sendo criada.

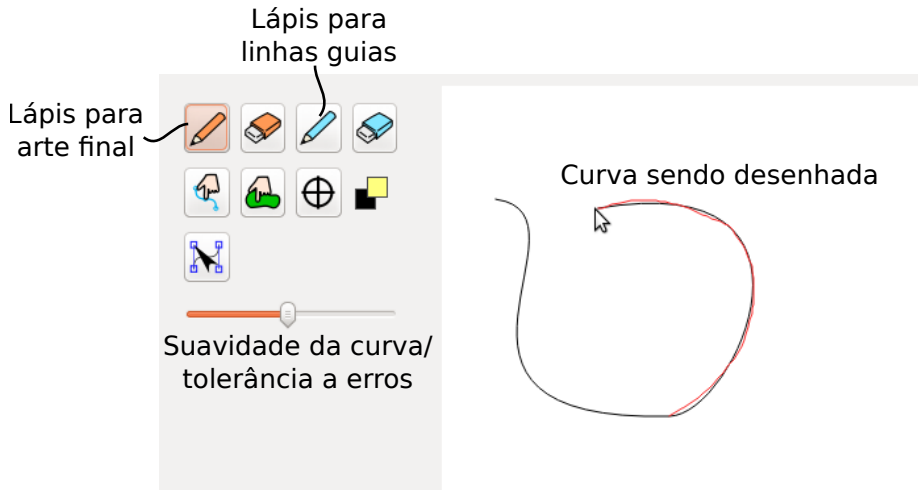


Figura 5.2: Ferramentas de criação de curvas. A curva vermelha é a entrada do usuário. A curva preta é a curva de Bézier ajustada à entrada do usuário.

É possível apagar as curvas com o uso de borrachas, sendo para as curvas da arte-final e a outra para as linhas-guias. As curvas podem também ser editadas, manipulando-se os pontos de controle. Para isso, o sistema contém a ferramenta que permite visualizar e deslocar cada ponto de controle. As curvas geradas nem sempre são suaves em todos os pontos, pois é possível haver descontinuidades entre dois segmentos de Bézier consecutivos formadas durante a criação da curva,

ou após a edição manual dos pontos de controle. Caso o usuário deseje eliminar uma descontinuidade, ele pode utilizar essa ferramenta também para forçar uma curva a ser G1-contínua em um determinado ponto. A Figura 5.3 mostra a ferramenta de edição de pontos de controle.

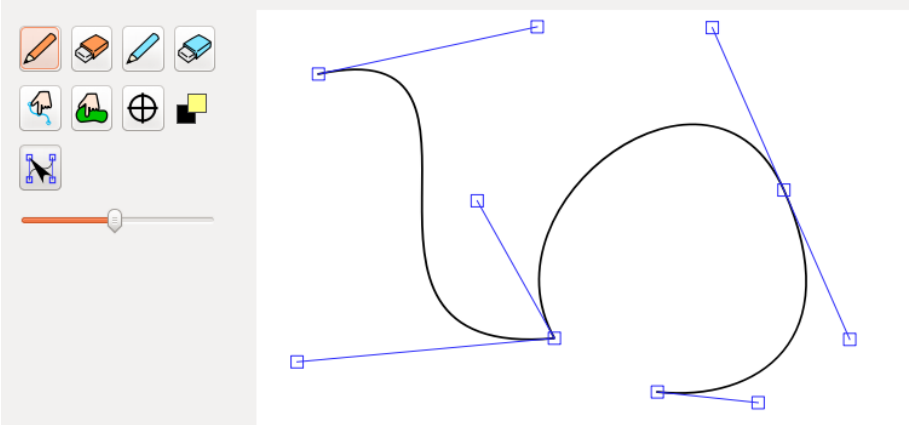


Figura 5.3: Edição de pontos de controle. O usuário pode mover os pontos de controle ou forçar continuidade $G1$ utilizando as ferramentas disponíveis.

5.2 Regiões

O usuário pode formar regiões definidas por curvas que definem seus contornos, bastando selecionar uma sequência de curvas pela ferramenta , e ativar a criação da região pelo menu, ou por um atalho no teclado. Segmentos de retas são adicionados entre curvas consecutivas para preencher o espaço entre elas, sendo assim possível criar regiões com contornos abertos, apesar de serem representadas com contornos fechados. É útil permitir contornos abertos pois o artista pode criar figuras complexas por meio da união de duas ou mais regiões, mantendo o contorno aberto na interseção das regiões.

Regiões também podem ser selecionadas pela ferramenta , o que é útil para copiar regiões para outros quadros, e para a remoção de regiões.

5.3 Quadros

Para controlar os quadros da animação, a interface do DiLight contém um editor de linha de tempo, que o usuário pode utilizar para selecionar o quadro editado no momento e definir os quadros-chaves. A Figura 5.4 ilustra o editor de linha de tempo.

O desenho do quadro-chave anterior pode ser visualizado por trás do quadro-chave atual, para ajudar o artista a fazer o desenho. Essa visualização do quadro

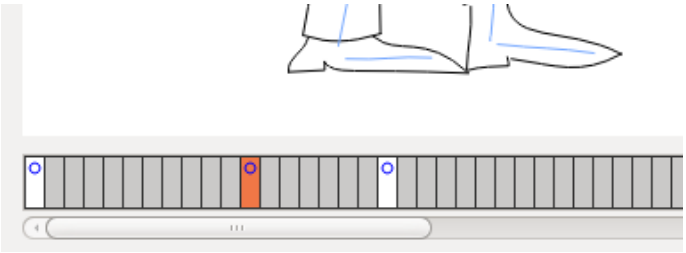


Figura 5.4: Gerenciamento de quadros. Cada quadro é representado por um retângulo, os quadros-chaves são demarcados por círculos dentro dos retângulos. O quadro atualmente visualizado/editado é destacado.

anterior é chamada de *desenho auxiliar*. O desenho auxiliar é modificado segundo uma operação de metamorfose definida pelas linhas-guias, conforme será detalhado no próximo capítulo, e o usuário pode ver a modificação enquanto cria as linhas-guias. A Figura 5.5 mostra como o usuário visualiza o desenho auxiliar durante a edição de um quadro-chave.

Figura 5.5: Exemplo de um quadro-chave sendo editado. O desenho auxiliar pode ser visualizado em linhas cinzas. Os traços pretos são parte da arte-final do quadro-chave atual.

Cada quadro-chave possui um centro de rotação, que é usado para controlar a trajetória dos objetos ao longo da animação. A ferramenta $\oplus$ pode ser utilizada para controlar a posição do centro de rotação no quadro-chave atual. Se não for definido pelo usuário, o centro de rotação é calculado automaticamente como o baricentro do desenho nesse quadro-chave.

Diversas animações são criadas de forma cíclica, ou seja após o último quadro-chave a animação volta ao início. Isso pode ser feito repetindo o primeiro quadro-chave após o último. Para facilitar essa tarefa, o usuário pode marcar a animação como sendo cíclica, e o sistema automaticamente fará com que o primeiro e último

quadros-chaves sejam iguais.

5.4 Camadas

O usuário pode separar os desenhos numa hierarquia de camadas, o que ajuda não apenas a organização como também o algoritmo de inbetweening, como será visto no próximo capítulo. Para o gerenciamento de camadas, a interface apresenta um controle que permite adicionar, renomear, visualizar ou ocultar camadas e sub-camadas. A Figura 5.6 mostra um exemplo de organização em camadas de um personagem.

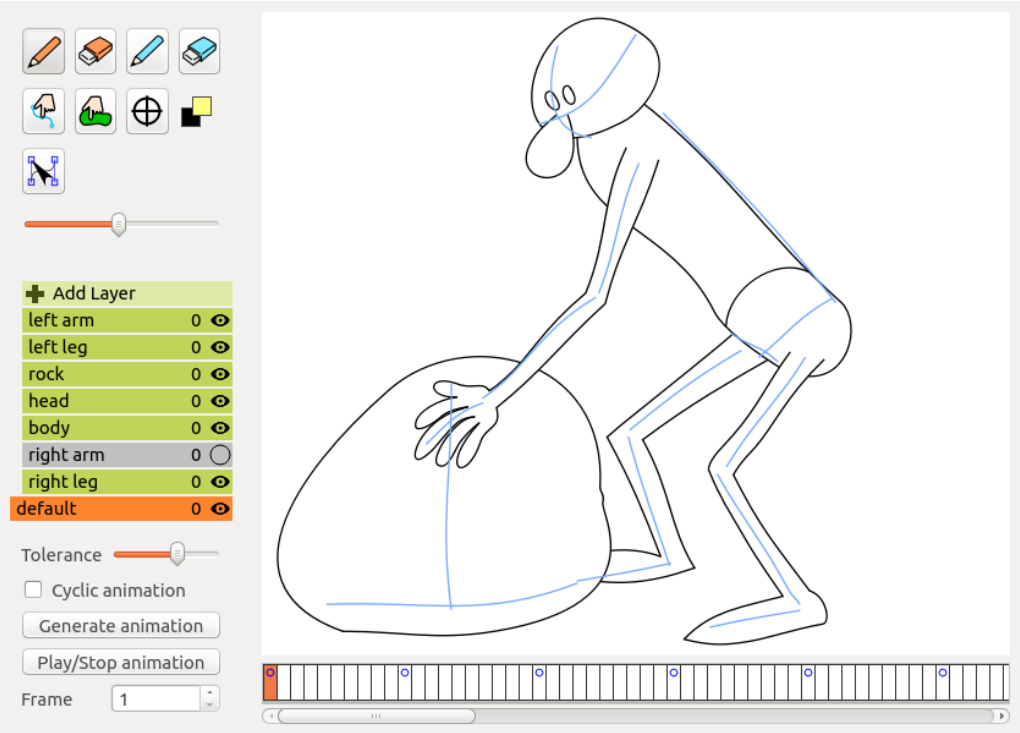


Figura 5.6: Gerenciamento de camadas. Cada parte do corpo do personagem foi colocada numa camada distinta. A cor cinza indica uma camada invisível na tela de edição. A camada atualmente editada é destacada pela cor laranja.

A camada que aparece na parte inferior é a camada base, ou seja, as outras camadas mostradas são as sub-camadas dessa camada. As camadas superiores nessa hierarquia não são visíveis no controle e são visualizadas com uma cor mais clara na tela com os desenhos. Para navegar na hierarquia, o usuário pode clicar com o botão do meio do mouse sobre uma camada, o que faz com que essa seja colocada na base, permitindo assim a manipulação das sub-camadas da camada clicada. Se a camada clicada já estiver na base, sobe-se na hierarquia.

Também é importante controlar a ordem entre camadas, de forma que pode-se especificar se uma camada deve aparecer em frente ou por trás de outra camada.

Essa ordem pode ser alterada durante a animação, por exemplo um braço pode estar na frente do corpo em alguns quadros, mas ele pode se mover para detrás do corpo num momento futuro. Para controlar a ordem, cada camada possui um índice para cada quadro, funcionando como uma terceira dimensão, tal que as camadas com índices mais baixos são posicionadas por detrás de camadas com índices mais altos. É necessário especificar esses índices apenas para os quadros-chaves, pois índices dos quadros intermediários são interpolados linearmente a partir dos índices dos quadros-chaves.

5.5 Controle de velocidade

É essencial que o animador seja capaz de controlar a temporização da animação, ou seja, ele ou ela pode fazer com que um movimento seja mais rápido ou devagar. Mais do que isso, a velocidade da animação pode ser variável. Por exemplo, uma bola quicando deve apresentar um movimento acelerado durante quedas e retardado durante subidas, caso contrário a animação não parecerá natural.

No DiLight o usuário pode controlar a temporização pela remoção e adição de quadros intermediários (o que causa aumento e redução de velocidades, respectivamente), e pela manipulação de curvas que guiam o espaçamento dos quadros. Isso é feito utilizando curvas de Bézier cúbicas que controlam como a animação deve fluir.

A Figura 5.7 mostra um exemplo de como controlar o movimento de queda de uma bola manipulando-se dessa curva.

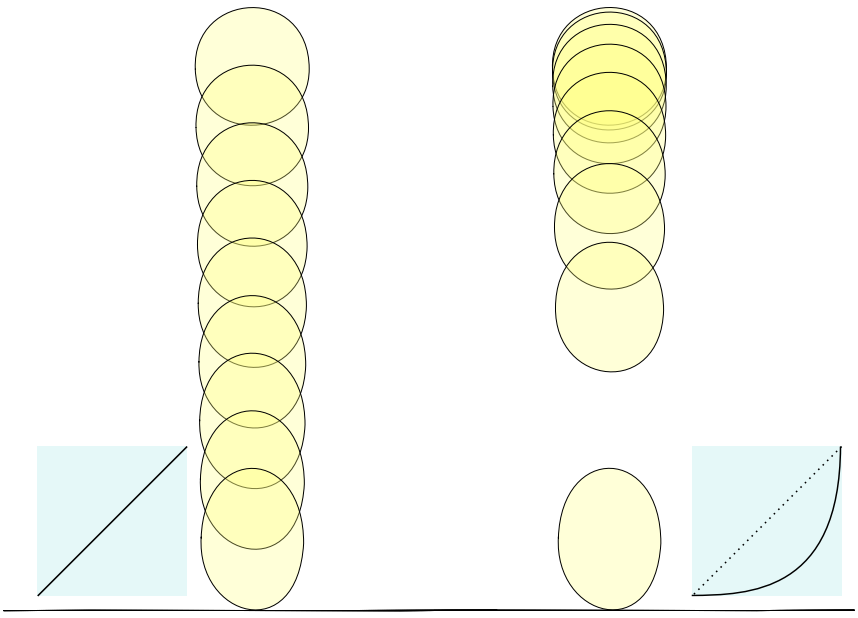


Figura 5.7: Uma bola caindo com velocidade constante (à esquerda) e acelerada (à direita). Os desenhos estão translúcidos para facilitar a visualização.

Capítulo 6

Inbetweening

Neste capítulo, descrevemos em detalhes o método proposto para inbetweening. Cada quadro-chave é separado em duas partes. A primeira corresponde às linhas-guias, em que o artista desenha um conjunto de traços para auxiliar o desenho. A segunda contém os desenhos detalhados, que chamamos de arte-final. Linhas-guias costumam ser usadas pelos animadores para auxiliar o desenho. Nosso método também as utiliza para guiar uma transformação de um quadro para o próximo. Essa transformação auxilia os algoritmos de inferência de correspondências e interpolação das formas das artes finais. A Figura 6.1 retrata a linha de execução completa do método proposto.

Nosso sistema recebe inicialmente como entrada dois quadros-chaves consecutivos i e j, e cria automaticamente correspondências entre os traços das linhas-guias. Então, uma transformação de similaridade A é calculada e aplicada às linhas-guias do quadro-chave i. A transformação é refinada calculando-se uma função de metamorfose não-linear M que aproxima ambos os conjuntos de linhas-guias (i transformado e j). Caso o quadro-chave i já contenha sua arte-final, essa é transformada pela composição $M \circ A$, o que permite um retorno visual para ajudar o artista a desenhar o quadro-chave j. É importante ressaltar que nosso objetivo final é criar quadros intermediários que interpolam a arte-final, não apenas os esboços. Portanto, o método continua computando a correlação entre a arte-final do quadro-chave i transformado e a do quadro-chave j. Finalmente, definimos uma função para cada par de traços correlacionados para criar traços interpolados dos quadros intermediários.

