COPPE/UFRJ

DEFORMAÇÃO INTERATIVA DE MODELOS 3D USANDO MÍNIMOS QUADRADOS MÓVEIS

Alvaro Ernesto Cuno Parari

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.

Orientadores: Cláudio Esperança
Antônio Alberto Fernandes de Oliveira

Rio de Janeiro
Setembro de 2008

DEFORMAÇÃO INTERATIVA DE MODELOS 3D USANDO MÍNIMOS QUADRADOS MÓVEIS

Alvaro Ernesto Cuno Parari

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA 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.

Aprovada por:

Prof. Claudio Esperança, Ph.D.

Prof. Antonio Alberto Fernandes de Oliveira, D.Sc.

Prof. Paulo Roma Cavalcanti, D.Sc.

Prof. Ricardo Cordeiro de Farias, Ph.D.

Prof. Luiz Henrique de Figueiredo, D.Sc.

Prof. Manuel Menezes de Oliveira Neto, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

SETEMBRO DE 2008

Cuno Parari, Alvaro Ernesto

Deformação Interativa de Modelos 3D Usando Mínimos Quadrados Móveis/Alvaro Ernesto Cuno Parari. – Rio de Janeiro: UFRJ/COPPE, 2008.

XVII, 108 p.: il.; 29, 7cm.

Orientadores: Cláudio Esperança

Antônio Alberto Fernandes de Oliveira

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2008.

Referências Bibliográficas: p. 98 – 108.

1. Deformações de Modelos 3D. 2. Mínimos Quadrados Móveis. I. Esperança, Cláudio *et al.*. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Aos meus pais.

Agradecimentos

Agradeço ao meu orientador Cláudio Esperança por tudo o que aprendi ao trabalhar junto com ele. Em especial, por permitir sermos amigos e por nos mostrar constantemente que a "arte de programar", a "arte de ensinar" e a "arte de escrever" corretamente não são trilhas impossíveis. Ao professor Antônio Oliveira, meu co-orientador, que sempre esteve prestes a resolver inúmeras dúvidas que apareceram no decorrer desta jornada. Ao professor Paulo Roma por muitos auxílios recebidos.

Minha gratidão à banca examinadora pelas sugestões e críticas que ajudaram a melhorar este trabalho.

À comunidade do LCG pelos muitos anos de convivência. Foram "vários carnavais" e muitas vivências inesquecíveis junto com o Ricardo, o Disney, o Saulo e o André. Não posso deixar de agradecer ao Flávio, que ajudou a tirar vários erros de Português coloquial deste documento. Da mesma forma, ao Zezim, Felipe e Alberto, por terem lido vários trechos deste trabalho e ao Yalmar por aliviar os custos das impressões coloridas. Ao Leandro, meu companheiro de moradia, com quem pude encontrar aquele espaço de música especial no Rio de Janeiro (valeu "semente"!). Novamente, ao Andre e ao Yalmar por me ajudar com a papelada pós-defesa.

Ao pessoal da CoppeTex, Helano e George, pelo *template* LaTeX que facilitou enormente a elaboração e formatação deste documento. À Hila por ter me ajudado a colocar um ponto final emocionante neste trabalho, além das "perguntas difíceis" e as inúmeras aventuras. Ao Rio de Janeiro (por continuar sendo a "cidade mais maravilhosa") e a todos meus amigos brasileiros.

Um agradecimento especial ao pessoal administrativo do PESC pelos "quebra-galos" de sempre: à Lúcia, Mercedes, Cláudia, Sônia e Solange. Finalmente, agradeço à CAPES/CNPQ e ao Programa de Estudante-Convênio de Pós-Graduação (PEC-PG) pelo suporte financeiro (IEL Nacional - Brasil).

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

DEFORMAÇÃO INTERATIVA DE MODELOS 3D USANDO MÍNIMOS QUADRADOS MÓVEIS

Alvaro Ernesto Cuno Parari

Setembro/2008

Orientadores: Cláudio Esperança
Antônio Alberto Fernandes de Oliveira

Programa: Engenharia de Sistemas e Computação

Este trabalho aborda o problema de deformação interativa de modelos tridimensionais. Dito de outra forma, estuda-se como colocar, controladamente, um modelo 3D em diferentes poses. Através de uma interface, o usuário especifica um conjunto de pontos de controle, os quais podem ser colocados em qualquer posição do espaço, inclusive, sobre a superfície do modelo. Estes, então, podem ser arrastados livremente sobre a tela do computador. Simultâneo ao arraste, o modelo é deformado automaticamente aplicando-se a seus vértices transformações rígidas obtidas pela minimização de um funcional de erro. Tal erro é computado ponderando-se o deslocamento dos pontos de controle pela inversa da distância à posição do vértice sendo deformado. O custo computacional é reduzido por um método para determinação eficiente da componente rotacional das transformações. Foram feitos experimentos que mostram um desempenho superior ao de métodos baseados em matrizes ortonormais e quatérnios unitários. São apresentadas também extensões ao método de maneira a fazê-lo mais sensível à forma do modelo original, a saber, a utilização de uma métrica baseada em geodésicas e um esquema apoiado no uso de esqueletos.

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

INTERACTIVE MOVING LEAST SQUARES MESH DEFORMATION

Alvaro Ernesto Cuno Parari

September/2008

Advisors: Cláudio Esperança
Antônio Alberto Fernandes de Oliveira

Department: Systems Engineering and Computer Science

This work discusses the problem of interactive deformation of 3D models. In particular, a technique is proposed for obtaining several poses for a 3D mesh by means of an interface where a collection of control points can be displaced in space. As control points are dragged across the display, vertices of the model undergo rigid transformations obtained by minimizing an error function. The error metric weights control point displacements according to the inverse of their distance to each vertex. Computational complexity is reduced by employing an efficient method for computing the rotational component of transformations. This was demonstrated by the results of several experiments comparing the proposed method to others based on orthonormal matrices and on unit quaternions. Several extensions are also presented which aim at making the technique more sensitive to model shape. These include the use of a geodesic distance metric and a scheme where skeletons are used to control the deformation.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O problema de deformação, no contexto da presente tese, se refere à atividade de alterar de maneira controlada a forma de um modelo 3D. Técnicas de deformação de modelos 3D são de grande importância em aplicações tais como modelagem, animação, interação háptica, simulação cirúrgica, ilustração assistida por computador, entretenimento, etc. [CIJ+05].

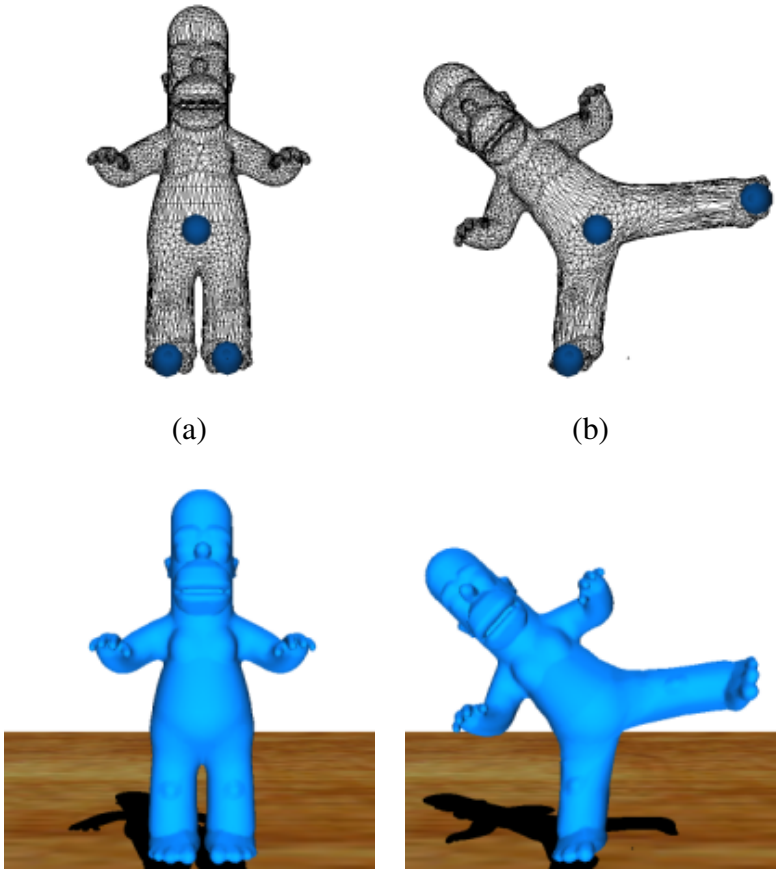
De forma geral, podemos dizer que duas grandes classes de abordagens têm sido propostas para o problema. Por um lado, os métodos baseados em física são capazes de simular a variação de propriedades tais como elasticidade e plasticidade de diferentes materiais, oferecendo um arcabouço teórico consistente para aplicações de deformação. Por outro lado, o cômputo exato de parâmetros físicos, tais como tensões e flexões, é desnecessário quando os requerimentos são resultados aproximados, visualmente plausíveis[1] e efetuados em sessões interativas, para os quais técnicas baseadas unicamente em informação geométrica costumam apresentar resultados também satisfatórios. O presente trabalho está inserido neste último contexto.

Para deformações feitas de forma interativa, o paradigma mais utilizado baseia-se na manipulação de estruturas de controle chamadas *handles* (algo como puxadores, em inglês), os quais podem ser grades regulares [SP86] ou de topologia arbitrária [MJ96], pontos de controle [SCOL+04, LSLCO05, SA07], ou outros objetos. Tradicionalmente, o usuário especifica transformações explícitas sobre os *handles*, as quais depois são propagadas para a superfície do modelo. É também comum requerer que o usuário possa ma-

[1] Neste trabalho, a palavra "plausível" se refere a um aspecto dentro do esperado intuitivamente, e na ausência de informações ou conhecimentos em contrário.

nipular os *handles* livremente, usando uma interface simplificada. Além disso, deseja-se que o arcabouço matemático permaneça transparente e que o modelo deformado mantenha suas características iniciais tanto quanto possível. Em outras palavras, deseja-se que o usuário consiga mudar a pose de um modelo utilizando um mínimo de parâmetros.

Neste trabalho, tal problema é abordado como um problema de otimização que pode ser definido como: dada a malha da superfície de um modelo 3D e um conjunto de pontos de controle, como calcular uma nova pose de tal forma que distorções no modelo sejam minimizadas e as novas posições dos pontos de controle sejam respeitadas. Veja na Figura 1.1 uma ilustração da deformação do modelo "Homer" através da manipulação de três pontos de controle.



(a) (b)

(c) (d)

Figura 1.1: Ilustração do processo de deformação. (a) Modelo inicial e pontos de controle. (b) Malha deformada após o usuário mover dois pontos de controle. Em (c) e (d) observa-se a mesma deformação em um ambiente iluminado.

Técnicas recentes abordam este problema como um problema de otimização variacional global, cuja solução é a malha deformada. Normalmente, um funcional quadrático

é minimizado resolvendo um sistema de equações lineares esparso de tamanho proporcional ao número de vértices do modelo. Tais técnicas são projetadas para trabalhar diretamente sobre malhas triangulares de alta resolução, isto é, malhas com muitos detalhes. Conforme observado por Botsch e Sorkine [BS08], embora bons resultados tenham sido alcançados, propostas alternativas que suportem deformações arbitrárias e que não compliquem a interação com o usuário merecem ainda ser pesquisadas.

Entre os métodos propostos recentemente, aqueles que evitam achatamentos e escalamentos arbitrários são de especial interesse, posto que a forma dos modelos e seus detalhes tendem a ser preservados depois das operações de edição [SA07]. Recentemente, Schaefer et al. [SMW06] usaram este paradigma para realizar deformações de imagens 2D interativamente. Nesse trabalho, a deformação é conduzida por transformações rígidas ótimas que mapeiam a imagem inicial numa imagem deformada. O problema chave a ser resolvido consistia em encontrar a componente rotacional eficientemente. Eles apresentaram uma solução fechada usando a relação entre transformações rígidas e de similaridade, obtendo resultados convincentes. Seu algoritmo usa como entrada uma imagem 2D e um conjunto de pontos de controle sobre o plano. Através do deslocamento de alguns pontos de controle, a imagem é alterada através do cálculo, para cada elemento da imagem, de uma transformação restrita a ser tão rígida quanto possível. Para tanto, é empregada a técnica de otimização por mínimos quadrados móveis que define uma solução diferente para cada elemento da imagem.

A extensão dessa técnica para 3D é simples para a componente de translação. No entanto, a solução do problema para a componente rotacional é mais complicada, já que surge a necessidade da decomposição em valores singulares ou de resolver um problema de autovalores. Na verdade, a questão pode ser vista como o problema de registro de dois conjuntos de amostras, para a qual várias soluções têm sido propostas, em particular as formulações iterativas [PHYH06] e as formulações fechadas [ELF97]. Neste trabalho descrevemos uma abordagem onde a minimização de um funcional de erro resulta em um único vetor que encapsula tanto o eixo quanto o ângulo de rotação. Este é encontrado computando a maior raiz real de uma equação quártica e resolvendo um sistema linear 3×3. O algoritmo foi implementado tanto em CPU quanto em GPU e vários experimentos foram realizados com o objetivo de avaliar tanto o aspecto visual das deformações quanto ao desempenho e precisão da técnica.

Na segunda parte deste trabalho estuda-se como adaptar esta abordagem de forma a torná-la mais sensível à forma do modelo. Além de estudar a mudança da métrica de distância utilizada, é proposto um esquema de deformação MLS guiado por esqueletos. Assim, dada uma malha e seu esqueleto associado computa-se transformações ótimas para mudar as posições das juntas. Depois, essas transformações são projetadas na malha segundo transformações interpoladas linearmente. A sensibilidade à forma do modelo é alcançada pelo uso de uma métrica de distância definida sobre os ossos do esqueleto. Apresenta-se vários exemplos onde resultados obtidos se mostram mais plausíveis que os obtidos com a proposta inicial.

1.1 Contribuições

Na primeira parte deste trabalho podem ser distinguidas as seguintes contribuições:

- É apresentado um algoritmo para a deformação de modelos 3D sob o enfoque da otimização por mínimos quadrados móveis (também conhecido como MLS, abreviatura do termo em inglês *Moving Least Squares*).

- É definida uma formulação direta e exata para computar a componente rotacional do algoritmo em 3D. Experimentos mostram um melhor desempenho frente a técnicas tradicionais.

- São descritas implementações do algoritmo tanto em hardware convencional (CPU) quanto em placas gráficas (GPU). Avaliações experimentais dessas implementações foram conduzidas de maneira a medir tanto rapidez quanto precisão relativas.

- Discute-se vantagens e deficiências do método frente a dois algoritmos de deformação introduzidos recentemente. Além disso, o algoritmo é submetido as deformações extremas propostas por Botsch e Sorkine [BS08] para efeitos de comparação.

As contribuições da segunda parte deste trabalho são:

- Uma proposta para aplicar "transformações tão rígidas quanto possíveis" sensíveis à forma do modelo.

- Uma reformulação do esquema MLS quando aplicado a esqueletos a fim de proporcionar um controle de curvatura.

- Um processo simplificado de associação entre malha e esqueleto (*rigging*) capaz de ser empregado mesmo com esqueletos grosseiros ou construídos automaticamente.

1.2 Organização

O restante desta tese é assim organizado: o Capítulo 2 versa sobre o problema de deformação *per se*, discute as abordagens relacionadas mais importantes, os desafios atuais e as diferentes formas como estão sendo abordados. O Capítulo 3 descreve o enfoque para deformação de imagens adotado por Schaefer et al. [SMW06] e apresenta uma maneira de estendê-lo para 3D. No Capítulo 4 mostra-se como efetuar deformações MLS mais sensíveis à forma do modelo. No último capítulo são apresentadas conclusões e perspectivas para a continuação deste trabalho.

Capítulo 2

Deformações geométricas

Deformações podem ser entendidas como processos de alteração da geometria de objetos em resposta a uma influência externa. Para tal tarefa, uma grande variedade de técnicas, sujeitas a um certo número de requerimentos, têm sido desenvolvidas. Os requerimentos podem ser, por exemplo, preservação de detalhes, da aparência, ou do volume, ter desempenho suficiente para tratamento interativo de massas de dados consideráveis, reproduzir efeitos próximos a um modelo físico, impedir auto-interseções, favorecer um suporte local, isto é, fazer com que a deformação numa posição do espaço seja determinada apenas em função de seus vizinhos, etc. Em deformações por manipulação direta, as instâncias deformadas de um modelo não são conhecidas a priori. O usuário obtém o resultado após a manipulação interativa das estruturas de controle.

Os métodos usados para efetuar a deformação podem ser divididos em dois grupos principais: métodos geométricos e métodos baseados em física.

Nos métodos geométricos, o objeto ou seu espaço ao redor são modificados tendo como base apenas manipulações geométricas aplicadas a estruturas de controle – pontos, arestas, simplexos, cubos – posicionadas ao redor do objeto. Já as deformações baseadas em física visam emular o modelo físico envolvido no movimento determinado pela dinâmica das interações. Assim, os modelos utilizados simulam o comportamento de objetos reais sob o efeito de forças internas e externas a eles. Entre os métodos mais destacados dessa classe temos: métodos de volumes/diferenças/elementos finitos, sistemas massa-mola, sistemas de partículas e sistemas baseados em análise modal.

Embora deformações baseadas em física sejam mais realistas, elas são também mais difíceis de serem implementadas. Normalmente, são necessárias a geração de malhas de

suporte arbitrárias e a especificação de uma série de parâmetros físicos, o que dificulta usá-las em aplicações desenvolvidas para usuários casuais. Além disso, resultados exatos não são relevantes em várias áreas de aplicação, tais como a indústria do cinema, jogos e a visualização e ilustração assistida por computador. A medida de qualidade mais importante nestes contextos é a aparência visual. Maiores detalhes e informações recentes sobre o desenvolvimento de modelos de deformação baseada em física podem ser encontrados nas referências [ZT00, NMK+05, MTPS08].

2.1 Métodos geométricos

Deformação geométrica é um tópico muito estudado pela comunidade de processamento geométrico e um grande número de técnicas já foi proposto. Modelos e técnicas de deformação baseadas na geometria podem ser classificados em procedurais e paramétricos [CIJ+05]:

- **Modelos procedurais/funcionais**. Modelos nesta categoria são definidos matematicamente e sem estarem vinculados a uma representação geométrica discreta, tal como pontos de controle. O controle da deformação é executado através da mudança de parâmetros globais (ex., raio de uma esfera) ou locais (ex., parâmetros de uma função primitiva em superfícies implícitas [Bli82]), e aplicados em objetos representados por quádricas, super-quádricas, cilindros generalizados ou objetos implícitos. Trabalhos pioneiros em deformações locais e globais de sólidos foram introduzidos por Barr [Bar84] e Blanc [Bla94]. Basicamente, são técnicas que deformam o espaço através de transformações especificadas por matrizes predefinidas, cujos componentes são funções de uma coordenada do espaço. Barr et al. [Bar84] introduziram matrizes para efetuar operações tais como: estreitamento, torção, dobra, etc.

  Um inconveniente dos métodos desta classe é a dependência da posição inicial do objeto em relação ao sistema de coordenadas escolhido. Assim, se ele estiver excessivamente inclinado em relação aos eixos coordenados, o resultado de algumas dessas operações pode evoluir de uma forma não esperada, embora esse inconveniente possa ser minorado pela aplicação prévia de um processo de alinhamento.

- **Modelos paramétricos**. Modelos paramétricos são definidos matematicamente via

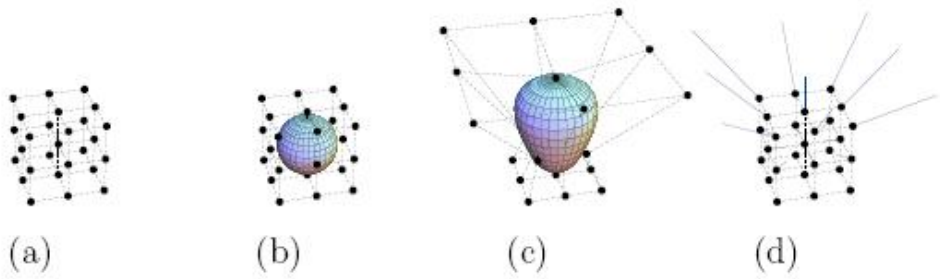


Figura 2.1: Exemplo de deformação *free-form*. (a) *Lattice* de controle de tamanho 3×3. (b) Modelo inicial. (c) Interação com um *lattice* de controle consiste no deslocamento dos pontos de controle. Isto origina a deformação do modelo inicial. (d) Campo vetorial do *lattice*.

uma representação geométrica discreta através da qual é controlada a deformação. Técnicas de deformação *free-form* (FFD: *Free Form Deformation*) [SP86, Coq90, CJ91, HHK92, MJ96] são as mais conhecidas nesta categoria. Nessa técnica, que é ilustrada na Figura 2.1, a deformação do modelo é induzida após as posições dos pontos de controle terem sido modificadas.

Outra forma de estudar os métodos de deformação geométricos é classificá-los em função da origem dos dados usados para computar a deformação e da aplicação explícita da deformação aos modelos. Com este enfoque podemos agrupá-los em: deformações baseadas na superfície, deformações do espaço, deformações multi-resolução e deformações baseadas em coordenadas diferenciais[1].

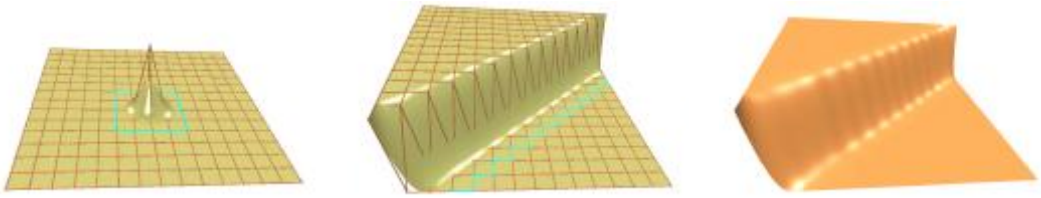
2.2 Deformação de superfícies

Técnicas de deformação desta categoria visam encontrar uma função de deslocamento $d : \mathcal{S} \rightarrow \Re^3$ que mapeia a superfície $\mathcal{S}$ na versão deformada $\mathcal{S}'$:

$$\mathcal{S}' = \{\mathbf{p} + d(\mathbf{p}) \quad | \quad \mathbf{p} \in \mathcal{S}\}.$$

Em muitas aplicações é de grande importância o controle exato do processo de deformação, isto é, a especificação precisa dos deslocamentos quando aplicados a um conjunto

[1] O material da Seção 2.2 para frente foi baseado nas notas de curso *Geometric Modeling Based on Polygonal Meshes* [BPK+08], que contêm uma excelente revisão bibliográfica de métodos recentes de deformação baseada em geometria.



(a) (b) (c)

Figura 2.2: Um exemplo de modelagem usando uma superfície *spline* de produto tensorial de duas variáveis. Cada ponto de controle está associado com uma função base de suporte retangular fixo (a). Este suporte fixo junto com o posicionamento retangular fixo dos pontos de controle impedem uma especificação do suporte preciso que pode deixar artefatos na superfície resultante (b), os quais podem ser revelados por um algoritmo de *shading* sensível (c). Figura retirada de [BPK+08].

de pontos de controle $\mathcal{C}$:

$$d(\mathbf{p}_i) = \mathbf{d}_i, \quad \forall \mathbf{p}_i \in \mathcal{C}.$$

Outro aspecto importante é a complexidade da interação do usuário requerida para especificar a função de deformação desejada.

2.2.1 Deformação de superfícies *spline*

A representação de superfície com maior uso em aplicações CAGD (*Computer Aided Geometric Design*) são as superfícies *spline*, as quais podem ser convenientemente descritas por funções base B-*spline* $N_i^n(\cdot)$. Uma superfície *spline* é definida por

$$\mathbf{f}\,(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{m} \mathbf{c}_{ij} N_i^n(u) N_j^n(v),$$

onde cada ponto de controle $\mathbf{c}_{ij}$ está associado a uma função base suave e com suporte local $N_{ij}(u,v) = N_i^n(u)N_j^n(v)$. A translação de um ponto de controle gera uma protuberância suave de suporte retangular na superfície (Figura 2.2(a)). Operações de modelagem sofisticadas têm que ser compostas empregando essas modificações elementares. Sendo assim, a função de deslocamento tem a forma

$$d\,(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{m} \delta\mathbf{c}_{ij} N_{ij}(u,v),$$

onde $\delta\mathbf{c}_{ij}$ denota a mudança do ponto de controle $\mathbf{c}_{ij}$. Desta forma, o suporte da deformação é a união do suporte das funções base individuais. Devido a dimensão e

orientação das funções base estarem vinculadas à resolução da grade inicial, é difícil um controle fino da porção de superfície que se quer alterar. Isto pode deixar artefatos na superfície resultante. A Figura 2.2 ilustra este problema.

É de conhecimento geral que superfícies *spline* obtidas via produto tensorial estão restritas a domínios retangulares e que superfícies complexas têm que ser compostas por um grande número de retalhos. Adicionalmente, a especificação de deformações complexas (em termos de movimentos de pontos de controle) requer uma interação não simples com o usuário, posto que a suavidade entre as bordas dos retalhos tem que ser considerada durante todo o processo de deformação. Nota-se também que a especificação de restrições $d(u_i, v_i) = \mathbf{d}_i$ requer resolver um sistema linear para determinar o deslocamento dos pontos de controle $\delta\mathbf{c}_{ij}$. Este sistema pode ser sobre-determinado ou sub-determinado e, tipicamente, é resolvido por técnicas de mínimos quadrados ou de norma mínima. No entanto, no primeiro caso, o sistema não pode ser resolvido de forma exata e, no segundo caso, resolver a indeterminação fazendo a minimização do deslocamento dos pontos de controle não produz necessariamente deformações suaves.

2.2.2 Deformação por propagação

Para efetuar deformações por propagação é preciso que a superfície do modelo seja dividida numa região fixa e outra região deformável. Dentro da região deformável é definida uma região manipulável (chamada também manipulador ou *handle*). Na Figura 2.3 são ilustradas estas componentes. Uma transformação aplicada diretamente ao *handle* será suavemente interpolada sobre a região deformável, combinando a contribuição da parte fixa da superfície com o efeito do deslocamento do *handle*.

Esta combinação suave é controlada por um campo escalar $s : \mathcal{S} \rightarrow [0, 1]$, tal que $s(\mathbf{p})$ é 1 na região manipulável (deformação total), 0 fora da região deformável (sem deformação) e varia entre 0 e 1 dentro da região deformável. Uma forma de construir o campo escalar consiste em computar as distâncias (euclidianas ou geodésicas): $\text{dist}_0(\mathbf{p})$ entre $\mathbf{p}$ e a parte fixa e $\text{dist}_1(\mathbf{p})$ entre $\mathbf{p}$ e a região de manipulação. Com isso, conforme proposto por [BK03, PKKG03], o campo escalar é definido como:

$$s(\mathbf{p}) = \frac{\text{dist}_0(\mathbf{p})}{\text{dist}_0(\mathbf{p}) + \text{dist}_1(\mathbf{p})}.$$

Adicionalmente, o campo escalar pode ser computado com maior sofisticação usando

Figura 2.3: Após a especificação da região de deformação azul e da região de manipulação (esquerda) constrói-se um campo escalar que é igual a 1 dentro do manipulador e 0 fora da região de deformação (meio). O campo escalar é usado para propagar a transformação sofrida pelo manipulador (direita). Figura retirada de [BPK$^+$08].

uma função de transferência $t(s(\mathbf{p}_i))$, a qual oferece maior controle no processo combinatório [BK03, PKKG03]. Desta forma, a transformação sofrida pelo manipulador pode ser propagada separadamente para cada componente da transformação (rotação R e translação T). De acordo com [PKKG03], a posição deformada **p'** de **p** é computada como sendo:

$$\begin{aligned}
\mathbf{p'} &= f(\mathbf{p}, t(s(\mathbf{p}))), \quad \text{onde} \quad t : [0,1] \rightarrow [0,1], \\
f(\mathbf{p}, t) &= f_T(\mathbf{p}, t) + f_R(\mathbf{p}, t), \\
f_T(\mathbf{p}, t) &= \mathbf{p} + t\mathbf{v}, \\
f_R(\mathbf{p}, t) &= R(\mathbf{a}, t\alpha)\ \mathbf{p}.
\end{aligned}$$

No caso em que componentes individuais da transformação não sejam dados, estes podem ser computados usando decomposição polar [SD92].

No entanto, quando regiões de manipulação côncavas são especificadas, os campos de distância podem não ser suaves. Neste caso, um campo harmônico, o qual oferece suavidade garantida, pode ser usado como alternativa [ZRKS05]. Esse campo resolve o seguinte sistema linear para os valores $s(\mathbf{p}_i)$ nos vértices $\mathbf{p}_i$ que pertencem à região deformável:

$$\begin{aligned}
\Delta s(\mathbf{p}_i) &= 0, \qquad \mathbf{p}_i \in \text{região deformável}, \\
s(\mathbf{p}_i) &= 1, \qquad \mathbf{p}_i \in \text{região de manipulação}, \\
s(\mathbf{p}_i) &= 0, \qquad \mathbf{p}_i \in \text{região fixa},
\end{aligned}$$

onde $\Delta s(\mathbf{p}_i)$ é uma representação discreta do Laplaciano de $\mathcal{S}$ em $\mathbf{p}_i$ (por exemplo, o

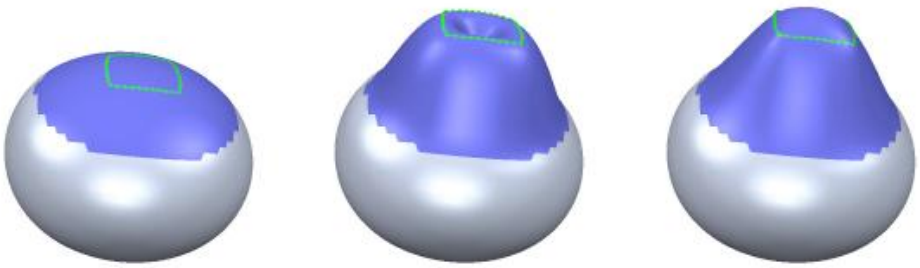


Figura 2.4: Um modelo é deformado levantando um manipulador definido por um polígono (esquerda). A propagação da translação baseada na distância geodésica causa uma cavidade no interior do polígono (meio). A solução mais intuitiva de uma interpolação suave (direita) foi obtida por um outro método baseado na interpolação de condições de borda. Figura retirada de [BPK+08].

operador de Laplace-Beltrami).

Como ilustrado na Figura 2.4, o maior problema deste método é a possibilidade da propagação baseada em distância não produzir uma solução geometricamente intuitiva.

2.2.3 Deformação por minimização de energia variacional

Funções de deformação suaves $d(\cdot)$, com condições de borda $d(\mathbf{p}_i) = \mathbf{d}_i$, podem ser modeladas elegantemente pela minimização de energías elásticas inspiradas em física [MS92, WW92, KCVS98, BK04]. Assume-se que a superfície se comporta como uma lâmina metálica que se expande e se dobra conforme forças são aplicadas nela. Matematicamente, este comportamento pode ser capturado por um funcional de energia que penaliza a quantidade de expansão e dobra na superfície.

Sejam $\mathbf{I}$ e $\mathbf{II}$ a primeira e segunda forma fundamental de $\mathcal{S}$. Quando $\mathcal{S}$ é deformada em $\mathcal{S}$', e suas formas fundamentais mudam para $\mathbf{I}$' e $\mathbf{II}$', a diferença entre as formas fundamentais pode ser usada como o funcional de energia que penaliza a expansão e dobra [TPBF87]:

$$E_{shell}(\mathcal{S}) = \int_{\Omega} k_s \|\mathbf{I}' - \mathbf{I}\|^2 + k_b \|\mathbf{II}' - \mathbf{II}\|^2 d_u d_v.$$

Os parâmetros de rigidez k_s e k_b são usados para controlar a resistência à expansão e dobra, respectivamente. Numa aplicação de modelagem, a superfície deformada ótima é aquela que minimiza o funcional de energia desde que satisfaça as condições de borda estabelecidas, que são elementos da região fixa e da região *handle*.

No entanto, a minimização de E_{shell} é computacionalmente custosa para aplicações interativas. Portanto, ela é simplificada e linearizada substituindo a diferença das formas fundamentais pelas derivadas parciais da função de deslocamento d [CG91, WW92]:

$$\tilde{E}_{shell}(d) = \int_{\Omega} k_s(\|d_u\|^2 + \|d_v\|^2) + k_b(\|d_{uu}\|^2 + 2\|d_{uv}\|^2 + \|d_{vv}\|^2)d_u d_v,$$

onde $d_x = \frac{\partial}{\partial x}d$ e $d_{xy} = \frac{\partial^2}{\partial x \partial y}d$.

Para computar a solução ótima deste problema de otimização, aplica-se cálculo variacional para derivar as equações diferenciais parciais de Euler-Lagrange que caracterizam o minimizador de $\tilde{E}_{shell}$ [Kob97]:

$$\begin{aligned} -k_s\Delta_{\mathcal{S}} + k_b\Delta_{\mathcal{S}}^2 d &= 0, \qquad \mathbf{p}_i \in \text{região deformável}, \\ d(\mathbf{p}_i) &= \mathbf{d}_i, \qquad \mathbf{p}_i \in \text{região de manipulação}, \\ d(\mathbf{p}_i) &= 0, \qquad \mathbf{p}_i \in \text{região fixa}, \end{aligned} \tag{2.1}$$

onde Δ representa o operador Laplace-Beltrami e as duas últimas equações correspondem às condições de borda. Desta forma, a função de deformação ótima d pode ser computada resolvendo um sistema bi-Laplaciano. Duas vantagens desta formulação são: permitir usar restrições arbitrárias e determinar a suavidade da solução ótima com antecedência.