6.1 Método

Como mencionado anteriormente, cada quadro-chave possui uma parte de esboço que consiste em uma sequência de curvas, chamadas *linhas-guias*. Nossa abordagem de inbetweening envolve o processamento do conjunto de linhas-guias (Seção 6.1.1), cálculo e aplicação de transformações entre quadros-chaves (Seção 6.1.2), busca de

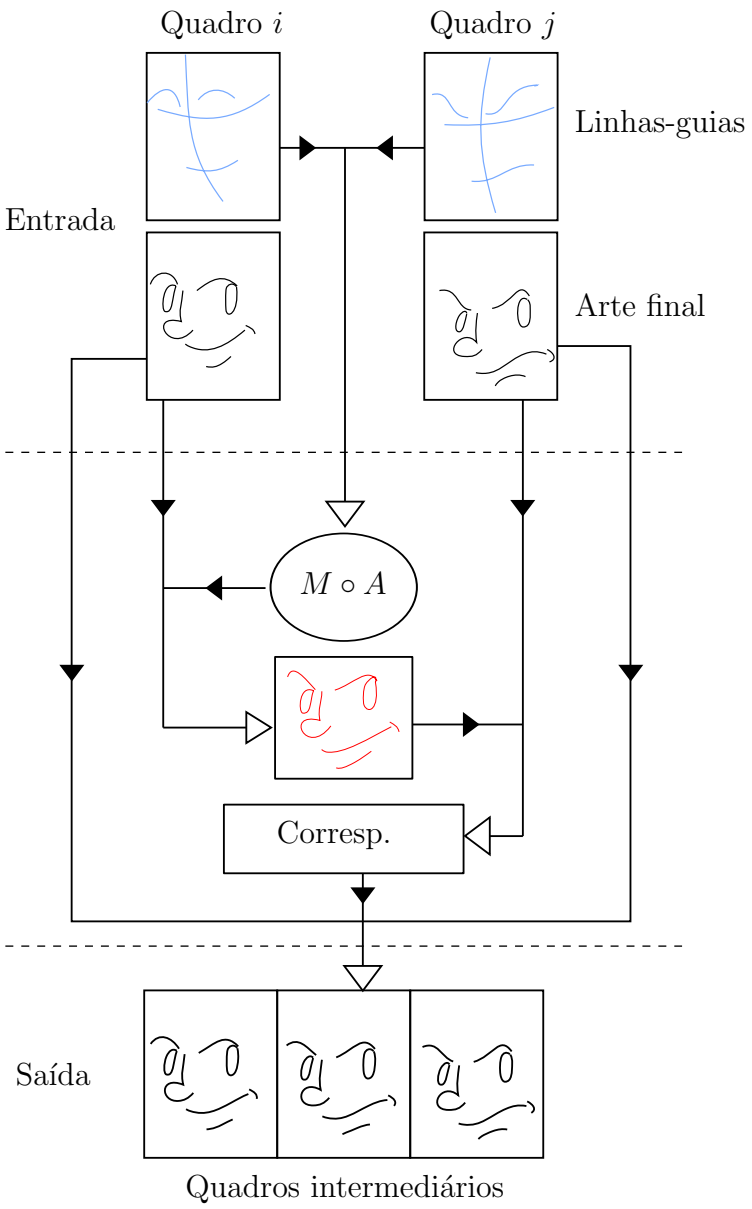


Figura 6.1: Linha de execução do método. Os esboços de dois quadros-chaves são processados para calcular uma transformação $M \circ A$, que é aplicada ao quadro-chave i para gerar um desenho deformado (vermelho). Correspondências são então encontradas entre o quadro transformado i e o quadro-chave j, e são usadas para gerar os quadros intermediários.

correspondências na arte-final (Seção 6.1.3), e a criação de novos quadros por meio da interpolação dos desenhos correspondentes (Seção 6.1.4).

6.1.1 Processamento de linhas-guias

O primeiro passo na geração de uma animação com nosso método é a criação de linhas-guias para cada quadro-chave em uma sequência. Espera-se que essas linhas-guias sejam muito mais simples do que a arte-final, simples o suficiente para descrever

apenas uma alteração média entre quadros-chaves. É importante para o método que haja uma correlação entre as linhas-guias de um quadro para outro, isto é, que linhas-guias num quadro-chave possuam correspondências no quadro-chave seguinte. Não é necessário que essa correspondência seja um-para-um, i.e., é possível haver linhas-guias sem correspondência no outro quadro-chave, mas o algoritmo funciona melhor quando a correspondência é um-para-um.

Quanto mais informação tivermos sobre os movimentos das curvas, mais confiável é o resultado da inferência de correspondência. Entretanto, durante esse primeiro passo nós ainda não temos essa informação. Logo, para computar as correspondências, primeiramente alinhamos as linhas-guias de cada quadro-chave e procuramos por uma rotação e escala que melhor descreve o movimento, como detalhado a seguir.

Inicialmente as linhas-guias de cada quadro-chave são transladadas, de forma que seus baricentros sejam movidos para a origem do plano $\mathbb{R}^2$. O baricentro de cada quadro é definido como o baricentro dos pontos de controle das linhas-guias nesse quadro. Essa operação ajuda a busca por linhas-guias correspondentes, evitando cálculos errados causados por uma possível translação.

Após isso uma escala é aplicada ao quadro i de forma que os quadros tenham dimensões semelhantes. Sejam w_i, h_i as dimensões (largura e altura, respectivamente) da menor caixa alinhada com os eixos que envolve os pontos do quadro-chave i, e w_j, h_j as dimensões da menor caixa alinhada com os eixos que envolve os pontos do quadro-chave j. O fator de escala aplicada ao quadro-chave i é dado por $\max(w_j, h_j)/\max(w_i, h_i)$. A Figura 6.2 ilustra as operações de translação e escala aplicadas às linhas-guias de um par de quadros-chaves.

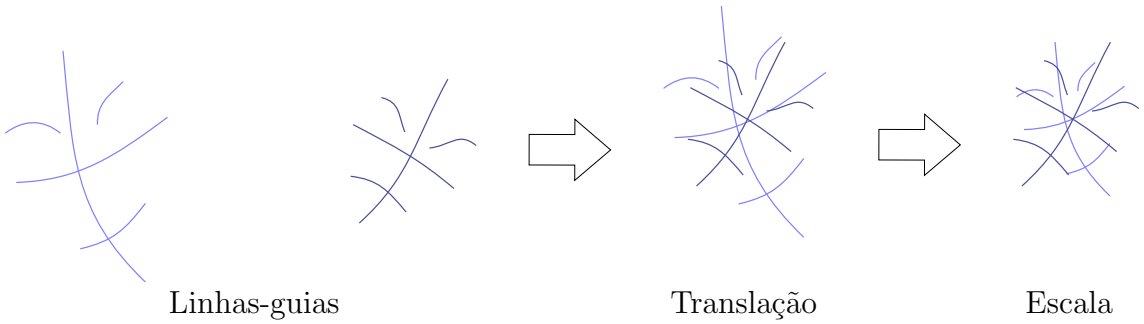


Figura 6.2: Translação e escala aplicadas a linhas-guias.

Sejam N_i e N_j a quantidade de linhas-guias nos quadros-chaves i e j, respectivamente. Definimos um grafo bipartido K_{N_i,N_j}, no qual cada vértice do grafo representa uma linha-guia, e cada aresta conecta uma linha-guia de i com uma em j. Nesse grafo, a aresta que conecta a linha-guia g_i com a linha-guia g_j possui um peso dado por $-d(g_i, g_j)$, onde d corresponde à distância entre curvas definida pela

Equação 3.2.

Uma vez que o grafo bipartido tenha sido criado, calcula-se então o casamento P de máxima cardinalidade e peso máximo, ou seja, as correspondências entre i e j cuja soma dos pesos seja a maior possível, utilizando o maior número possível de arestas. Como os pesos das arestas do grafo foram definidos como valores negativos, o casamento P indicará os pares de linhas-guias mais próximas umas das outras.

É possível que a figura em j tenha sido rotacionada, nesse caso o algoritmo de casamento provavelmente resultará em correspondências indevidas. Para minimizar erros devido a rotações, as linhas-guias do quadro i são rotacionadas por um conjunto de ângulos $\alpha \cdot k$, $k \in \{1, 2, \ldots, 360°/\alpha\}$. Para cada ângulo, calculamos P, e escolhemos aquele com o maior peso, o que corresponde ao casamento no qual as curvas correspondentes estão mais próximas umas das outras. Neste trabalho, usamos $\alpha = 45°$, que se mostrou suficiente em todos os nossos testes. A Figura 6.3 ilustra essa busca pela melhor rotação.

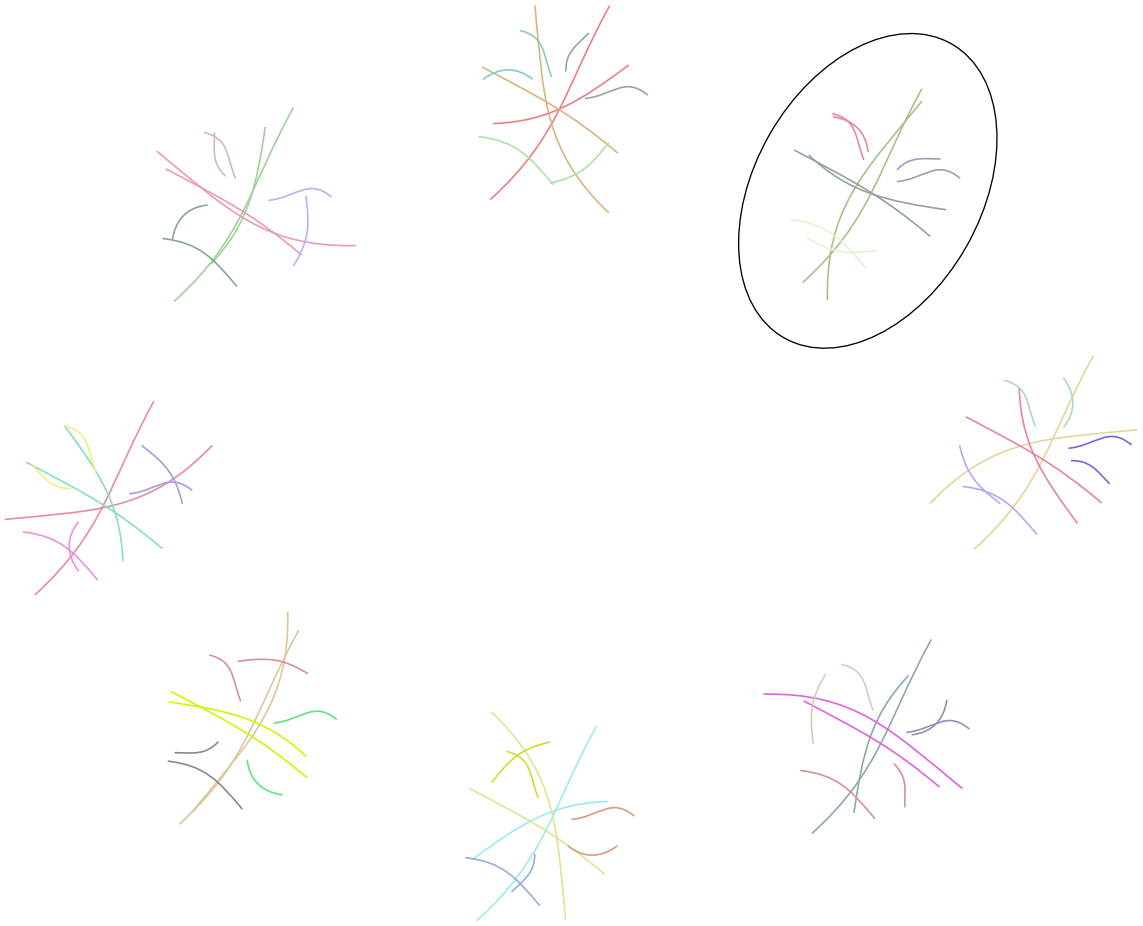


Figura 6.3: Linhas-guias rotacionadas para encontrar a melhor correspondência (identificada por uma elipse). As cores indicam as correspondências entre curvas encontradas em cada rotação.

Após encontrar as correspondências entre linhas-guias, é necessário encontrar o

melhor sentido de correspondência entre cada par de linhas-guias correspondentes. Esse sentido será utilizado na próxima seção para o cálculo de transformações entre quadros-chaves. Utilizando a rotação que forneceu a melhor correspondência entre as linhas-guias, calculamos as funções dfw e dbw definidas na Seção 3.2.7, caso dfw $<$ dbw utilizamos o sentido original de cada linha-guia, caso contrário invertemos uma das curvas.

6.1.2 Cálculo de transformações

Uma vez que temos as correspondências entre linhas-guias de dois quadros-chaves em uma sequência, é necessário encontrar uma transformação que leva as linhas-guias no quadro-chave i às linhas-guias correspondentes no quadro-chave j. O processo é descrito a seguir.

As linhas-guias são amostradas, formando dois conjuntos de pontos correspondentes: $G^i = \{p_k^i = (x_k^i, y_k^i) \in \mathbb{R}^2, k \in [1, N_S]\}$, $G^j = \{p_k^j = (x_k^j, y_k^j) \in \mathbb{R}^2, k \in [1, N_S]\}$, onde N_S é o número total de amostras em cada quadro. Para cada $k \in [1, N_S]$ os pontos p_k^i e p_k^j são correspondentes.

Calcula-se então uma transformação de similaridade (i.e., uma combinação de rotação, translação e escala) dada por $A : \mathbb{R}^2 \mapsto \mathbb{R}^2$,

$$A(x, y) = (cx - sy + t_x, sx + cy + t_y),$$

onde (t_x, t_y) indica a direção da translação, e $c = \alpha \cos(\theta)$, $s = \alpha \sin(\theta)$, sendo α o fator de escala, e θ o ângulo de rotação. Esses parâmetros são calculados de forma a minimizar a soma:

$$S(A) = \sum_{k=1}^{N_S} d(A(p_k^i) - p_k^j)^2.$$

Sejam as matrizes:

$$T_k = \begin{pmatrix} x_k^i & -y_k^i & 1 & 0 \\ y_k^i & x_k^i & 0 & 1 \end{pmatrix}, \quad Q_k = \begin{pmatrix} x_k^j \\ y_k^j \end{pmatrix},$$

$$a = \begin{pmatrix} c & s & t_x & t_y \end{pmatrix}^\top,$$

A soma acima pode ser reescrita como:

$$S(A) = \sum_{k=1}^{N_S} \|T_k a - Q_k\|^2 = \|Ta - Q\|^2,$$

onde

$$T = \begin{pmatrix} T_1 \\ T_2 \\ \vdots \\ T_{S_N} \end{pmatrix}, \qquad Q = \begin{pmatrix} Q_1 \\ Q_2 \\ \vdots \\ Q_{S_N} \end{pmatrix}.$$

Encontrar A que minimiza $S(A)$ é portanto um problema de mínimos quadrados, cuja solução é dada por $a = (T^\top T)^{-1} T^\top Q$. Computacionalmente calculamos a solução usando decomposição em valores singulares [36].

Essa operação será denotada pela equação

$$A = \text{similarity}(G^i, G^j). \tag{6.1}$$

Essa transformação posiciona aproximadamente um conjunto sobre o outro, porém as suas formas ainda são diferentes, pois estamos restritos a uma similaridade.

Para aproximar mais os dois conjuntos, uma segunda transformação M é calculada usando MFFD (*multilevel free-form deformation*) [37]. Esse método recebe um conjunto de pares de pontos no plano e gera uma função C^2-contínua um-para-um que move cada ponto o mais próximo o possível de seu par. Uma aproximação 2D com B-spline é aplicada a uma hierarquia de grades de controle, minimizando a distância entre pontos dos dois conjuntos de entrada. Neste trabalho, utilizamos uma hierarquia de grades em cinco níveis, na qual no primeiro nível a grade possui dimensões 10×10, e as dimensões dobram a cada nível. As linhas-guias são reamostradas de forma que quaisquer dois pontos consecutivos das amostras de Ag_i e de g_j podem estar contidos em uma célula na mais refinada grade de controle utilizada na definição do MFFD, e os conjuntos G^i e G^j são atualizados de acordo. A transformação M é calculada para aproximar mais os pontos AG^i e G^j, e pode ser aplicada a qualquer ponto no domínio do desenho (uma tela de pintura virtual).

Ao aplicarmos $M \circ A$ às curvas da arte-final do primeiro quadro, obtemos um desenho distorcido similar ao desenho no segundo quadro-chave. Como o cálculo desse desenho distorcido depende apenas das linhas-guias e do desenho do primeiro quadro-chave, o usuário do DiLight pode visualizá-lo e utilizá-lo como referência para o desenho do quadro-chave seguinte, isso é o que chamamos de desenho auxiliar. Como M é uma transformação não-linear, a curva resultante da aplicação de $M \circ A$ à curva original é obtida pelo método descrito na Seção 3.2.9. A Figura 6.4 mostra um exemplo dessa operação.

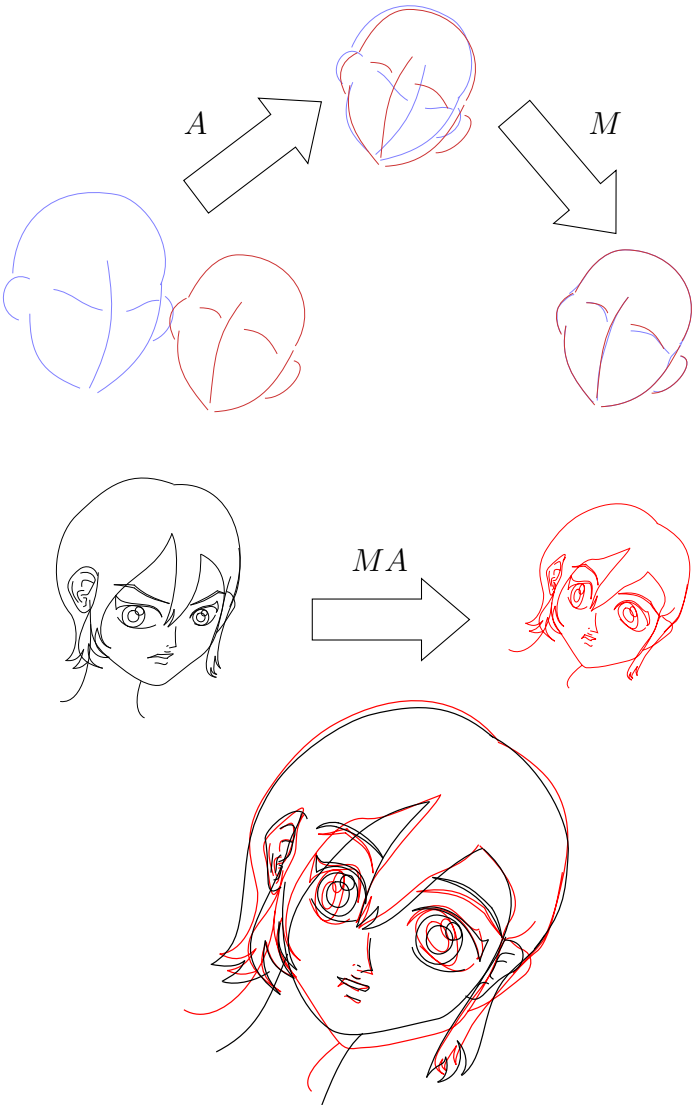


Figura 6.4: A transformação afim A e transformação MFFD M são calculadas a partir das linhas-guias (topo) e são compostas e transformadas ao primeiro quadro-chave (meio). Na parte inferior, o quadro-chave transformado (vermelho) é comparado com o segundo quadro-chave (preto).

É importante computar a transformação de similaridade A antes de M pois em certos casos a animação é predominantemente uma simples rotação e/ou escala entre os quadros-chaves. Isso é ilustrado na Figura 6.5.

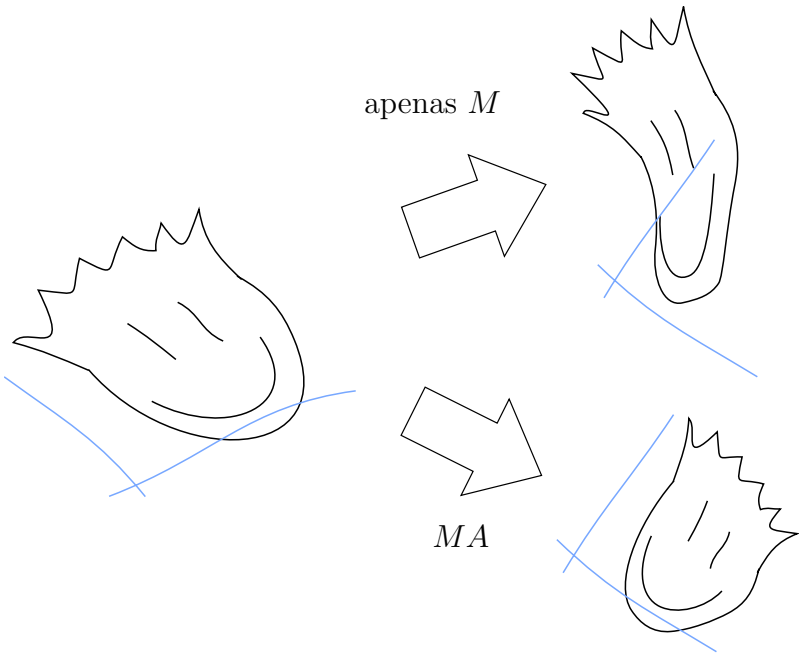


Figura 6.5: Quando a animação é predominantemente uma rotação e escala, o desenho transformado será muito menos distorcido quando A é calculado e aplicado antes de M (abaixo à direita) comparado ao resultado da aplicação de M apenas (acima à direita).

6.1.3 Cálculo de correspondências entre desenhos

Nesse momento, o desenho transformado do quadro-chave i é bastante similar ao desenho do próximo quadro-chave. O artista pode inclusive usar esse quadro-chave transformado como referência para criação do próximo quadro-chave. Essa semelhança entre quadros-chaves é crucial para ajudar o próximo passo do nosso método, i.e., o cálculo das correspondências entre as curvas da arte-final.

Para encontrar as correspondências resolve-se um outro problema de casamento de grafos bipartidos, em que cada vértice do grafo é uma curva do desenho na arte final, e cada aresta conecta curvas dos dois quadros-chaves. Definimos um grafo bipartido K_{N_i,N_j} onde N_i e N_j são o número de curvas na arte-final de i e j respectivamente. Cada aresta possui um peso definido como menos a distância entre as curvas, usando a mesma descrição da Equação 3.2.

Utilizamos o algoritmo descrito na Seção 3.4 para encontrar o casamento de máxima cardinalidade e peso máximo. Entretanto, algumas vezes são associadas curvas que estão muito distantes uma da outra, conforme ilustrado na Figura 6.6.

Essa situação normalmente é indesejável. Para evitar esses casamentos indesejáveis basta eliminar arestas que conectem vértices que estejam a distâncias acima de um determinado limiar. O usuário pode definir o limiar como um valor percentual da distância máxima entre as curvas. Dessa forma, quanto mais alto for o limiar, maior é o número de possíveis associações entre as curvas. O usuário pode ajustar dinamicamente esse limiar até que o resultado inclua todos as correspondências corretas e evita as incorretas. Esse é o único parâmetro no sistema que deve ser definido pelo usuário.

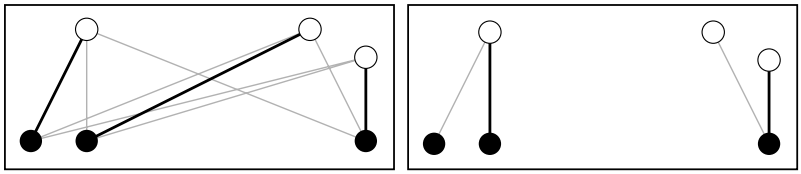


Figura 6.6: Exemplo de casamento em grafo bipartido. Vértices são representados com círculos. Queremos associar vértices brancos com vértices pretos. Arestas são marcadas com linhas. Linhas pretas grossas são os resultados. À esquerda, todas as arestas são utilizadas. À direita, arestas com vértices distantes não são consideradas pelo algoritmo de casamento.

Outra situação indesejada que pode ocorrer é o casamento indevido de curvas semelhantes. Mesmo que o casamento minimize as distâncias entre as curvas, o resultado pode não corresponder ao desejo do artista. Nessa situação, o artista pode indicar ao sistema que houve erro no casamento selecionando curvas que não tenham sido associadas de acordo com seu desejo. Ao fazer isso, o sistema elimina as arestas de K_{N_i,N_j} correspondentes às curvas selecionadas e executa novamente o algoritmo de casamento. A Figura 6.7 ilustra o que ocorre com o resultado do inbetweening quando acontece algum erro de casamento.

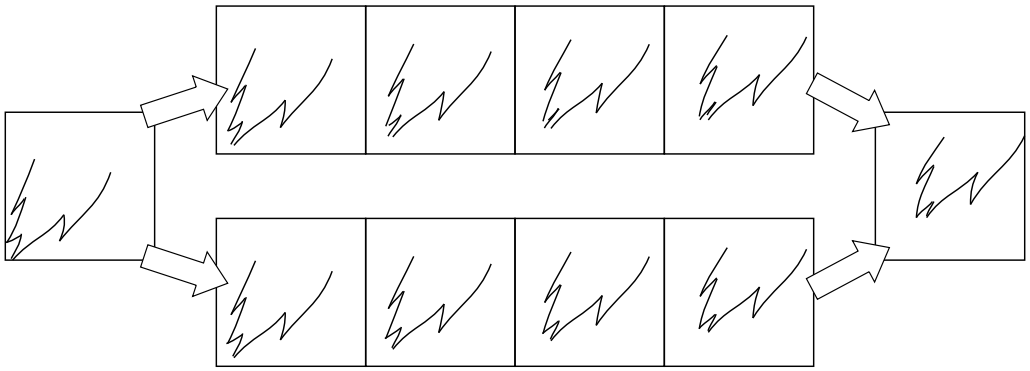


Figura 6.7: Casamento indevido. À esquerda e à direita vemos os quadros-chaves. Acima ao centro: quadros intermediários gerados com um casamento incorreto. Abaixo ao centro: quadros intermediários gerados após a correção do casamento.

Como N_i e N_j podem ser diferentes e algumas arestas não serem incluídas, é possível haver algumas curvas sem correspondências. Isso é comum, por exemplo, quando curvas representam partes do objeto que são obstruídas em um dos quadros-chaves. Essas curvas serão simplesmente ignoradas em nossa interpolação, e o animador deve tratar esses casos manualmente desenhando os traços que faltarem nos quadros intermediários.

6.1.4 Interpolação de quadros-chaves

Finalmente, quadros intermediários são calculados interpolando-se curvas dos quadros-chaves. Há diversas abordagens que podem ser utilizadas para esse propósito, como os métodos desenvolvidos por SEDERBERG e GREENWOOD [38], CONG e PARVIN [39], FU *et al.* [40], e WHITED e ROSSIGNAC [41]. Utilizamos um método que funciona associando formas similares, baseado no trabalho de SEDERBERG e GREENWOOD [9], que é simples e produz resultados satisfatórios. Entretanto o algoritmo de forma geral não depende desse método, de forma que ele pode ser substituído por outros métodos caso características específicas sejam desejáveis. O método que empregamos é descrito a seguir.

Para que seja possível interpolar duas curvas é necessário primeiramente fazer algumas verificações e ajustes, conforme descrevemos a seguir.

Suponha que c_i e c_j sejam um par de curvas que queremos interpolar. Primeiramente analisamos a orientação, pois há duas opções de correspondência entre essas curvas: usando a mesma orientação para ambas, ou invertendo a orientação de uma delas, veja Figura 6.8. Assim como foi feito com as linhas-guias, para verificar qual orientação produz as melhores correspondências, comparamos o valor de $\mathrm{dfw}(c_i, c_j)$ e $\mathrm{dbw}(c_i, c_j)$, de maneira que a orientação original das duas curvas é utilizada caso $\mathrm{dfw}(c_i, c_j) < \mathrm{dbw}(c_i, c_j)$, caso contrário a orientação de uma das curvas é invertida.

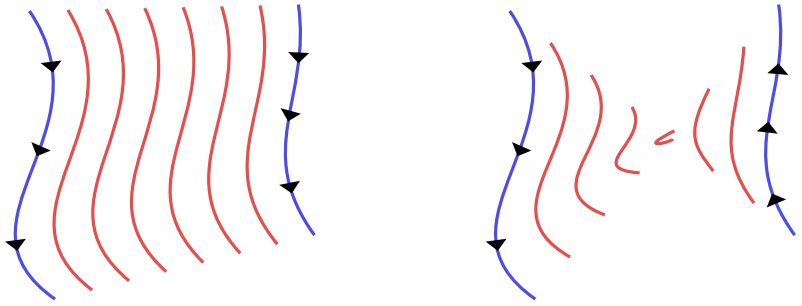
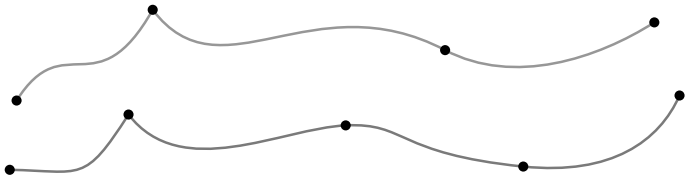


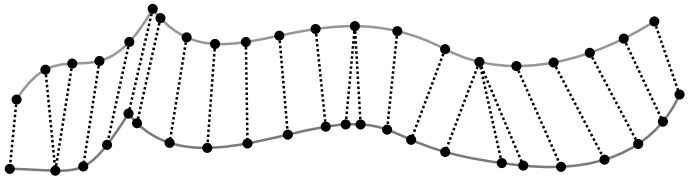
Figura 6.8: Curvas (em azul) podem ser interpoladas de duas formas de acordo com a ordem dos pontos de controle (curvas interpoladas são mostradas em vermelho). Setas indicam a orientação usada para cada curva. O modo preferencial é obtido pela comparação dos valores de dfw e dbw para essas curvas. Neste caso, é o mostrado à esquerda.

Para interpolar as curvas c_i e c_j é necessário associar cada segmento cúbico de uma curva para a outra, mas normalmente os segmentos não se encaixam naturalmente, i.e., as curvas contêm números diferentes de segmentos, ou mesmo quando são iguais, porções muito distintas podem ser associadas umas às outras.

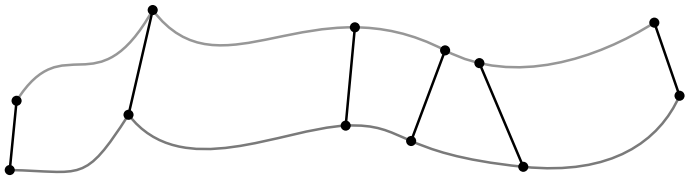
Para resolver esse problema, recortamos cada curva de forma a equalizar o número de segmentos e conectar segmentos semelhantes, ou seja, obtemos as curvas $\tilde{c}_i$ e $\tilde{c}_j$ que são visualmente iguais à c_i e c_j, respectivamente, contendo o mesmo número de pontos de controle, de forma que os segmentos de $\tilde{c}_i$ correspondam a segmentos semelhantes de $\tilde{c}_j$. Na Figura 6.9 vemos um exemplo dessa operação, que será detalhada a seguir.



(a) Curvas originais com seus pontos extremos.



(b) Correspondência entre amostras.



(c) Correspondências que contêm pontos extremos.

Figura 6.9: Equalização do número de segmentos entre duas curvas.