Transformando interativamente a região de manipulação, alteram-se as restrições de borda do problema de otimização. Posto que isto também muda o lado direito da Equação (2.1), este sistema tem que ser resolvido eficientemente em cada quadro[2]. Note-se que, aplicando apenas transformações afins na região *handle* é possível pré-computar as funções base da deformação. Isto significa que ao invés de resolver o sistema linear em cada quadro somente é preciso avaliar as funções base [BK04]. Note-se também que modelagem baseada em condições de borda é um método flexível (como mostrado em [BK04]), já que diferentes funções de energia podem ser escolhidas para controlar comportamentos rígidos ou flexíveis, os quais podem, inclusive, ser anisotrópicos. Mais ainda, a continuidade na borda da região de deformação pode ser controlada por vértice (Figura 2.5, direita).

[2]Neste trabalho, "quadro" refere-se à tradução do termo em inglês *frame*.

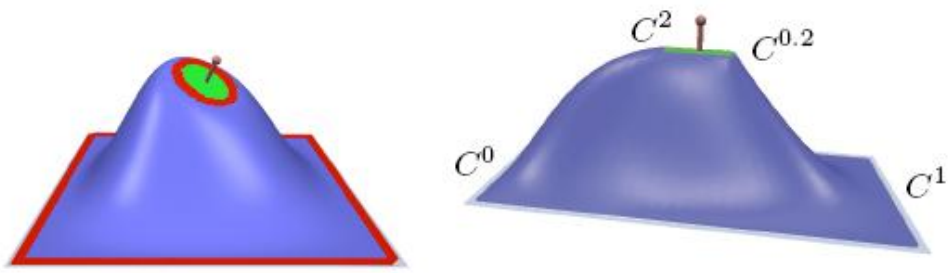


Figura 2.5: Deformação por minimização da energia permite restrições de borda de continuidade variável. Adicionalmente, permite comportamento de dobra anisotrópico e controle da continuidade das bordas por vértice.

2.3 Deformação baseada em coordenadas diferenciais

Métodos desta categoria se caracterizam por que otimizam as propriedades diferenciais da superfície ao invés das suas coordenadas espaciais. Após a especificação de posições e/ou normais em algumas partes da malha (região fixa e *handles*), a superfície deformada é reconstruída considerando as coordenadas diferenciais desejadas e minimizando a distorção do modelo. Representações diferenciais comumente usadas são: campos de gradiente e coordenadas Laplacianas.

Os métodos de [YZX+04, ZRKS05] deformam uma superfície de tal maneira que satisfaça um campo de gradiente prescrito g. Em outras palavras, o problema consiste em encontrar uma função $s : \Omega \rightarrow \Re$ que minimize

$$\int_{\Omega} \|\nabla s - g\|^2 d_u d_v.$$

Derivando as equações de Euler-Lagrange, obtém-se

$$\nabla s = \text{div } g,$$

que deve ser resolvida para encontrar uma função s ótima.

Para uma função linear por partes $s : \mathcal{S} \rightarrow \Re$ definida por seus valores $s_i = s(\mathbf{p}_i)$ nos vértices da malha, o gradiente $\nabla s : \mathcal{S} \rightarrow \Re^3$ é um vetor $\mathbf{g}_j \in \Re^3$ definido em cada triângulo f_j. Se, ao invés da função escalar s, consideramos a função $\mathbf{p}(v_i) = \mathbf{p}_i \in \Re^3$, então o gradiente numa face f_j será uma matriz 3×3:

$$\nabla \mathbf{p}|_{f_j} = \mathbf{G}_j \in \Re^{3\times 3}.$$

Logo, para uma malha com V vértices e F triângulos, o operador gradiente pode ser expressado por uma matriz **G** de tamanho $3F \times V$:

$$\begin{pmatrix} \mathbf{G}_1 \\ \vdots \\ \mathbf{G}_F \end{pmatrix} = \mathbf{G} \begin{pmatrix} \mathbf{p}_1^T \\ \vdots \\ \mathbf{p}_V^T \end{pmatrix}.$$

Técnicas com este enfoque modificam explicitamente os gradientes, produzindo novos gradientes $\mathbf{G'}_j$ por triângulo f_j. Reconstruir a malha deformada considerando os gradientes modificados é um problema sobre-determinado. Para tal utiliza-se o esquema de mínimos quadrados ponderados usando equações normais [GL89], gerando o seguinte sistema:

$$\mathbf{G}^T\mathbf{D}\mathbf{G} \begin{pmatrix} \mathbf{p'}_1^T \\ \vdots \\ \mathbf{p'}_V^T \end{pmatrix} = \mathbf{G}^T\mathbf{D} \begin{pmatrix} \mathbf{G'}_1 \\ \vdots \\ \mathbf{G'}_F \end{pmatrix},$$

onde **D** é uma matriz diagonal que contém a área da face como fator de peso. Posto que a matriz $\mathbf{G}^T\mathbf{D}$ corresponde ao operador divergente discreto, e posto que $\text{div}\nabla = \Delta$, este sistema representa a equação de Poisson:

$$\Delta \begin{pmatrix} \mathbf{p'}_1^T \\ \vdots \\ \mathbf{p'}_V^T \end{pmatrix} = \text{div} \begin{pmatrix} \mathbf{G'}_1 \\ \vdots \\ \mathbf{G'}_F \end{pmatrix}.$$

Em resumo, dado um campo de gradientes guia $(\mathbf{G'}_1, \ldots \mathbf{G'}_F)$, métodos deste grupo computam seu campo divergente e resolvem três sistemas de Poisson para as coordenadas x, y, e z dos vértices da malha $\mathbf{p}_i$'.

A maior dificuldade nesta categoria de propostas é projetar uma técnica que modifique os gradientes $\mathbf{G}_j$. Uma alternativa proposta em [YZX$^+$04, ZRKS05, SP04] consiste em usar os gradientes das transformações afins para transformar os gradientes da superfície $\mathbf{G}_j$. Como conseqüência, estes métodos trabalham bem com rotações, mas são insensíveis frente às translações. Ou seja, efetuar uma translação no manipulador do modelo não modifica o gradiente do mesmo, portanto, não tem influência nos gradientes da superfície resultante. Além disso, como há uma conexão (não-linear) entre translações e rotações locais de gradientes, estes métodos produzem resultados não intuitivos para modificações que contêm translações amplas. Embora Yu et al. [YZX$^+$04] tenham proposto um trata-

mento especial para translações puras, deformações contendo rotações e translações ainda são problemáticas.

Por outro lado, existem métodos que manipulam os Laplacianos dos vértices ao invés dos campos de gradiente. A idéia é computar as coordenadas Laplacianas iniciais $\delta_i = \Delta_{\mathcal{S}}(\mathbf{p}_i)$ e modificá-las para $\delta_i\text{'} = \mathbf{T}_i\delta_i$. O objetivo é encontrar uma função $\mathbf{p}$' que minimize

$$\int_{\Omega} \|\Delta_{\mathcal{S}}\mathbf{p}\text{'} - \delta\text{'}\|^2 d_u d_v.$$

Derivando as equações de Euler-Lagrange, obtém-se

$$\Delta_{\mathcal{S}}^2\mathbf{p}\text{'} = \Delta_{\mathcal{S}}\delta\text{'}.$$

Para o caso discreto, a equação anterior implica resolver o sistema bi-Laplaciano

$$\Delta_{\mathcal{S}}^2 \begin{pmatrix} \mathbf{p}\text{'}_1^T \\ \vdots \\ \mathbf{p}\text{'}_V^T \end{pmatrix} = \Delta_{\mathcal{S}} \begin{pmatrix} \delta_1^T \\ \vdots \\ \delta_V^T \end{pmatrix},$$

onde

$$\Delta(\mathbf{p}_i) = \sum_{j \in N(i)} w_{ij}(\mathbf{p}_j - \mathbf{p}_i),$$

e $N(i)$ são os vértices da vizinhança 1-anel de $\mathbf{p}_i$, w_{ij} é o peso da aresta (i, j) e $\mathbf{T}_i$ é uma matriz de transformação 3×3. Dois esquemas de ponderação são comumente usados:

- Ponderação uniforme: $w_{ij} = 1/N(i)$,
- Ponderação por cotangentes: $w_{ij} = \frac{1}{|\Omega_i|}(\cot\alpha_i + \cot\beta_i)$, onde $|\ \Omega_i\ |$ é a área da célula de Voronoi associada com $\mathbf{p}_i$ e α_i, β_i são os dois ângulos opostos à aresta (i, j).

Para evitar que as coordenadas Laplacianas sejam desviadas da direção das normais, requer-se que $\mathbf{T}_i$ seja uma combinação de rotação e escala uniforme. Dependendo da definição de $\mathbf{T}_i$, métodos desta categoria podem ser classificados como explícitos [YZX$^+$04, ZRKS05, ZHS$^+$05] ou implícitos [LSCO$^+$04, SCOL$^+$04].

Métodos explícitos definem $\mathbf{T}_i$ sem considerar a superfície deformada (desconhecida). Neste caso, $\mathbf{T}_i$ é definida na propagação das transformações dos manipuladores em toda a região deformável usando uma ponderação baseada na métrica geodésica [YZX$^+$04, ZHS$^+$05] ou num conjunto de campos harmônicos [ZRKS05]. Estes métodos podem

Figura 2.6: Métodos baseados em coordenadas diferenciais apresentam problemas de distorção frente a translações. As figuras da esquerda [ZRKS05] e do meio [LSCO$^+$04] ilustram este problema. Na direita, o resultado da proposta do Fu et al. [FAT06] introduzida para lidar com o problema. Eles aplicam uma transformação de similaridade nas coordenadas Laplacianas do modelo. Tal transformação é computada extraindo a parte rígida e a escala uniforme – com o uso de decomposição polar – de uma transformação afim. Figura retirada de [FAT06].

produzir boas deformações quando os *handles* sofrem grandes rotações, incluindo ângulos de rotação maiores que 2π. No entanto, se os manipuladores são apenas transladados, não há campos de orientação a serem propagados, portanto estes métodos não podem evitar distorções causadas pela translação do manipulador. A Figura 2.6 ilustra este problema.

Métodos implícitos definem $\mathbf{T}_i$ com respeito à superfície deformada. Este é um problema de dependência cíclica. Por um lado, a superfície deformada deve ser reconstruída usando as coordenadas Laplacianas transformadas e, por outro lado, as transformações são dependentes da superfície deformada. Lipman et al. [LSCO$^+$04] propuseram um método heurístico: primeiro reconstruíram uma superfície grosseira usando as coordenadas Laplacianas originais e, depois, usaram a superfície reconstruída para estimar uma rotação local para cada vértice. Sorkine et al. [SCOL$^+$04] representam $\mathbf{T}_i$ aproximadamente como uma função de posições de vértices desconhecidos. Métodos implícitos propostos trabalham bem para pequenas translações e/ou pequenos ângulos de rotação, porém não produzem resultados visualmente plausíveis quando os *handles* sofrem rotações amplas ou translações de grande escala.

2.4 Deformação do espaço

Os métodos de deformação descritos anteriormente computam um campo de deformação suave definido sobre a superfície $\mathcal{S}$. Se a representação da superfície é uma malha de triângulos, computar o campo de deformação tipicamente requer resolver um sistema linear em $\mathcal{S}$. Uma desvantagem óbvia que estes métodos apresentam se deve a que o esforço computacional e a robustez numérica estão fortemente relacionados à complexidade e à qualidade dos polígonos que representam $\mathcal{S}$.

Na presença de triângulos degenerados, o operador Laplaciano discreto que usa ponderação por cotangentes não está bem definido, portanto o sistema linear pode ser singular. Da mesma forma, problemas topológicos, como buracos e configurações não-variedade, também geram sérios problemas. Nesses casos, para poder computar deformações suaves, alguns esforços têm que ser feitos a fim de eliminar triângulos degenerados ou re-poligonizar a superfície. Mesmo que a qualidade da malha seja suficientemente alta, malhas extremamente complexas podem resultar em sistemas lineares que não podem ser resolvidos rapidamente devido ao seu tamanho.

Os problemas anteriores podem ser aliviados usando técnicas de deformação do espaço, que deformam o espaço 3D e, através dele, deformam indiretamente o objeto contido. Em contraste com os métodos baseados na superfície, estes usam uma função de deformação de três variáveis $d : \Re^3 \rightarrow \Re^3$ para transformar todos os pontos da superfície original $\mathcal{S}$ na superfície modificada: $\mathcal{S}$' $= \{\mathbf{p} + d(\mathbf{p}) \quad | \quad \mathbf{p} \in \mathcal{S}\}$. Desta vez, satisfazer as restrições $d(\mathbf{p}_i) = \mathbf{d}_i$ equivale a encontrar uma função $d : \Re^3 \rightarrow \Re^3$ que interpole as restrições, ao invés da interpolação sobre a variedade $\mathcal{S}$ como acontecia nos métodos baseados na superfície. Semelhante a estes, pode-se observar que métodos baseados na minimização de energia global tipicamente oferecem resultados de alta qualidade.

Posto que a função de deformação do espaço d não depende de uma representação de superfície particular, este esquema pode ser aplicado tanto em representações de superfície poligonais quanto em outras representações (por exemplo, nuvem de pontos). No entanto, não é trivial realizar deformações anisotrópicas ou garantir continuidade específica nas bordas como nos métodos baseados em superfície.

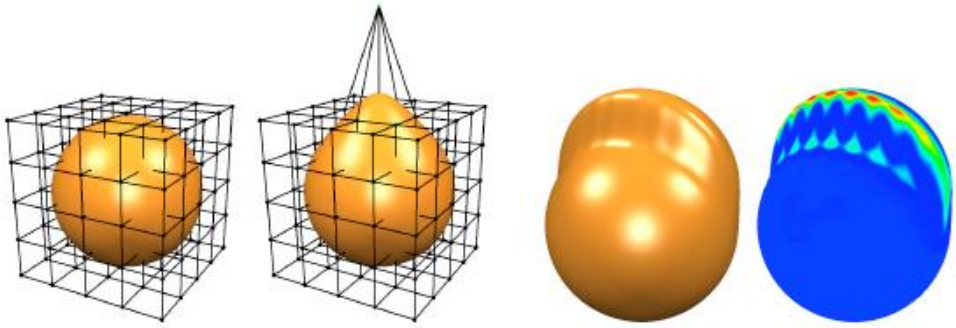


Figura 2.7: No método de deformação *free-form*, um *lattice* de controle 3D é usado para especificar uma função de deslocamento volumétrico (esquerda). Semelhante às superfícies *spline*, as *splines* de três variáveis podem produzir artefatos na superfície deformada (direita). Figura retirada de [BS08].

2.4.1 Deformações de formas livres (*free-form*)

O método clássico de deformação de formas livres (FFD, *Free-Form Deformation*) [SP86] representa o espaço de deformação por uma função *spline* ou Bézier

$$d(u,v,w) = \sum_i \sum_j \sum_k \delta \mathbf{c}_{ijk} N_i^l(u) N_j^n(v) N_k^m(w).$$

Devido aos mesmos motivos discutidos na Seção 2.2.1, estes métodos requerem interações complicadas com o usuário e podem causar problemas de cisalhamento como mostrado na Figura 2.7 (direita).

Para satisfazer as restrições estabelecidas, o método FFD inverso [HHK92] resolve um sistema linear para o movimento dos pontos da grade $\mathbf{c}_{ijk}$. Este sistema pode ser sobre-determinado ou sub-determinado e, portanto, é resolvido por métodos como mínimos quadrados ou norma mínima, respectivamente. No entanto, o primeiro não pode interpolar exatamente as restrições e o último minimiza o movimento dos pontos de controle, o que não necessariamente implica uma deformação com energia de curvatura baixa razoável.

2.4.2 Deformação baseada em interpolação usando RBFs

Funções de base radial (RBFs) são conhecidas por serem bem apropriadas para problemas de interpolação, especialmente para dados que não seguem nenhuma estrutura regular [Buh03]. Uma função de deformação RBF é definida em termos de centros $\mathbf{c}_j \in \Re^3$ e

pesos $\mathbf{w}_j \in \Re^3$ como

$$\mathbf{d}(\mathbf{x}) = \sum_j \mathbf{w}_j \phi(\|\mathbf{c}_j - \mathbf{x}\|) + \mathbf{P}(\mathbf{x}), \tag{2.2}$$

onde $\phi(\|\mathbf{c}_j - \cdot\|)$ é a função base correspondente ao j-ésimo centro $\mathbf{c}_j$ e $\mathbf{P}(\mathbf{x})$ é uma função polinomial de grau baixo.

A escolha de ϕ tem uma forte influência na complexidade computacional e na aparência da deformação. Enquanto funções de suporte compacto geram sistemas lineares esparsos (podendo interpolar vários milhares de pontos [MYC+01, OBS04]), eles não oferecem o mesmo grau de suavidade oferecido pelas funções base de suporte global [CBC+01]. Foi demonstrado por Duchon [Duc77] que, para a base $\phi(r) = r^3$ e o polinômio quadrático $P(\cdot) \in \Pi_2$, a função (2.2) é tri-harmônica ($\Delta^3 \mathbf{d} = 0$), logo, minimiza o funcional de energia

$$\int_{\Re^3} \|\mathbf{d}_{xxx}(\mathbf{x})\|^2 + \|\mathbf{d}_{xxy}(\mathbf{x})\|^2 + \cdots + \|\mathbf{d}_{zzz}(\mathbf{x})\|^2 d\mathbf{x}.$$

Deve-se notar que estas funções são conceitualmente equivalentes às superfícies de variação mínima de [MS92] e às superfícies tria-harmônicas usadas em [BK04], portanto oferecem o mesmo grau de suavidade.

Para construir uma RBF que interpole as restrições $\mathbf{d}(\mathbf{p}_i) = \mathbf{d}_i$, os centros são posicionados nas restrições, isto é, $\mathbf{c}_i = \mathbf{d}_i$. Em seguida, resolve-se o sistema linear para encontrar os pesos $\mathbf{w}_i$ e os coeficientes do polinômio. Devido ao suporte global da função base tri-harmônica $\phi(r) = r^3$, este sistema linear é denso, o que implica complexidade cúbica para métodos de resolução de equações tradicionais. Maiores detalhes sobre deformações usando RBFs podem ser encontrados em [BK05a].

2.5 Deformações multi-resolução

Técnicas de deformação multi-resolução aparecem no cenário pois muitas das técnicas já apresentadas não conseguem preservar os detalhes das superfícies frente a algumas especificações da deformação (veja, por exemplo, a Figura 2.6). Estas técnicas realizam uma decomposição das freqüências do modelo para poder realizar deformações globais com preservação de detalhes intuitiva.

Primeiramente, dada uma superfície $\mathcal{S}$, calcula-se uma representação de baixa freqüência $\mathcal{B}$. Os detalhes geométricos da superfície, $\mathcal{D} = \mathcal{S} \ominus \mathcal{B}$, que foram removidos,

representam as altas freqüências de $\mathcal{S}$ e são armazenados como vetores de deslocamento paralelos ao campo de normais de $\mathcal{B}$ [KVS99]. Desta forma, é possível reconstruir a superfície original $\mathcal{S}$ agregando os detalhes geométricos sobre a superfície base $\mathcal{S} = \mathcal{B} \oplus \mathcal{D}$. Os operadores especiais $\ominus$ e $\oplus$ são chamados operadores de decomposição e reconstrução, respectivamente. Logo, um operador de "adição" é usado para deformar a superfície base $\mathcal{B}$ numa versão modificada $\mathcal{B}'$. Finalmente, a adição dos detalhes geométricos na superfície base produz a deformação multi-resolução $\mathcal{S}' = \mathcal{B}' \oplus \mathcal{D}$.

Uma inerente limitação deste método está na necessidade da diferença entre $\mathcal{S}'$ e $\mathcal{B}'$ ser suficientemente pequena, tal que $\mathcal{S}$ possa ser representado como um campo de altura sobre $\mathcal{B}$. Se este critério não puder ser alcançado por uma hierarquia simples de dois níveis, mais níveis serão necessários $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_k = \mathcal{B}$.

Ademais, posto que vetores de deslocamento vizinhos não são acoplados, a superfície de deslocamento resultante pode gerar auto-interseções locais quando a curvatura da superfície base deformada $\mathcal{B}'$ é muito alta em relação à longitude de deslocamento. Volumes de deslocamento [BK05b] evitam estas auto-interseções locais representando os detalhes geométricos $\mathcal{D}'$ por elementos de prisma de volume constante, que requerem uma reconstrução não-linear custosa.

2.6 Preservação de volume

Entre estes métodos, pode-se citar aqueles que procuram uma construção explícita capaz de representar o interior de um modelo [RSB95, AB97, HML99, ZHS+05]. Subseqüentemente, técnicas baseadas em física são aplicadas para conduzir a deformação. Os principais problemas com estes métodos são a construção complicada das estruturas de suporte, a instabilidade dos sistemas e o baixo desempenho da computação da deformação. Além disso, estas propostas podem ser aplicadas apenas em representações específicas e/ou em modelos cujos volumes podem ser computados.

Lipman et al. [LCOGL06] introduziram um método onde a preservação do volume acontece implicitamente devido a um esquema de representação da deformação baseado nas formas fundamentais diferenciais. Botsch and Kobelt [MB03] introduziram um método para a preservação volumétrica baseado num paradigma multi-resolução. Eles usaram elementos de volume (prismas) entre a superfície e a sua formulação base.

Foi empregada relaxação hierárquica para resolver o sistema não-linear que corrige as posições dos vértices para otimizar o volume local. Uma extensão deste método, que estuda como preservar detalhes finos da superfície, foi apresentado por Botsch et al. [BPGK06]. A comparação deste método com alguns outros foi efetuada recentemente por Botsch e Sorkine [BS08] e um resumo dos resultados é mostrado na Figura 2.8.

Angelidis et al. [ACWK06] introduziram uma ferramenta de edição que preserva volume. A proposta é baseada num operador chamado *swirling-sweepers*, que são combinações de matrizes elevadas a potências de funções escalares. O operador é aplicado incrementalmente ao longo de um caminho, onde, em cada passo, o *swirl* deforma o espaço ao redor enquanto preserva o volume. Pode ser classificada como uma técnica de deformação do espaço, portanto é aplicável a todas as representações de superfície.

Recentemente, Funk et al. [vFTS06] apresentaram um método baseado na construção e modificação interativa de um campo vetorial contínuo de classes C^1. As posições dos vértices deformados são obtidas pela integração de um caminho no campo vetorial. Desta forma, propriedades simples do campo vetorial produzem propriedades úteis na deformação: um campo vetorial de divergente zero gera uma deformação que preserva volume e evita auto-interseções, ao mesmo tempo que características afiadas são preservadas e pequenos detalhes são deformados de forma realista.

Por último, entre as técnicas que não optimizam uma medida de erro ou alguma propriedade dos modelos, mas que produzem bons resultados estão as propostas de Blanco [BO08] e Forstamann [FO06, FOKGM07]. Eles apresentam técnicas de deformação baseadas na manipulação de curvas paramétricas. Embora as duas propostas sejam diferentes, possuem funcionalidade semelhante. As coordenadas dos vértices do modelo são definidas em termos de sistemas locais (*frames*) sobre as curvas. Quando o usuário modifica a curva, os *frames* são recomputados, o que a sua vez origina a modificação dos vértices do modelo.

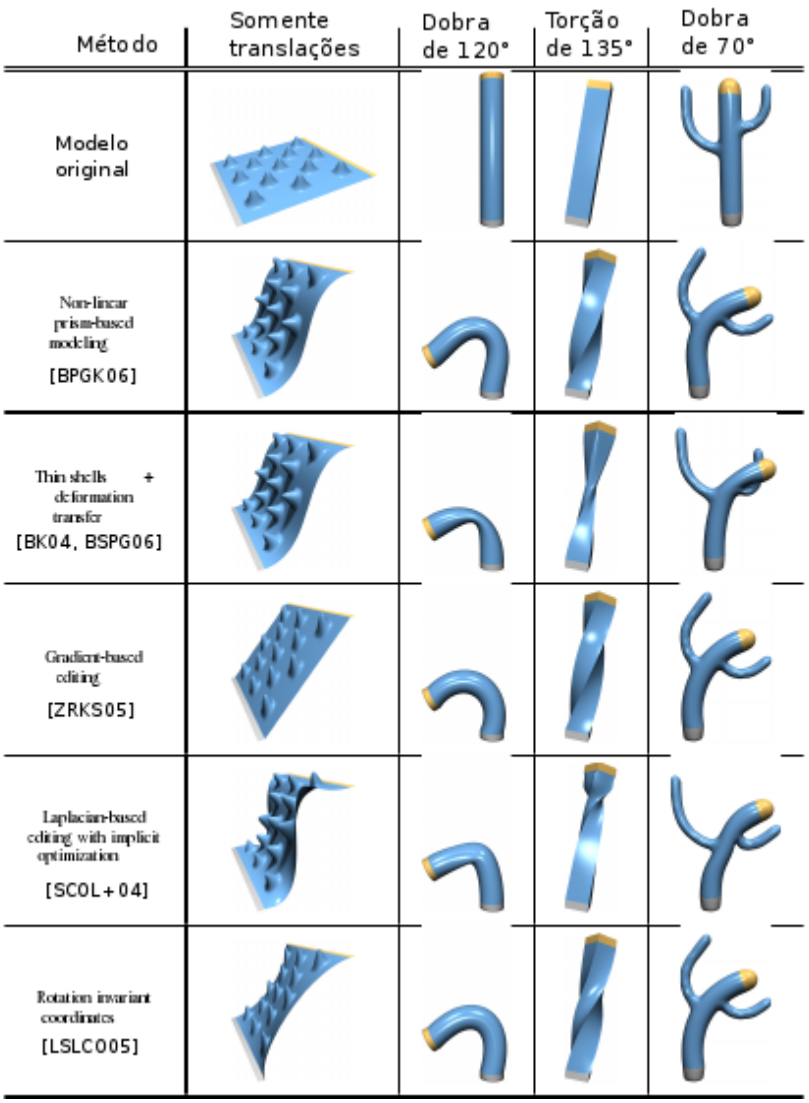


Figura 2.8: Comparação de várias técnicas de deformação frente à translação, dobra e torção de pontos de controle. Todos os métodos, exceto [BPGK06], apresentam algum problema em frente dessas configurações dos pontos de controle. Figura retirada de [BS08].

Capítulo 3

Deformações MLS em 3D

Várias técnicas de deformação de imagens (*image warping*) que utilizam métodos de interpolação para computar a função de deformação têm sido propostas [Boo89, BN92, RM93, RM95]. Tais técnicas usam um conjunto de manipuladores (normalmente um conjunto de pontos) para controlar a deformação. Após a modificação da posição e/ou orientação dos manipuladores, a imagem é deformada através da aplicação de uma função $f : \Omega \subseteq \Re^2 \rightarrow \Re^2$.

Considere uma imagem 2D e um conjunto de manipuladores $\{\mathrm{p}_0, \ldots, \mathrm{p}_n\}, \mathrm{p}_i \in \Re^2$, que o usuário move para novas posições $\{\mathrm{q}_0, \ldots, \mathrm{q}_n\}$. A deformação pode ser vista como uma função f que mapeia pontos v da imagem original para pontos na imagem deformada. Normalmente, f é construída pela interpolação separada das componentes $(\mathrm{q}_{ix}, \mathrm{q}_{iy})$ nas posições p_i. Isto é, $f = (f_x, f_y)$ com $f_x : \Re^2 \rightarrow \Re$ e $f_y : \Re^2 \rightarrow \Re$. É desejável, também, que f satisfaça as seguintes propriedades [SMW06]:

- **Interpolação**: o conjunto de manipuladores $\{\mathrm{p}_i\}$ deve ser mapeado diretamente para $\{\mathrm{q}_i\}$ sob a ação de uma deformação. Isto é, $f(\mathrm{p}_i) = \mathrm{q}_i$.

- **Suavidade**: f deve produzir deformações suaves. Ou seja, a imagem deformada não deve conter quebras.

- **Identidade**: Se os manipuladores deformados $\{\mathrm{q}_i\}$ são iguais aos $\{\mathrm{p}_i\}$, então f é a função identidade. Isto é, $\mathrm{q}_i = \mathrm{p}_i \Rightarrow f(\mathrm{v}) = \mathrm{v}$.

Como estas propriedades são semelhantes às da interpolação, métodos de deformação podem ser interpretados como um problema de interpolação de dados.

Como observado por Schaefer et al. [SMW06], muitos métodos de deformação não se preocupam com o tipo de transformação aplicado aos pontos da imagem, o que é importante, por exemplo, quando se quer que uma medida de rigidez na imagem seja mantida. Recentemente, Schaefer et al. introduziram um método para deformação de imagens restrito a transformações afins, de similaridade ou rígidas, usando um esquema de otimização baseado em uma técnica de interpolação via mínimos quadrados móveis (MLS: *Moving Least Squares* [LS81, Lev98]). Foram apresentadas, para o caso bidimensional, fórmulas fechadas para computar a deformação em cada ponto do domínio, a qual podia ser especificada através de pontos ou segmentos de controle.

Este capítulo trata especificamente da extensão para o caso 3D da metodologia de deformação apresentada por Schaefer et al. Na próxima seção são descritos trabalhos relacionados e a técnica de deformação de imagens 2D introduzida por Schaefer et al. A Seção 3.2 aborda a extensão dessa técnica para o caso 3D. O algoritmo proposto e discussões dos resultados são apresentados nas Seções 3.3 e 3.4. Em seguida, as Seções 3.5 e 3.6 discutem o uso de transformações de similaridade e afins na formulação em 3D. Finalmente, resultados de implementações em arquiteturas paralelas são apresentados na Seção 3.7.

3.1 Trabalhos relacionados

O problema da interpolação de dados multi-dimensionais consiste em estimar valores de uma função em posições arbitrárias a partir de um conjunto de dados com valores e posições conhecidas. Dado um conjunto de atributos escalares v_i e suas correspondentes posições x_i, onde $\mathrm{x}_i \in \Re^d$ para $i = 1 \ldots n$, pede-se construir uma função interpolante $f : \Omega \subseteq \Re^d \rightarrow \Re$ tal que $f(\mathrm{x}_i) = v_i$.

Entre os métodos de interpolação de dados multi-dimensionais não estruturados relacionados com o problema de deformação podem-se citar:

- Soluções baseadas em RBFs (*Radial Basis Functions* [Buh03]): A solução do problema consiste na construção de uma função formada pela combinação linear de translações de alguma função de base $\Theta : \Re^+ \rightarrow \Re$. Normalmente, a solução tem a forma

$$f(\mathrm{x}) = \sum_i \lambda_i \Theta(\|\mathrm{x}_i - \mathrm{x}\|),$$

onde $\lambda_i \in \Re$, $\mathrm{x}_i \in \Re^d$ são os centros de interpolação e $\|\cdot\|$ representa a norma euclidiana. Basicamente, tenta-se minimizar o funcional de erro $\|f(\mathrm{x}_i) - v_i\|^2 + \gamma\|f\|^2$, onde $\|f\|$ é um regularizador – definido empregando integrais de derivadas de f de alguma ordem – que toma valores altos para funções não suaves. Embora seja necessário resolver um sistema linear de tamanho $n \times n$, esta técnica ainda é vantajosa porque é multi-dimensional e não requer nenhuma organização especial dos dados de entrada ou informação acerca de sua conectividade.

- Soluções baseadas no método de Shepard [She68]: Este método se baseia na construção de combinações convexas das coordenadas de $\{\mathrm{q}_i\}$ e não requer a solução de nenhum sistema linear. Problemas de estabilidade podem ocorrer se o método for usado para interpolar grandes massas de dados. Soluções deste método têm a forma

$$f_{i\{x,y\}}(\mathrm{x}) = \frac{\sum_i \mathrm{q}_{i\{x,y\}}\Theta(||\mathrm{x} - \mathrm{x}_i||)}{\sum_i \Theta(||\mathrm{x} - \mathrm{x}_i||)},$$

  onde $\Theta(d) = d^{-\mu}$. A suavidade é determinada pelo expoente μ, sendo que valores $\mu > 1$ asseguram a continuidade da primeira derivada. No entanto, o método de Shepard mostra um comportamento desfavorável quando aplicado ao problema de deformação: o uso da média ponderada dos pontos de controle faz com que os pontos da imagem tendam a se posicionar no centro geométrico dos pontos de controle deslocados. Isto pode ocasionar distorções desagradáveis inclusive quando $\mathrm{q}_i = \mathrm{p}_i$.

- Soluções baseadas em WLS: A formulação de mínimos quadrados ponderados (WLS: *Weighted Least Squares*) consiste em substituir os valores $\{\mathrm{q}_i\}$, na formulação de Shepard, por um interpolante local $f_i(\mathrm{x}) : \Re^2 \rightarrow \Re$,

$$f_{\overline{\mathrm{x}}}(\mathrm{x}) = \frac{\sum_i f_i(\mathrm{x})\Theta(||\overline{\mathrm{x}} - \mathrm{x}_i||)}{\sum_i \Theta(||\overline{\mathrm{x}} - \mathrm{x}_i||)}.$$

  Dado um $\overline{\mathrm{x}} \in \Re^2$, constrói-se a função $f_{\overline{\mathrm{x}}}$ em $\overline{\mathrm{x}}$ minimizando o funcional de erro $\Theta(||\overline{\mathrm{x}} - \mathrm{x}_i||)||f(\mathrm{x}_i) - v_i||^2$.

- Soluções baseadas em MLS [LS81, Lev98]: A idéia deste método é resolver para cada x um problema de mínimos quadrados ponderados (WLS). Pode ser visto como uma generalização do método de Shepard (onde $m = 0$) para polinômios de grau $m > 0$. Neste método, diferente dos métodos baseados em RBFs, não é necessário montar um sistema de equações de tamanho $n \times n$. Foi demonstrado

por Levin [Lev98] que as soluções são continuamente diferenciáveis se a função de peso Θ é continuamente diferenciável.