A Figura 6.9(a) mostra um exemplo de duas curvas, em que é possível ver também os pontos extremos (i.e., pontos que estão no início ou no fim de cada segmento cúbico). Seja $0 = t_0^a < t_1^a < \cdots < t_{n_a}^a = 1$ a sequência utilizada na parametrização da curva c_i, onde n_a é o número de segmentos de Bézier cúbicos em c_i, e similarmente $0 = t_0^b < t_1^b < \cdots < t_{n_b}^b = 1$ para a curva c_j. As curva c_i e c_j são amostradas uniformemente, usando $n = 5 \cdot \max(n_a, n_b)$ amostras para cada curva. Esse número n é utilizado em ambas as curvas porque o algoritmo descrito em [9] apresenta melhores resultados quando ambas as curvas possuem um número semelhante de pontos. Nem todos os pontos extremos são incluídos nas amostras, de fato os pontos em sua maioria muito provavelmente não serão inclusos, então esses pontos extremos são adicionados às amostras. Para evitar componentes demasiadamente pequenas,

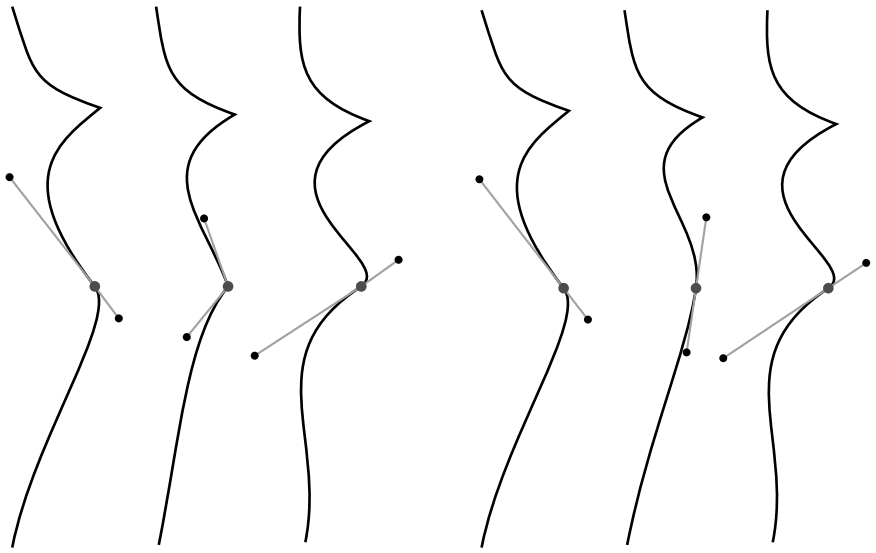
se um ponto extremo estiver próximo demais (dada uma determinada tolerância) de outra amostra, essa é eliminada da amostragem. O critério utilizado nesse caso é quando $|t - t_k| < 0.2/(n-1)$, onde t é o parâmetro usado para a amostra, e t_k é o parâmetro utilizado para o ponto extremo. Isso é feito em ambas as curvas c_i e c_j. Aplicamos o algoritmo de [9] a essas amostras, o que nos dá um casamento com a melhor combinação entre pontos das duas sequências de amostras.

A Figura 6.9(b) mostra o resultado obtido desse casamento das amostras. Utilizando esse casamento, selecionamos os pares de pontos que contêm pelo menos um ponto extremo, criando assim duas sequências de parâmetros $0 < \tilde{t}_0^a < \cdots < \tilde{t}_{\tilde{n}}^a = 1$ e $0 < \tilde{t}_0^b < \cdots < \tilde{t}_{\tilde{n}}^b = 1$, de forma que a amostra $c_i(\tilde{t}_k^a)$ corresponde a $c_j(\tilde{t}_k^b)$, para $k \in [0, \ldots, \tilde{n}]$, onde $\tilde{n}+1$ amostras foram consideradas.

Criamos então a curva $\tilde{c}_i$ que é igual a c_i, mas com $\tilde{n}$ segmentos de Bézier cúbicos, criada por cortes da curva c_i em cada valor t_k^a que não seja um ponto extremo, e de forma similar para a curva c_j, criando a curva $\tilde{c}_j$.

Essa operação é mostrada na Figura 6.9(c), em que é possível ver as correspondências que contêm pontos extremos, que serão os pontos extremos das curvas resultantes após os cortes. Agora temos um par de curvas com o mesmo número de pontos de controle e cada par de segmentos possui forma similar. Com isso, podemos obter curvas intermediárias por meio de uma interpolação linear dos pontos de controle de $\tilde{c}_i$ com $\tilde{c}_j$.

Entretanto, essa interpolação não garante que pontos G1-contínuos mantenham essa propriedade nas curvas intermediárias, conforme pode ser visto no exemplo da Figura 6.10.



(a) Descontinuidade formada após interpolação linear.

(b) Continuidade G1 imposta.

Figura 6.10: Interpolação e controle de continuidade G1.

Para evitar isso, podemos simplesmente impor continuidade G1 (conforme visto na Seção 3.2.11) sempre que ambas as curvas originais forem G1-contínuas nos pontos utilizados.

Definimos então a função interp(c^1, c^2, α) que computa uma nova curva que é uma interpolação das curvas c^1 e c^2, ambas com o mesmo número de pontos de controle. O parâmetro $\alpha \in [0, 1]$ controla a interpolação de forma que interp$(c^1, c^2, 0) = c^1$ e interp$(c^1, c^2, 1) = c^2$. Primeiramente, os pontos de controle das curvas c^1 e c^2 são interpolados linearmente com parâmetro α. Então, continuidade G1 é imposta para cada ponto de controle se as curvas c^1 e c^2 forem ambas G1-contínuas nos pontos de controles correspondentes. Alguns resultados da função interp podem ser vistos na Figura 6.11.

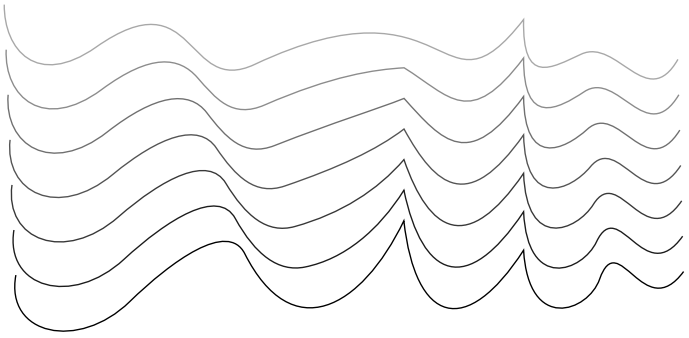


Figura 6.11: Resultado da função interp: as curvas mais abaixo e mais acima são interpoladas utilizando 7 valores diferentes para $\alpha \in [0, 1]$.

Dado um quadro k entre os quadros-chaves i e j, seja c_k o resultado da interpolação de curvas correspondentes c_i e c_j. Um método simples para obtermos c_k seria por meio da interpolação linear das curvas c_i e c_j, ou seja, utilizando $c_k = \text{interp}(\tilde{c}_i, \tilde{c}_j, \alpha,$ para $\alpha \in (0, 1)$. Entretanto essa maneira pode gerar resultados distorcidos, como mostrado na Figura 6.12.

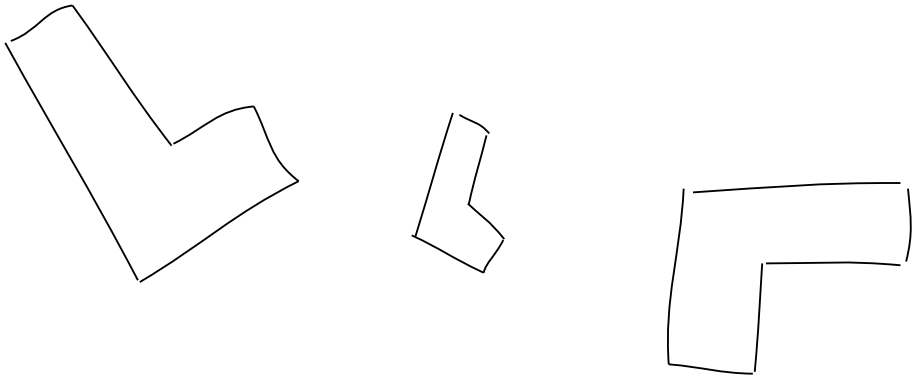


Figura 6.12: Distorções causadas pela interpolação linear das curvas. O desenho do meio é o resultado da interpolação linear dos outros desenhos.

Para evitar esse problema, utilizamos a transformação de similaridade A que mapeia linhas-guias calculada no início do método (Seção 6.1.2). Inicialmente aplicamos A à curva $\tilde{c}_i$. O resultado é interpolado com a curva $\tilde{c}_j$ por meio da função interp e depois transformado para uma posição intermediária. A curva resultante c_k é definida então por

$$c_k = A_k(\text{interp}(A(\tilde{c}_i), \tilde{c}_j, \alpha)), \tag{6.2}$$

onde $A(\tilde{c}_i)$ é a curva obtida aplicando-se a transformação A aos pontos de controle de $\tilde{c}_i$, e A_k é uma transformação de similaridade que move a curva interpolada para uma posição intermediária entre as curvas originais c_i e c_j. O parâmetro $\alpha \in (0, 1)$ é descrito na Seção 5.5. Para calcular A_k, sejam s, θ e t os parâmetros de A^{-1} (a inversa de A), representando sua escala, ângulo de rotação, e translação, respectivamente. Defina $A_{i\alpha}$ como a transformação de similaridade obtida pela interpolação de A^{-1} com a transformação identidade, i.e., seu parâmetro de escala é dado por $(1-\alpha)s+\alpha$, o ângulo de rotação é $(1-\alpha)\theta$, e o vetor de translação é $(1-\alpha)t$. Como essas transformações são centradas no ponto $(0, 0)$, se α varia entre 0 e 1 enquanto se aplica $A_{i\alpha}$ a um ponto, será formada uma trajetória que normalmente é indesejável, pois a forma rotaciona ao redor de $(0, 0)$ ao invés do seu centro de rotação. Para corrigir isso o usuário pode especificar centros de rotação, que são usados para transladar os pontos resultantes para uma posição desejável. Sejam o_i e o_j os centros de rotação de dois quadros-chaves sendo processados, $o_\alpha = (1-\alpha)o_i + \alpha o_j$, e T uma translação com direção $o_\alpha - A_{i\alpha}(o_j)$. Finalmente, defina $A_k = A_{i\alpha} \circ T$. Figura 6.13 mostra o resultado de uma interpolação completa entre duas curvas, usando centros de rotação em posições diferentes.

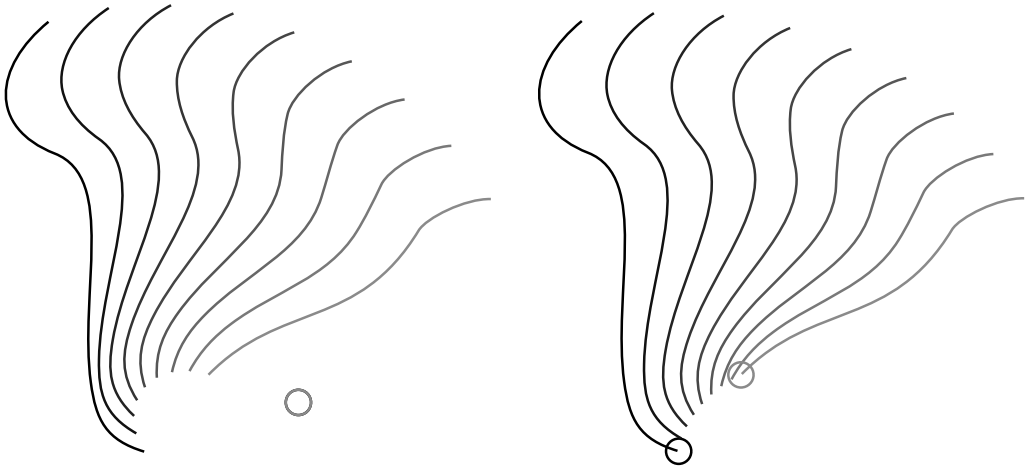


Figura 6.13: Exemplo de interpolação de curvas usando centros de rotações diferentes (marcados com círculos). À esquerda, os centros para ambos os quadros são os mesmos. À direita, os centros são posicionados em pontos extremos das curvas, produzindo um resultado diferente.

A Figura 6.14 ilustra o resultado desse método ao exemplo da Figura 6.12, observe que agora a forma do desenho é preservada no desenho interpolado.

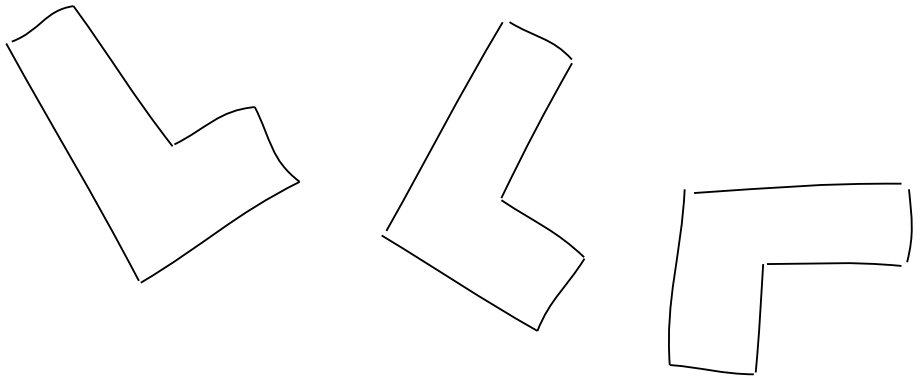


Figura 6.14: A distorção ilustrada na figura anterior é reduzida ao se utiliza a informação da transformação A para interpolar os dois quadros.

6.2 Regiões

Para permitir objetos coloridos, o artista pode selecionar uma sequência de curvas para definir o contorno de uma região, que pode ser colorida como desejado. As curvas podem ser removidas do desenho, mas mantidas na representação das regiões.

Após a inferência de correspondências entre curvas é necessário inferir também a correspondência entre regiões. Mais uma vez, o algoritmo de casamento de grafos bipartidos é usado. Cada nó do grafo agora representa uma região, e as arestas conectam cada par de regiões que possuem pelo menos um par de curvas correspondentes em comum. Os pesos das arestas são dados por menos a distância entre regiões, definida como o máximo entre as distâncias das curvas correspondentes (como antes, considerando o primeiro quadro-chave sendo transformado por $M \circ A$).

Para interpolar regiões, precisamos interpolar todas as curvas usadas em seus contornos. O algoritmo da Seção 6.1.3 pode não ter encontrado correspondências para algumas curvas, então novas correspondências são inferidas para essas curvas. Seja R_i uma região definida por m_i curvas de contorno $c_i^1, c_i^2, \ldots, c_i^{m_i}$, e R_j uma região definida por m_j curvas $c_j^1, c_j^2, \ldots, c_j^{m_j}$. Suponha, sem perda de generalidade, que as curvas seguem a mesma orientação. Sejam (c_i^p, c_j^q) um par de curvas correspondentes, e (c_i^r, c_j^s) o próximo par de curvas correspondentes seguindo a orientação das regiões. Observe que os pares podem ser disjuntos, então uma nova correspondência é criada casando a curva formada pela sequência de curvas entre c_i^p e c_i^r com a curva formada pela sequência entre c_j^q and c_j^s. Se não há curva entre c_i^p e c_i^r, então uma nova curva é criada formada por um único ponto (o ponto de interseção entre essas curvas).

De forma semelhante para curvas c_j^q e c_j^s. Finalmente, interpolamos as novas curvas usando o método descrito na Seção 6.1.4. A Figura 6.15 ilustra esse processo.

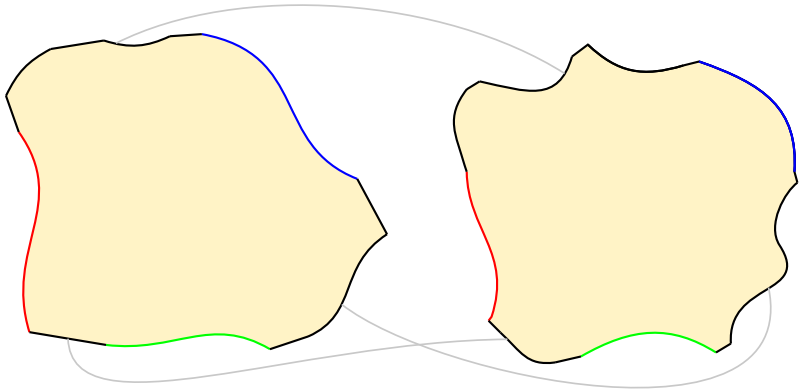


Figura 6.15: Correspondências entre regiões. Curvas correspondentes são marcadas com as mesmas cores. As linhas cinzas indicam novas correspondências para curvas intermediárias (em preto).

6.3 Composição em camadas

É comum que desenhos contenham curvas que se sobrepõem em diferentes partes daquilo que é retratado. Isso pode induzir um erro na busca por curvas correspondentes, em que pares não desejáveis podem ser formados se seus pesos de correspondência são menores do que os pesos dos pares esperados. Como comentamos na Seção 4.1.2, o DiLight permite dividir partes do desenho em camadas, e o método é aplicado separadamente para cada camada.

A composição de camadas não apenas evita a formação de pares indesejáveis, como também permite elementos cruzarem uns aos outros, o que é mais conveniente do que restringir a animação a elementos que não se cruzem. Além disso, separar desenhos em camadas é um processo tipicamente usado por artistas [1].

Algumas vezes é útil especificar uma correlação entre camadas, de forma que o o movimento de uma camada "mãe" é aplicado às suas camadas "filhas".

Isso nos leva a uma hierarquia de camadas, na qual a transformação aplicada a uma camada também é aplicada a suas sub-camadas. Se a camada atual não é uma sub-camada, então as equações anteriores 6.1 e 6.2 são usadas. Caso contrário as equações precisam ser reformuladas. Suponha que a camada atualmente processada é uma sub-camada da camada p. Primeiramente, a transformação de similaridade (Equação 6.1) é redefinida como

$$A = \text{similarity}(A_p(G^i), G^j), \tag{6.3}$$

onde A_p é a transformação de similaridade da camada p. A Equação 6.2 é alterada de acordo para

$$c^k = A_{kp} \circ A_k(\text{interp}(A \circ A_p(\tilde{c}_i), \tilde{c}_j, \alpha)), \tag{6.4}$$

onde A_{kp} corresponde à transformação intermediária A_k calculada para a camada p. Ao computar A_k, o centro de rotação muda para $o_\alpha = (1 - \alpha)A_p(o_i) + \alpha o_j$.

6.4 Controle de tempo

Como vimos na Seção 5.5, a interface do DiLight possui também um controle da velocidade. Para cada par de quadros-chaves existe uma curva de Bézier cúbica, que ao ser manipulada altera a velocidade da animação entre esses dois quadros-chaves.

Essa curva é utilizada para definir o parâmetro α utilizado durante a interpolação de quadros-chaves (ver Seção 6.1.4), de forma que $\alpha = C((k - i)/(j - i))$, onde C é uma parametrização dessa curva pela coordenada x.

A curva é definida por uma curva de Bézier cúbica, cujos primeiro ponto de controle está fixo em $(0, 0)$ e o último ponto de controle está fixo em $(1, 1)$. Os outros dois pontos de controle podem ser manipulados, restritos apenas ao quadrado unitário. Inicialmente a curva corresponde à função identidade. Um movimento acelerado é definido quando a curva se encontra abaixo da linha de identidade, e é desacelerado caso a linha se encontre acima da identidade. Esse comportamento é ilustrado na Figura 6.16.

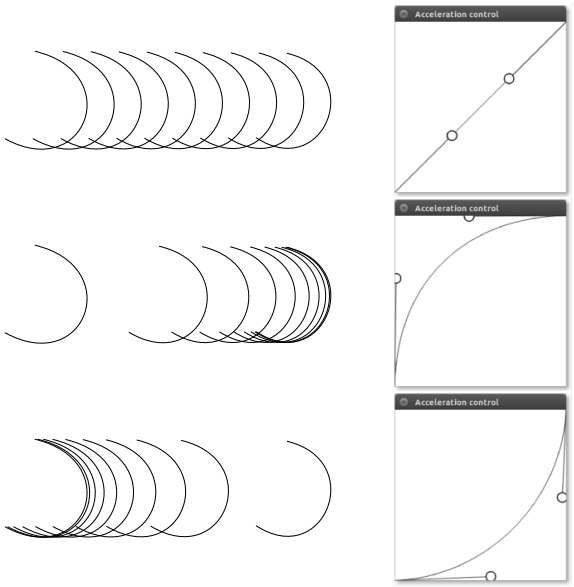


Figura 6.16: Controle de aceleração. À esquerda vemos uma animação utilizando três curvas de velocidade (à direita). Acima: variação constante. Ao centro: movimento desacelerado. Abaixo: movimento acelerado.

Capítulo 7

Resultados

Neste capítulo serão mostrados resultados obtidos por meio do sistema desenvolvido conforme descrito nos capítulos anteriores.

7.1 Animações

As Figuras 4.2 e 7.1 mostram uma reprodução baseada em Tenzin, um personagem da série de animação *The Legend of Korra* [42]. A Figura 7.1 ilustra o resultado do inbetweening entre dois quadros-chaves representando o movimento do rosto de Tenzin.

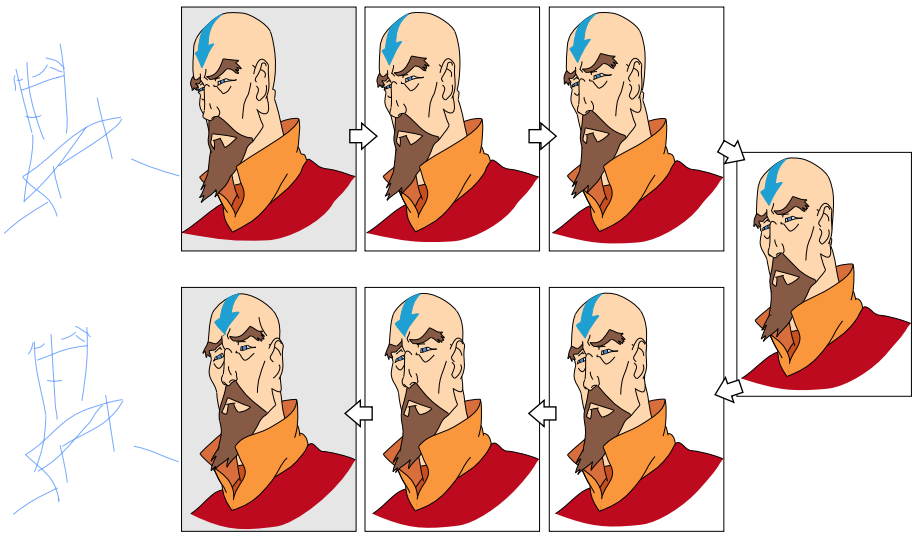


Figura 7.1: Tenzin. Os quadros-chaves (dentro de retângulos cinzas) com suas linhas-guias (à esquerda), e cinco quadros intermediários gerados pelo método.

O conjunto de linhas-guias é simples (19 traços) mas é suficiente para definir a metamorfose entre os quadros. Alguns detalhes são parcialmente obstruídos no primeiro quadro-chave, como as rugas abaixo do olho direito, e a parte detrás da

orelha. Graças à separação em camadas, isso não foi um problema para o método.

A Figura 7.2 mostra alguns quadros de uma sequência de um cavalo andando. Cinco quadros intermediários foram gerados para cada par de quadros-chaves. As setas indicam a sequência de quadros da animação.

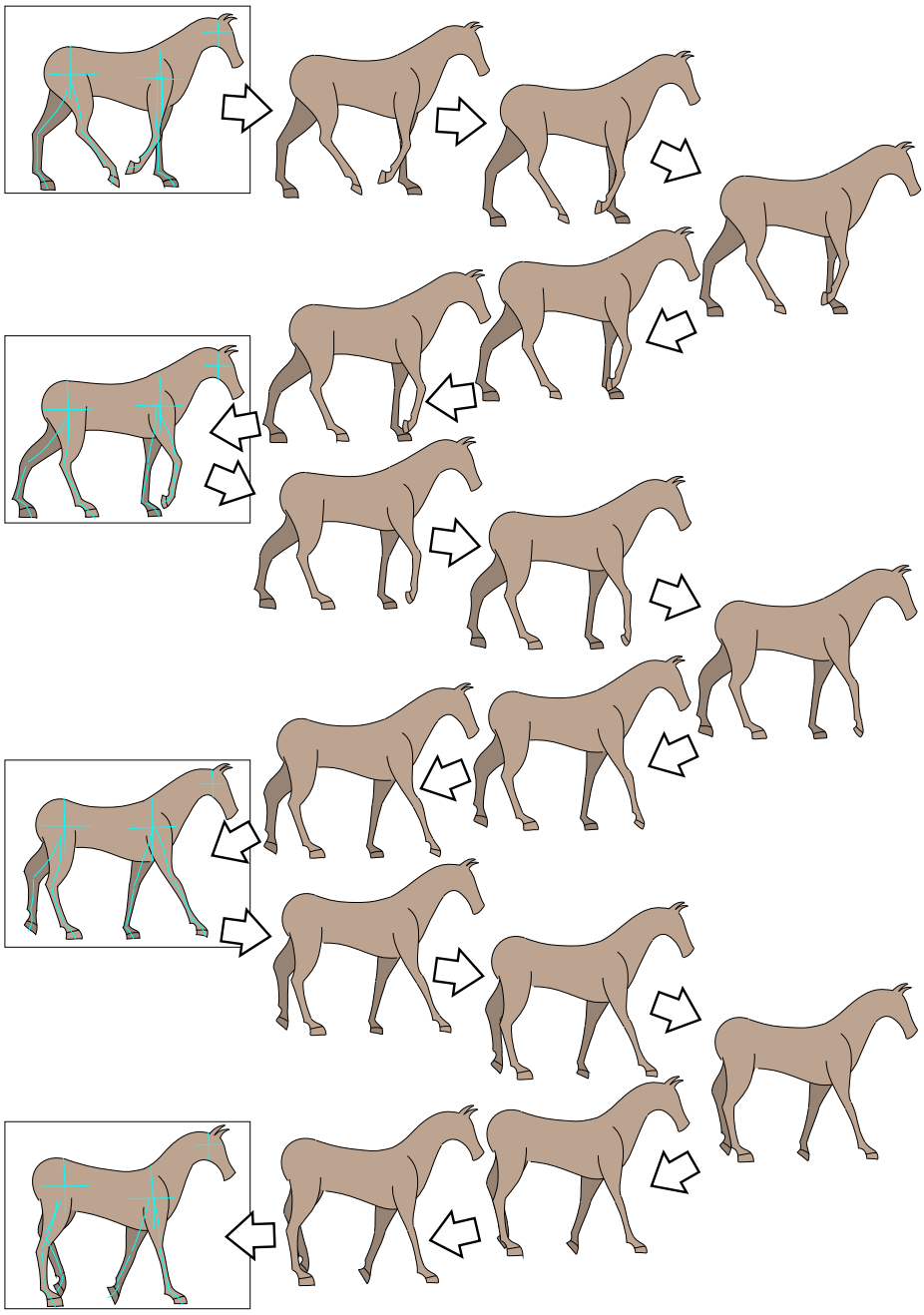


Figura 7.2: Cavalo andando, quadros-chaves estão dentro de retângulos. Linhas-guias são mostradas em ciano.

Apesar de haver grandes obstruções quando as pernas se cruzam, elas são corretamente tratadas por estarem em camadas separadas. Foram usadas cinco camadas, cada perna em uma camada e uma camada para o corpo e cabeça do cavalo.

A Figura 7.3 mostra uma animação de um tiranossauro-rex. São mostrados três quadros intermediários. Foi necessário separar em camadas os dentes que estão na frente dos dentes que estão na parte de trás, caso contrário haveria correspondências erradas devido à semelhança entre os dentes.

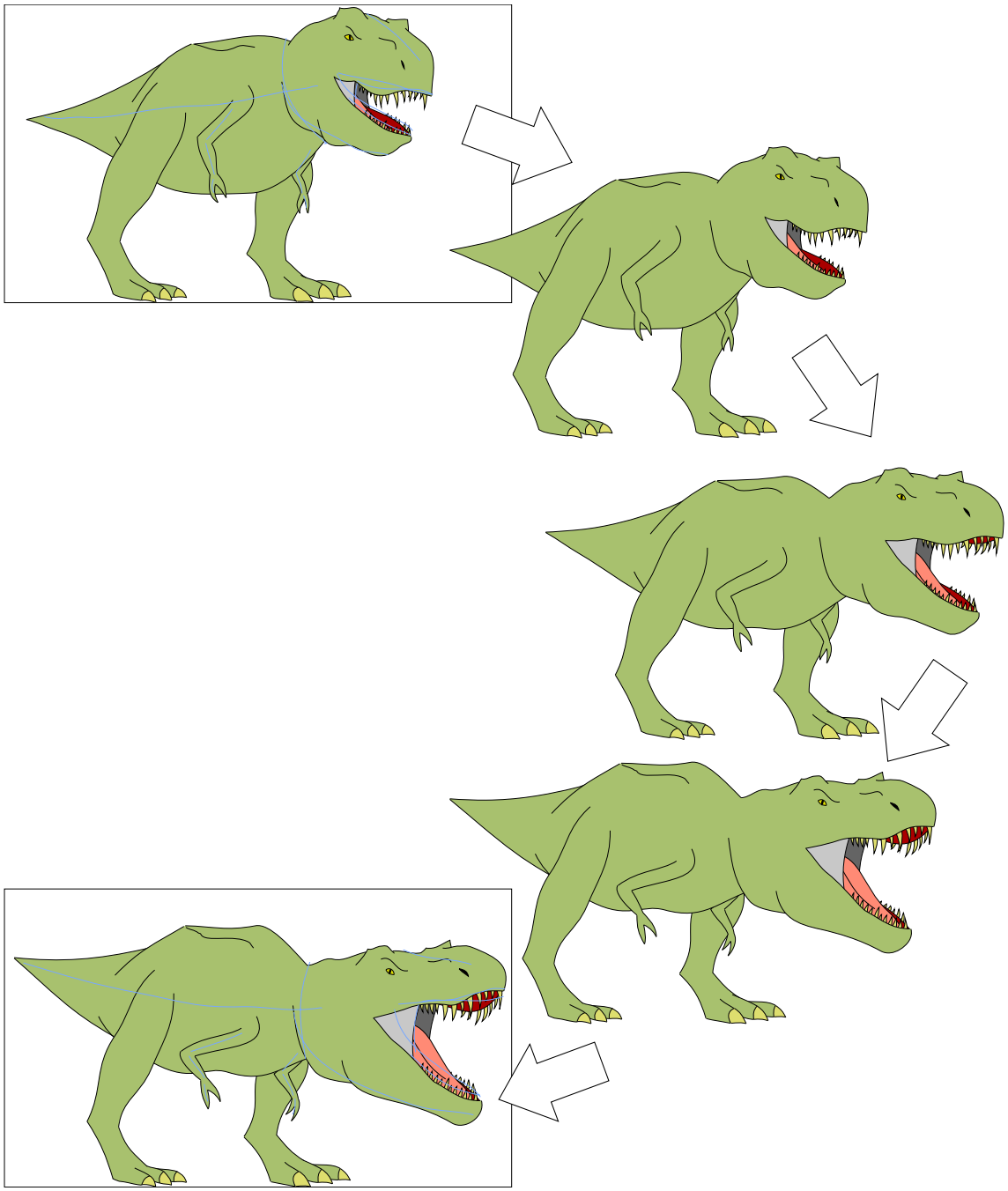


Figura 7.3: Tiranossauro-rex, os quadros-chaves estão dentro de retângulos. Linhas-guias são mostradas em azul. As setas indicam a sequência de quadros da animação.

A Figura 7.4 mostra alguns quadros de uma mão se movendo. Quatro quadros intermediários foram gerados para cada par de quadros-chaves. Apenas o polegar foi posicionado em uma camada separada, visto que ele passa por cima das outras partes da mão.