3.1.1 Deformação MLS em 2D

Problema 1: Dados os conjuntos de pontos $\{\mathrm{p}_1, \ldots, \mathrm{p}_n\}$ e $\{\mathrm{q}_1, \ldots, \mathrm{q}_n\}$ em $\Re^2$, e um ponto v na imagem encontrar a melhor transformação afim $l_{\mathrm{v}}(\mathrm{x})$ que minimize

$$\sum_i w_i \|\mathrm{q}_i - l_{\mathrm{v}}(\mathrm{p}_i)\|^2, \tag{3.1}$$

onde $\mathrm{p}_i = [\mathrm{p}_{ix}\ \mathrm{p}_{iy}]$, $\mathrm{q}_i = [\mathrm{q}_{ix}\ \mathrm{q}_{iy}]$ e w_i é um valor positivo correspondente a uma função de peso da forma:

$$w_i = \frac{1}{\|\mathrm{p}_i - \mathrm{v}\|^2}.$$

Como os pesos w_i dependem do ponto de avaliação v, este problema de otimização é conhecido como um problema de minimização de mínimos quadrados móveis. Portanto, diferentes transformações serão obtidas para cada v.

Posto que l_{v} é uma transformação afim, esta deve ser composta de uma matriz de transformação linear L e uma componente de translação T,

$$l_{\mathrm{v}}(\mathrm{x}) = \mathrm{xL} + \mathrm{T}. \tag{3.2}$$

Resolvendo a Equação (3.1) para T em termos de L, tem-se que

$$\mathrm{T} = \mathrm{q}^* - \mathrm{p}^*\mathrm{L},$$

onde p* e q* são os centróides

$$\mathrm{p}^* = \frac{\sum_i w_i \mathrm{p}_i}{\sum_i w_i} \quad , \quad \mathrm{q}^* = \frac{\sum_i w_i \mathrm{q}_i}{\sum_i w_i}.$$

Substituindo T na Equação (3.2), tem-se

$$l_{\mathrm{v}}(\mathrm{x}) = (\mathrm{x} - \mathrm{p}^*)\mathrm{L} + \mathrm{q}^*.$$

Portanto, o problema da Equação (3.1) pode ser escrito como a minimização de

$$\sum_i w_i \|\hat{\mathrm{q}}_i - \hat{\mathrm{p}}_i\mathrm{L}\|^2, \tag{3.3}$$

$$\text{onde,} \quad \hat{\mathrm{p}}_i = \mathrm{p}_i - \mathrm{p}^* \quad \text{e} \quad \hat{\mathrm{q}}_i = \mathrm{q}_i - \mathrm{q}^*.$$

Para este problema de otimização, considerando L como uma matriz que representa diferentes classes de transformações, Schaefer et al. [SMW06] apresentaram as seguintes soluções analíticas para deformações baseadas em transformações afins f_a, de similaridade f_s e rígidas f_r.

- Deformações afins:

$$f_a(\mathrm{v}) = (\mathrm{v} - \mathrm{p}^*)(\sum_i \hat{\mathrm{p}}_i^{\mathsf{T}} w_i \, \hat{\mathrm{p}}_i)^{-1} \sum_j \hat{\mathrm{p}}_j^{\mathsf{T}} w_j \, \hat{\mathrm{q}}_j + \mathrm{q}^*. \tag{3.4}$$

  Ou a expressão que permite pré-computar algumas operações aritméticas

$$f_a(\mathrm{v}) = \sum_j A_j \hat{\mathrm{q}}_j + \mathrm{q}^*,$$

$$\text{onde} \quad A_j = (\mathrm{v} - \mathrm{p}^*)(\sum_i \hat{\mathrm{p}}_i^{\mathsf{T}} w_i \, \hat{\mathrm{p}}_i)^{-1} \sum_j \hat{\mathrm{p}}_i^{\mathsf{T}} w_j.$$

  Observe que esta pre-computação é possível devido ao fato do conjunto $\{\mathrm{p}_i\}$ ser constante durante uma sessão de deformação.

- Deformações de similaridade ou semelhança:

$$f_s(\mathrm{v}) = (\mathrm{v} - \mathrm{p}^*)(\frac{1}{\mu_s}) \sum_j \hat{\mathrm{p}}_j^{\mathsf{T}} w_j \, \hat{\mathrm{q}}_j + \mathrm{q}^*, \tag{3.5}$$

$$\text{onde} \quad \mu_s = \sum_i w_i \, \hat{\mathrm{p}}_i \hat{\mathrm{p}}_i^{\mathsf{T}}.$$

  Ou a expressão que permite pré-computar algumas operações aritméticas

$$f_s(\mathrm{v}) = \sum_i \hat{\mathrm{q}}_i \frac{1}{\mu_s} A_i + \mathrm{q}^*,$$

$$\text{onde} \quad A_i = w_i \begin{pmatrix} \hat{\mathrm{p}}_i \\ -\hat{\mathrm{p}}_i^{\perp} \end{pmatrix} \begin{pmatrix} \mathrm{v} - \mathrm{p}^* \\ -(\mathrm{v} - \mathrm{p}^*)^{\perp} \end{pmatrix}^{\mathsf{T}} \quad \text{e} \quad (x, y)^{\perp} = (-y, x).$$

- Deformações rígidas

$$f_r(\mathrm{v}) = (\mathrm{v} - \mathrm{p}^*)(\frac{1}{\mu_r}) \sum_j \hat{\mathrm{p}}_j^{\mathsf{T}} w_j \, \hat{\mathrm{q}}_j + \mathrm{q}^*, \tag{3.6}$$

$$\text{onde} \quad \mu_r = \sqrt{\left(\sum_i w_i \, \hat{\mathrm{q}}_i \hat{\mathrm{p}}_i^{\mathsf{T}} \right)^2 + \left(\sum_i w_i \, \hat{\mathrm{q}}_i \hat{\mathrm{p}}_i^{\perp \mathsf{T}} \right)^2}.$$

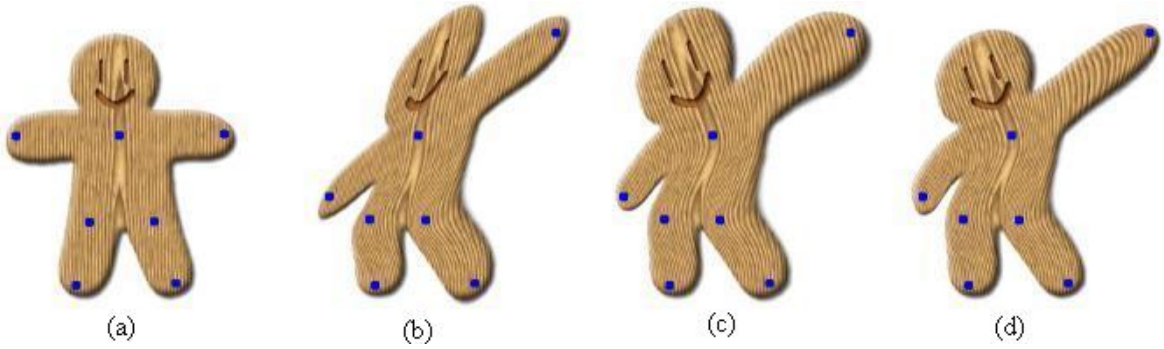


Figura 3.1: Resultados da técnica de deformação de imagens usando MLS apresentada por Schaefer et al. [SMW06]. (a) Imagem original com os pontos de controle. (b) Deformação usando transformações afins. (c) Deformação usando transformações de similaridade. (d) Deformação usando transformações rígidas. Figura retirada de [SMW06].

Ou a expressão que permite pré-computar algumas operações aritméticas

$$f_r(\mathrm{v}) = |\mathrm{v} - \mathrm{p}^*| \frac{\vec{f_r}(\mathrm{v})}{|\vec{f_r}(\mathrm{v})|} + \mathrm{q}^*,$$

$$\text{onde} \quad \vec{f_r}(\mathrm{v}) = \sum_i \hat{\mathrm{q}}_i A_i.$$

A Figura 3.1 ilustra as diferenças entre as três técnicas aplicadas na deformação de uma mesma imagem. Pode-se observar que a deformação usando transformações rígidas apresenta um resultado mais natural do que usando transformações de similaridade, a qual, ao mesmo tempo, produz uma deformação mais plausível do que as transformações afins. Ademais, conforme ilustrado na Figura 3.2, a técnica pode ser usada para deformar malhas 2D sem precisar modificar sua formulação. Observe, também, que embora transformações rígidas estejam sendo aplicadas nos vértices a rigidez dos polígonos não é mantida.

3.2 Deformações 3D usando transformações rígidas

Como foi notado por [ACOL00, IMH05, SMW06] (veja também a Figura 3.1), deformações que aplicam transformações rígidas nos pontos das imagens produzem resultados visualmente mais naturais do que os obtidos por outras classes de transformações. Por exemplo, distorções na Figura 3.1 (b) e (c) não são notadas na Figura 3.1(d).

Uma extensão natural consiste em estudar esses enfoques na manipulação de modelos tridimensionais. Existem algumas propostas para computar a transformação rígida L da

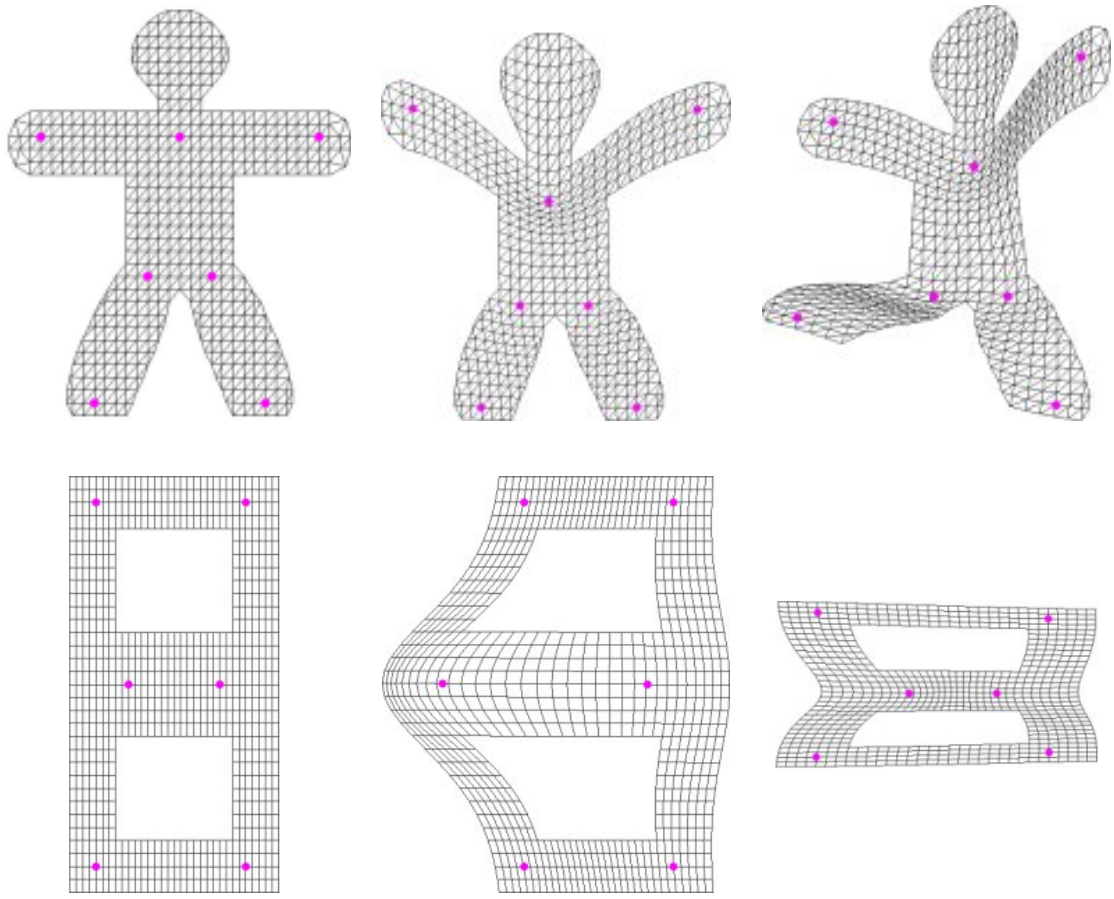


Figura 3.2: Deformações MLS usando transformações rígidas aplicadas em malhas 2D compostas de triângulos (fila superior) e de quadriláteros (fila inferior).

Equação (3.2) para o caso tridimensional. Embora tais propostas tenham sido introduzidas no contexto dos problemas de casamento e registro de conjuntos de amostras; elas podem ser usadas no contexto do problema de deformação. Por exemplo, foram introduzidas soluções analíticas baseadas na decomposição de valores singulares (SVD: *Singular Value Decomposition*) [AHB87], quatérnios unitários [Hor87, Kan94], matrizes ortonormais [HHN88] e quatérnios duais [WSV91]. Um outro grupo de técnicas propostas para este problema consiste naquelas baseadas em esquemas iterativos, tal como o algoritmo ICP (*Iterative Closest Point*) [BM92] e suas variantes.

No entanto, alguns dos métodos analíticos podem gerar matrizes de rotação incorretas (determinante = -1) frente a determinadas disposições dos pontos de controle. Isto ocasiona movimentos bruscos nos modelos afetando a intuitividade da deformação. Além disso, já que o interesse deste trabalho é estudar técnicas de deformação interativas, faz-se necessário pensar em enfoques que possam ser implementados de forma simples e direta em hardware moderno, por exemplo GPUs (*Graphics Processing Units*). É nesse contexto

que aborda-se a extensão da metodologia baseada em MLS para o caso tridimensional.

Problema 2: Dados os conjuntos de pontos $\{\mathrm{p}_1, \ldots, \mathrm{p}_n\}$ e $\{\mathrm{q}_1, \ldots, \mathrm{q}_n\}$ em $\Re^3$, e um ponto $\mathrm{v} \in \Re^3$, computar a transformação rígida l_v que minimize

$$\sum_i w_i \|\mathrm{q}_i - l_\mathrm{v}(\mathrm{p}_i)\|^2, \tag{3.7}$$

onde $\mathrm{p}_i = [\mathrm{p}_{ix}\, \mathrm{p}_{iy}\, \mathrm{p}_{iz}]$, $\mathrm{q}_i = [\mathrm{q}_{ix}\, \mathrm{q}_{iy}\, \mathrm{q}_{iz}]$, $\|\cdot\|$ é a norma euclidiana e w_i é um valor positivo correspondente à função de peso

$$w_i = \frac{1}{\|\mathrm{p}_i - \mathrm{v}\|^2}.$$

Posto que l_v é uma transformação rígida, ela é composta de um vetor de translação T e uma componente rotacional R,

$$l(\mathrm{x}) = \mathrm{xR} + \mathrm{T}. \tag{3.8}$$

Para reduzir a complexidade do problema de otimização (devido à restrição não linear $\mathrm{I} = \mathrm{RR}^\mathsf{T}$), resolve-se em duas partes: primeiro para T e depois para a componente rotacional.

Expandindo o funcional de erro E da Equação (3.7) e removendo os termos que não dependem de T, tem-se

$$\begin{aligned}
E &= \sum_i w_i \|\mathrm{q}_i - l_\mathrm{v}(\mathrm{p}_i)\|^2 \\
&= \sum_i w_i (\mathrm{q}_i - \mathrm{p}_i\mathrm{R} - \mathrm{T})(\mathrm{q}_i - \mathrm{p}_i\mathrm{R} - \mathrm{T})^\mathsf{T} \\
&= \sum_i w_i (-\mathrm{T}\mathrm{q}_i^\mathsf{T} + \mathrm{T}(\mathrm{p}_i\mathrm{R})^\mathsf{T} - \mathrm{q}_i\mathrm{T}^\mathsf{T} + \mathrm{p}_i\mathrm{R}\mathrm{T}^\mathsf{T} + \mathrm{T}\,\mathrm{T}^\mathsf{T}) \\
&= \sum_i w_i (-2\mathrm{T}\mathrm{q}_i^\mathsf{T} + 2\mathrm{T}(\mathrm{p}_i\mathrm{R})^\mathsf{T} + \mathrm{T}\,\mathrm{T}^\mathsf{T}).
\end{aligned}$$

O valor de E é minimizado quando as derivadas parciais em função de T são iguais a zero:

$$\begin{aligned}
\frac{\partial}{\partial\, \mathrm{T}} E = \frac{\partial \sum_i w_i (-2\mathrm{T}\mathrm{q}_i^\mathsf{T} + 2\mathrm{T}(\mathrm{p}_i\mathrm{R})^\mathsf{T} + \mathrm{T}\,\mathrm{T}^\mathsf{T})}{\partial\, \mathrm{T}} &= 0 \\
-2\sum_i w_i \mathrm{q}_i^\mathsf{T} + 2\sum_i w_i (\mathrm{p}_i\mathrm{R})^\mathsf{T} + 2\sum_i w_i \mathrm{T}^\mathsf{T} &= 0 \\
\sum_i w_i \mathrm{T} &= \sum_i w_i \mathrm{q}_i - \sum_i w_i \mathrm{p}_i\mathrm{R} \\
\mathrm{T} &= \frac{\sum_i w_i \mathrm{q}_i}{\sum_i w_i} - \frac{\sum_i w_i \mathrm{p}_i}{\sum_i w_i}\mathrm{R}.
\end{aligned}$$

Então, pode-se escrever T como sendo

$$\mathrm{T} = \mathrm{q}^* - \mathrm{p}^*\mathrm{R}, \tag{3.9}$$

onde p* e q* são os centróides

$$\mathrm{p}^* = \frac{\sum_i w_i \, \mathrm{p}_i}{\sum_i w_i}, \quad \mathrm{q}^* = \frac{\sum_i w_i \, \mathrm{q}_i}{\sum_i w_i}. \tag{3.10}$$

Para encontrar a componente rotacional R substitui-se T na Equação (3.8):

$$l(\mathrm{x}) = (\mathrm{x} - \mathrm{p}^*)\mathrm{R} + \mathrm{q}^*.$$

Com isso, o problema da Equação (3.7) pode ser escrito da seguinte forma:

$$\min_{R \in SO(3)} \sum_i w_i \|\hat{\mathrm{q}}_i - \hat{\mathrm{p}}_i \mathrm{R}\|^2, \tag{3.11}$$

onde, $\hat{\mathrm{p}}_i = \mathrm{p}_i - \mathrm{p}^*$, $\hat{\mathrm{q}}_i = \mathrm{q}_i - \mathrm{q}^*$ e $SO(3)$ representa o espaço das transformações ortogonais positivas.

A expressão (3.11) pode ser expandida como

$$\begin{aligned}
\sum_i w_i \|\hat{\mathrm{q}}_i - \hat{\mathrm{p}}_i \mathrm{R}\|^2 &= \sum_i w_i (\hat{\mathrm{q}}_i - \hat{\mathrm{p}}_i \mathrm{R})(\hat{\mathrm{q}}_i - \hat{\mathrm{p}}_i \mathrm{R})^\mathsf{T} \\
&= \sum_i w_i (\hat{\mathrm{q}}_i \hat{\mathrm{q}}_i^\mathsf{T} - \hat{\mathrm{q}}_i (\hat{\mathrm{p}}_i \mathrm{R})^\mathsf{T} - \hat{\mathrm{p}}_i \mathrm{R} \hat{\mathrm{q}}_i^\mathsf{T} + \hat{\mathrm{p}}_i \mathrm{R} (\hat{\mathrm{p}}_i \mathrm{R})^\mathsf{T}) \\
&= \sum_i w_i (\hat{\mathrm{q}}_i \hat{\mathrm{q}}_i^\mathsf{T} - \hat{\mathrm{q}}_i \mathrm{R}^\mathsf{T} \hat{\mathrm{p}}_i^\mathsf{T} - \hat{\mathrm{p}}_i (\hat{\mathrm{q}}_i \mathrm{R})^\mathsf{T} + \hat{\mathrm{p}}_i \mathrm{R}\mathrm{R}^\mathsf{T} \hat{\mathrm{p}}_i^\mathsf{T}) \\
&= -2 \sum_i w_i \hat{\mathrm{q}}_i \mathrm{R} \hat{\mathrm{p}}_i^\mathsf{T} + \sum_i w_i \|\hat{\mathrm{q}}_i\|^2 + \sum_i w_i \|\hat{\mathrm{p}}_i\|^2.
\end{aligned}$$

Sendo que o segundo e terceiro termos são constantes (não dependem de R), a minimização de (3.11) equivale a:

$$\max_{R \in SO(3)} \sum_i w_i \hat{\mathrm{q}}_i \mathrm{R} \hat{\mathrm{p}}_i^\mathsf{T}. \tag{3.12}$$

Vamos definir R como uma matriz que efetua uma rotação de α graus ao redor do eixo **e**, como

$$\mathrm{R}_{\mathbf{e},\alpha}(\mathbf{v}^\mathsf{T}) = \mathbf{e}^\mathsf{T}\mathbf{e}\mathbf{v}^\mathsf{T} + \cos(\alpha)(\mathbf{I} - \mathbf{e}^\mathsf{T}\mathbf{e})\mathbf{v}^\mathsf{T} + \sin(\alpha) \begin{pmatrix} 0 & -\mathbf{e}_z & \mathbf{e}_y \\ \mathbf{e}_z & 0 & -\mathbf{e}_x \\ -\mathbf{e}_y & \mathbf{e}_x & 0 \end{pmatrix} \mathbf{v}^\mathsf{T}. \tag{3.13}$$

Então, substituindo R na equação (3.12), deve-se maximizar

$$\sum_i w_i \hat{\mathrm{q}}_i \mathbf{e}^{\mathsf{T}} \mathbf{e}\, \hat{\mathrm{p}}_i^{\mathsf{T}} + \cos(\alpha) \sum_i w_i \hat{\mathrm{q}}_i (\mathrm{I} - \mathbf{e}^{\mathsf{T}}\mathbf{e}) \hat{\mathrm{p}}_i^{\mathsf{T}} + \sin(\alpha) \sum_i w_i \hat{\mathrm{q}}_i \begin{pmatrix} 0 & -\mathbf{e}_z & \mathbf{e}_y \\ \mathbf{e}_z & 0 & -\mathbf{e}_x \\ -\mathbf{e}_y & \mathbf{e}_x & 0 \end{pmatrix} \hat{\mathrm{p}}_i^{\mathsf{T}},$$

que equivale a maximizar

$$\begin{aligned} & \sum_i w_i \hat{\mathrm{q}}_i \mathbf{e}^{\mathsf{T}} \mathbf{e}\, \hat{\mathrm{p}}_i^{\mathsf{T}} \\ & + \cos(\alpha)\Big(\sum_i w_i \hat{\mathrm{q}}_i \hat{\mathrm{p}}_i^{\mathsf{T}} - \sum_i w_i \mathbf{e}\, \hat{\mathrm{q}}_i^{\mathsf{T}} \hat{\mathrm{p}}_i \mathbf{e}^{\mathsf{T}}\Big) \\ & + \sin(\alpha) \sum_i w_i \hat{\mathrm{q}}_i \begin{pmatrix} 0 & -\mathbf{e}_z & \mathbf{e}_y \\ \mathbf{e}_z & 0 & -\mathbf{e}_x \\ -\mathbf{e}_y & \mathbf{e}_x & 0 \end{pmatrix} \hat{\mathrm{p}}_i^{\mathsf{T}}. \end{aligned} \tag{3.14}$$

Observe que:

$$\hat{\mathrm{q}}_i \begin{pmatrix} 0 & -\mathbf{e}_z & \mathbf{e}_y \\ \mathbf{e}_z & 0 & -\mathbf{e}_x \\ -\mathbf{e}_y & \mathbf{e}_x & 0 \end{pmatrix} \hat{\mathrm{p}}_i^{\mathsf{T}} = \hat{\mathrm{q}}_i \begin{pmatrix} 0 & \hat{\mathrm{p}}_{iz} & -\hat{\mathrm{p}}_{iy} \\ -\hat{\mathrm{p}}_{iz} & 0 & \hat{\mathrm{p}}_{ix} \\ \hat{\mathrm{p}}_{iy} & -\hat{\mathrm{p}}_{ix} & 0 \end{pmatrix} \mathbf{e}^{\mathsf{T}} = (\hat{\mathrm{p}}_i \times \hat{\mathrm{q}}_i)\mathbf{e}^{\mathsf{T}}.$$

Portanto, a expressão (3.14) pode ser reescrita assim:

$$\mathbf{e}\Big(\sum_i w_i \hat{\mathrm{q}}_i^{\mathsf{T}} \hat{\mathrm{p}}_i\Big)\mathbf{e}^{\mathsf{T}} + \cos(\alpha)\Big(\sum_i w_i \hat{\mathrm{q}}_i \hat{\mathrm{p}}_i^{\mathsf{T}}\Big) - \cos(\alpha)\mathbf{e}\Big(\sum_i w_i \hat{\mathrm{q}}_i^{\mathsf{T}} \hat{\mathrm{p}}_i\Big)\mathbf{e}^{\mathsf{T}} + \sin(\alpha)\Big(\sum_i w_i \hat{\mathrm{p}}_i \times \hat{\mathrm{q}}_i\Big)\mathbf{e}^{\mathsf{T}}.$$

Finalmente, se as variáveis M, E, e V são definidas como sendo:

$$\mathrm{M} = \sum_i w_i \hat{\mathrm{q}}_i^{\mathsf{T}} \hat{\mathrm{p}}_i = \begin{pmatrix} \sum_i w_i \hat{\mathrm{q}}_{ix} \hat{\mathrm{p}}_{ix} & \sum_i w_i \hat{\mathrm{q}}_{ix} \hat{\mathrm{p}}_{iy} & \sum_i w_i \hat{\mathrm{q}}_{ix} \hat{\mathrm{p}}_{iz} \\ \sum_i w_i \hat{\mathrm{q}}_{iy} \hat{\mathrm{p}}_{ix} & \sum_i w_i \hat{\mathrm{q}}_{iy} \hat{\mathrm{p}}_{iy} & \sum_i w_i \hat{\mathrm{q}}_{iy} \hat{\mathrm{p}}_{iz} \\ \sum_i w_i \hat{\mathrm{q}}_{iz} \hat{\mathrm{p}}_{ix} & \sum_i w_i \hat{\mathrm{q}}_{iz} \hat{\mathrm{p}}_{iy} & \sum_i w_i \hat{\mathrm{q}}_{iz} \hat{\mathrm{p}}_{iz} \end{pmatrix}, \tag{3.15}$$

$$\mathrm{E} = \sum_i w_i \hat{\mathrm{q}}_i\, \hat{\mathrm{p}}_i^{\mathsf{T}} = \sum_i w_i (\hat{\mathrm{q}}_{ix}\hat{\mathrm{p}}_{ix} + \hat{\mathrm{q}}_{iy}\hat{\mathrm{p}}_{iy} + \hat{\mathrm{q}}_{iz}\hat{\mathrm{p}}_{iz}) = \mathrm{Trace}(\mathrm{M}),$$

$$\mathrm{V} = \sum_i w_i \hat{\mathrm{p}}_i \times \hat{\mathrm{q}}_i = \begin{pmatrix} (\mathrm{M}_{32} - \mathrm{M}_{23} & \mathrm{M}_{13} - \mathrm{M}_{31} & \mathrm{M}_{21} - \mathrm{M}_{12}) \end{pmatrix}.$$

Então,

$$\sum_i w_i \hat{\mathrm{q}}_i \mathrm{R} \hat{\mathrm{p}}_i^{\mathsf{T}} = \mathbf{e}\,\mathrm{M}\,\mathbf{e}^{\mathsf{T}} + \cos(\alpha)(\mathrm{E} - \mathbf{e}\,\mathrm{M}\,\mathbf{e}^{\mathsf{T}}) + \sin(\alpha)\mathrm{V}\,\mathbf{e}^{\mathsf{T}}. \tag{3.16}$$

3.2.1 Rotação ótima

Considerando as expressões (3.12) e (3.16), observa-se que o problema de otimização pode ser escrito como

$$\begin{aligned} \text{maximizar} \quad & \mathbf{e}\,\mathrm{M}\,\mathbf{e}^{\mathsf{T}} + \cos(\alpha)(\mathrm{E} - \mathbf{e}\,\mathrm{M}\,\mathbf{e}^{\mathsf{T}}) + \sin(\alpha)\mathrm{V}\,\mathbf{e}^{\mathsf{T}} \\ \text{sujeito a} \quad & \|\mathbf{e}\| = 1, \\ & \cos(\alpha)^2 + \sin(\alpha)^2 = 1. \end{aligned} \tag{3.17}$$

Seja L uma função de Lagrange definida por

$$\begin{aligned} L(\mathbf{e}, \sin(\alpha), \cos(\alpha)) = \mathbf{e}\,\mathrm{M}\,\mathbf{e}^{\mathsf{T}} + \cos(\alpha)(\mathrm{E} - \mathbf{e}\,\mathrm{M}\,\mathbf{e}^{\mathsf{T}}) + \sin(\alpha)\mathrm{V}\,\mathbf{e}^{\mathsf{T}} - \\ k_1(\|\mathbf{e}\| - 1) - \frac{k_2}{2}(\cos(\alpha)^2 + \sin(\alpha)^2 - 1). \end{aligned}$$

Então, as soluções ótimas do problema (3.17) devem satisfazer $\nabla L = 0$. Isto acontece quando

$$\left(\frac{\partial L}{\partial\, \mathbf{e}}, \frac{\partial L}{\partial \sin(\alpha)}, \frac{\partial L}{\partial \cos(\alpha)}\right) = (0, 0, 0).$$

Dessa forma, deriva-se L parcialmente e obtém-se as seguintes equações:

$$\begin{aligned} \mathbf{M}\mathbf{e}^{\mathsf{T}} + \mathbf{e}\mathbf{M} - \cos(\alpha)(\mathbf{M}\mathbf{e}^{\mathsf{T}} + \mathbf{e}\mathbf{M}) + \sin(\alpha)\mathbf{V} = & \\ (1 - \cos(\alpha))\,\mathbf{e}(\mathrm{M} + \mathrm{M}^{\mathsf{T}}) + \sin(\alpha)\mathrm{V} = k_1\,\mathbf{e}, & \quad (3.18) \\ \mathrm{E} - \mathbf{e}\,\mathrm{M}\,\mathbf{e}^{\mathsf{T}} = k_2\cos(\alpha), & \quad (3.19) \\ \mathrm{V}\,\mathbf{e}^{\mathsf{T}} = k_2\sin(\alpha). & \quad (3.20) \end{aligned}$$

Da Equação (3.20), tem-se que $\sin(\alpha) = \mathrm{V}\mathbf{e}^{\mathsf{T}}/k_2$, e considerando $\mathrm{N} = \mathrm{M} + \mathrm{M}^{\mathsf{T}}$, pode-se escrever a Equação (3.18) como

$$\begin{aligned} \left((1 - \cos(\alpha))\,\mathbf{e}\mathrm{N} + \frac{\mathrm{V}\mathbf{e}^{\mathsf{T}}\mathrm{V}}{k_2}\right)^{\mathsf{T}} &= (k_1\,\mathbf{e})^{\mathsf{T}} \quad \therefore \\ (1 - \cos(\alpha))\,\mathrm{N}\mathbf{e}^{\mathsf{T}} + \frac{\mathrm{V}^{\mathsf{T}}\mathrm{V}\mathbf{e}^{\mathsf{T}}}{k_2} &= k_1\,\mathbf{e}^{\mathsf{T}} \quad \therefore \\ \mathrm{N}\mathbf{e}^{\mathsf{T}} + \frac{\mathrm{V}^{\mathsf{T}}\mathrm{V}\mathbf{e}^{\mathsf{T}}}{k_2(1 - \cos(\alpha))} &= \frac{k_1}{(1 - \cos(\alpha))}\,\mathbf{e}^{\mathsf{T}}. \end{aligned}$$

Assim,

$$(\mathrm{N} + a\mathrm{V}^{\mathsf{T}}\mathrm{V})\mathbf{e}^{\mathsf{T}} = \lambda\,\mathbf{e}^{\mathsf{T}}, \tag{3.21}$$

onde

$$a = \frac{1}{k_2(1 - \cos(\alpha))} \qquad \text{e} \qquad \lambda = \frac{k_1}{(1 - \cos(\alpha))}. \tag{3.22}$$

Sendo o sistema (3.21) uma instância de um problema de autovalores, o eixo ótimo de rotação corresponde ao autovetor da matriz $(\mathrm{N} + a\mathrm{V}^\mathsf{T}\mathrm{V})$ associado ao autovalor λ. Ocorre entretanto que, expressos dessa maneira, os valores de a e λ são dependentes das variáveis k_1 e k_2. Portanto, não se tem um problema de determinação de autovalores típico. Para reduzir o grau de indeterminação, é possível mostrar que as condições de otimalidade permitem relacionar λ e a sem fazer uso dessas variáveis.