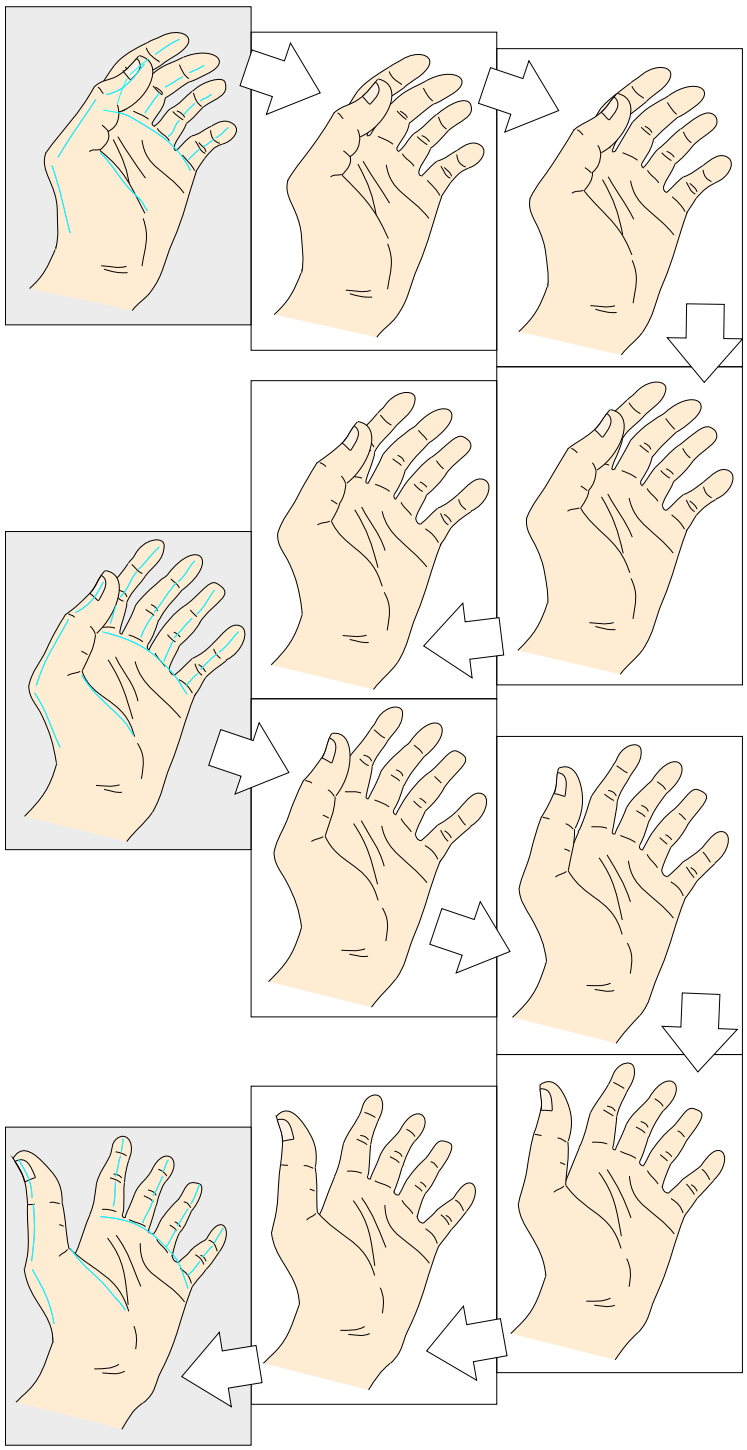


Figura 7.4: Mão se movendo, quadros-chaves estão dentro de retângulos cinzas. Linhas-guias são mostradas em ciano.

Pedimos a alguns animadores profissionais para testar o DiLight, criando algumas animações. A Figura 7.5 mostra um exemplo de animação criada por um animador profissional, usando sete camadas.

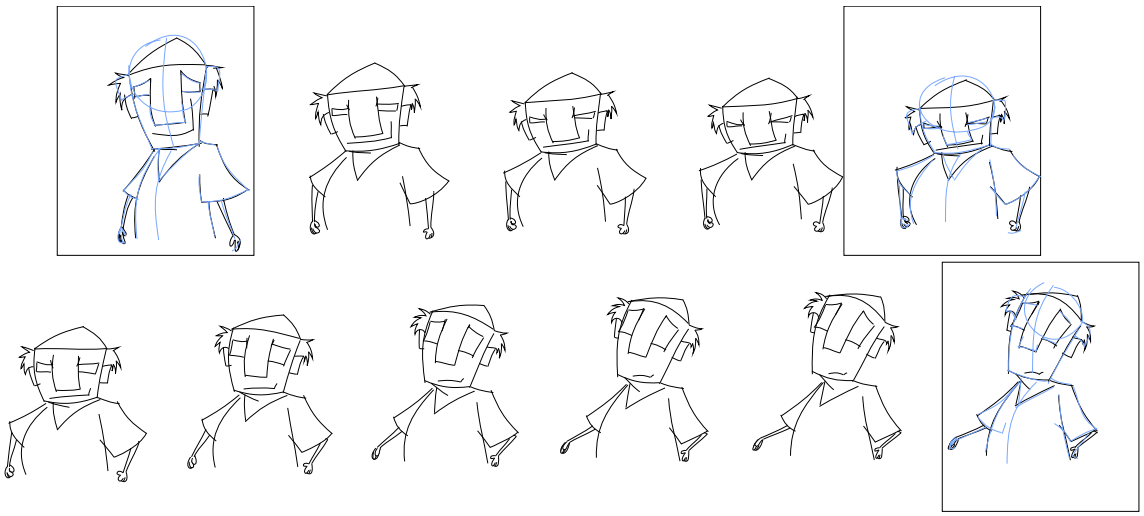


Figura 7.5: Exemplo de animação. Os quadros-chaves estão dentro de retângulos. Linhas-guias são mostradas em azul. ©Diogo Viegas

Na Figura 7.6 há um exemplo de animação criado por outro animador profissional. Essa animação possui apenas dois quadros-chaves, porém é dividida em 28 camadas para criar diferentes efeitos de sombreamento.

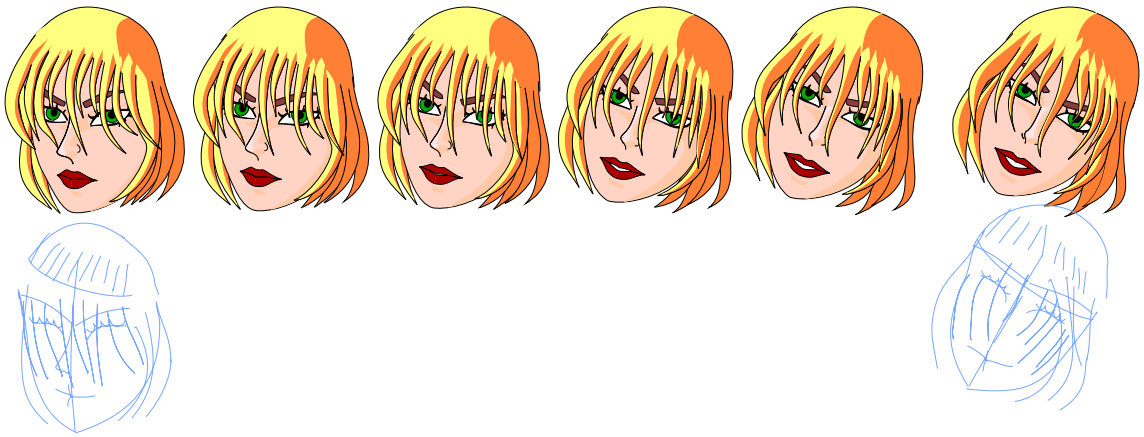


Figura 7.6: Exemplo de animação. As linhas-guias são exibidas em azul, abaixo dos quadros-chaves. ©Leandro Araujo

Na Figura 7.7 há uma reprodução de Edward da animação Fullmetal Alchemist: Brotherhood [43].

Figura 7.7: Edward. Quadros-chaves estão dentro de retângulos. Linhas-guias são mostradas em azul. A animação segue da esquerda para a direita, de cima para baixo.

Os próximos resultados foram criados baseados em exemplos do livro *The Animator's Survival Kit* [1], uma obra feita por Richard Williams, um animador profissional renomado, bastante utilizada por artistas que querem aprender a criar animações. O livro é útil para diversos estilos de animação, seja tradicional, *stop motion*, ou criadas com o auxílio de computadores.

A Figura 7.8 mostra um exemplo de animação de um homem levantando uma caixa, usando quatro quadros-chaves e três quadros intermediários entre cada par de quadros-chaves. O desenho foi separado em seis camadas para permitir sobreposição entre pernas, braço, corpo, cabeça e a caixa.

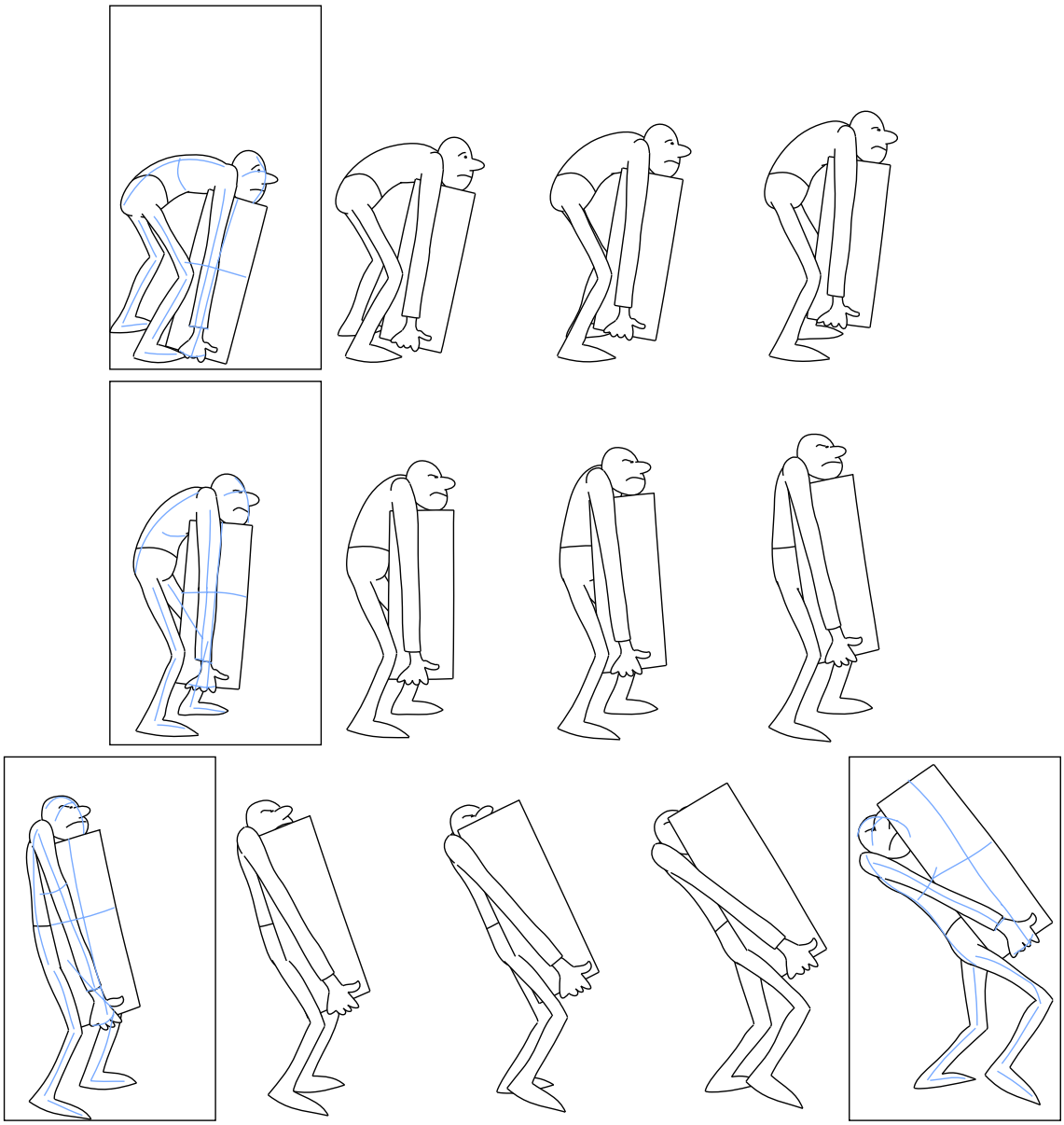


Figura 7.8: Levantamento de uma caixa. Quadros-chaves são mostrados dentro de retângulos. Linhas-guias são mostradas em azul. A animação segue da esquerda para a direita, de cima para baixo.

A Figura 7.9 mostra um homem caminhando sorrateiramente, com três quadros-chaves e cinco quadros intermediários entre cada par de quadros-chaves. Esse exemplo apresenta uma grande distorção entre os desenhos, entretanto, é tratado adequadamente pelo nosso método.

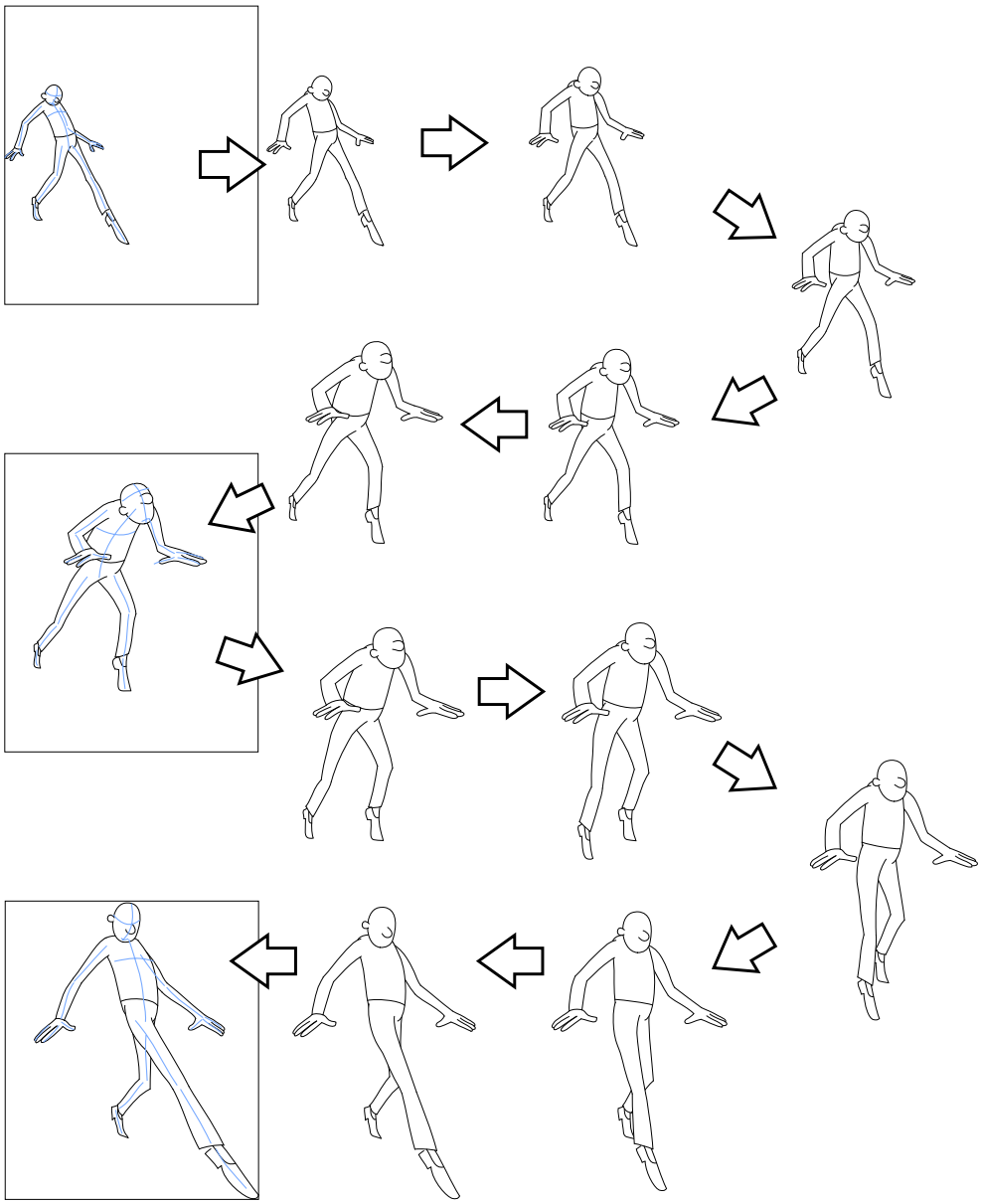


Figura 7.9: Caminhando sorrateiramente. Quadros-chaves estão dentro de retângulos. Linhas-guias são mostradas em azul.