Para isso começa-se multiplicando por $\mathbf{e}^\mathsf{T}$ ambos os lados da Equação (3.19) obtendo:

$$(1 - \cos(\alpha))\,\mathbf{e}(\mathrm{M} + \mathrm{M}^\mathsf{T})\mathbf{e}^\mathsf{T} + \sin(\alpha)\mathrm{V}\mathbf{e}^\mathsf{T} = k_1\,\mathbf{e}\,\mathbf{e}^\mathsf{T}. \tag{3.23}$$

Mas como $\mathbf{e}\mathrm{M}\mathbf{e}^\mathsf{T} = \mathbf{e}\mathrm{M}^\mathsf{T}\mathbf{e}^\mathsf{T}$, então

$$\mathbf{e}(\mathrm{M} + \mathrm{M}^\mathsf{T})\,\mathbf{e}^\mathsf{T} = 2\mathbf{e}\mathrm{M}\mathbf{e}^\mathsf{T} \overset{3.19}{=} 2(\mathrm{E} - k_2\cos(\alpha)).$$

Além disso, $\mathrm{V}\mathbf{e}^\mathsf{T} = k_2\sin(\alpha)$. Como $\mathbf{e}$ é um vetor unitário, a expressão acima toma a forma

$$\begin{aligned}(1 - \cos(\alpha))2(\mathrm{E} - k_2\cos(\alpha)) + \sin(\alpha)(k_2\sin(\alpha)) =& k_1 \quad \therefore\\ 2\mathrm{E} - 2\mathrm{E}\cos(\alpha) - 2k_2\cos(\alpha) + 2k_2\cos^2(\alpha) + k_2\sin^2(\alpha) =& k_1.\end{aligned}$$

Como $2k_2\cos^2(\alpha) + k_2\sin^2(\alpha) = k_2(\cos^2(\alpha) + \cos^2(\alpha) + \sin^2(\alpha)) = k_2 + k_2\cos^2(\alpha)$ e $k_2 + k_2\cos^2(\alpha) - 2k_2\cos(\alpha) = k_2(1 - 2\cos\alpha + \cos^2(\alpha)) = k_2(1 - \cos(\alpha))^2$, a expressão acima se torna:

$$2\mathrm{E}(1 - \cos(\alpha)) + k_2(1 - \cos(\alpha))^2 = k_1.$$

Dividindo ambos os lados por $(1 - \cos(\alpha))$, tem-se

$$2\mathrm{E} + k_2(1 - \cos(\alpha)) = \frac{k_1}{(1 - \cos(\alpha))},$$

o que permite concluir que:

$$2\mathrm{E} + \frac{1}{a} = \lambda,$$

ou

$$a = \frac{1}{\lambda - 2\mathrm{E}}. \tag{3.24}$$

3.2.2 Computando o autovalor λ

Se A é uma matriz 3×3, então o seu polinômio característico está definido por:

$$P(\lambda) = \lambda^3 - \text{Trace(A)}\lambda^2 + \frac{1}{2}(\text{Trace(A)}^2 - \text{Trace(A}^2))\lambda - \det(\text{A}).$$

Adicionalmente, as relações

$$\text{Trace}(\text{A} + \text{b}^\text{T}\text{b}) = \text{Trace}(\text{A}) + \text{Trace}(\text{b}^\text{T}\text{b}) = \text{Trace}(\text{A}) + \|\text{b}\|^2 \quad \text{e}$$
$$\det(\text{A} + \text{b}^\text{T}\text{b}) = \det(\text{A})(1 + \text{bA}^{-1}\text{b}^\text{T}),$$

são satisfeitas quando b é um vetor fila de dimensão 3.

Como λ é um autovalor da matriz $(\text{N} + a\text{V}^\text{T}\text{V})$, ele deve ser uma raiz do polinômio característico $P(\lambda)$ que tem a forma:

$$\begin{aligned} P(\lambda) = \lambda^3 - \\ \text{Trace}(\text{N} + a\text{V}^\text{T}\text{V})\,\lambda^2 + \\ \tfrac{1}{2}(\text{Trace}(\text{N} + a\text{V}^\text{T}\text{V})^2 - \text{Trace}((\text{N} + a\text{V}^\text{T}\text{V})^2))\,\lambda - \\ \det(\text{N} + a\text{V}^\text{T}\text{V}) \quad \therefore \end{aligned}$$

$$\begin{aligned} P(\lambda) = \lambda^3 - \\ (\text{Trace}(\text{N}) + a\|\text{V}\|^2)\,\lambda^2 + \\ ((\tfrac{1}{2}(\text{Trace}(\text{N})^2 - \|\text{N}\|^2) + a(\|\text{V}\|^2\text{Trace}(\text{N}) - \text{VNV}^\text{T}))\,\lambda - \\ \det(\text{N})(1 + \text{VN}^{-1}\text{V}^\text{T}a). \end{aligned}$$

Substituindo $a = \dfrac{1}{(\lambda - 2\text{E})}$ e multiplicando cada termo por $(\lambda - 2\text{E})$, a equação $P(\lambda) = 0$ se torna:

$$\begin{aligned} (\lambda - 2\text{E})\,\lambda^3 - \\ (\text{Trace}(\text{N})(\lambda - 2\text{E}) + \|\text{V}\|^2)\,\lambda^2 + \\ ((\tfrac{1}{2}(\text{Trace}(\text{N})^2 - \|\text{N}\|^2)(\lambda - 2\text{E}) + (\|\text{V}\|^2\text{Trace}(\text{N}) - \text{VNV}^\text{T}))\,\lambda - \\ \det(\text{N})((\lambda - 2\text{E}) + \text{VN}^{-1}\text{V}^\text{T}) = 0. \end{aligned}$$

Desenvolvendo essa expressão em termos das potências de λ, vem:

$$\begin{aligned} \lambda^4 + \\ (-2\text{E} - \text{Trace}(\text{N}))\,\lambda^3 + \\ (2\text{ETrace}(\text{N}) - \|\text{V}\|^2 + \tfrac{1}{2}\text{Trace}(\text{N})^2 - \tfrac{1}{2}\|\text{N}\|^2)\,\lambda^2 + \\ (-\text{Trace}(\text{N})^2\text{E} + \|\text{N}\|^2\text{E} + \|\text{V}\|^2\text{Trace}(\text{N}) - \text{VNV}^\text{T} - \det(\text{N}))\,\lambda - \\ \det(\text{N})(2\text{E} - \text{VN}^{-1}\text{V}^\text{T}) = 0. \end{aligned} \tag{3.25}$$

Observando agora que $\text{Trace(N)} = 2\text{Trace(M)} = 2\text{E}$, $2\|\text{M}\|^2 = \frac{1}{2}\|\text{N}\|^2 + \|\text{V}\|^2$ e $\text{VNV}^\text{T} = 2\text{VMV}^\text{T}$, obtém-se a equação

$$\begin{aligned} \lambda^4 - \\ 4\text{E}\,\lambda^3 + \\ (6\text{E}^2 - 2\|\text{M}\|^2)\,\lambda^2 + \\ ((\|\text{M}\|^2 - \text{E}^2)4\text{E} - 2\text{VMV}^\text{T} - \det(\text{N}))\,\lambda + \\ \det(\text{N})(2\text{E} - \text{VN}^{-1}\text{V}^\text{T}) = 0. \end{aligned} \tag{3.26}$$

Para eliminar o termo de grau 3, faz-se uma mudança de variável: $y = \lambda - \text{E}$. Em função de y, a Equação (3.26) se torna:

$$\begin{aligned} y^4 - 2\|\text{M}\|^2(y - \text{E})^2 - \\ (4\|\text{M}\|^2\text{E} + 2\text{VMV}^\text{T} + \det(\text{N}))(y + \text{E}) + \\ 8\|\text{M}\|^2\text{E}^2 - \text{E}^4 + \det(\text{N})(2\text{E} - \text{VN}^{-1}\text{V}^\text{T}) = 0 \quad \therefore \end{aligned}$$

$$\begin{aligned} y^4 - \\ 2\|\text{M}\|^2\,y^2 - \\ (2\text{VMV}^\text{T} + \det(\text{N}))\,y + \\ 2\|\text{M}\|^2\text{E}^2 - \text{E}^4 - (2\text{VMV}^\text{T} + \det(\text{N}))\text{E} + \det(\text{N})(2\text{E} - \text{VN}^{-1}\text{V}^\text{T}) = 0. \end{aligned}$$

Substituindo $8\det(\text{M}) = 2\text{VMV}^\text{T} + \det(\text{N})$ na equação anterior, tem-se

$$\begin{aligned} y^4 - \\ 2\|\text{M}\|^2\,y^2 - \\ 8\det(\text{M})\,y + \\ 2\|\text{M}\|^2\text{E}^2 - \text{E}^4 - 8\det(\text{M})\text{E} + \det(\text{N})(2\text{E} - \text{VN}^{-1}\text{V}^\text{T}) = 0. \end{aligned} \tag{3.27}$$

Observe agora que se **e** satisfaz as condições de otimalidade, então a função objetivo aplicada a **e** se reduz a

$$\mathbf{e}\text{M}\mathbf{e}^\text{T} + \cos(\alpha)(\text{E} - \mathbf{e}\text{M}\mathbf{e}^\text{T}) + \sin(\alpha)\text{V}\mathbf{e}^\text{T} = \text{E}\cos(\alpha) + (1 - \cos(\alpha))\mathbf{e}\text{M}\mathbf{e}^\text{T} + \sin(\alpha)\text{V}\mathbf{e}^\text{T}.$$

Lembrando uma vez mais que $2(\mathbf{e}\text{M}\mathbf{e}^\text{T}) = \mathbf{e}\text{N}\mathbf{e}^\text{T}$, tem-se,

$$\overbrace{\text{E}\cos(\alpha) - (1 - \cos(\alpha))\mathbf{e}\text{M}\mathbf{e}^\text{T}}^{A} + \overbrace{(1 - \cos(\alpha))\mathbf{e}\text{N}\mathbf{e}^\text{T} + \sin(\alpha)\text{V}\mathbf{e}^\text{T}}^{B}.$$

Usando (3.19), o termo A pode ser reescrito da seguinte maneira: $A = \text{E}\cos(\alpha) - (1 - \cos(\alpha))(\text{E} - k_2\cos(\alpha)) = 2\text{E}\cos(\alpha) - \text{E} + \cos(\alpha)(k_2(1 - \cos(\alpha))) = 2\text{E}\cos(\alpha) - \text{E} + \cos(\alpha)(\alpha - 2\text{E}) = \lambda\cos(\alpha) - \text{E}$.

Já (3.20) permite reescrever o termo B da seguinte forma $B = (k_1\mathbf{e})\mathbf{e}^\mathsf{T} = k_1 = \lambda(1-\cos(\alpha))$.

Assim $A+B = \lambda\cos(\alpha)-\mathrm{E}+\lambda(1-\cos(\alpha)) = \lambda-\mathrm{E} = y$, o que significa que o valor da função objetivo (Equação 3.17) para um **e** que satisfaz as condições de otimalidade é dado pelo próprio valor de y. Portanto, como deseja-se maximizar essa função objetivo, deve-se fazer y ser a maior solução real da Equação (3.27).

3.2.3 Resolvendo a equação de quarto grau

Um polinômio de quarto grau da forma $y^4 + ay^2 + by + c$ pode ser fatorado como $(y^2 + py + q)(y^2 - py + s)$ se p satisfaz $p^6 + 2ap^4 + (a^2 - 4c)p^2 - b^2 = 0$, que é uma equação cúbica em z para $z = p^2$.

Se o polinômio de quarto grau está definido pela expressão (3.27), então o polinômio cúbico correspondente é

$$\begin{aligned} Q(z) = z^3 - 4\|\mathbf{M}\|^2 z^2 + \\ 16(\textstyle\sum_i\sum_{j<i}(\|\mathbf{M}_i\|\,\|\mathbf{M}_j\|)^2 - (\mathbf{M}_i\mathbf{M}_j^\mathsf{T})^2)z - 64\det(\mathbf{M})^2. \end{aligned} \tag{3.28}$$

Agora, seja Q^* o polinômio característico de $\mathrm{M}^\mathsf{T}\mathrm{M}$. Posto que $Q(z) = 64Q^*(z/4)$, então uma raiz r de Q estará relacionada com algum autovalor μ de $(\mathrm{M}^\mathsf{T}\mathrm{M})$ pela relação $r = 4\mu$. Isto implica que $r \geq 0$, posto que todos os autovalores da matriz semi-definida positiva $\mathrm{M}^\mathsf{T}\mathrm{M}$ devem ser não negativos.

Uma raiz de $Q(z) = 0$ pode ser computada com o seguinte algoritmo, o qual segue o método analítico publicado por Cardano[1]:

Seja $p = \sqrt{r}$. Então, o valor de y é igual ao maior valor entre

$$-p/2 \;+\; \sqrt{-p^2/4 + \|\mathbf{M}\|^2 - 4\det(\mathbf{M})/p} \qquad \text{e}$$

$$p/2 \;+\; \sqrt{-p^2/4 + \|\mathbf{M}\|^2 + 4\det(\mathbf{M})/p}.$$

Observe que $Q(z)$ é essencialmente o mesmo polinômio usado para encontrar os autovalores de $\mathrm{M}^\mathsf{T}\mathrm{M}$ pelo método matricial de Horn [HHN88]. No entanto, se $\det(\mathrm{M}) > 0$, o valor de y obtido pelo método proposto é $\mathrm{Trace}((\mathrm{M}^\mathsf{T}\mathrm{M})^{1/2})$. De outra forma, é $\mathrm{Trace}((\mathrm{M}^\mathsf{T}\mathrm{M})^{1/2}) - 2\sqrt{\mu_{min}}$, onde μ_{min} é o menor autovalor de $\mathrm{M}^\mathsf{T}\mathrm{M}$.

[1] Uma discussão recente sobre métodos analíticos para resolver equações cúbicas pode ser encontrada no trabalho de Blinn [Bli06].

Algoritmo 1: Computar uma raiz real de uma equação cúbica.

Input: a, b, c: coeficientes do polinômio cúbico $Q(z) = z^3 + az^2 + bz^2 + c$

Output: r: uma raiz da equação $Q(z) = 0$

$p \leftarrow (3b - a^2)/9$;

$q \leftarrow (9ab - 27c - 2a^3)/54$;

$D \leftarrow p^3 + q^2$;

if $D \geq 0$ **then**

    $r \leftarrow \sqrt[3]{q + \sqrt{D}} + \sqrt[3]{q - \sqrt{D}} - a/3$;

else

    $\theta \leftarrow \arccos(q/\sqrt{-p^3})$;

    $r \leftarrow 2\sqrt{-p}\cos(\theta/3) - a/3$;

end

return r ;

Baseado no raciocínio anterior, pode-se notar que encontrar λ não demanda mais esforço que obter todos os autovalores de $\mathrm{M}^\mathsf{T}\mathrm{M}$, como requerido por métodos matriciais. Deste ponto de vista, o método descrito neste trabalho é ligeiramente vantajoso. Enquanto o método de Horn determina uma base de autovetores unitários de $\mathrm{M}^\mathsf{T}\mathrm{M}$ para computar $\mathrm{M}(\mathrm{M}^\mathsf{T}\mathrm{M})^{1/2}$, o método proposto requer, após ter encontrado λ, resolver um sistema linear 3×3 para obter a rotação definida por $\{\mathbf{e}, \cos(\alpha), \sin(\alpha)\}$, conforme o procedimento descrito a seguir.

3.2.4 Computando o eixo de rotação

A Equação (3.21) pode ser reescrita como

$$(\mathrm{N} - \lambda\mathrm{I}) \left(\frac{\mathbf{e}^\mathsf{T}}{\phi} \right) = \mathrm{V}^\mathsf{T}, \tag{3.29}$$

onde $\phi = -a(\mathrm{V}\mathbf{e}^\mathsf{T})$. Este sistema está bem definido para $\mathrm{V}\mathbf{e}^\mathsf{T} \neq 0$ visto que neste caso $\phi \neq 0$. Portanto, considerando que $\pm(\mathbf{e}, \alpha)$ representam a mesma rotação, podemos inferir que $\mathbf{e}$ pode ser determinado resolvendo

$$(\mathrm{N} - \lambda\mathrm{I})\, \mathbf{u}^\mathsf{T} = \mathrm{V}^\mathsf{T} \tag{3.30}$$

e computando o vetor unitário de $\mathbf{u}$. Para pequenas deformações, em especial, este método é mais apropriado que a computação direta de um autovalor unitário de $(\mathrm{N} +$

$a\mathrm{VV}^{\mathsf{T}}$) posto que a pode assumir valores grandes.

Resta considerar o caso onde $\mathbf{Ve}^{\mathsf{T}} = 0$. Assumindo que $k_2 \neq 0$ e $\alpha \neq 0$, as três seguintes condições são equivalentes:

(i) $\mathbf{Ve}^{\mathsf{T}} = 0$;

(ii) (3.30) é um sistema indeterminado;

(iii) λ é um autovalor de N. Neste caso, **e** é um autovetor associado a λ.

Assim, se $\mathbf{Ve}^{\mathsf{T}} = 0$, a condição (ii) é satisfeita, e quando identificada pode-se computar **e** como sendo o autovetor unitário de N associado ao autovalor λ.

3.2.5 Computando $\sin(\alpha)$ e $\cos(\alpha)$

Das equações (3.22) e (3.24) infere-se que $\lambda - 2\mathrm{E} = k_2(1 - \cos(\alpha))$. Adicionalmente, a Equação (3.19) estabelece que $\mathrm{E} - \mathbf{eMe}^{\mathsf{T}} = k_2 \cos(\alpha)$. Somando estas duas igualdades obtém-se $k_2 = \lambda - \mathrm{E} - \mathbf{eMe}^{\mathsf{T}}$. Isto permite usar a Equação (3.19) para obter as seguintes expressões para $\cos(\alpha)$ e $\sin(\alpha)$:

$$\cos(\alpha) = \frac{\mathrm{E} - \mathbf{eMe}^{\mathsf{T}}}{\lambda - \mathrm{E} - \mathbf{eMe}^{\mathsf{T}}}, \quad \sin(\alpha) = \frac{\mathbf{Ve}^{\mathsf{T}}}{\lambda - \mathrm{E} - \mathbf{eMe}^{\mathsf{T}}}.$$

Por outro lado, da condição de otimalidade (3.20) vem que $k_2 = \dfrac{\mathbf{Ve}^{\mathsf{T}}}{\sin(\alpha)}$, e das equações (3.22) e (3.24) infere-se que $k_2 = \dfrac{\lambda - 2\mathrm{E}}{(1 - \cos(\alpha))}$. Assim,

$$\begin{aligned}
\frac{\mathbf{Ve}^{\mathsf{T}}}{\sin(\alpha)} &= \frac{\lambda - 2\mathrm{E}}{1 - \cos(\alpha)}, \\
\frac{(\mathbf{Ve}^{\mathsf{T}})^2}{(\sin(\alpha))^2} &= \frac{(\lambda - 2\mathrm{E})^2}{(1 - \cos(\alpha))^2}, \\
\frac{(\mathbf{Ve}^{\mathsf{T}})^2}{1 - (\cos(\alpha))^2} &= \frac{(\lambda - 2\mathrm{E})^2}{(1 - \cos(\alpha))^2} \quad \therefore \\
\frac{(\mathbf{Ve}^{\mathsf{T}})^2}{(\lambda - 2\mathrm{E})^2} &= \frac{1 - (\cos(\alpha))^2}{(1 - \cos(\alpha))^2} \quad \therefore \\
\frac{(\mathbf{Ve}^{\mathsf{T}})^2}{(\lambda - 2\mathrm{E})^2} &= \frac{(1 - \cos(\alpha))(1 + \cos(\alpha))}{(1 - \cos(\alpha))^2} \quad \therefore \\
\frac{(\mathbf{Ve}^{\mathsf{T}})^2}{(\lambda - 2\mathrm{E})^2} &= \frac{1 + \cos(\alpha)}{1 - \cos(\alpha)} \quad \therefore \\
(\mathbf{Ve}^{\mathsf{T}})^2 - (\lambda - 2\mathrm{E})^2 &= \cos(\alpha)((\mathbf{Ve}^{\mathsf{T}})^2 + (\lambda - 2\mathrm{E})^2).
\end{aligned}$$

Daqui vem:

$$\cos(\alpha) = \frac{(\mathbf{V}\mathbf{e}^{\mathsf{T}})^2 - (\lambda - 2\mathrm{E})^2}{(\mathbf{V}\mathbf{e}^{\mathsf{T}})^2 + (\lambda - 2\mathrm{E})^2}, \tag{3.31}$$

e como $\sin(\alpha) = \sqrt{1 - \cos^2(\alpha)}$, tem-se:

$$\sin(\alpha) = \frac{2(\mathbf{V}\mathbf{e}^{\mathsf{T}})(\lambda - 2\mathrm{E})}{(\mathbf{V}\mathbf{e}^{\mathsf{T}})^2 + (\lambda - 2\mathrm{E})^2}. \tag{3.32}$$

Da Equação (3.29), infere-se que $\mathbf{u}^{\mathsf{T}} = \dfrac{\mathbf{e}^{\mathsf{T}}}{-a(\mathbf{V}\mathbf{e}^{\mathsf{T}})}$. Posto que $\mathbf{e}^{\mathsf{T}}$ é unitário, então

$$\|\mathbf{u}\| = \frac{1}{-a(\mathbf{V}\mathbf{e}^{\mathsf{T}})}.$$

Substituindo $a = \dfrac{1}{\lambda - 2\mathrm{E}}$ na expressaõ anterior, vem:

$$\mathbf{V}\mathbf{e}^{\mathsf{T}} = \frac{-(\lambda - 2\mathrm{E})}{\|\mathbf{u}\|}.$$

Agora, de substituir $\mathbf{V}\mathbf{e}^{\mathsf{T}}$ nas Equações (3.31) e (3.32), vêm:

$$\cos(\alpha) = \frac{(\mathbf{V}\mathbf{e}^{\mathsf{T}})^2 - (\lambda - 2\mathrm{E})^2}{(\mathbf{V}\mathbf{e}^{\mathsf{T}})^2 + (\lambda - 2\mathrm{E})^2} = \frac{1 - \|\mathbf{u}\|^2}{1 + \|\mathbf{u}\|^2} \quad \text{e}$$

$$\sin(\alpha) = \frac{2(\mathbf{V}\mathbf{e}^{\mathsf{T}})(\lambda - 2\mathrm{E})}{(\mathbf{V}\mathbf{e}^{\mathsf{T}})^2 + (\lambda - 2\mathrm{E})^2} = \frac{-2\|\mathbf{u}\|}{1 + \|\mathbf{u}\|^2},$$

que requerem poucas operações, considerando que $\|\mathbf{u}\|^2$ e $\|\mathbf{u}\|$ já foram computados na determinação de **e**. Estas equações também verificam que os valores obtidos para $\sin(\alpha)$ e $\cos(\alpha)$ estão dentro do intervalo $[-1, 1]$.

Finalmente, se as expressões

$$\mathbf{e}^{\mathsf{T}} = \frac{\mathbf{u}^{\mathsf{T}}}{\|\mathbf{u}\|}, \quad \cos(\alpha) = \frac{1 - \|\mathbf{u}\|^2}{1 + \|\mathbf{u}\|^2}, \quad \sin(\alpha) = \frac{-2\|\mathbf{u}\|}{1 + \|\mathbf{u}\|^2},$$

são introduzidas na fórmula da rotação definida pela Equação (3.13), encontra-se que a rotação ótima aplicada ao vetor **v** é dada por:

$$\mathrm{R}(\mathbf{v}) = \mathbf{v} - \frac{2((\mathbf{u} \times \mathbf{v}) \times \mathbf{u} - (\mathbf{u} \times \mathbf{v}))}{1 + \|\mathbf{u}\|^2}, \tag{3.33}$$

a qual só depende de $\mathbf{u}$.

Observe que até o ponto onde o vetor $\mathbf{u}$ é computado, a proposta é computacionalmente equivalente à técnica baseada em quatérnios de Kanatani [Kan94]. No entanto, em nosso caso, tudo o que resta fazer é aplicar diretamente a fórmula (3.33), enquanto que para efetuar uma transformação baseada em quatérnios mais operações são necessárias.

Entretanto, observe que a rotação obtida é exatamente $\mathrm{M}(\mathrm{M}^{\mathsf{T}}\mathrm{M})^{1/2}$ se $\det(\mathrm{M}) >= 0$. Caso contrário é $\mathrm{M}(\mathrm{M}^{\mathsf{T}}\mathrm{M})^{1/2}(\mathrm{I} - \mathbf{e}_{min}\mathbf{e}_{min}^{\mathsf{T}})$, onde $\mathbf{e}_{min}$ é um autovetor de $\mathrm{M}^{\mathsf{T}}\mathrm{M}$ associado ao autovalor μ_{min}. Essas expressões indicam que a rotação é uma função contínua de M enquanto $\det(\mathrm{M}) > 0$. Uma descontinuidade pode ocorrer se M é singular ou $\det(\mathrm{M}) < 0$ e μ_{min} é um autovalor múltiplo de $\mathrm{M}^{\mathsf{T}}\mathrm{M}$. Em pontos de descontinuidade, é observado um efeito de *twisting* quando aplicada uma operação de deformação num modelo. No entanto, é importante notar que qualquer método para resolver (3.17) exibirá este comportamento.

3.3 Sumário do método

A deformação de um modelo é controlada pela manipulação de pontos de controle (*Control Points*-CP) e segue os seguintes passos: primeiro, o usuário define posições iniciais para o conjunto CP, as quais podem estar em qualquer posição do espaço. Normalmente, estas são especificadas sobre a superfície do modelo que se quer deformar. Assim, quando o usuário arrasta elementos de CP com o mouse, a superfície do modelo é deformada de maneira suave e controlada. A influência que um ponto de controle exerce num vértice depende da proximidade entre eles, vértices próximos a um ponto de controle acompanham o movimento do mesmo.

Uma sessão de deformação recebe como entrada um modelo 3D na forma de uma malha de polígonos $\mathcal{M}_o$, composta de n vértices, além de um conjunto de pontos de controle com posições $\{\mathbf{p}_i\}$, onde $\mathbf{p}_i \in \Re^3$ e $i = 0 \ldots k-1$. Cada vez que o usuário especifica novas posições $\{\mathbf{q}_i\}$, correspondentes às posições iniciais $\{\mathbf{p}_i\}$, acontecem duas etapas:

1. **Inicialização**: onde podem ser pré-computados uma série de valores (por exemplo, os centróides ponderados correspondentes as posições iniciais dos pontos de controle).

2. **Manipulação interativa**: na qual – após o usuário especificar as novas posições $\{\mathbf{q}_i\}$ – o Algoritmo 2 é invocado para compor o modelo deformado $\mathcal{M}_d$.

No Algoritmo 2, a rotina `WeightedCentroid` computa os centróides ponderados $\mathbf{p}*$ e $\mathbf{q}*$ relativos ao vértice sendo processado, utilizando a Equação (3.10).

Algoritmo 2: Computar a posição deformada de um vértice

Input: $\mathbf{v}$: posição do vértice,

$\{\mathbf{p}_i\}$: posições iniciais dos pontos de controle,

$\{\mathbf{q}_i\}$: novas posições dos pontos de controle

Output: $\mathbf{v}'$: posição deformada do vértice

```
1 p* ← WeightedCentroid(v, {p_i});
2 q* ← WeightedCentroid(v, {q_i});
3 M ← CorrelationMatrix(v, {p_i}, {q_i}, p*, q*);
4 λ ← MaximumRootOfP(M) + Trace(M);                    // autovalor
5 u ← RotationAxis(M, λ);
6 v' ← q* + Rotate(v − p*, u);
7 return v';
```

A rotina `CorrelationMatrix` implementa a Equação (3.15) para computar $\mathrm{M} = \sum_{i=0}^{k-1} w_i \hat{\mathrm{q}}_i^{\mathsf{T}} \hat{\mathrm{p}}_i$ e, da mesma forma que a rotina anterior, depende do número de pontos de controle k. Embora a rotina `MaximumRootOfP` possa computar a maior raiz real a partir das equações (3.25) ou (3.27), optou-se por elaborar uma formulação mais direta que segue o raciocínio de Lodovico Ferrari para resolver equações de quarto grau do tipo $y^4 + ay^2 + by + c = 0$. Assim, ao invés de resolver a Equação quártica (3.27) para obter a maior raiz real, ela é obtida resolvendo a Equação cúbica (3.28), segundo o processo descrito na Seção 3.2.3. Em seguida, a rotina `RotationAxis` é responsável por computar o vetor de rotação $\mathbf{u}$ resolvendo o sistema 3×3 da Equação (3.30). Finalmente, a posição deformada $\mathbf{v}'$ do vértice $\mathbf{v}$ é computada pela rotina `Rotate`, a qual está definida pela Equação (3.33).

3.4 Resultados e discussões

Com o intuito de verificar a validade do Algoritmo 2, foi implementado um protótipo escrito em C++ onde a API do OpenGL foi usada para efeitos de visualização. O protótipo fornece uma interface na qual o usuário modifica as posições dos pontos de controle arrastando-os com o mouse. Algumas otimizações foram realizadas com a finalidade de melhorar o desempenho do algoritmo, por exemplo na etapa de inicialização, os valores

Modelo	Vértices	# de CP	FPS
Dolphin	2811	4	125
Dragon	16096	6	37
Girl	9912	5	105
Horse	19851	9	9

Tabela 3.1: Quadros por segundo (FPS) para a deformação dos modelos da Figura 3.3. A coluna # de CP refere-se ao número de pontos de controle.

para w, $\mathbf{v} - \mathbf{p}*$, e $\hat{\mathrm{p}}_i$ foram devidamente pré-computados.

Na Figura 3.3 são mostrados resultados do algoritmo de deformação, aplicando-o a quatro modelos diferentes. Na parte superior, um golfinho é deformado usando 4 pontos de controle. Depois, a boca do Dragão é aberta e fechada usando 6 pontos de controle. A seguir, um modelo de garota é dobrado usando 5 pontos de controle. Na parte inferior são mostradas duas poses de um cavalo, as quais foram obtidas usando 9 (esquerda) e 8 (direita) pontos de controle. Exceto na deformação do cavalo, as poses foram obtidas em sessões interativas de deformação. Na Tabela 3.1 podem ser vistos os quadros por segundo (*Frames per Second* - FPS) gastos na deformação destes modelos. As normais dos vértices foram modificadas fazendo uso do mesmo vetor de rotação $\mathbf{u}$ computado para modificar a posição do vértice. Um outro grupo de exemplos é mostrado na Figura 3.4. Neste caso, diferente das deformações da Figura 3.3, os pontos de controle não foram especificados sobre a superfície dos modelos e sim no espaço ao redor deles.

As deformações foram conduzidas num PC equipado com um processador Pentium-IV *dual-core* rodando a 2.4 GHz, 1GB de memória RAM e uma placa gráfica GeForce 8800GTS. Foi utilizada a representação ponto flutuante de precisão simples para realizar todas as operações aritméticas. Estas configurações foram utilizadas para executar todos os experimentos desta tese.

Conforme notado por Botsch et al. [BPGK06], técnicas de deformação podem produzir resultados não intuitivos quando os pontos de controle são sujeitos a grandes deslocamentos. Similarmente, alguns resultados não intuitivos podem ser obtidos com a técnica proposta, conforme ilustrado na Figura 3.5. Uma observação que oferece uma explicação para este comportamento é que os centróides ponderados $\mathbf{p}*_i$ e $\mathbf{q}*_i$ são, de fato, combinações convexas das posições dos pontos de controle $\{\mathbf{p}_i\}$ e $\{\mathbf{q}_i\}$. Uma tenta-

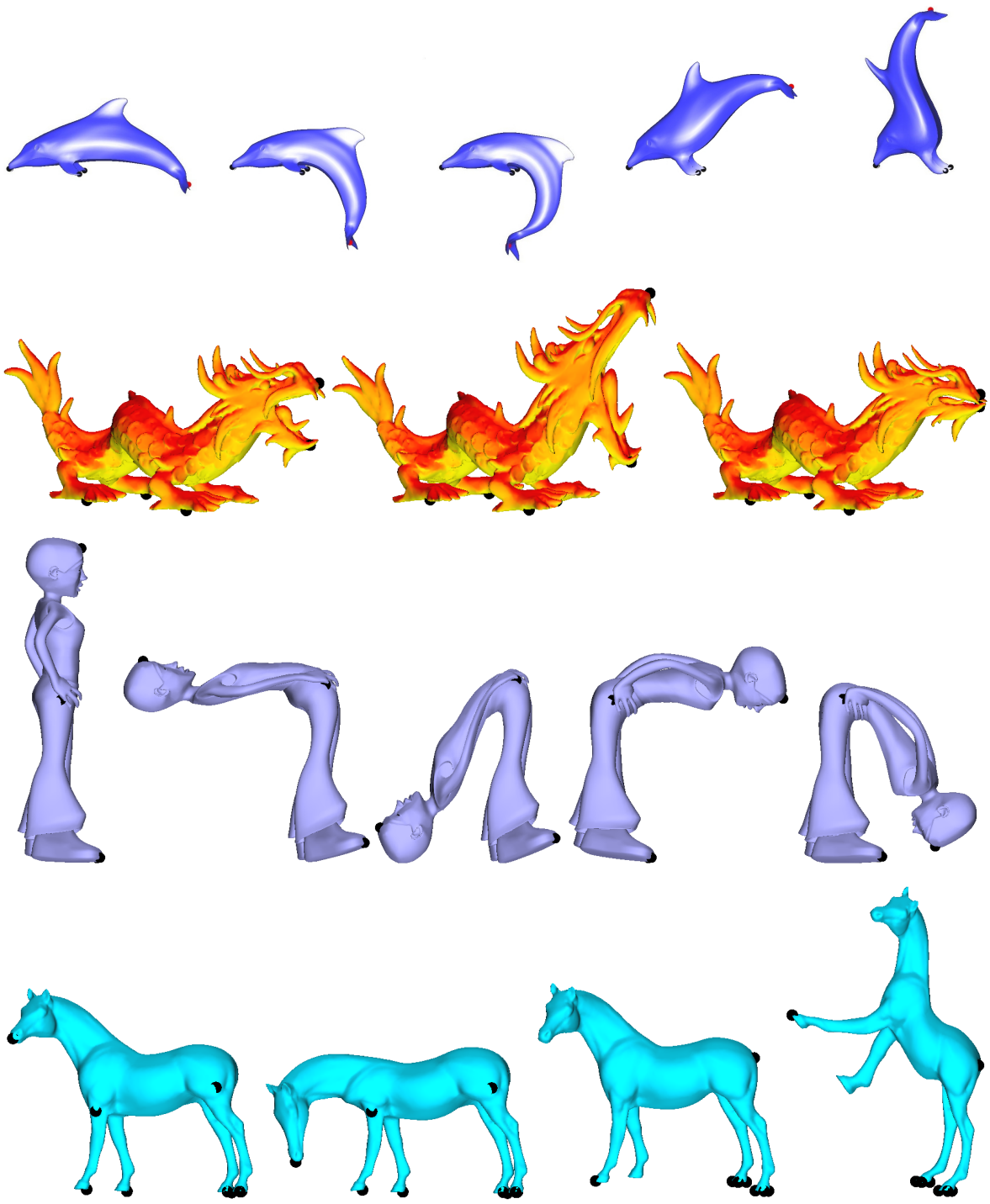
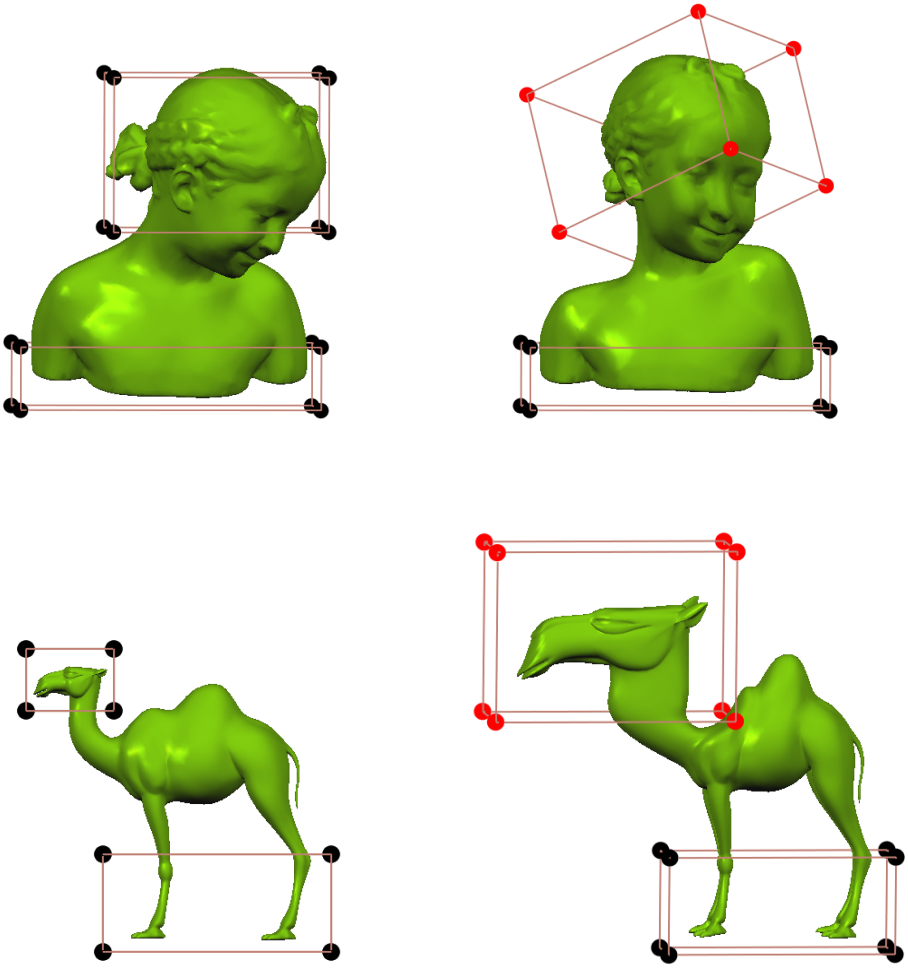


Figura 3.3: Resultados obtidos aplicando o algoritmo de deformação baseado em MLS. Os pontos pretos sobre os modelos representam os pontos de controle.



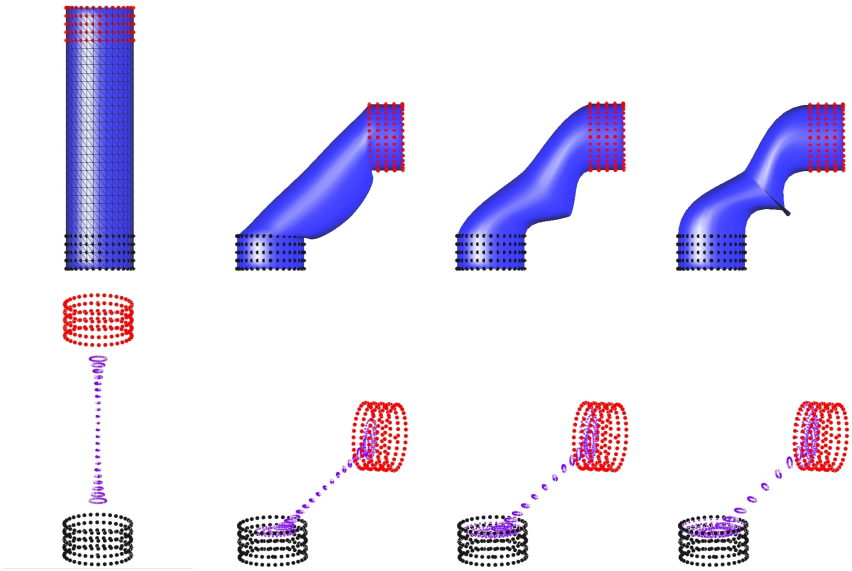
(a) (b)

Figura 3.4: Resultados obtidos usando pontos de controle especificados no espaço ao redor dos modelos. (a) Configurações iniciais. Deformações obtidas produto do rotacionamento e escalamento dos pontos de controle em vermelho.

tiva errada para obter distribuições curvas dos pontos de controle é modificar o expoente da função de peso w. Mas, em geral, esta abordagem não produz resultados satisfatórios (veja a Figura 3.5).

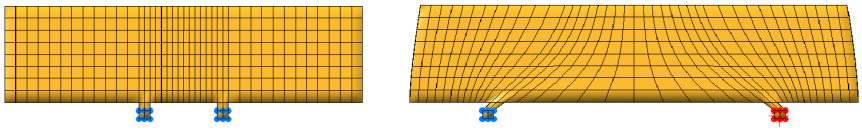
Outro efeito interessante pode ser observado quando os pontos de controle são sujeitos somente à aplicação de translações (veja a Figura 3.6). Note que, em particular, retas paralelas ao movimento dos pontos de controle não são distorcidas.

Por outro lado, nota-se que a deformação MLS se comporta razoavelmente bem, frente a translações amplas dos pontos de controle, quando comparada com métodos baseados em RBFs (*Radial Basis Functions*) ou em coordenadas Laplacianas. Isso acontece porque transformações de corpos rígidos tendem a manter a distância entre dois pontos



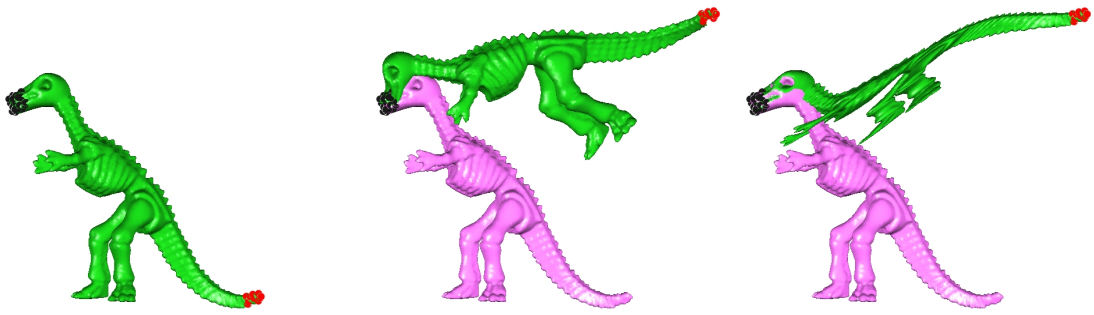
(a) (b) (c) (d)

Figura 3.5: Deformação não intuitiva de um cilindro: (a) Modelo inicial e pontos de controle. Deformações induzidas rotacionando um conjunto de pontos de controle posicionados na parte superior do cilindro e usando diferentes funções de peso: (b) $w(d) = d^{-2}$, (c) $w(d) = d^{-3}$ e (d) $w(d) = d^{-4}$. A parte inferior da figura mostra as posições dos centróides $\mathbf{q}_i^*$.



(a) (b)

Figura 3.6: Efeito de deformação causada pela translação dos pontos de controle. (a) Modelo inicial e pontos de controle. (b) Modelo deformado.



(a) (b) (c)

Figura 3.7: Deformações com pontos de controle deslocados amplamente. (a) Modelo inicial e pontos de controle. (b) Deformação MLS. (c) Deformação RBF.

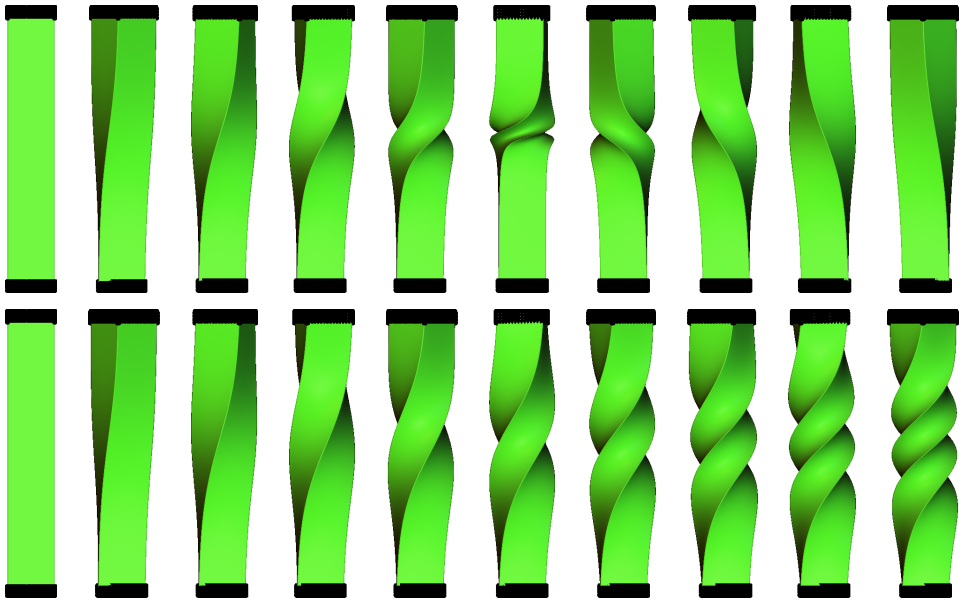


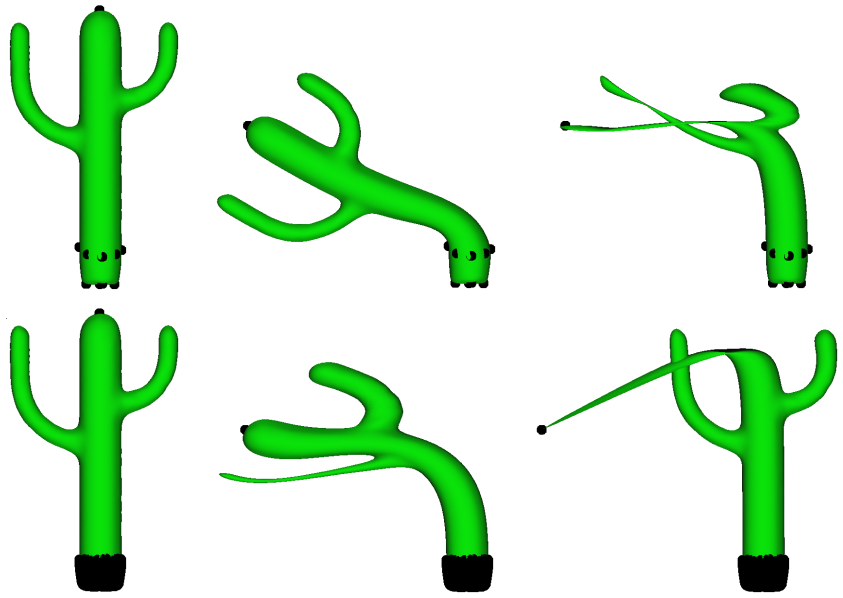
Figura 3.8: *Twisting* de um paralelepípedo. Na parte superior, as "torções" são computadas usando o mesmo modelo de referência $\mathcal{M}_o$. Na parte inferior, o efeito é obtido efetuando múltiplas torções de 35^o sucessivamente. Observe que, neste caso, o modelo de referência para a k-ésima torção $\mathcal{M}_d$ é o resultado da $(k-1)$-ésima torção.

quaisquer após aplicada a transformação. Isso pode ser observado quando comparamos a Figura 3.7(b) com a Figura 3.7(c) e com a Figura 2.6.

Observe que, no Algoritmo 2, não são usados resultados prévios como novas configurações iniciais, isto é, vértices do mesmo modelo de referência $\mathcal{M}_o$ são usados para computar diferentes instâncias deformadas $\mathcal{M}_d$. Como pode ser visto na Figura 3.8, diferentes resultados são obtidos quando $\mathcal{M}_o$ é atualizada constantemente.

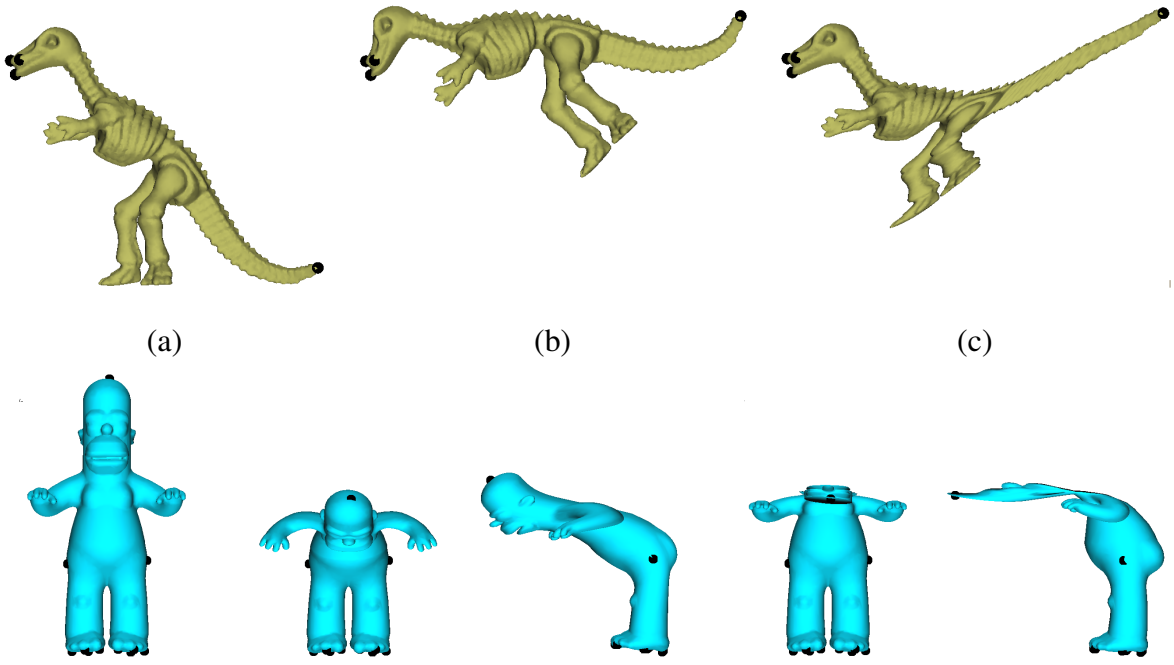
Uma outra observação sugere que resultados não intuitivos podem ser obtidos se os pontos de controle forem especificados arbitrariamente (tanto em quantidade quanto em posicionamento). Considere, por exemplo, a deformação da "garota" ilustrada na Figura 3.3, onde é possível perceber que o resultado desejado é que o modelo se dobre pela cintura. No entanto, para que isso aconteça, o posicionamento de dois pontos de controle sobre os quadris foi crucial. Por outro lado, a Figura 3.9(b) ilustra como pontos de controle colocados em posições semelhantes, mas não na mesma quantidade, produzem deformação diferentes. Isso pode ser explicado pela forte influência da grande quantidade de pontos de controle posicionados na base do cactus, frente ao único ponto no topo.

Outra observação refere-se à influência da componente rotacional no processo de deformação. Nas figuras 3.9(c) e 3.10 pode ser notado que, quando a componente de



(a) (b) (c)

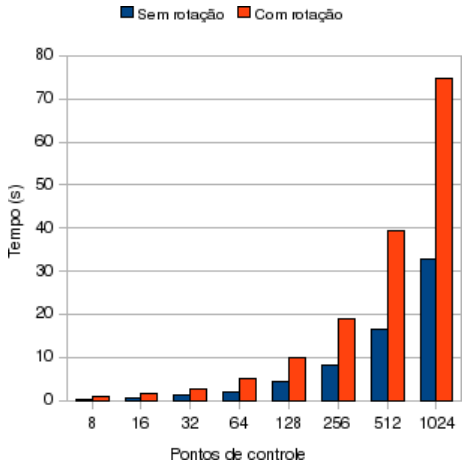
Figura 3.9: Ilustração das diferentes deformações obtidas frente a quantidades variáveis de pontos de controle (a) e (b). Por outro lado, para as mesmas configurações iniciais, em (c) são mostradas deformações que não computam a componente rotacional.



(a) (b) (c)

(a) (b) (c)

Figura 3.10: Ilustração da influência da componente rotacional no algoritmo de deformação. Em (a) modelos iniciais e pontos de controle, (b) deformações com componente rotacional e (c) deformações sem componente rotacional.



Pontos de controle	Sem rotação	Com rotação
8	0.3242	1.0170
16	0.5758	1.6076
32	1.0849	2.7504
64	2.0967	5.0755
128	4.2118	9.7790
256	8.2590	19.1054
512	16.3884	39.4772
1024	32.7863	74.9341

Figura 3.11: Tempos (em segundos) gastos para computar deformações sem e com rotações ótimas. Foram processados 1024×1024 vértices com um número variável de pontos de controle, ambos gerados aleatoriamente.

rotação está ausente (rotação nula), deformações não intuitivas são obtidas. Adicionalmente, é possível fazer uma análise da relação entre o tempo gasto pela computação da componente de translação e da componente rotacional, pois o conjunto de expressões usadas para computar a posição deformada de um vértice pode ser dividido em duas partes: expressões dependentes do número de pontos de controle e expressões com número fixo de operações que independem deles. O cômputo da componente de translação, isto é, a posição dos centróides, depende inteiramente do número de pontos de controle. A componente de rotação é computada usando os centróides, a matriz de correlação e um número constante de operações aritméticas. A Figura 3.11 mostra um gráfico que ilustra a diferença dos tempos gastos por cada etapa. Pode-se observar que, quando o número de CP é pequeno, o cômputo da componente rotacional gasta mais de 70% do tempo de processamento. Embora o número de CP seja maior, o tempo gasto pela computação da rotação ótima não chega a ser desprezível; note que nos experimentos realizados o tempo gasto é superior a 50% do tempo total de processamento. Isto mostra a importância de computar a rotação ótima da maneira mais eficiente possível.

Além do método descrito, existem outros métodos para computar a componente de rotação (uma descrição mais detalhada destes métodos pode ser encontrada nos artigos de Kanatani [Kan94] e Eggert et al. [ELF97]). Como este trabalho de pesquisa centra-se em apresentar uma formulação eficiente para encontrar a componente rotacional em 3D, foram realizados alguns experimentos que comparam o desempenho do algoritmo

proposto (*Axis-Angle* - AA) contra técnicas baseadas em:

1. Matrizes ortonormais (*Orthonormal Matrices* - OM). Nesta formulação a matriz de rotação é computada como:

$$R = \mathrm{M}(\mathrm{M}^\mathsf{T}\mathrm{M})^{-1/2} = \mathrm{M}\mathrm{A}^\mathsf{T} \begin{pmatrix} 1/\sqrt{\lambda_1} & 0 & 0 \\ 0 & 1/\sqrt{\lambda_2} & 0 \\ 0 & 0 & 1/\sqrt{\lambda_3} \end{pmatrix} \mathrm{A},$$

   onde as linhas da matriz A são formadas pelos autovetores de $\mathrm{M}^\mathsf{T}\mathrm{M}$ e as variáveis λ_1, λ_2 e λ_3 denotam os autovalores de $\mathrm{M}^\mathsf{T}\mathrm{M}$.

2. Quatérnios unitários (*Unitary Quaternions* - UQ). A matriz de rotação é computada como:

$$R = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_2q_1 + q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_3q_1 - q_0q_2) & 2(q_3q_2 + q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{pmatrix},$$

   onde $q = \{q_0\ q_1\ q_2\ q_3\}$ é o autovetor correspondente ao maior autovalor da matriz

$$A = \begin{pmatrix} \text{M[0][0]+M[1][1]+M[2][2]} & \text{M[2][1]-M[1][2]} & \text{M[0][2]-M[2][0]} & \text{M[1][0]-M[0][1]} \\ \text{M[2][1]-M[1][2]} & \text{M[0][0]-M[1][1]-M[2][2]} & \text{M[0][1]+M[1][0]} & \text{M[2][0]+M[0][2]} \\ \text{M[0][2]-M[2][0]} & \text{M[0][1]+M[1][0]} & \text{-M[0][0]+M[1][1]-M[2][2]} & \text{M[1][2]+M[2][1]} \\ \text{M[1][0]-M[0][1]} & \text{M[2][0]+M[0][2]} & \text{M[1][2]+M[2][1]} & \text{-M[0][0]-M[1][1]+M[2][2]} \end{pmatrix}.$$

3. Decomposição em valores singulares (*Singular Value Decomposition* - SVD). A matriz de rotação é computada como:

$$R = UV^\mathsf{T},$$

   onde U e V são matrizes da decomposição em valores singulares de $\mathrm{M} = U\Lambda V^\mathsf{T}$.

A Tabela 3.2 mostra os resultados destas comparações. Para os experimentos realizados foram utilizadas 1024×1024 posições e $\{4, 40$ e $400\}$ pontos de controle. Para resolver os sistemas de autovalores (para OM e UQ) e a decomposição SVD – e para efeitos de comparação – foram utilizada duas bibliotecas: LAPACK [LAP90] e *Numerical Recipes* [NR92], as quais são conhecidas por implementar seus algoritmos de forma eficiente e robusta. Como pode ser observado nos gráficos da Figura 3.12, a técnica

			LAPACK			NR		
Vértices	# de CP	AA	OM	UQ	SVD	OM	UQ	SVD
10K	4	0.01	0.13	0.17	0.11	0.03	0.04	0.02
20K	4	0.02	0.29	0.38	0.25	0.06	0.09	0.05
50K	4	0.04	0.66	0.85	0.56	0.13	0.20	0.13
100K	4	0.07	1.27	1.66	1.11	0.28	0.41	0.27

			LAPACK			NR		
Vértices	# de CP	AA	OM	UQ	SVD	OM	UQ	SVD
10K	40	0.03	0.15	0.19	0.14	0.05	0.06	0.05
20K	40	0.07	0.35	0.43	0.31	0.11	0.15	0.11
50K	40	0.16	0.78	0.96	0.70	0.25	0.32	0.26
100K	40	0.33	1.57	1.98	1.41	0.52	0.66	0.52

			LAPACK			NR		
Vértices	# de CP	AA	OM	UQ	SVD	OM	UQ	SVD
10K	400	0.28	0.41	0.44	0.39	0.30	0.31	0.30
20K	400	0.63	0.91	1.00	0.88	0.68	0.71	0.68
50K	400	1.43	2.05	2.23	1.97	1.52	1.60	1.53
100K	400	2.89	4.14	4.50	3.97	3.08	3.22	3.08

Tabela 3.2: Tempos de deformação (em segundos) de uma quantidade variável de posições usando 4, 40 e 400 pontos de controle (# de CP). Pode-se observar que, em todos os experimentos, o tempo consumido pelo algoritmo proposto (AA) é bem menor do que o gasto pelas outras técnicas.

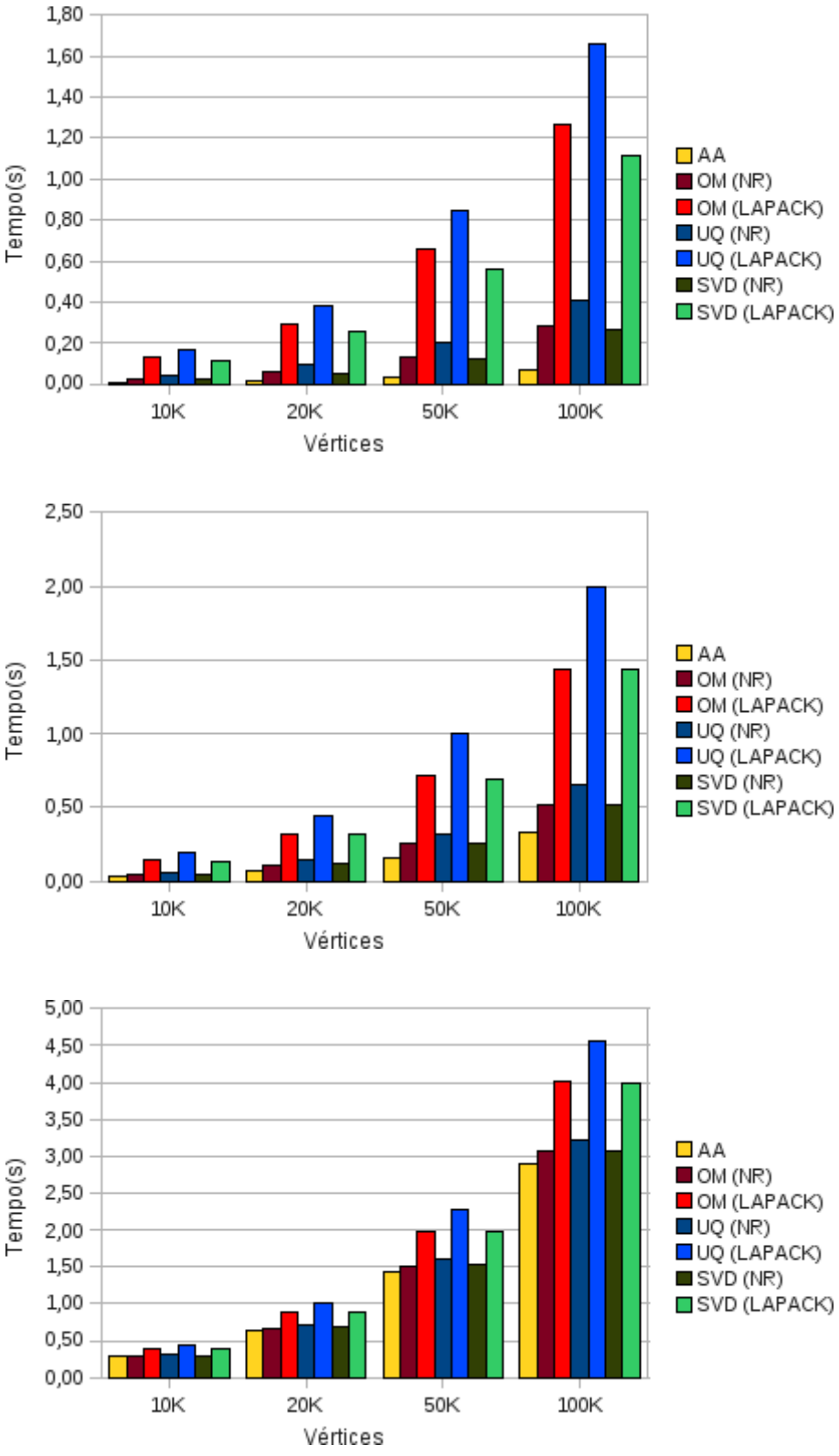


Figura 3.12: Gráficos correspondentes aos tempos de deformação da Tabela 3.2.

proposta apresenta um melhor desempenho do que as outras técnicas, inclusive quando a complexidade computacional do problema é dominada pelos somatórios, os quais dependem do número de pontos de controle (observe o gráfico inferior da Figura 3.12). A razão principal deste comportamento é devido à formulação direta da proposta, pois requer menos operações do que as outras técnicas. Outra explicação baseia-se no fato que as bibliotecas usadas implementam algoritmos genéricos que podem estar baseados em abordagens iterativas.

Um outro grupo de comparações refere-se aos resultados obtidos pela técnica proposta frente a deformações conduzidas por RBFs [BK05a] e as baseadas na minimização de energía variacional [BK04]. Três modelos foram deformados usando estas técnicas sob a mesma configuração de pontos de controle (a Figura 3.13 mostra os resultados obtidos). Pode-se observar que nestes experimentos o algoritmo proposto oferece resultados mais intuitivos do que as outras técnicas.

Finalmente, no artigo do Botsch e Sorkine [BS08] é efetuada uma comparação de algoritmos de deformação recentes, frente a uma massa de dados comum, com o intuito de mostrar sob quais circunstâncias os algoritmos produzem deformações não intuitivas. O resultado desta comparação é mostrado na Figura 2.8. Da mesma forma, os mesmos modelos utilizados nesse trabalho foram deformados com o algoritmo proposto (Algoritmo 2). Os resultados são mostrados na Figura 3.14. Observa-se que, para as configurações estabelecidas, as deformações produzidas não são intuitivas e a qualidade é inferior das obtidas com técnicas não-lineares (compare a Figura 3.14 com a Figura 2.8 retirada do artigo de Botsch e Sorkine [BS08]). Isto pode ser explicado pelo fato da técnica proposta estar definida como um problema de otimização sujeito a uma única restrição (restrição posicional) e pelo fato das transformações rígidas serem funções lineares.

3.5 Deformação usando transformações de similaridade

Se ao invés de usar transformações rígidas requer-se uma deformação computada usando transformações de similaridade, então um fator de escala uniforme $\mu_s \in \Re$ deve ser introduzido no problema de otimização. Dessa forma, a Equação (3.11) passa a ser

$$\min_{R \in SO(3)} \sum_i w_i \|\hat{\mathrm{q}}_i - \hat{\mathrm{p}}_i \mathrm{R} \mu_s\|^2,$$

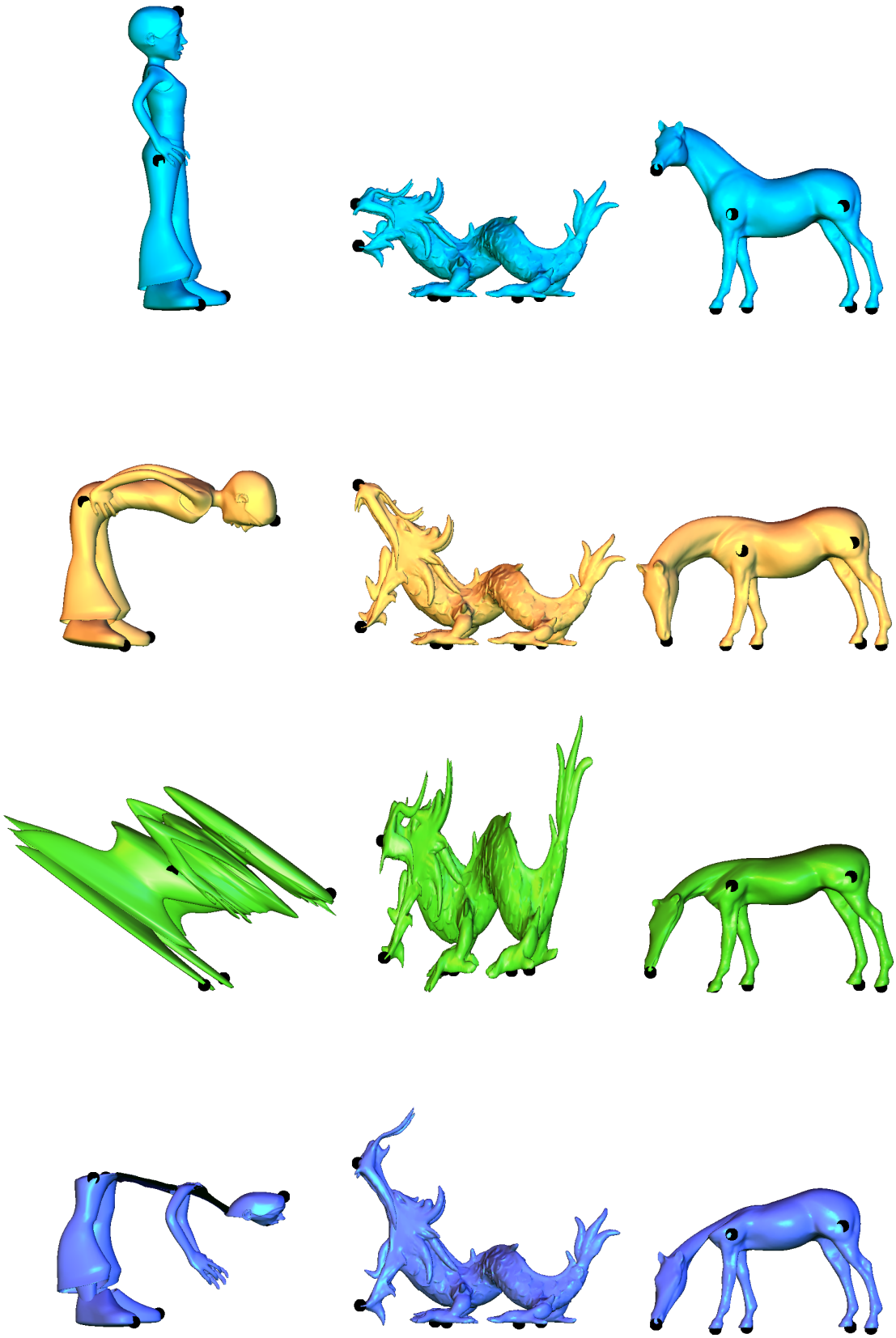


Figura 3.13: Comparação de técnicas de deformação. Na primeira linha, modelos iniciais e pontos de controle. Na seqüência, deformações baseadas em MLS (segunda linha), RBF (terceira linha) e em minimização de energía variacional (última linha).