Na Figura 7.10, uma mulher move sua mão e quadris, oito quadros-chaves foram usados, com três quadros intermediários entre cada par de quadros-chaves. O movimento do braço é inferido adequadamente pela separação do braço em sub-camadas, e pelo posicionamento dos centros de rotação de cada sub-camada em uma junta.

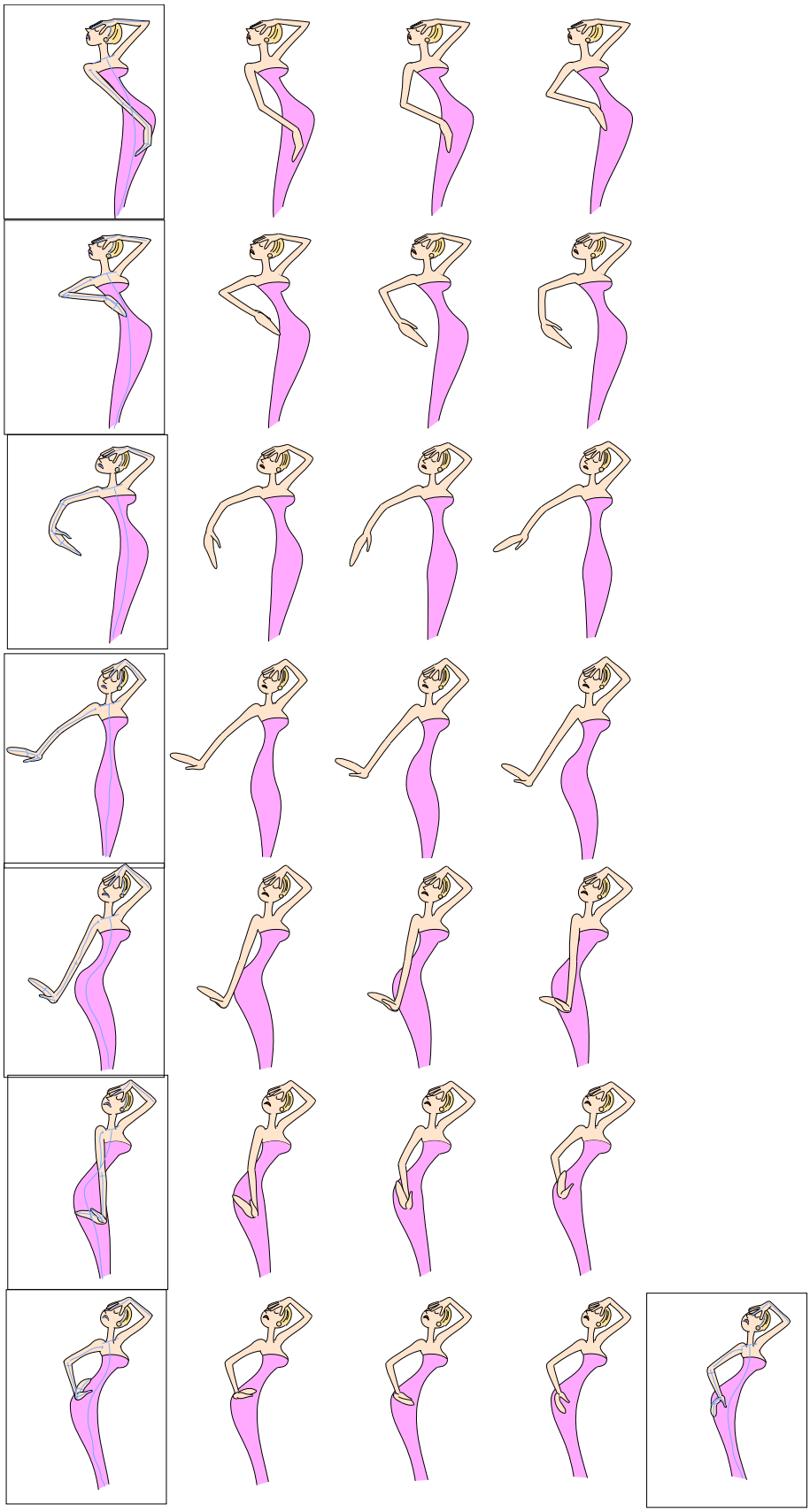


Figura 7.10: Mulher movendo sua mão, quadros-chaves estão dentro de retângulos. Linhas-guias são mostradas em azul. Animação segue da esquerda para a direita, de cima para baixo.

Na Figura 7.11 há um homem gritando. Foram utilizados oito quadros-chaves, com três quadros intermediários entre cada par de quadros-chaves.

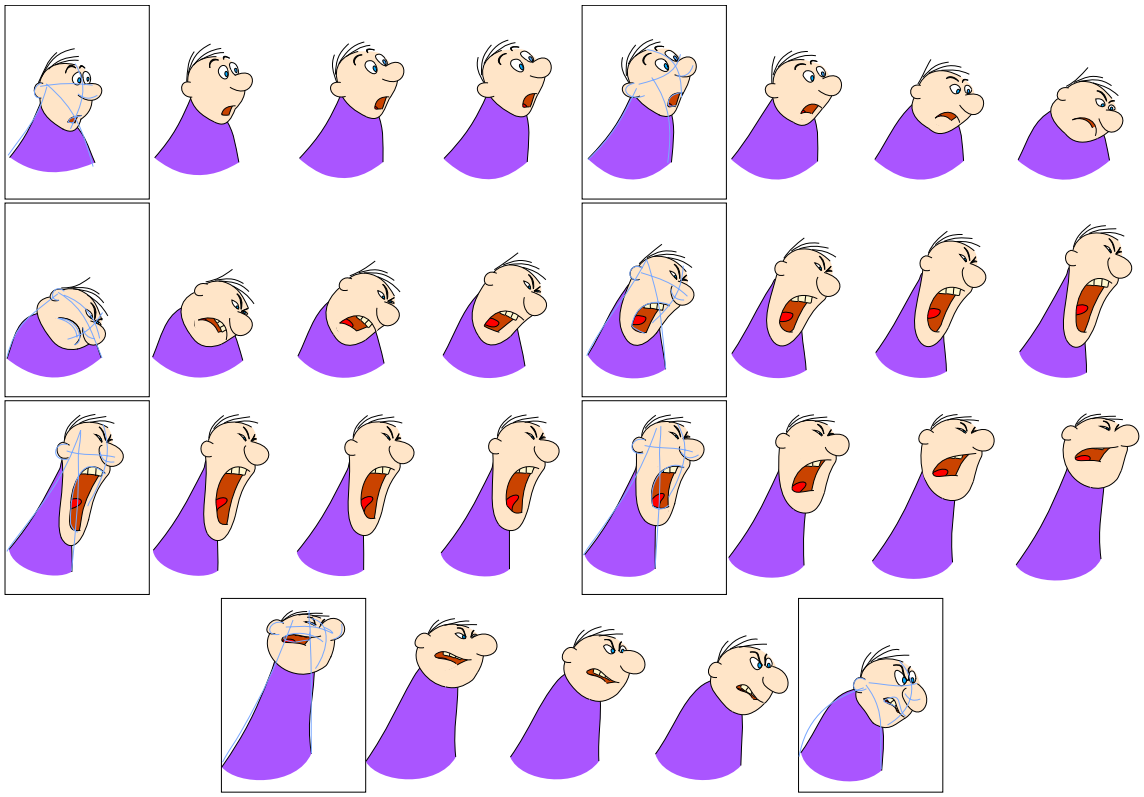


Figura 7.11: Homem gritando. Quadros-chaves estão dentro de retângulos. Linhas-guias são mostradas em azul. A animação segue da esquerda para a direita, de cima para baixo.

7.2 Desempenho

A aplicação foi desenvolvida em C++, utilizando a biblioteca Qt5 para a interface gráfica. Foi utilizado o compilador GCC-4.8.4 configurado com as opções de otimização: "-march=native -O3". Os resultados foram gerados com um Intel Core i7, com 16GB de memória RAM, porém conseguimos desempenho similar mesmo com um netbook de configuração simples.

O procedimento mais caro computacionalmente é o cálculo das funções de MFFD, que envolve o cálculo de diversos valores em grades de grandes dimensões.

Os resultados mostrados na seção anterior foram simplificados para melhor apresentação neste trabalho. Nas animações completas foram gerados mais quadros intermediários, e algumas possuem mais quadros-chaves.

A tabela 7.2 apresenta um comparativo entre os resultados, indicando para cada exemplo: a quantidade de camadas, linhas-guias e curvas/regiões no primeiro

quadro-chave (há pouca variação desses dados nos quadros-chaves seguintes); a quantidade de quadros-chaves e quadros intermediários; o limiar de corte de arestas no grafo de correspondências em relação à distância máxima entre as curvas; o tempo em milissegundos que o sistema levou para executar o método completo nos testes realizados.

	Camadas	Linhas-guias	Curvas e regiões	Quadros-chaves	Quadros intermediários	Limiar de corte	Tempo de execução
Tenzin	9	19	127	3	48	20%	320ms
Cavalo	6	18	50	9	40	20%	189ms
Tiranossauro-rex	9	15	189	3	49	20%	356ms
Mão	3	18	73	4	27	20%	123ms
Diogo Viegas	7	32	56	4	30	40%	125ms
Leandro Araujo	28	51	291	2	37	50%	404ms
Edward	17	46	281	3	65	55%	858ms
Levantando caixa	8	15	46	4	57	60%	150ms
Sorrateiro	7	22	56	3	78	55%	193ms
Mulher	6	14	58	8	35	40%	282ms
Gritando	5	7	38	8	70	58%	224ms

Tabela 7.1: Comparativo entre os resultados.

É possível ainda reduzir o tempo de processamento consideravelmente se vez em de calcular toda a animação, o método só for usado quando houver mudança nos quadros-chaves, evitando assim processamento desnecessário.

7.3 Limitações

Como é comum em métodos de inbetweening assistidos por computador, o método não é capaz de tratar mudanças exageradas no desenho. Como a entrada possui $2^1/_2$ dimensões, ela não contém toda informação necessária para tratar apropriadamente a forma dos elementos nos quadros intermediários. Ainda assim o método é capaz de tratar mudanças bem amplas, como na Figura 7.9.

O método que descrevemos funciona bem quando o movimento em uma camada de um quadro-chave ao seguinte pode ser descrito aproximadamente por uma transformação de similaridade, caso contrário os algoritmos estão propensos a encontrar correspondências indesejáveis. O desenho precisa ser organizado em camadas para o sistema funcionar com movimentos mais complexos. Isso é ilustrado na Figura 7.12.

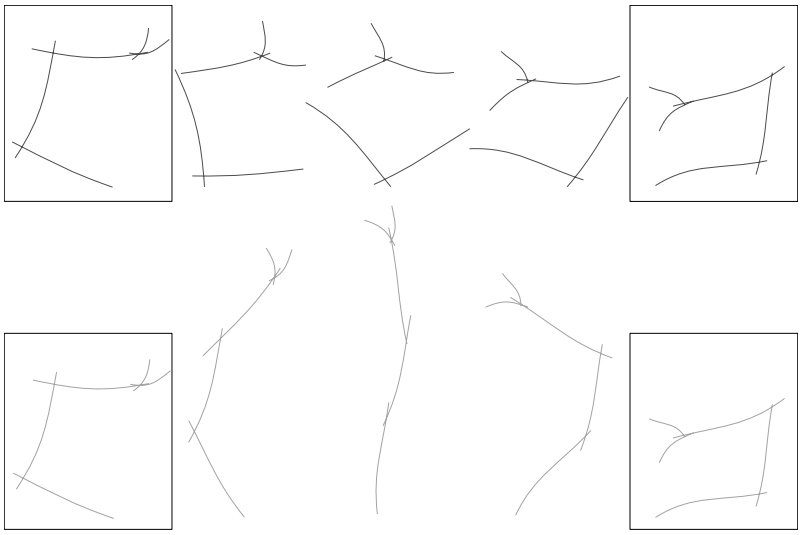


Figura 7.12: Acima: caso de falha, o algoritmo de casamento falha quando o movimento em uma camada é mais complexo do que uma transformação de similaridade. Abaixo: o mesmo exemplo, separado em camadas para corrigir as correspondências. Quadros-chaves estão dentro de retângulos.

Outra limitação é que se grandes partes do objeto estão completamente obstruídas de um quadro para o outro, normalmente não será gerado um resultado adequado. Pode funcionar caso a parte obstruída e a que está obstruindo estejam localizadas em diferentes camadas e suas formas não são relacionadas, como as pernas do cavalo na Figura 7.2. De forma similar, se um olho está escondido pela própria face, ele deve ser definido em outra camada, como ocorre entre o segundo e o terceiro quadro-chave do exemplo da Figura 7.10), quando a mulher virou sua cabeça revelando seu olho direito.

Além disso, entre o último par de quadros-chaves na Figura 7.10, a mão direita muda de orientação, de forma que a palma da mão é visível em um quadro-chave, mas a parte de trás da mão é visível no quadro-chave seguinte. Essa parte da animação pode ser vista mais detalhadamente na Figura 7.13. Nesse caso, os quadros interpolados não refletem o movimento desejado da mão. Esses artefatos normalmente acontecem em alguns poucos lugares específicos, e podem ser corrigidos manualmente pelo artista. Apesar disso parecer ser uma tarefa laboriosa, é muito mais simples do que criar todos os quadros intermediários manualmente. Na verdade, uma vez que o artista se acostuma com o sistema, ele pode até mesmo prever onde isso acontecerá e tratar prontamente.

Acreditamos que a restrição mais forte é que o método requer que todos os quadros-chaves possuam aproximadamente o mesmo número de linhas-guias. Idealmente o artista deveria ser livre para desenhar os esboços. Outro problema é possíveis ambiguidades entre as linhas-guias. Por exemplo, existem quatro formas

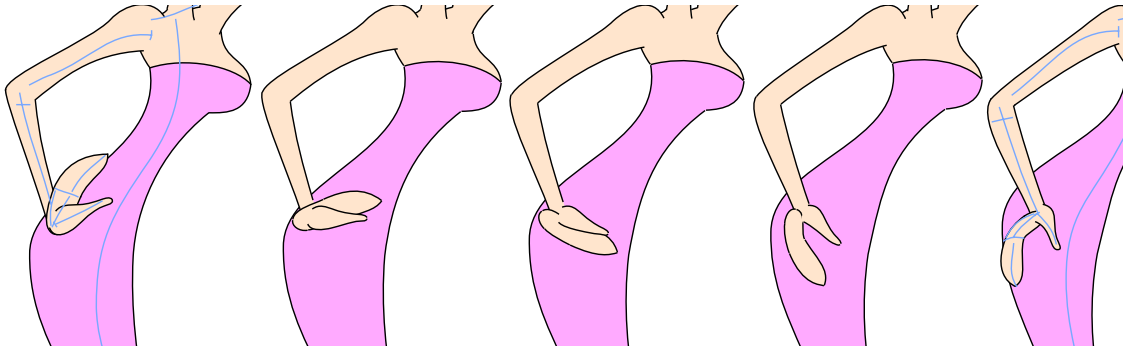


Figura 7.13: Inferência errada no movimento da mão. Como o movimento é mais complexo e não temos informação volumétrica a animação resultante não corresponde ao que o animador espera.

possíveis de correspondências entre uma cruz de um quadro para uma cruz no quadro seguinte, conforme ilustrado na Figura 7.14. Esse problema pode ser resolvido adicionando-se mais linhas-guias para forçar as correspondências desejadas. Assim, o usuário pode precisar adicionar algumas linhas-guias extras para indicar suas intenções.

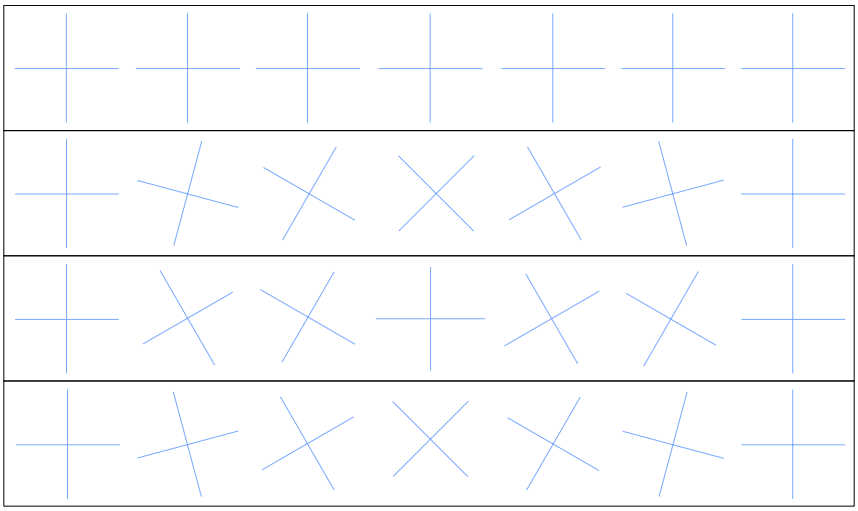


Figura 7.14: Casos de ambiguidade de correspondências. Cada linha ilustra a interpolação entre uma cruz (à esquerda) e outra (à direita).

Além disso, a orientação do movimento calculado pelo método pode não corresponder ao desejado. Por exemplo, o artista pode deseja criar um movimento de rotação no sentido horário, mas o sistema gerar a rotação no sentido anti-horário. O sistema sempre vai fornecer o caminho mais curto, então esse caso pode ser resolvido pela adição de quadros-chaves. Felizmente, essa situação se mostrou pouco frequente nos testes realizados.

Capítulo 8

Conclusões e trabalhos futuros

Neste trabalho, apresentamos um método para a criação de animações 2D que utiliza linhas-guias para obter uma informação sobre o movimento que acontece entre um par de quadros-chaves. Essa informação é útil para auxiliar o artista a desenhar quadros-chaves e para a geração dos quadros intermediários.

As linhas-guias não impõem trabalho extra para os animadores pois a criação dessas já é um passo natural em seus fluxos de trabalho. Apesar disso, nenhum trabalho conhecido até então utilizou as linhas-guias para auxiliar a produção de animações 2D. Os trabalhos anteriores processavam diretamente com a arte-final ou havia a adição de estruturas para indicar o movimento, como esqueletos. As animações produzidas com esqueletos são mais restritas, pois cada parte do esqueleto move-se de maneira rígida, produzindo resultados semelhantes a animações de recortes. Os trabalhos que processam apenas a arte-final em geral requerem que os quadros-chaves sejam similares para facilitar a busca por correspondências. A utilização de linhas-guias permite ao artista trabalhar com uma diversidade maior de animações, com figuras mais flexíveis e quadros-chaves mais distintos uns dos outros. Isso acontece porque as linhas-guias são mais simples do que a arte-final e podem incluir informações que não aparecem na arte-final mas que são úteis para a definição da transformação de um quadro para o outro, como por exemplo linhas que dividem o rosto de uma pessoa pela metade. Essas informações favorecem a busca por correspondências e a inferência do movimento entre quadros-chaves. Além disso, optamos pelo uso de camadas, o que também segue o processo de criação de animação tradicional, em vez de restringir as classes de animações possíveis.

O método foi incorporado a um protótipo de sistema que permite a criação e edição das animações. A interface do sistema exibe uma pré-visualização de um quadro-chave, o qual funciona como uma mesa de luz virtual aprimorada, que o animador pode utilizar como referência para o desenho desse quadro-chave. Conforme visto na seção 7.2, o sistema funciona de forma interativa, demorando menos de um segundo para calcular todas as transformações e gerar os quadros in-

termediários em cada exemplo. O código-fonte do sistema pode ser obtido em `https://gitlab.com/lcarvalho/sketch_project`. Além do código, nesse endereço encontram-se os exemplos mostrados no Capítulo 7 e outros testes. Cada animação é definida em um arquivo no formato Json, contendo todas as definições dos quadros-chaves e camadas. Para compilar é necessário um compilador com suporte ao padrão C++11. As únicas dependências externas são as bibliotecas Qt 5 `http://qt-project.org/` e Eigen 3 `http://eigen.tuxfamily.org/`.

Utilizando este método fomos capazes de produzir diversas animações utilizando nosso sistema mesmo em situações difíceis, como em movimentos e distorções grandes, e até tratando corretamente casos de obstruções em muitas situações. Apenas pequenos problemas são deixados para o artista tratar, como resolver casos de ambiguidade e situações em que o sistema não é capaz de resolver por falta de informação, como o que ocorreu no exemplo da Figura 7.10, discutido na seção 7.3. Isto reduz consideravelmente o tempo de produção, comparando com o tempo que seria necessário para criar toda a animação quadro a quadro.

O procedimento não pretende substituir outros métodos de inbetweening, como os descritos em [16, 19, 21], mas potencialmente ser utilizado em conjunção com eles para uma melhoria de resultados. De fato, a maioria das limitações pode ser evitada ou reduzida utilizando outros métodos em cada parte do algoritmo. Por exemplo, é possível incorporar o método descrito por XING *et al.* [26] para o processamento de linhas-guias, resultando numa transformação com menos distorção, além de reduzir o número de camadas necessárias. O nosso método funcionou bem mesmo utilizando algoritmos simples que não seriam apropriados de serem usados diretamente com a arte-final (sem utilizar a informação das linhas-guias). Portanto, acreditamos que nosso método pode reduzir fortemente o trabalho dos artistas, mantendo-se dentro do método de produção de animação tradicional.

Como trabalho futuro, vamos investigar como tratar apropriadamente o caso no qual curvas quebram em duas ou mais partes, ou o contrário, quando curvas num quadro-chave se tornam uma única curva no quadro-chave seguinte. Isso daria mais flexibilidade ao artista.

Podemos também utilizar diferentes estilos de linhas e para o desenho das curvas e de preenchimentos para as regiões, testando as maneiras possíveis de se gerar esses elementos nos quadros intermediários.

Outra possibilidade é utilizar o trabalho de NORIS *et al.* [44], no qual um desenho é segmentado utilizando apenas traços sobre algumas curvas do desenho. O método analisa esses traços para inferir quais curvas devem pertencer ao mesmo grupo. Essa ideia poderia ser incorporada em nosso trabalho para definir as camadas de modo alternativo, por exemplo.

Referências Bibliográficas

[1] WILLIAMS, R. *The Animator's Survival Kit*. Second ed. , Faber & Faber, December 2009.

[2] *Parameter values for ultra-high definition television systems for production and international programme exchange*. International Telecomunication Union, ago. 2012. Recommendation ITU-R BT.2020.

[3] CATMULL, E. "The problems of computer-assisted animation", *SIGGRAPH Comput. Graph.*, v. 12, n. 3, pp. 348–353, ago. 1978.

[4] MIURA, T., IWATA, J., TSUDA, J. "An application of hybrid curve generation: cartoon animation by electronic computers." In: *AFIPS Spring Joint Computing Conference*, v. 30, *AFIPS Conference Proceedings*, pp. 141–148. AFIPS / ACM / Thomson Book Company, Washington D.C., 1967.

[5] BURTNYK, N., WEIN, M. "Computer-Generated Key Frame Animation", *Journal of the Society of Motion Picture and Television Engineers*, v. 80, n. 3, pp. 149–153, mar. 1971.

[6] BURTNYK, N., WEIN, M. "Interactive Skeleton Techniques for Enhancing Motion Dynamics in Key Frame Animation", *Commun. ACM*, v. 19, n. 10, pp. 564–569, out. 1976.

[7] REEVES, W. T. "Inbetweening for Computer Animation Utilizing Moving Point Constraints", *SIGGRAPH Comput. Graph.*, v. 15, n. 3, pp. 263–269, ago. 1981.

[8] DURAND, C. X. "The "TOON"project: Requirements for a computerized 2D animation system", *Computers & Graphics*, v. 15, n. 2, pp. 285–293, 1991.

[9] SEDERBERG, T. W., GREENWOOD, E. "A Physically Based Approach to 2&Ndash;D Shape Blending". v. 26, pp. 25–34, New York, NY, USA, jul. 1992. ACM.

[10] PATTERSON, J.W.;WILLIS, P. "Computer-Assisted Animation: 2D or not 2D", *The Computer Journal*, v. 37, n. 10, pp. 829–839, 1995.

[11] FEKETE, J.-D., BIZOUARN, E., COURNARIE, E., et al. "TicTacToon: a paperless system for professional 2D animation". In: *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pp. 79–90, New York, NY, USA, 1995. ACM.

[12] XIE, M. "Feature matching and affine transformation for 2D cell animation." *The Visual Computer*, v. 11, n. 8, pp. 419–428, 1995.

[13] REETH, F. V. "Integrating 2 1/2-D Computer Animation Techniques for Supporting Traditional Animation". In: *Proceedings of the Computer Animation*, CA '96, pp. 118–, Washington, DC, USA, 1996. IEEE Computer Society.

[14] FIORE, F. D., SCHAEKEN, P., ELENS, K., et al. "Automatic Inbetweening in Computer Assisted Animation by Exploiting 2.5D Modelling Techniques". In: *The Fourteenth Conference on Computer Animation (CA2001)*, pp. 192–200, 2001.

[15] BREGLER, C., LOEB, L., CHUANG, E., et al. "Turning to the Masters: Motion Capturing Cartoons". In: *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pp. 399–407, New York, NY, USA, 2002. ACM.

[16] KORT, A. "Computer aided inbetweening". In: *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, NPAR '02, pp. 125–132, New York, NY, USA, 2002. ACM.

[17] GOMES, J., DARSA, L., COSTA, B., et al. *Warping and Morphing of Graphical Objects*. San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 1998.

[18] MELIKHOV, K., TIAN, F., SOON, S. H., et al. "Frame Skeleton Based Auto-Inbetweening in Computer Assisted Cel Animation." In: *CW*, pp. 216–223. IEEE Computer Society, 2004.

[19] DE JUAN, C. N., BODENHEIMER, B. "Re-using traditional animation: methods for semi-automatic segmentation and inbetweening". In: *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '06, pp. 223–232, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[20] SÝKORA, D., DINGLIANA, J., COLLINS, S. "As-rigid-as-possible Image Registration for Hand-drawn Cartoon Animations". In: *Proceedings of International Symposium on Non-photorealistic Animation and Rendering*, pp. 25–33, 2009.

[21] WHITED, B., NORIS, G., SIMMONS, M., et al. "BetweenIT: An Interactive Tool for Tight Inbetweening", *Comput. Graphics Forum (Proc. Eurographics)*, v. 29, n. 2, pp. 605–614, 2010.

[22] NORIS, G., SÝKORA, D., COROS, S., et al. "Temporal noise control for sketchy animation". In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*, NPAR '11, pp. 93–98, New York, NY, USA, 2011. ACM.

[23] YU, J., LIU, D., TAO, D., et al. "Complex Object Correspondence Construction in Two-Dimensional Animation." *IEEE Transactions on Image Processing*, v. 20, n. 11, pp. 3257–3269, 2011.

[24] SONG, Z., YU, J., ZHOU, C., et al. "Automatic cartoon matching in computer-assisted animation production", *Neurocomputing*, v. 120, n. 0, pp. 397–403, 2013. Image Feature Detection and Description.

[25] DALSTEIN, B., RONFARD, R., VAN DE PANNE, M. "Vector Graphics Animation with Time-Varying Topology", *ACM Trans. Graph.*, v. 34, n. 4, jul. 2015.

[26] XING, J., WEI, L.-Y., SHIRATORI, T., et al. "Autocomplete Hand-drawn Animations", *ACM Trans. Graph.*, v. 34, n. 6, pp. 169:1–169:11, out. 2015.

[27] SANTOS, W., ALENCAR, H. *Geometria diferencial das curvas planas*. IMPA, 2003.

[28] PIEGL, L., TILLER, W. *The NURBS Book*. Second ed. New York, NY, USA, Springer-Verlag New York, Inc., 1997.

[29] LAWSON, C. L., HANSON, R. J. *Solving Least Squares Problems*. Series in Automatic Computation. Englewood Cliffs, NJ 07632, USA, Prentice-Hall, 1974.

[30] GRAVESEN, J. "Adaptive subdivision and the length and energy of Bézier curves", *Computational Geometry*, v. 8, n. 1, pp. 13–31, 1997.

[31] ADOBE SYSTEMS INC. *PostScript Language Reference Manual*. 3rd ed. Reading, Massachusetts, Addison Wesley, January 1999.

[32] JACKSON, D., NORTHWAY, C. "Scalable Vector Graphics (SVG) Full 1.2 Specification". World Wide Web Consortium, Working Draft WD-SVG12-20050413, April 2005.

[33] WALTER, M., FOURNIER, A. "Approximate Arc Length Parametrization". 1996.

[34] WEBSTER, C. *Animation: The Mechanics of Motion*. N. v. 1, Animation: The Mechanics of Motion. Elsevier Focal Press, 2005.

[35] FRISKEN, S. F. "Efficient Curve Fitting", *J. Graphics Tools*, v. 13, n. 2, pp. 37–54, 2008.

[36] GOLUB, G., REINSCH, C. E. "Singular value decomposition and least squares solutions", *Numerische Mathematik*, v. 14, n. 5, pp. 403–420, 4 1970.

[37] LEE, S.-Y., CHWA, K.-Y., SHIN, S. Y. "Image metamorphosis using snakes and free-form deformations". In: *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pp. 439–448, New York, NY, USA, 1995. ACM.

[38] SEDERBERG, T. W., GREENWOOD, E. "Shape Blending of 2-D Piecewise Curves". In: *Mathematical Methods for Curves and Surfaces*, pp. 497–506, 1995.

[39] CONG, G., PARVIN, B. "A New Regularized Approach for Contour Morphing". In: *2000 Conference on Computer Vision and Pattern Recognition (CVPR 2000), 13-15 June 2000, Hilton Head, SC, USA*, pp. 1458–1463. IEEE Computer Society, 2000.

[40] FU, H., LAN TAI, C., CHUNG AU, O. K. "Morphing with Laplacian Coordinates and Spatial-Temporal Texture". 2005.

[41] WHITED, B., ROSSIGNAC, J. "B-morphs Between B-compatible Curves in the Plane". In: *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, SPM '09, pp. 187–198, New York, NY, USA, 2009. ACM.

[42] DIMARTINO, M. D., KONIETZKO, B., HEDRICK, T., et al. "The Legend of Korra". 2012. Nickelodeon Animation Studio, Ginormous Madman, Studio Mir, Studio Pierrot.

[43] IRIE, Y., MARUYAMA, H., YONAI, N., et al. "Fullmetal Alchemist: Brotherhood". 2009. Bones Studio.

[44] NORIS, G., SÝKORA, D., SHAMIR, A., et al. "Smart Scribbles for Sketch Segmentation", *Computer Graphics Forum*, v. 31, n. 8, pp. 2516–2527, 2012.