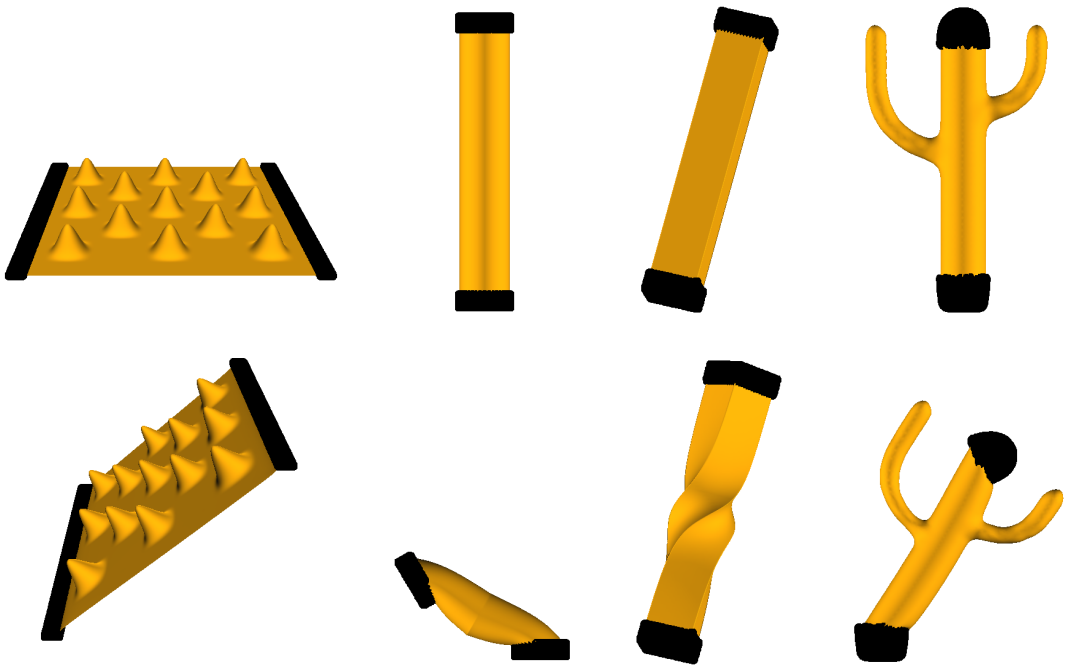


Figura 3.14: Resultados da deformação MLS nos mesmos modelos efetuados por Botsch e Sorkine [BS08]. Na parte superior tem-se modelos inicias e pontos de controle. As deformações são mostradas na parte inferior.

e para otimizá-la deve-se resolver

$$\max \quad \mu_s \sum_i w_i \hat{\mathrm{q}}_i \mathrm{R} \hat{\mathrm{p}}_i^{\mathsf{T}} - \mu_s^2 \sum_i w_i \|\hat{\mathrm{p}}_i\|^2.$$

Este problema obedece as mesmas condições de otimalidade de (3.17). Adicionalmente, surge uma condição relacionada com μ_s, a qual é dada por

$$\sum_i w_i \hat{\mathrm{q}}_i \mathrm{R} \hat{\mathrm{p}}_i^{\mathsf{T}} - \mu_s \sum_i w_i \|\hat{\mathrm{p}}_i\|^2 = 0.$$

Posto que as condições de otimalidade de (3.17) originam que a solução ótima satisfaça $\sum_i w_i \hat{\mathrm{q}}_i \mathrm{R} \hat{\mathrm{p}}_i^{\mathsf{T}} = y$, pode-se concluir que

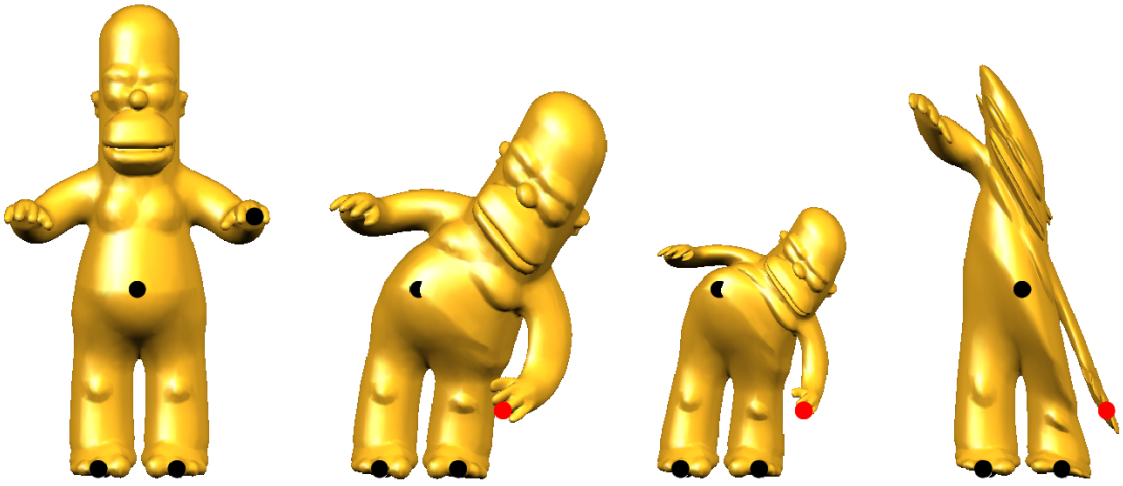
$$\mu_s = \frac{y}{\sum_i w_i \|\hat{\mathrm{p}}_i\|^2},$$

onde y é a maior raiz real da Equação (3.27). Com isso, o passo 6 do Algoritmo 2, que era

$$\mathbf{v}' \leftarrow \mathbf{q}* \ + \ \texttt{Rotate}(\mathbf{v} - \mathbf{p}*, \mathbf{u}),$$

passa a ser:

$$\mathbf{v}' \leftarrow \mathbf{q}* \ + \ \mu_s \ \texttt{Rotate}(\mathbf{v} - \mathbf{p}*, \mathbf{u}).$$



(a) (b) (c) (d)

Figura 3.15: Deformações MLS usando diferentes funções lineares. (a) Modelo original e pontos de controle. Deformação usando transformações rígidas (b), de similaridade (c), e transformações afins (d).

A Figura 3.15(c) mostra o resultado de deformar um modelo usando transformações de similaridade. Quando comparado com a deformação rígida (Figura 3.15(b)), pode-se notar como o fator de escala uniforme μ_s influencia na obtenção de um resultado menos intuitivo.

3.6 Deformação usando transformações afins

Se é preciso que a deformação seja computada usando transformações afins, então a restrição que define R como uma matriz de rotação em $l(\mathrm{x}) = \mathrm{xR} + \mathrm{T}$ (veja a Seção 3.2) não é mais necessária. Dessa forma, a solução ótima para o problema da Equação (3.11) passa a ser a solução do problema

$$\min \sum_i w_i \|\hat{\mathrm{q}}_i - \hat{\mathrm{p}}_i \mathrm{L}\|^2,$$

onde L é uma matriz 3×3. Assim, L é uma solução ótima quando:

$$\begin{aligned}
\frac{\partial \sum_i w_i \|\hat{\mathrm{q}}_i - \hat{\mathrm{p}}_i \mathrm{L}\|^2}{\partial L} &= \frac{\partial \sum_i w_i (\hat{\mathrm{q}}_i - \hat{\mathrm{p}}_i \mathrm{L})(\hat{\mathrm{q}}_i - \hat{\mathrm{p}}_i \mathrm{L})^\mathsf{T}}{\partial L} = 0 \\
&= \frac{\partial \sum_i w_i (\hat{\mathrm{q}}_i \hat{\mathrm{q}}_i^\mathsf{T} - \hat{\mathrm{q}}_i (\hat{\mathrm{p}}_i \mathrm{L})^\mathsf{T} - \hat{\mathrm{p}}_i \mathrm{L} \hat{\mathrm{q}}_i^\mathsf{T} + \hat{\mathrm{p}}_i \mathrm{L} (\hat{\mathrm{p}}_i \mathrm{L})^\mathsf{T})}{\partial L} = 0 \\
&= \sum_i w_i \, \hat{\mathrm{p}}_i^\mathsf{T} \hat{\mathrm{p}}_i \mathrm{L} - \sum_j w_j \, \hat{\mathrm{q}}_j^\mathsf{T} \hat{\mathrm{p}}_j = 0.
\end{aligned}$$

Portanto,

$$\mathrm{L} = (\sum_i w_i \, \hat{\mathrm{p}}_i^{\mathsf{T}} \hat{\mathrm{p}}_i)^{-1} \sum_j w_j \, \hat{\mathrm{q}}_j^{\mathsf{T}} \hat{\mathrm{p}}_j.$$

Finalmente, a deformação afim v' do vértice v é dada pela expressão:

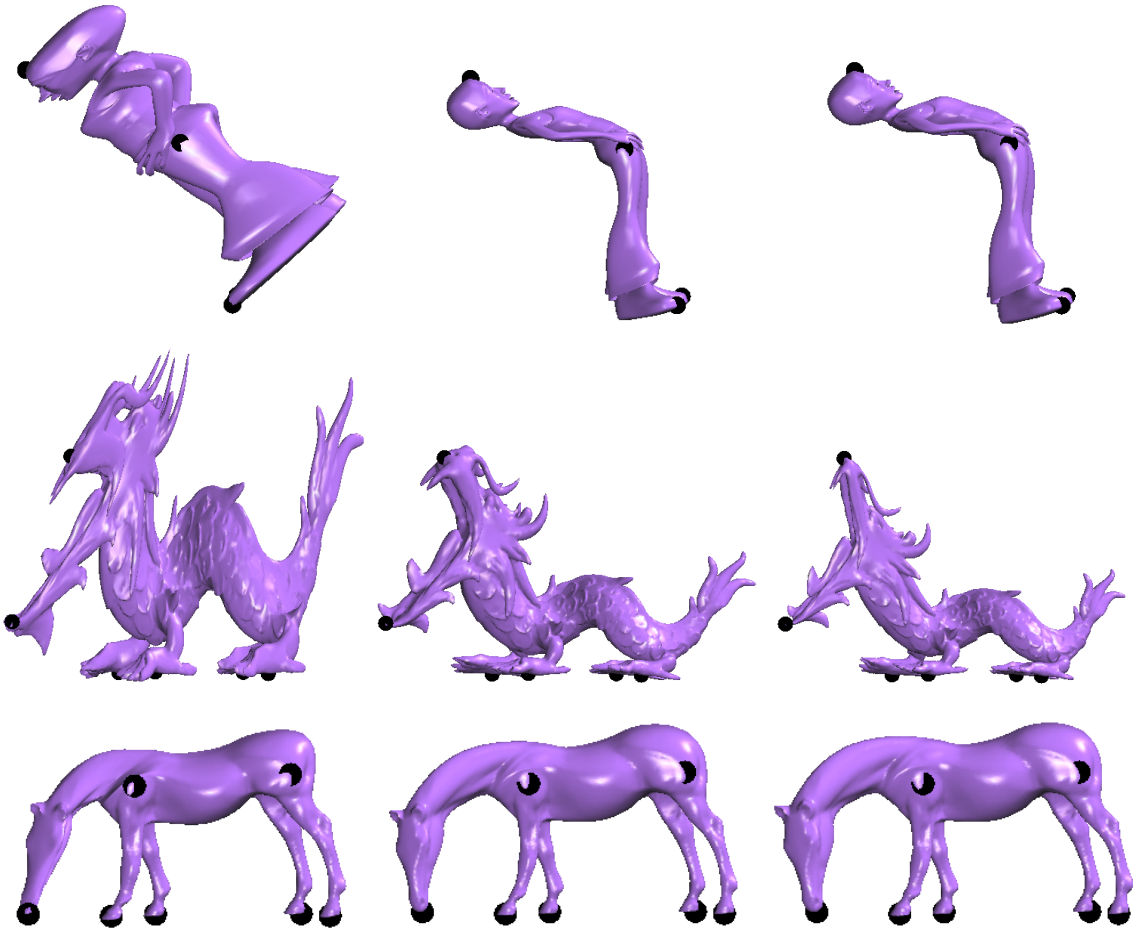
$$\mathrm{v}' = (\mathrm{v} - \mathrm{p}^*)(\sum_i w_i \, \hat{\mathrm{p}}_i^{\mathsf{T}} \hat{\mathrm{p}}_i)^{-1} \sum_j w_j \, \hat{\mathrm{q}}_j^{\mathsf{T}} \hat{\mathrm{p}}_j + \mathrm{q}^*.$$

Conforme visto na Figura 3.15(d), os resultados obtidos usando transformações afins são menos intuitivos do que os obtidos com transformações rígidas e de similaridade. O fato de L não estar sujeita a nenhuma restrição resulta em achatamentos e escalamentos não uniformes, tais como podem ser observados no braço esquerdo do modelo. Deformações adicionais são apresentadas na Figura 3.16. Embora as deformações obtidas usando transformações de similaridade (Figura 3.16(b)) sejam semelhantes com as obtidas usando transformações rígidas (Figura 3.16(c)), em muitas situações isso não é verdade. Veja, por exemplo, artefatos relativos com a escala nos modelos da Figura 3.17.

3.7 Implementação em paralelo

O principal fator que explica a necessidade do processamento paralelo é a busca por maior desempenho. As diversas áreas, nas quais a computação se aplica, requerem cada vez mais poder computacional devido, principalmente, ao tamanho crescente da massa de dados a ser processada. Além disso, várias aplicações são inerentemente paralelas, no entanto, perde-se desempenho pelo uso costumeiro de modelos tradicionais de programação seqüencial. Esta filosofia de programação, já firmemente estabelecida por décadas, não aproveita a potência do hardware moderno.

Nos últimos anos, a acessibilidade a arquiteturas de processamento paralelo de baixo custo tornou-se crescente. Por exemplo, entre aquelas com maior popularidade destacam-se: as GPUs (*Graphics Processing Units*), as CPUs de múltiplos núcleos de processamento (*multi-core CPUs*) e a arquitetura CELL (*STI Cell Broadband Engine*). Isso, origina a necessidade de novas maneiras de organização do processamento computacional. Uma descrição detalhada destas arquiteturas está fora do escopo desta tese. No entanto, no Apêndice A são brevemente descritos os conceitos usados no decorrer desta seção.

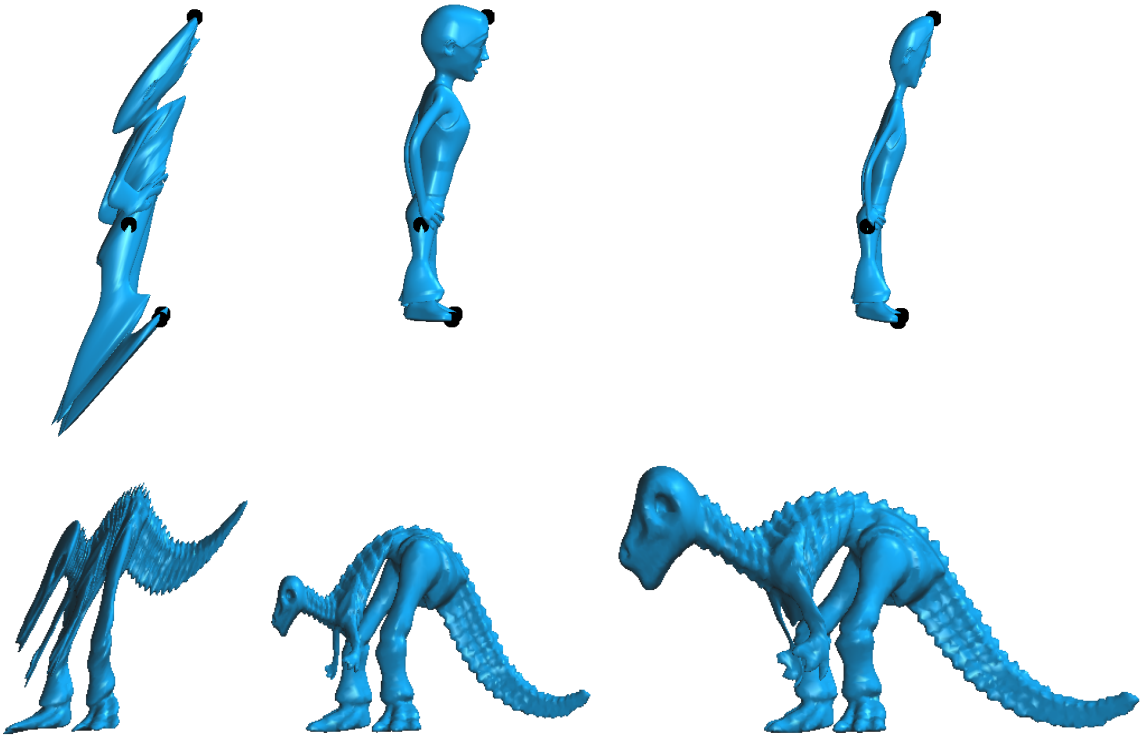


(a) (b) (c)

Figura 3.16: Resultados de deformações computadas com transformações afins (a), de similaridade (b) e rígidas (c).

3.7.1 MLS em GPU

Uma das motivações para o estudo de deformações conduzidas por MLS foi a procura de um método que pudesse ser implementado com o mínimo necessário de ferramentas e/ou bibliotecas externas. Isto é importante quando, por exemplo, deseja-se que o algoritmo seja portável em plataformas múltiplas ou implementado em hardware não convencional, por exemplo GPUs. Técnicas baseadas em MLS computam uma solução diferente para cada ponto do domínio e, em geral, esta solução não depende da vizinhança do ponto. Com isso é possível evitar construir sistemas lineares de grandes dimensões (como acontece com técnicas de deformação baseadas em otimização global [BS08]) e a necessidade de empregar bibliotecas para resolvê-los é minimizada. Para mostrar a adaptabil-



(a) (b) (c)

Figura 3.17: Ilustração do escalamento gerado pelo uso de transformações de similaridade (b), que não é observado no caso das transformações rígidas (c). Para as mesmas configurações, em (a) apresenta-se deformações obtidas usando transformações afins.

idade da deformação MLS no modelo de programação em GPUs foram efetuadas três implementações do Algoritmo 2, as quais são descritas a seguir.

Implementação no processador de vértices

Foi utilizada a biblioteca OpenGL e a linguagem GLSL (*OpenGL Shading Language*) para codificar o Algoritmo 2 num programa de vértices (*vertex shader*) K_{vs}. A estratégia consiste em fazer que a geometria da malha, enviada para a memória da placa gráfica, seja deformada antes de ser renderizada. Assim, após especificadas as posições dos pontos de controle ($\{\mathbf{p}_i\}$, $\{\mathbf{q}_i\}$) e a geometria do modelo, o *shader* de deformação K_{vs} computa, em cada *frame*, uma transformação ótima para cada vértice e modifica sua posição. O *shader* K_{vs} é invocado pela API gráfica para processar os vértices do modelo em forma paralela. A Figura 3.18 mostra um diagrama que ilustra esta idéia. Observe que, contrário ao caso da implementação em CPU (veja a seção anterior), nenhuma pré-computação foi feita devido à natureza do hardware fazer operações de leitura em memória de textura

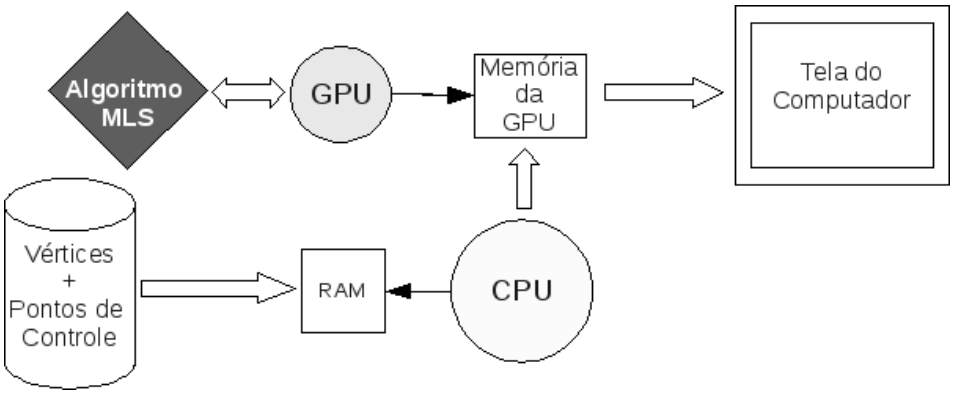


Figura 3.18: Diagrama que ilustra a implementação no *vertex shader*. O processador de vértices da GPU, que teve sua funcionalidade padrão alterada, modifica as posições dos vértices do modelo – aplicando o Algoritmo MLS – antes de renderizá-lo.

mais custosas do que o cômputo repetitivo de operações aritméticas.

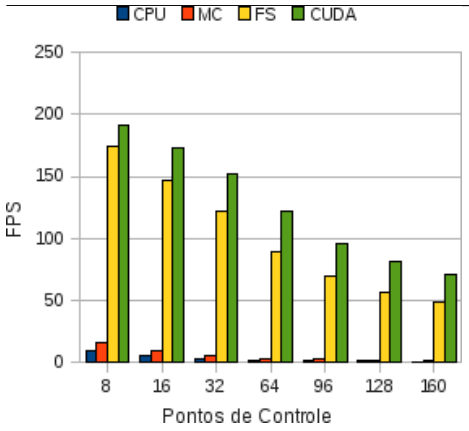
Com esta implementação foram conseguidas sessões de deformação interativas (> 20 FPS) para um modelo composto de 115,154 vértices. Na Tabela 3.19 são mostrados os tempos correspondentes à deformação do modelo com um número variável de pontos de controle. Pode-se observar que, no pior caso, o desempenho é 20 vezes superior do que a implementação tradicional em CPU. Nota-se também que, quanto mais pontos de controle são usados maior o ganho (97 vezes superior), uma vez que mais operações aritméticas são necessárias. É importante destacar que a implementação no processador de vértices não precisa de operações explícitas de transferência de dados entre a memória principal do sistema e a memória da placa gráfica, dado que isto é feito de forma transparente pela API gráfica.

Implementação no processador de fragmentos

Uma outra abordagem pode ser adotada para a implementação do algoritmo de deformação em GPU. A estratégia consiste em usar a GPU como um co-processador de propósito geral[2]. Em outras palavras deseja-se usar a GPU como uma caixa preta. Uma vez computadas as posições deformadas da malha, estas são reenviadas à memória principal do sistema para outros processamentos. Diferente da deformação no *vertex shader*, a renderização [3] dos vértices processados é opcional. Veja uma ilustração destas idéias na

[2]Esta estratégia é conhecida com o nome de GPGPU (*General-Purpose computation on GPUs*).

[3]A palavra renderização é um estrangeirismo derivado do verbo em inglês *render*, que significa desenhar.



# de CP	CPU	MC	VS	FS	CUDA
8	9.0	15.5	174.0	174.0	334.0
16	6.0	10.0	147.0	147.0	274.5
32	3.5	6.0	122.5	122.5	203.0
64	2.0	3.5	89.5	89.5	131.5
96	1.5	2.5	69.5	69.5	99.5
128	1.0	2.0	57.0	57.0	77.5
160	0.5	1.5	48.5	48.5	64.5

Figura 3.19: Comparação dos quadros por segundo da implementações tradicional (CPU) do Algoritmo 2, frente a implementações *multicore* (MC), processador de fragmentos (FS), processador de vértices (VS) e CUDA para a deformação de um modelo de 115,154 vértices com número variável de pontos de controle (# de CP).

Figura 3.20.

A implementação visa explorar a natureza paralela das GPUs, gerando uma série de fragmentos que são processados em paralelo e de forma independente. Nesta abordagem a memória de textura é usada explicitamente como repositório para a entrada e saída de dados.

Assim, cada vértice é associado a um fragmento e cada fragmento é processado pelo programa de fragmentos (*fragment shader*) K_{fs} , que computa a transformação ótima conforme o Algoritmo 2. A codificação de K_{fs} é similar ao código de K_{vs} com a diferença que o *shader* de fragmentos lê as posições dos vértices da memória de textura e escreve a posição deformada em outra textura que, posteriormente, é transferida para a memória principal do sistema. Embora exista o requerimento de transferência explícita das posições dos vértices entre a memória RAM e a memória da placa gráfica, foram obtidos desempenhos similares com os da implementação no processador de vértices e superiores quando comparados com a implementação em CPU descrita no início deste capítulo (observe a Tabela 3.19). Em placas NVidia© modernas (a partir da série 8) os recursos usados pelo processador de vértices e pelo processador de fragmentos são os mesmos, o que não acontecia em hardware antigo, onde os recursos dos processadores de fragmentos eram mais potentes. Além disso, a quantidade de processadores de vértices era inferior à quantidade de processadores de fragmentos.

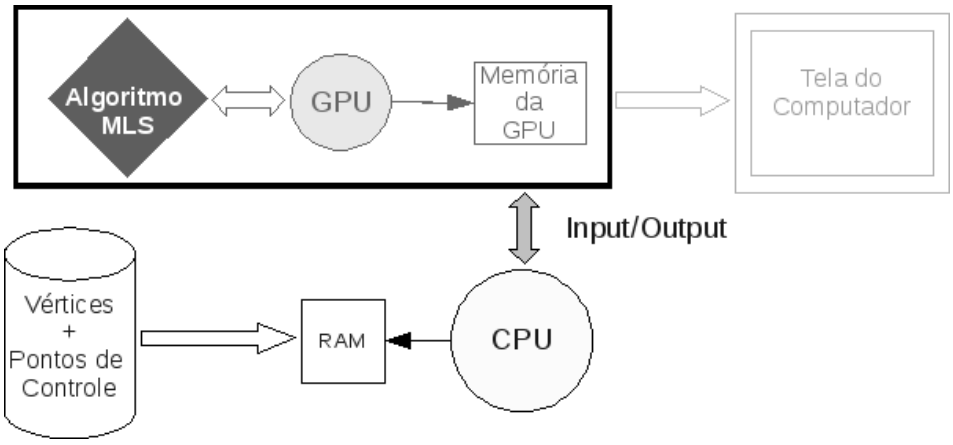


Figura 3.20: Diagrama que ilustra a implementação no *fragment shader*. A GPU é usada como uma caixa preta, isto é, recebe os vértices originais e retorna as sua posições deformadas, sem necessidade de renderizá-los imediatamente.

Implementação em CUDA

A implementação em CUDA (*Compute Unified Device Architecture*) segue a mesma estratégia usada no processador de fragmentos. É baseada na codificação de um único *kernel* que implementa o algoritmo de deformação MLS. Na fase de inicialização são estabelecidos dois parâmetros de entrada, os quais são passados através de texturas 2D e 1D, respectivamente: as posições iniciais dos vértices e as posições dos pontos de controle. Após especificadas as posições modificadas dos pontos de controle, o *kernel* é invocado para computar as posições deformadas dos vértices. Em seguida, as posições modificadas podem ser renderizadas ou processadas para outros fins (por exemplo, medição de erros).

Vários testes foram realizados para determinar o tamanho ótimo dos blocos e a grade de computação. Foi observado que blocos de $8 \times 8 \times 1$ e uma grade de $\sqrt{n}/8 \times \sqrt{n}/8 \times 1$ o desempenho é melhor do que com outras configurações. Na Figura 3.19 pode-se observar que, embora o mesmo hardware esteja sendo usado, o desempenho da implementação em CUDA é superior às implementações nos *shaders* de vértices e fragmentos uma vez que esta tecnologia não depende do *pipeline* gráfico, portanto, faz um uso mais eficiente dos recursos da placa gráfica. No melhor caso, a implementação é quase 2 vezes superior, no entanto, quando mais pontos de controle são utilizados (o que implica em mais operações aritméticas) o desempenho converge para os tempos gastos pelas implementações no processador de vértices ou fragmentos. Quando comparada com a implementação em CPU é possível observar uma superioridade de 37 vezes no pior caso e 129 vezes no melhor

Modelos	Vértices	Implementações	Erro Absoluto	Erro Relativo
Dino	14050	CPU-FS	3.83470e-07	1.10698e-07
		CPU-CUDA	4.00118e-07	1.15504e-07
		CUDA-FS	3.16891e-07	9.14784e-08
Dragon	16096	CPU-FS	2.40736e-07	6.94944e-08
		CPU-CUDA	2.41771e-07	6.97933e-08
		CUDA-FS	1.49012e-07	4.30159e-08
Girl	9912	CPU-FS	4.33224e-07	1.25061e-07
		CPU-CUDA	5.54756e-07	1.60144e-07
		CUDA-FS	5.47352e-07	1.58007e-07
Horse	19851	CPU-FS	2.66560e-07	7.69493e-08
		CPU-CUDA	2.99510e-07	8.64610e-08
		CUDA-FS	2.14908e-07	6.20385e-08

Tabela 3.3: Precisão das implementações. Cada modelo foi deformado por três implementações (CPU, CUDA e processador de fragmentos), em seguida, foi medida a diferença entre os resultados. A coluna Erro Absoluto denota a maior diferença entre vértices de dois modelos deformados. Por outro lado, o Erro Relativo se refere a razão entre o Erro Absoluto e o tamanho da diagonal do cubo envolvente dos modelos.

caso.

Por último, foram executados experimentos que permitem analisar a exatidão das diferentes implementações. Para isso, foram selecionados 4 modelos de geometria diferente e alguns pontos de controle sobre sua superfície. Os modelos foram reescalados para estarem contidos dentro do cubo envolvente definido pelos pontos $[-1\ -1\ -1]$ e $[111]$. A Figura 3.21(a) mostra as configurações iniciais das deformações. Em seguida, foram usadas as implementações na CPU, no processador de fragmentos (FS) e em CUDA para deformá-los. Finalmente foram medidas as diferenças entre: o resultado da CPU frente ao resultado do FS, o resultado da CPU frente ao resultado do CUDA e o resultado do CUDA frente ao resultado do FS. A Tabela 3.3 mostra a lista dos maiores valores obtidos, onde o Erro Absoluto se refere à maior distância entre todos os vértices e seus correspondentes. Embora não exista diferença visual entre os modelos deformados (ver Figura 3.21) existe ainda uma pequena diferença numérica entre suas coordenadas. Pode-se observar que o maior erro absoluto não é mais que 1e-07 e que o maior erro relativo está ao redor de 1e-

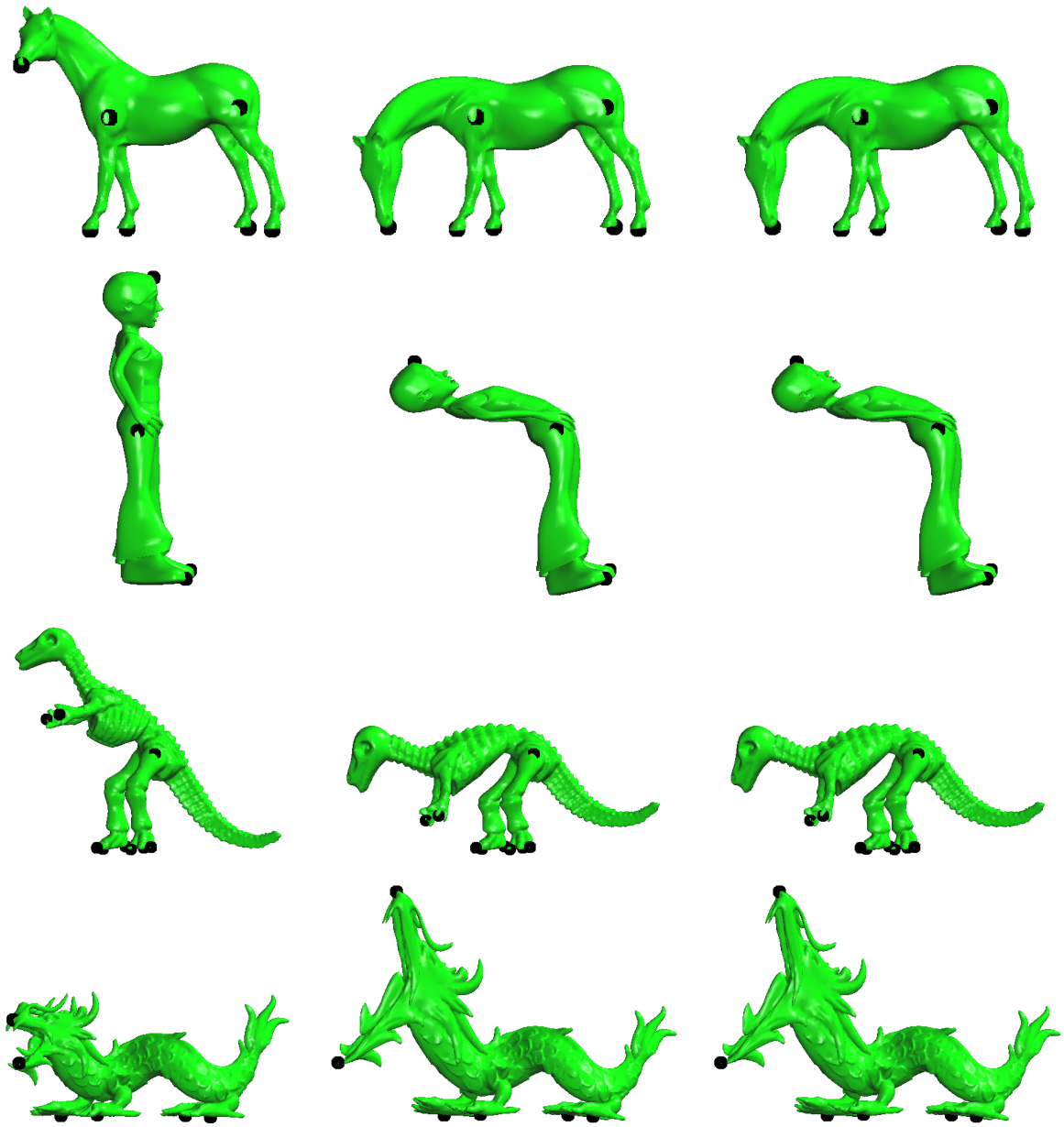
08. Tais diferenças podem ser atribuídas ao fato de que as operações aritméticas não são necessariamente aplicadas na mesma ordem e ao fato de que as GPUs não implementam fielmente o padrão de aritmética de ponto flutuante IEEE 754 [NVI07].

3.7.2 MLS em múltiplos núcleos

Para a implementação em múltiplos núcleos foi utilizada a biblioteca Pthreads [Law95]. A implementação é semelhante à versão em CPU. No entanto, a divisão de trabalho é explicitamente estabelecida para duas linhas de processamento (*threads*), sendo que cada *thread* processa metade dos vértices. Tal divisão do trabalho foi feita após vários experimentos com um número variável de *threads*, onde foi observado um melhor desempenho (menor tempo de processamento) com duas *threads* do que com mais. Isto se deve ao fato do processador onde os experimentos foram rodados conter dois núcleos.

As freqüências de renderização em quadros por segundo mostradas na Tabela 3.19 sugerem um ganho em desempenho de 100% frente à implementação convencional em núcleo simples (*single-core*) em CPU. No entanto, o desempenho é ainda bem inferior quando comparado com as implementações em GPU.

Finalmente, para confirmar a validade destes resultados, foram feitos experimentos onde não acontece nenhum processo de renderização e a deformação é aplicada na massa de dados uma única vez. Optou-se por usar uma massa de dados aleatória, tanto para as posições dos vértices quanto para os pontos de controle. Quantidades variáveis destes dados foram gerados e o tempo gasto foi medido na deformação (veja os tempos na Tabela 3.4 e os gráficos da Figura 3.22). Observa-se que a implementação em GPU usando CUDA, no melhor caso, é 110 vezes superior à implementação em CPU. Da mesma forma, a implementação no processador de fragmentos mostra um desempenho 74 vezes superior.



(a) (b) (c)

Figura 3.21: Ilustração da correta funcionalidade das implementações em GPU. Além de oferecerem um melhor desempenho o erro numérico existente entre os resultados é desprezível. (a) Modelos iniciais e pontos de controle. (b) Deformações computadas pela implementação em CPU. (c) Deformações computadas usando CUDA.

PC	CPU	MC	FS	CUDA
16	0.40	0.22	0.91	0.01
32	0.70	0.35	0.20	0.01
64	1.27	0.64	0.20	0.02
128	2.44	1.29	0.22	0.03
256	4.78	2.39	0.25	0.06
512	9.42	4.73	0.30	0.13
1024	18.73	9.41	0.41	0.25

(a) 512x512 vértices

PC	CPU	MC	FS	CUDA
16	1.60	0.88	0.26	0.03
32	2.76	1.42	0.28	0.04
64	5.16	2.55	0.31	0.07
128	9.72	4.90	0.36	0.13
256	19.09	9.63	0.47	0.26
512	37.65	18.98	0.69	0.50
1024	75.04	37.75	1.42	0.98

(b) 1024x1024 vértices

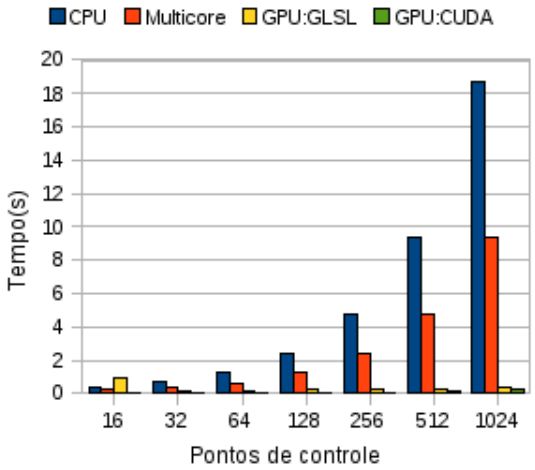
PC	CPU	MC	FS	CUDA
16	6.43	3.23	0.57	0.12
32	11.01	5.62	0.61	0.18
64	20.47	10.17	0.72	0.29
128	39.14	19.55	0.93	0.54
256	79.28	38.28	1.38	1.02
512	150.86	75.65	2.26	1.98
1024	300.10	150.54	4.03	3.92

(c) 2048x2048 vértices

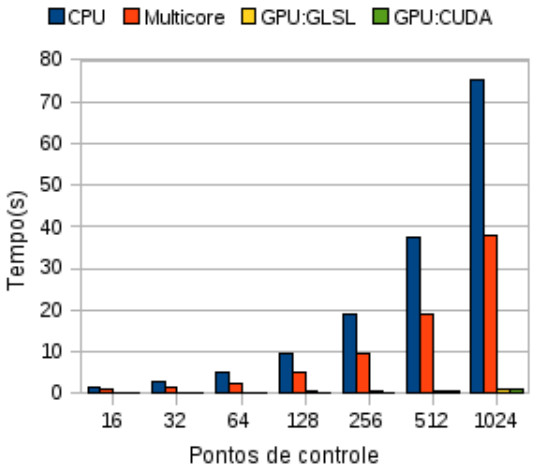
PC	CPU	MC	FS	CUDA
16	25.6	13.06	6.51	0.48
32	44.0	22.51	3.11	0.71
64	81.17	40.70	2.42	1.15
128	156.19	78.29	3.26	2.12
256	305.66	153.05	5.05	4.07
512	603.288	302.58	8.57	7.90
1024	1413.32	601.19	55.77	12.41

(d) 4096x4096 vértices

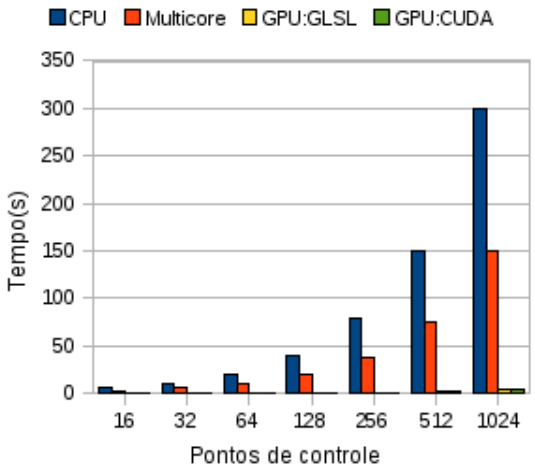
Tabela 3.4: Tempos gastos (em segundos) por diferentes implementações do Algoritmo 2. Cada quadro representa um experimento de deformação onde foi estabelecida uma quantidade fixa de vértices e um número variável de pontos de controle (CP), ambos gerados aleatoriamente. Os acrônimos CPU, MC, FS, CUDA referem-se as implementações serial em CPU, múltiplos núcleos em CPU, processador de fragmentos (FS) e CUDA em GPU.



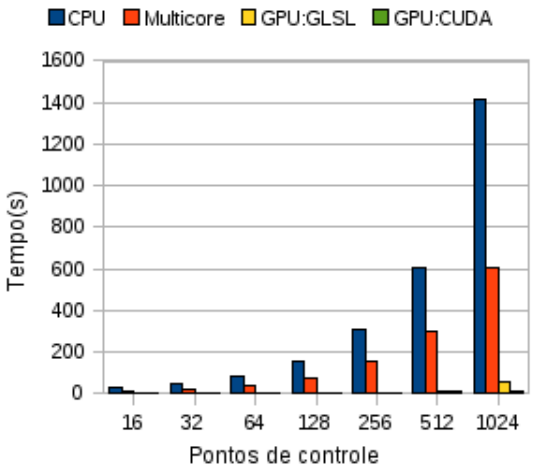
(a) 512x512 vértices



(b) 1024x1024 vértices



(c) 2048x2048 vértices



(d) 4096x4096 vértices

Figura 3.22: Comparação dos tempos gastos (em segundos) por diferentes implementações do Algoritmo 2 aplicados na deformação de numa massa de dados gerada aleatoriamente. Cada gráfico corresponde aos valores dos quadros na Tabela 3.4.

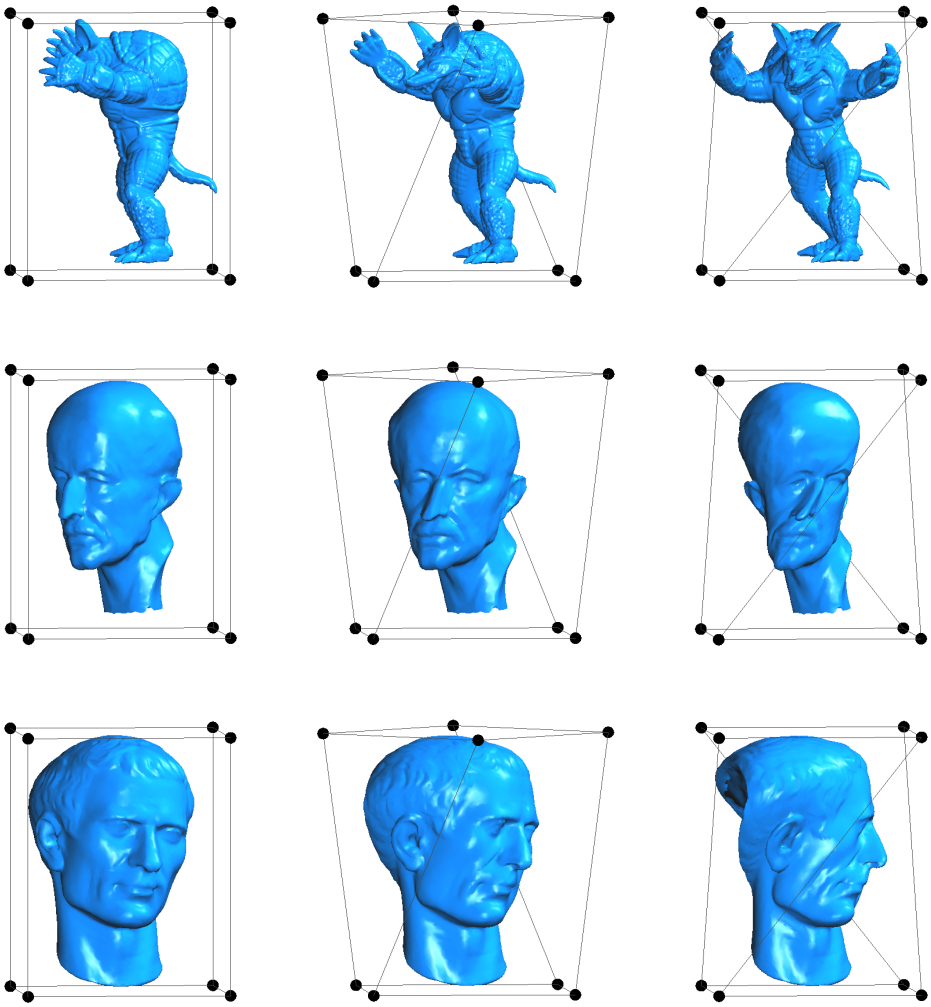
Capítulo 4

Deformação de superfícies

A deformação de malhas de superfícies – especialmente no contexto de animação de personagens – pode ser definida como um processo de modificação de sua geometria sem alterar a forma geral do modelo. Tradicionalmente, técnicas de cinemática inversa são as mais utilizadas neste tipo de tarefa [F$\hat{0}$3]. Estas, no entanto, requerem que o personagem a ser deformado seja cuidadosamente acoplado a um esqueleto. Além disso, as restrições de movimento de ossos e juntas devem ser estabelecidas manualmente, o que pode tornar a execução de uma simples deformação bastante tediosa.

Recentemente, várias técnicas baseadas em esquemas de otimização têm sido propostas para esta tarefa [Ale06, BS08, SA07]. A maior parte delas procura obter uma nova pose de um personagem por meio de um processo de deformação guiado pelo deslocamento de pontos de controle posicionados sobre a superfície do modelo. A interação é efetuada arrastando um mouse 2D na tela do computador, provendo, assim, uma interface bastante simples para a geração de deformações. Costuma-se não estabelecer nenhuma restrição ao arraste do mouse, isto é, os pontos de controle podem ser deslocados livremente e rotações não precisam ser especificadas de forma explícita. Entretanto, requer-se que a forma do modelo seja preservada tanto quanto possível, dando ao usuário a impressão de que ele está manipulando um objeto real.

O método descrito no Capítulo 3 pode ser classificado como pertencente a essa categoria. No entanto, como efetua deformações do espaço, isto é, define um mapeamento $f : \Re^3 \rightarrow \Re^3$, o processo é influenciado apenas pelo posicionamento dos pontos de controle e do vértice sendo deformado. Praticamente, qualquer modelo imerso no espaço está sujeito à mesma deformação, sem qualquer consideração à sua forma. Veja, por



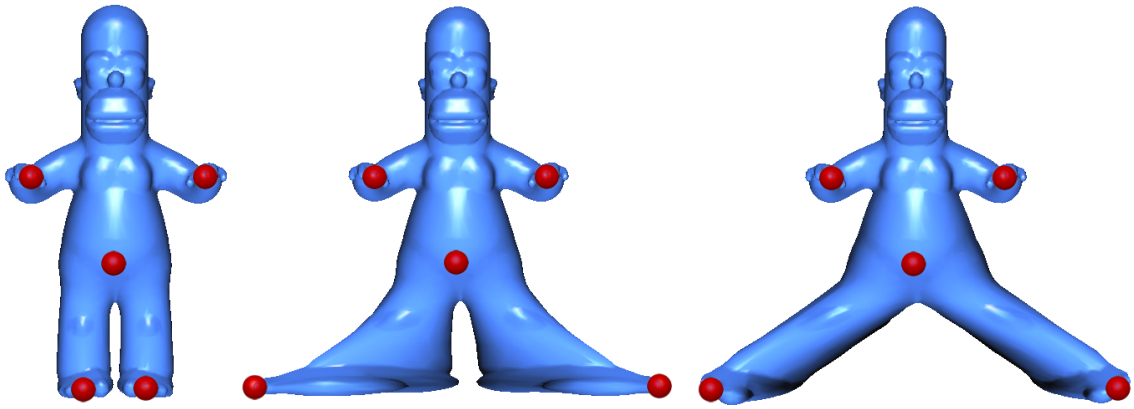
(a) (b) (c)

Figura 4.1: Deformação do espaço. (a) Modelos iniciais e pontos de controle. (b) e (c) Deformações – guidas pelos mesmos pontos de controle – obtidas com o algoritmo apresentado no Capítulo 3.

exemplo, na Figura 4.1 a deformação de dois modelos sujeitas aos mesmos pontos de controle.

Em algumas aplicações, os resultados dessa abordagem não são os desejáveis. Por exemplo, em animação de personagens, a deformação deve levar em consideração peculiaridades do modelo, tais como extremidades, articulações, etc. Como ilustração, compare a deformação gerada pelo algoritmo descrito no Capítulo 3 (mostrada na Figura 4.2(b)) com um resultado mais razoável exibido na Figura 4.2(c).

Este capítulo apresenta alternativas para tornar o algoritmo de deformação apresentado no capítulo anterior (Algoritmo 2) em um esquema sensível à forma do modelo. A



(a) (b) (c)

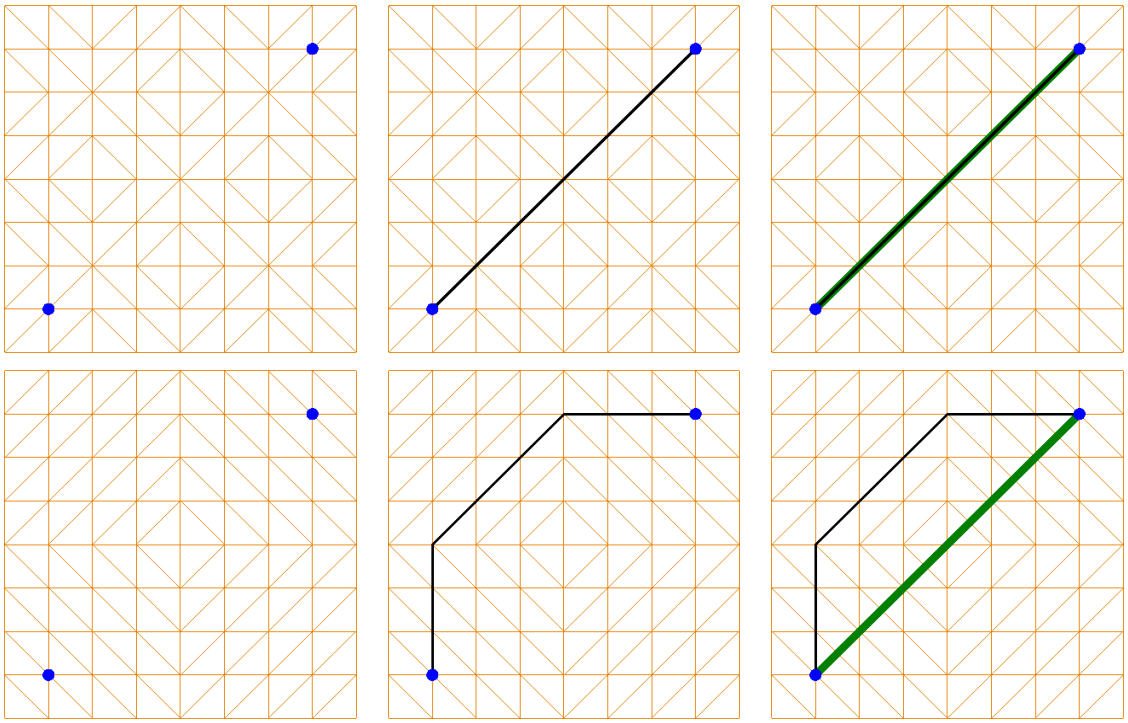
Figura 4.2: Deformação do espaço vs. deformação da superfície. (a) Modelo inicial e pontos de controle. (b) Resultado do algoritmo de deformação do espaço apresentado no Capítulo 3. (c) Resultado da deformação usando uma métrica de distância definida sobre a superfície.

Seção 4.1 mostra como uma métrica mais sensível à malha permite gerar deformações mais plausíveis do que as geradas por uma métrica euclidiana. Depois, uma alternativa que visa maior controle na deformação MLS, através do uso de esqueletos, é proposta na Seção 4.2.

4.1 Métrica sobre a superfície

Uma alternativa para tornar o Algoritmo 2 mais sensível à forma do modelo consiste em substituir a métrica euclidiana usada na formulação do problema por uma métrica restrita à superfície, isto é, uma métrica "geodésica". Neste caso, a distância entre dois pontos é medida em termos do tamanho da curva mais curta na superfície que conecta esses pontos. Usualmente, tal curva é chamada de geodésica.

Na literatura, podem ser encontrados diversos algoritmos que resolvem o problema de encontrar geodésicas numa malha de triângulos. Tais algoritmos podem ser classificados em duas categorias: aqueles que oferecem soluções exatas [MMP87, CH90, Kap99] e aqueles que são baseados em aproximações. O cômputo de geodésicas exatas é muito custoso, enquanto que, em muitas aplicações, o uso de geodésicas aproximadas é suficiente. Entre as propostas aproximadas destacam-se: os esquemas baseados em subdivisão de arestas [Kan00], os esquemas iterativos [MVC04] e o algoritmo baseado na definição de



(a) (b) (c)

Figura 4.3: Geodésicas aproximadas numa malha de triângulos. (a) Duas malhas de triângulos com conectividade diferente e dois vértices selecionados. (b) Geodésicas aproximadas (em preto). (c) No pior caso, o tamanho das geodésicas aproximadas é uma sobre-estimativa do tamanho do menor caminho (em verde) entre os dois vértices.

intervalos sobre as arestas da triangulação proposto por Surazhsky et al. [SSK$^+$05].

Embora os métodos aproximados computem caminhos geodésicos eficientemente, a sua necessidade não é vital na deformação MLS. Na verdade, o algoritmo de deformação apresentado no Capítulo 3 precisa simplesmente de campos de distância definidos sobre a superfície, mas não os caminhos. Além disso, se os triângulos da malha a deformar forem suficientemente pequenos, os campos de distância podem ser aproximados considerando o tamanho das arestas da superfície como medida de distância entre vértices vizinhos. Dessa forma, a distância mais curta entre os vértices v_0 e v_1 de uma malha $\mathcal{M}$ é definida pelo tamanho do conjunto de arestas que, quando conectadas, formam o caminho mais curto entre v_0 e v_1. Conforme ilustrado na Figura 4.3, com esta aproximação, no pior caso, o tamanho das geodésicas aproximadas é uma super-estimativa do tamanho do menor caminho entre os dois vértices. Apesar desse fato, resultados razoáveis ainda podem ser obtidos com esta abordagem quando aplicados em modelos de alta resolução. Veja na Figura 4.4 uma geodésica, computada com esta aproximação, entre dois vértices sobre a

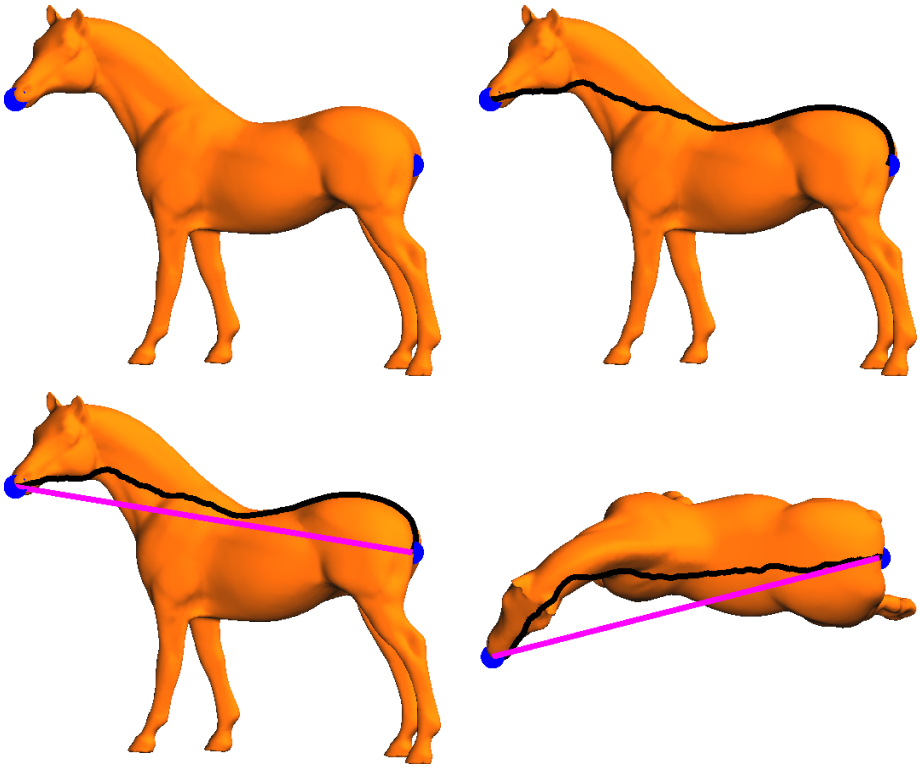


Figura 4.4: Geodésicas entre dois pontos da superfície do cavalo. Na parte superior, dois pontos (em azul) sobre a superfície e a geodésica (em preto). Na parte inferior, duas vistas que ilustram a diferença entre a distância euclidiana (em rosa) e a geodésica aproximada (em preto).

superfície do cavalo.

Assim, dada uma malha $\mathcal{M}$ e um vértice fonte v, um campo de distância $d_v : \Omega \subseteq \Re^3 \mapsto \Re$ é aproximado pelo conjunto de geodésicas definidas entre v e v_i, onde $i = 0 \cdots n$, Ω é o conjunto formado pelos vértices de $\mathcal{M}$ e n é a cardinalidade de Ω. Na Figura 4.5 é mostrado o resultado do cálculo do campo de distância definido por um vértice posicionado na boca do cavalo. Dessa forma, na deformação MLS, os pesos $w_i = d_i^{-2}$ podem ser determinados através da construção de campos de distância d_c usando o Algoritmo de Dijkstra sobre os vértices de $\mathcal{M}$. Cada ponto de controle c_i é usado como elemento fonte de um processo de expansão sobre o grafo da malha.

Deformações computadas com este enfoque de medição de distâncias são mostradas na Figura 4.6. Na deformação do cavalo, a métrica "geodésica" permite deformar uma perna sem influenciar nas outras. No segundo exemplo, os braços da garota não ficam atrelados ao tronco como acontece quando usada uma métrica euclidiana (veja a Figura 3.3). O mesmo comportamento pode ser observado entre a cauda e o tronco do dinossauro.

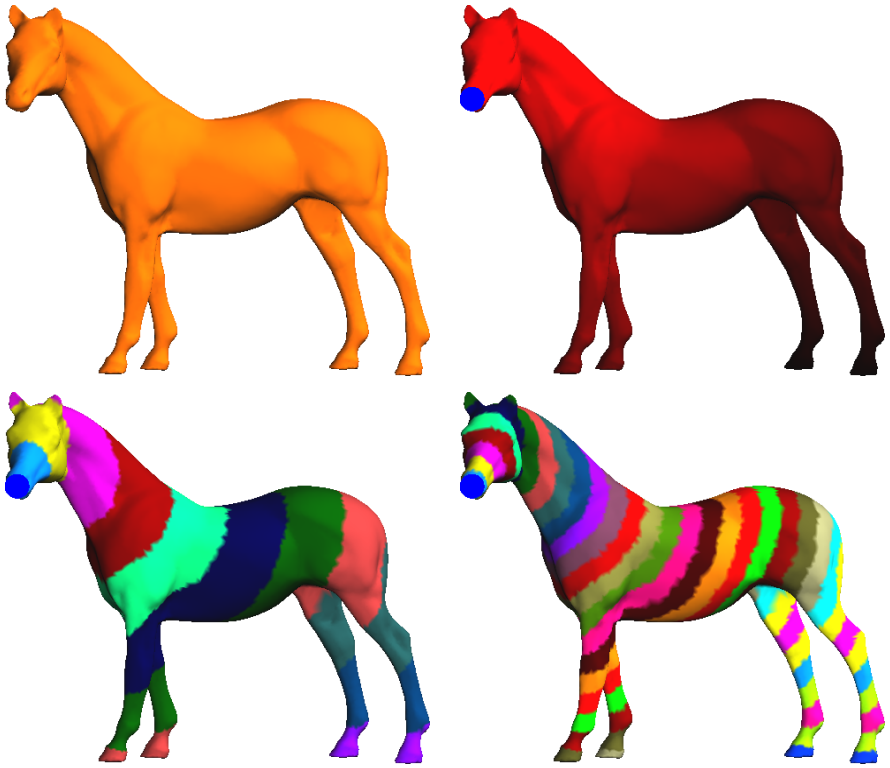
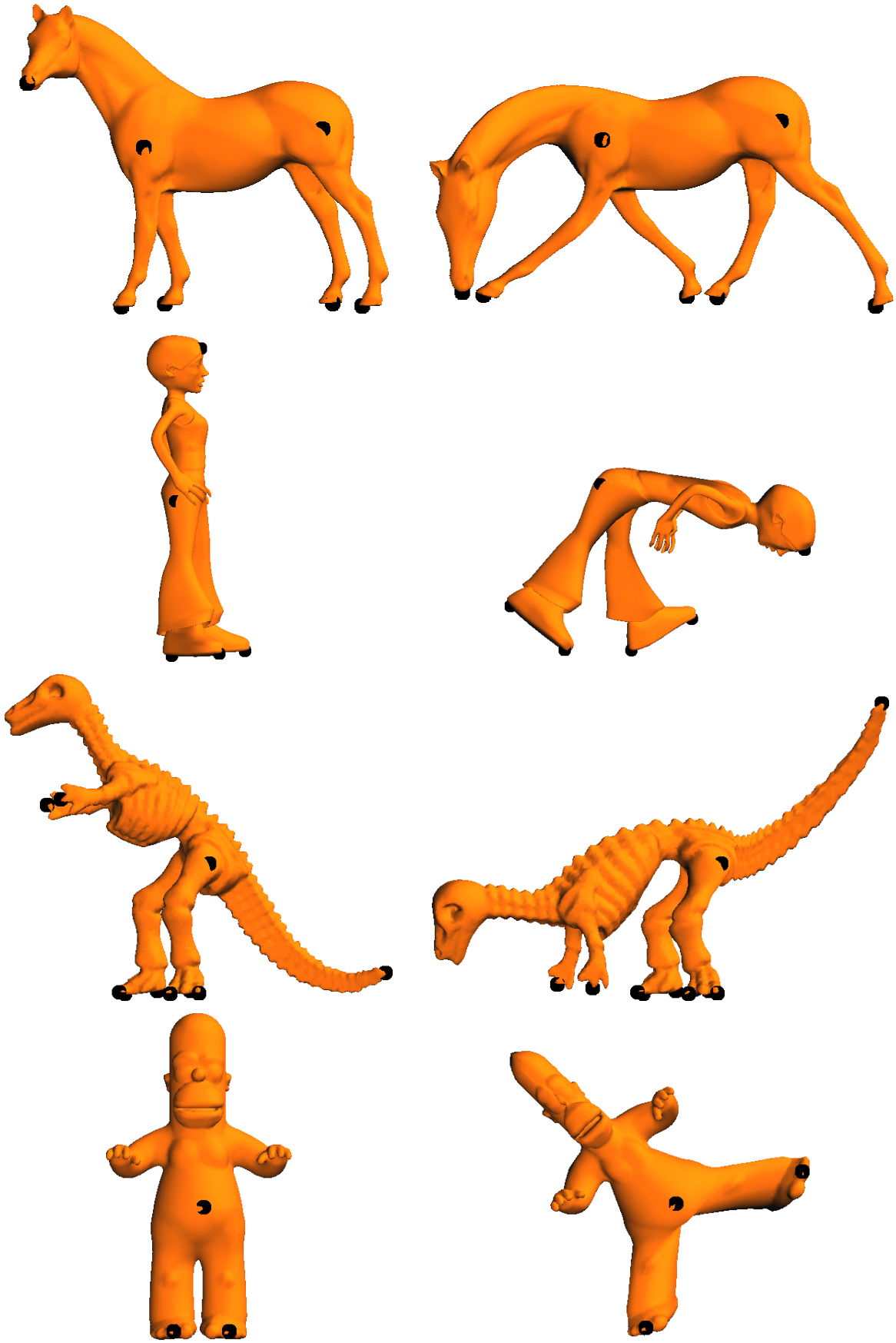


Figura 4.5: Aproximações de campos de distância. Na parte superior, é mostrado o campo de distância ao redor de um vértice v (em azul), onde a cor em um vértice v_i é mais escura quanto mais afastado de v estiver. Para uma melhor ilustração, na parte inferior, a coloração do mesmo campo de distância é efetuada através de um filtro de intervalos, onde vértices com a mesma cor encontram-se a uma distância similar a v.

Na deformação do modelo "Homer", observa-se um resultado intuitivo nas pernas, no entanto, a cabeça apresenta um achatamento indesejado. A interatividade da deformação não é afetada visto que a aplicação do Algoritmo de Dijkstra na malha para um conjunto pequeno de pontos de controle é rápida. Além disso, tal operação é feita uma única vez, isto é, antes da etapa de manipulação interativa.

4.2 Deformação guiada por esqueletos

Muitos dos problemas vistos até agora podem ser aliviados por um esquema híbrido que combina a deformação MLS e técnicas tradicionais de animação conduzida por esqueletos. Existem duas vantagens ao usar esqueletos. Primeiro, o esqueleto é uma boa representação da forma global do modelo, o que significa que pequenos detalhes presentes na superfície do modelo não afetam a sua forma. Segundo, é possível influenciar o com-



(a) (b)

Figura 4.6: Deformações MLS usando uma métrica geodésica aproximada.

portamento do método alterando as distâncias computadas ao longo dos ossos e juntas, podendo fazer algumas partes mais "rígidas" que outras. Esta seção mostra como essas questões podem ser abordadas.

O esquema é dividido em duas fases: a fase de preparação (*rigging*), efetuada apenas uma vez, e a fase de deformação, efetuada uma vez para cada pose desejada. Na fase de preparação, primeiro extrai-se do modelo um esqueleto aproximado, isto é, uma estrutura de árvore composta de juntas e ossos (Figura 4.7(a)). Depois, cada vértice da malha é associado a uma junta do esqueleto (Figura 4.7(b)). Por último, um processo de distribuição de pesos estabelece a influência relativa de cada junta na deformação dos vértices da malha (Figura 4.7(c)).

Na fase de deformação o usuário define um conjunto de pontos de controle $\{\mathbf{p}_i\}$ sobre as juntas do esqueleto (pontos vermelhos na Figura 4.7(d)). Estes, depois, são movidos interativamente para posições alvo $\{\mathbf{q}_i\}$, que farão com que as juntas sejam modificadas seguindo o esquema MLS introduzido no Capítulo 3. No entanto, as distâncias são medidas ao longo do esqueleto, ao invés de medidas usando a norma euclidiana (Figura 4.7(e)). Observe que, diferente da animação baseada em esqueletos tradicional, os ossos não são restritos a ser transformados rigidamente. Para cada junta j, os centróides $\{\mathbf{p*}_j\}$ e $\{\mathbf{q*}_j\}$ assim como os vetores de rotação $\{\mathbf{u}_j\}$ são armazenados. Finalmente, uma combinação linear de transformações é computada para cada vértice da malha utilizando os valores armazenados (Figura 4.7(e)).

4.2.1 Extração de esqueletos

Todos os métodos de extração de esqueletos – veja o artigo de Cornea e Min [CM07] para maiores detalhes – podem ser usados no esquema proposto. No entanto, diferentemente dos esquemas de animação tradicionais, esta proposta não requer esqueletos com topologia específica ou correspondência cuidadosa entre esqueleto e malha. Neste trabalho, o esqueleto é esboçado manualmente, obtendo-se resultados adequados para malhas que representam modelos simples.

4.2.2 Segmentação da malha

Uma vez conhecido o esqueleto, todos os vértices da malha são separados em conjuntos S_j, onde cada conjunto S_j contém todos os vértices que serão influenciados pela trans-

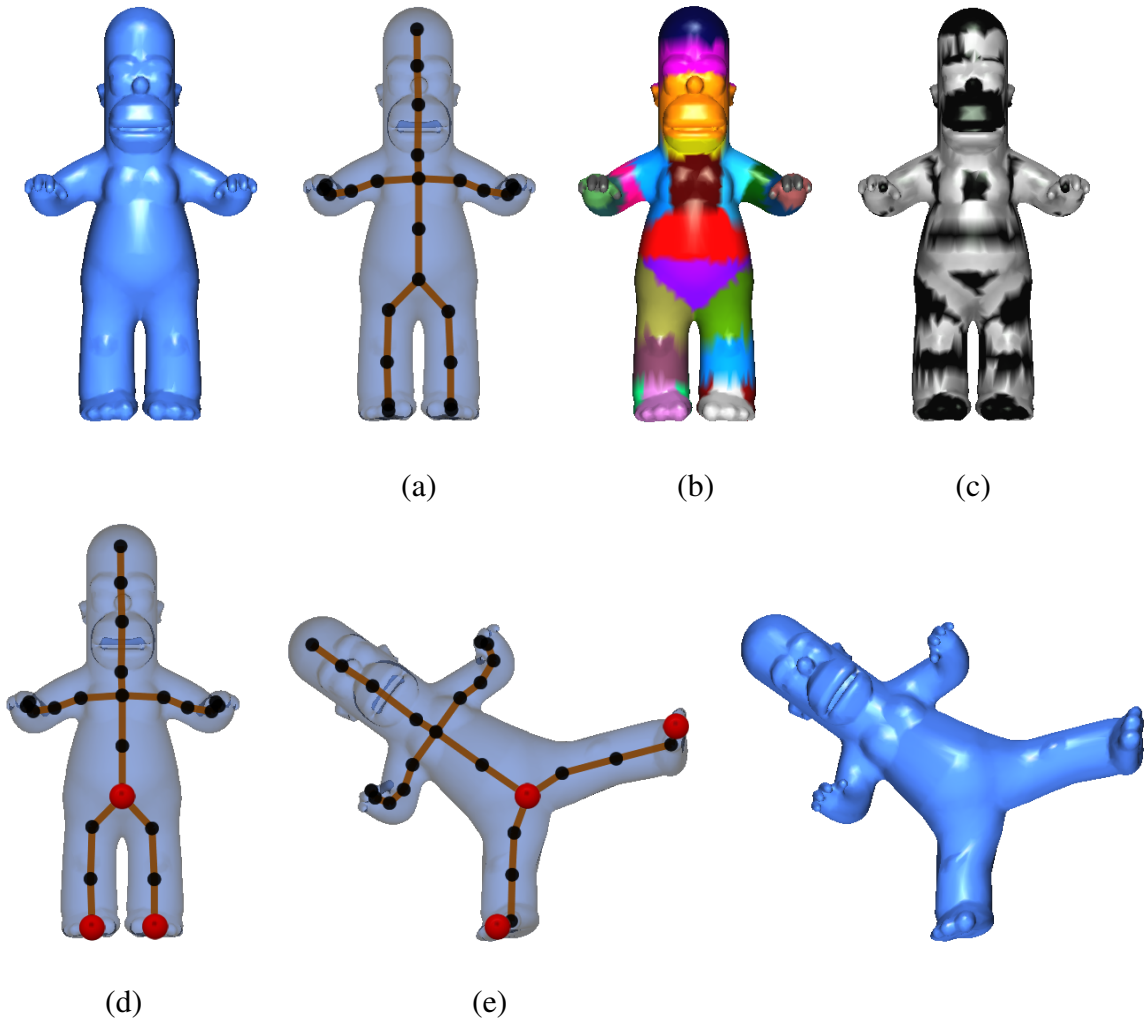


Figura 4.7: Deformação conduzida por esqueletos. Na fase de preparação, um esqueleto é inserido na malha (a), depois a malha é dividida em grupos (b) e a influência das juntas é definida para cada vértice (c). Os vértices nas regiões escuras serão influenciados por apenas uma junta e os vértices nas outras regiões serão influenciados por mais de uma junta. Na fase de deformação, os pontos de controle (pontos vermelhos) são definidos sobre as juntas (pontos pretos) do esqueleto (d). Depois de mover um ponto de controle (e), o esqueleto é deformado e as posições dos vértices da malha são modificadas, projetando as transformações computadas nas juntas associadas.

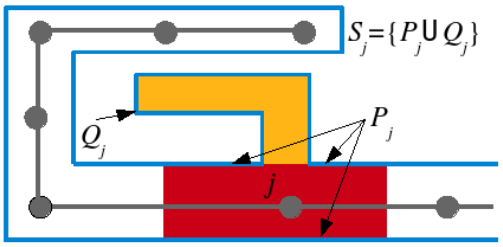


Figura 4.8: Área de influência da junta j. O conjunto P_j da junta j é composto pela borda da região vermelha e o conjunto Q_j é a borda da região amarela.

formação de uma junta j. Um conjunto S_j é definido como a união de dois conjuntos de vértices: os primários P_j e os secundários Q_j. Veja na Figura 4.8 uma ilustração da definição destes conjuntos.

P_j é o conjunto de vértices **v** tal que, entre todas as juntas do esqueleto que são visíveis a **v**, j é a mais próxima. Visível significa que um segmento definido por **v** e a junta j não intercepta a malha. Por outro lado, os vértices secundarios permitem segmentar regiões tubulares sem ossos. Dessa forma, o conjunto Q_j de uma junta j é constituído por vértices **v** para os quais a junta j é a solução do problema

$$\min_{j \in \text{juntas}} g(\mathbf{v}, P_j),$$

onde $g(a, X)$ é a distância geodésica entre um ponto a e um conjunto de pontos X. Lembrando que a distância geodésica entre dois pontos de uma superfície é a distância do caminho mais curto (geodésica) que os conecta.

É importante mencionar que algoritmos exatos para computar estes conjuntos são custosos. Logo, ao invés de computar os comprimentos exatos dos caminhos geodésicos, são usadas geodésicas compostas apenas pelas arestas da malha. Isto, semelhante à Seção 4.1, torna possível empregar o algoritmo de Dijkstra para encontrar os elementos do conjunto Q_j começando nos vértices da borda de P_j.

Da mesma forma, a determinação da visibilidade usando, por exemplo, um algoritmo de traçado de raios é desnecessariamente custoso para a tarefa em questão. Ao invés disso, utiliza-se um algoritmo de propagação que emprega propriedades de visibilidade "local". Uma junta j é localmente visível a um vértice **v** se o ângulo entre a normal (voltada para dentro) de **v** e o vetor $j - \mathbf{v}$ for agudo (assume-se que o esqueleto está sempre colocado dentro da malha). Seja $L(j)$ o conjunto de vértices da malha localmente visíveis a j e que estão mais perto de j do que de qualquer junta k para a qual também são localmente

visíveis. Então, o algoritmo para encontrar o conjunto P_j da junta j é o seguinte:

1. Inicializar $P(j)$ com $\{$**semente**$\}$, onde **semente** é o vértice em $L(j)$ que é mais próximo a j.

2. Seja $V = L(j) - P(j)$. Procurar em V por um vértice $\mathbf{v}'$ tal que $\mathbf{v}'$ é uma aresta vizinha de algum vértice $\mathbf{v} \in P(j)$.

3. Se tal vértice for encontrado, coloque-o em $P(j)$ e repita o passo (2).

4. Se $V \neq \emptyset$, selecione outra **semente** $\in V$, tendo certeza de que **semente** é globalmente visível utilizando traçado de raios. Se nenhuma puder ser encontrada, pare. De outra forma, inclua **semente** em $P(j)$ e repita o passo (2).

Este algoritmo é muito rápido, já que encontrar arestas vizinhas e testes de visibilidade local são operações de tempo constante. Se outras sementes têm de ser encontradas por traçado de raios (passo 4 acima), o processo é restrito à vizinhança de j. De acordo com os experimentos realizados, mesmo se um vértice que deveria pertencer a P_j não for encontrado, na maioria das vezes o algoritmo que computa Q_j determinará a junta correta para aquele vértice. Além disso, deve-se ressaltar que o processo de *skinning* usado para deformar o modelo é robusto suficiente para lidar com erros de atribuição, se houver.

A Figura 4.9 ilustra a utilidade da definição dos conjuntos P_j e Q_j. Nesse exemplo não foram definidos ossos dentro da cauda nem dentro das orelhas do modelo Armadillo, no entanto, o algoritmo associa os vértices destas partes da malha às juntas mais próximas. Portanto, as partes do modelo que conformam a cauda e as orelhas se modificaram como se fossem componentes rígidas. A Figura 4.10 mostra resultados da segmentação de alguns modelos.

4.2.3 Determinação dos pesos para *Skinning*

Apesar de cada vértice da malha estar associado a um único conjunto S_j, outras juntas não j podem, também, influenciar sua posição deformada. Deve-se, portanto, determinar um esquema de atribuição de pesos ω[1] capaz de manter a suavidade na malha deformada. Abordagens recentes a este problema (veja, por exemplo, a proposta de Baran e Popovic

[1]Não confundir ω com os pesos w usados na formulação MLS.

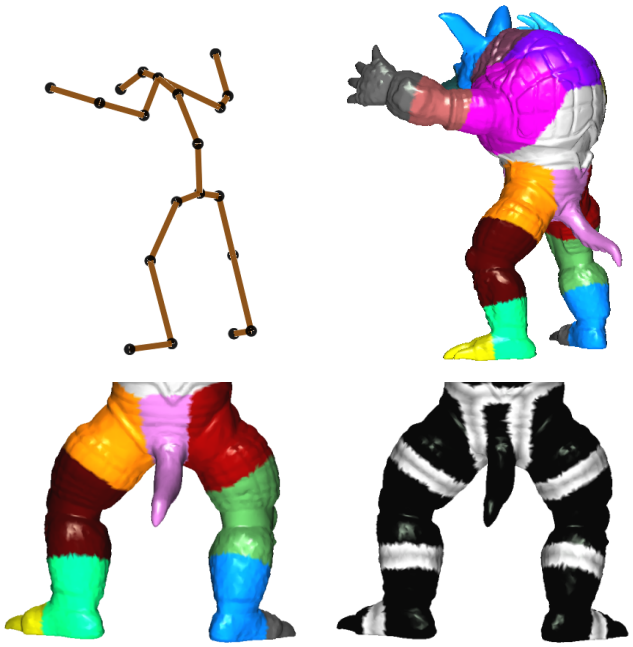


Figura 4.9: Segmentação de regiões tubulares sem ossos. Observe a ausência de ossos na cauda e nas orelhas. Na parte inferior, uma vista mais detalhada do resultado da segmentação e distribuição dos pesos (a coloração é similar à da Figura 4.7(c)).

[BP07]) simulam um processo de difusão de calor do esqueleto à malha, que ocorre somente no seu interior como se a superfície da malha fosse um isolante térmico perfeito. Para evitar a complexidade de se resolver um difícil problema de difusão de calor em 3D restrito a um poliedro, considera-se que o calor emitido pela junta j é totalmente direcionado ao conjunto S_j. A partir de S_j o calor é difundido para toda a malha. Agora suponha que a junta j tem temperatura 1 enquanto todas as outras juntas têm temperatura 0. Seja $\omega_{\mathbf{v}}$ a atual temperatura de um vértice da malha $\mathbf{v}$. Então, posto que a energia é espalhada de forma radial das juntas à malha (e vice-versa), o fluxo de calor entre o vértice $\mathbf{v} \in S_k$ e a junta k é dada por

$$\begin{array}{ll} \dfrac{\alpha(1-\omega_{\mathbf{v}})}{||\mathbf{v}-k||^2} & \text{se} \quad k=j, \quad \text{e} \\ \dfrac{-\alpha\,\omega_{\mathbf{v}}}{||\mathbf{v}-k||^2} & \text{caso contrário,} \end{array}$$

onde α é uma constante global. Quando um estado de equilíbrio é atingido, o fluxo entre $\mathbf{v}$ e k deve ser igual ao fluxo entre $\mathbf{v}$ e seus vizinhos, que pode ser estimado por

$$\sum_{\mathbf{u} \in N(\mathbf{v})} (\omega_{\mathbf{u}} - \omega_{\mathbf{v}}) \Delta_{\mathbf{uv}},$$

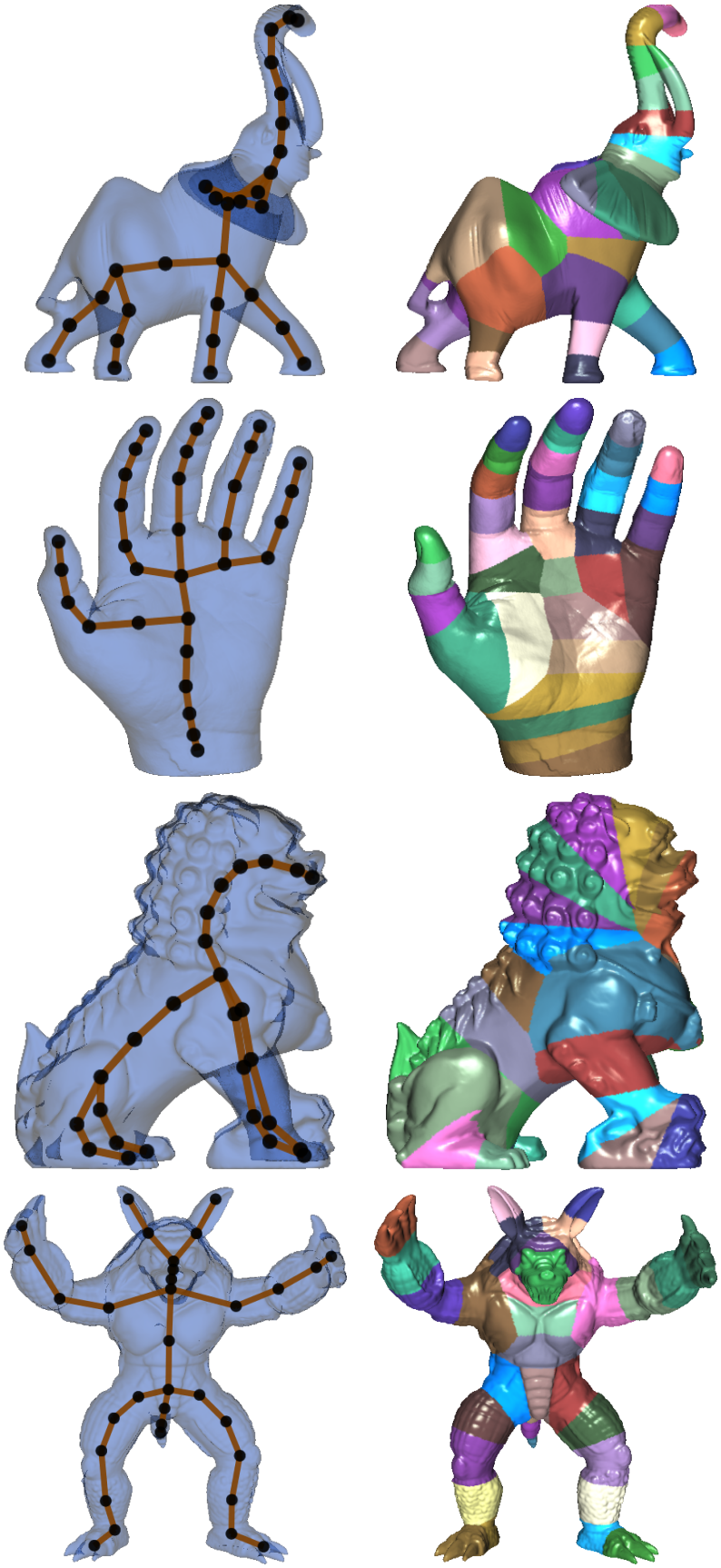


Figura 4.10: Resultados da segmentação.

onde $N(\mathbf{v})$ é o conjunto de arestas incidentes a $\mathbf{v}$ e $\Delta_{\mathbf{uv}}$ é o peso atribuído à aresta $(\mathbf{u}, \mathbf{v})$ que, neste trabalho, é dado pelo operador Laplace-Beltrami [BS08]. Temperaturas de estado de equilíbrio determinadas pelo calor emitido por j são, então, obtidas resolvendo um sistema de n equações (uma para cada vértice $\mathbf{v}$) da forma

$$\alpha\left(R_j(\mathbf{v}) - \frac{\omega_{\mathbf{v}}}{||\mathbf{v}-k||^2}\right) + \sum_{\mathbf{u}\in N(\mathbf{v})}(\omega_{\mathbf{u}} - \omega_{\mathbf{v}})\Delta_{\mathbf{uv}} = 0,$$

onde $R_j(\mathbf{v})$ é a função característica de S_j, i.e., igual a 1 se $\mathbf{v} \in S_j$ e igual a 0 caso contrário. Cada temperatura de vértice $\omega_{\mathbf{v}}$ obtida é considerada como o peso da junta j na combinação linear de transformações aplicadas em $\mathbf{v}$. Para determinar todos os pesos, um sistema parecido deve ser resolvido para todas as juntas.

Deve-se mencionar que esta formulação possui várias propriedades interessantes. Primeiramente, os pesos obtidos por cada junta j minimizam a discretização de um funcional da forma

$$\int_{\mathbf{v}\in M}(\nabla(\omega_{\mathbf{v}})^2 + \alpha(R_j(\mathbf{v}) - \omega_{\mathbf{v}})^2)d\mathbf{v}.$$

Em outras palavras, não só produz valores suaves de $\omega_{\mathbf{v}}$, mas também respeita a informação referente ao *rigging*. Além disso, assegura que a soma dos pesos seja 1 em cada vértice e faz com que a "quantidade de combinação" em um vértice seja regulada por sua distância ao esqueleto.

4.3 Resultados

A Figura 4.11 mostra quatro poses do modelo Armadillo obtido com esta técnica. Embora as transformações ótimas tenham sido computadas nas juntas do esqueleto, detalhes ainda são preservados (perceba, por exemplo, as rugas e as protuberâncias nas mãos, pernas e quadril). Mais resultados podem ser vistos na Figura 4.12. Estes exemplos foram obtidos usando poucos pontos de controle (apenas quatro ou cinco). O tempo para computar as deformações é dominado pela fase de preparação, que consiste em construir a estrutura de suporte ou *rigging* (isto é, o esqueleto e os pesos das juntas). Em contrapartida, a fase de deformação deve computar as transformações MLS apenas para as juntas, enquanto a deformação da malha é feita através de um processo de combinação linear de transformações (*Linear Blending Skinning* - LBS). Assim, a transformação aplicada para cada vértice da malha $\mathbf{v}$ está definida pelo vetor de rotação $\bar{\mathbf{u}}_{\mathbf{v}}$ e os centros ponderados

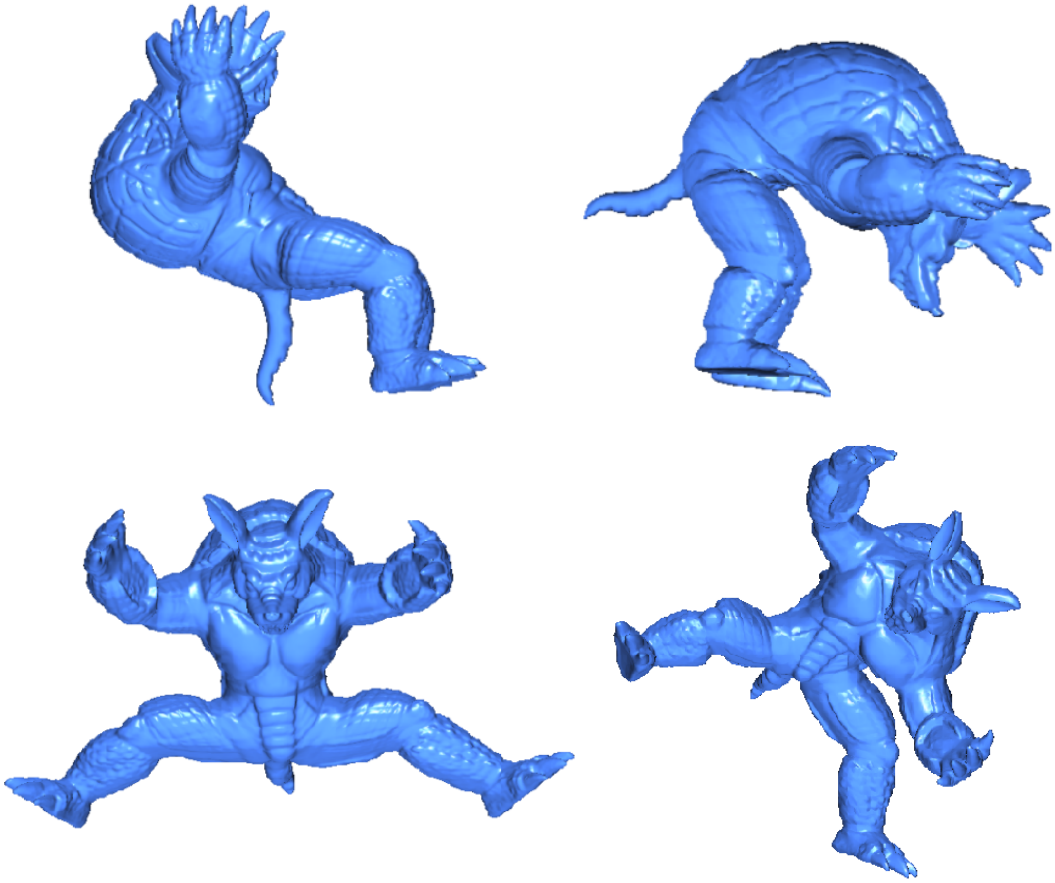


Figura 4.11: Diversas poses do Armadillo obtidas com o esquema proposto.

$\bar{\mathbf{p}}^*_{\mathbf{v}}$ e $\bar{\mathbf{q}}^*_{\mathbf{v}}$:

$$\bar{\mathbf{u}}_{\mathbf{v}} = \frac{\sum_j \omega_j \mathbf{u}_j}{\sum_j \omega_j}, \quad \bar{\mathbf{p}}^*_{\mathbf{v}} = \frac{\sum_j \omega_j \mathbf{p}^*_j}{\sum_j \omega_j}, \quad \text{e} \quad \bar{\mathbf{q}}^*_{\mathbf{v}} = \frac{\sum_j \omega_j \mathbf{q}^*_j}{\sum_j \omega_j}.$$

As deformações na Figura 4.12 foram obtidas interativamente (com, aproximadamente, 15 FPS). O tamanho dos modelos, quantidade de juntas e pontos de controle são apresentados na Tabela 4.1.

Modelo	Vértices	Juntas	Pontos de controle
Armadillo	165K	33	5
Elephant	185K	29	5
Hand	195K	32	6
Dragon	247K	26	5

Tabela 4.1: Dados dos modelos da Figura 4.12.

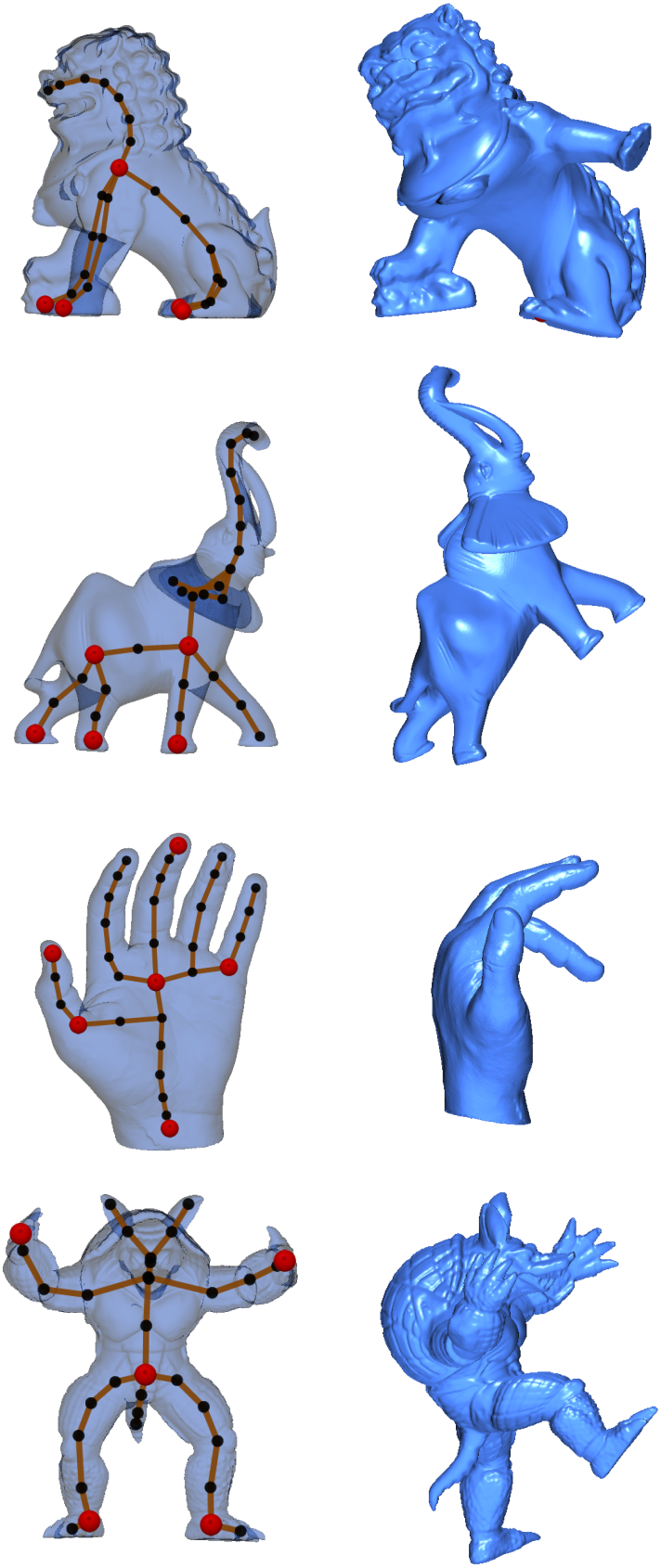


Figura 4.12: Exemplos de deformações.

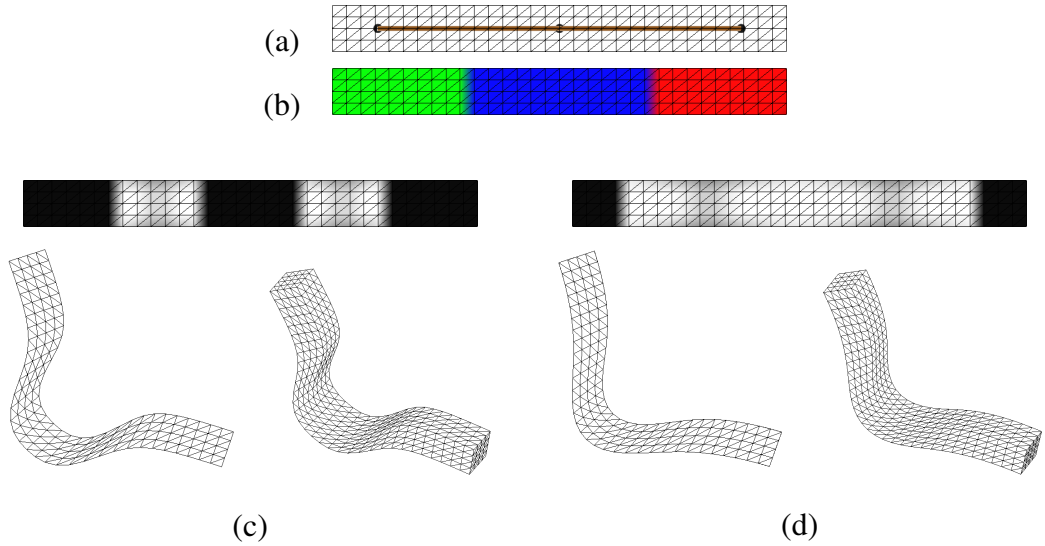


Figura 4.13: Distorções obtidas utilizando diferentes distribuições de pesos. (a) Um modelo de barra e seu esqueleto. (b) A malha segmentada, onde cada vértice é associado à junta mais próxima. (c) e (d) ilustram duas distribuições de pesos e as correspondentes deformações.

4.3.1 Limitações

O uso único de restrições posicionais pode ser insuficiente para expressar certas poses. Por exemplo, a rotação da mão em torno do eixo do antebraço não pode ser realizada se os ossos da mão não forem transversais aos do antebraço.

Embora o esquema proposto seja muito barato computacionalmente, a qualidade dos resultados são, de certa maneira, visualmente inferiores se comparados a métodos sofisticados de deformação, tais como aqueles baseados em técnicas Laplacianas [BS08] ou métodos não-lineares [BPWG07].

Outra questão importante é o fato de que as transformações são computadas somente nas posições das juntas e depois projetadas sobre a superfície. Isto pode resultar em dobras não naturais como mostrado na Figura 4.13, as quais podem ser aliviadas por um processo de determinação de pesos mais cuidadoso. Alternativamente, o esquema pode ser modificado para que as transformações sejam computadas para os ossos e combinadas nas juntas. Deve ser notado, entretanto, que esta limitação é herdada do esquema clássico de LBS.

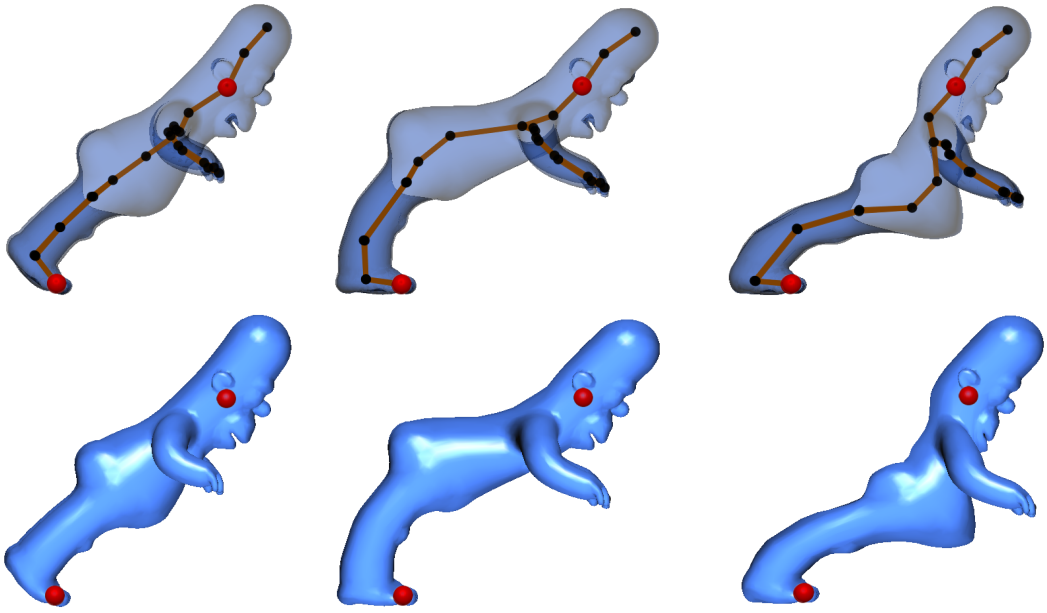
4.4 Controle de dobra

A proposta para o controle de dobra se diferencia do esquema de deformação padrão (Capítulo 3) porque computa as transformações ótimas sobre a estrutura de um esqueleto, ao invés de computá-las diretamente nos vértices do modelo. A idéia é semelhante ao método baseado em distâncias geodésicas discutido anteriormente, mas, neste caso, as distâncias são medidas ao longo dos ossos e juntas do esqueleto. Mudanças no esqueleto são propagadas pela malha usando LBS.

A idéia de influenciar o comportamento de dobra aparece da observação da componente de translação do método. Observando as equações (3.9) e (3.10), pode-se notar que a simples mudança do valor w_i não altera a forma do conjunto **q***. Em outras palavras, considere três juntas A, B e C colineares em um esqueleto. Alterar o comprimento dos ossos pode variar suas distâncias relativas, porém, se a rotação for ignorada, essas juntas continuarão sendo colineares. Isto sugere que o cômputo dos **q*** seja feito com pesos w_i diferentes dos usados para computar **p***. Dessa forma, a equação para **q*** pode ser reescrita assim:

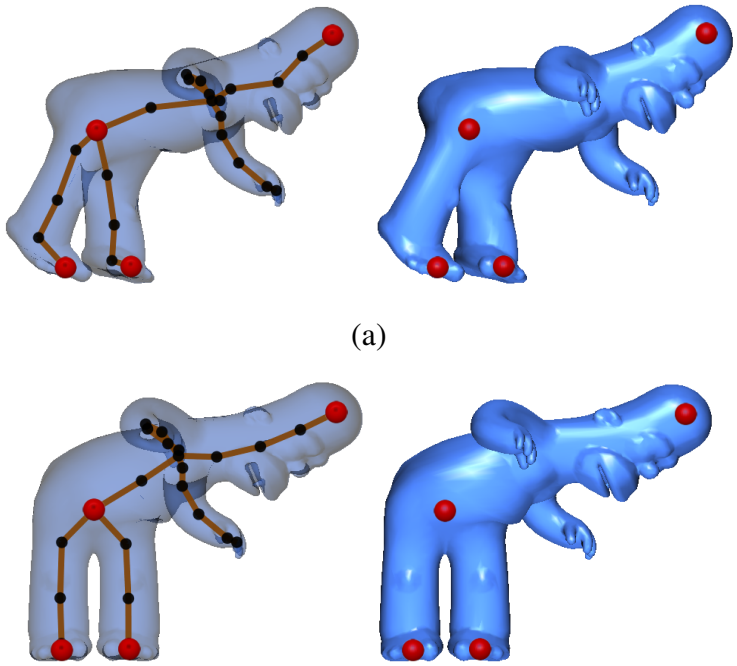
$$\mathbf{q}^* = \mathbf{p}^* + \frac{\sum_i w'_i (\mathbf{q}_i - \mathbf{p}_i)}{\sum_i w'_i}. \tag{4.1}$$

Observe que, quando $w'_i = w_i$, a formulação é idêntica a Equação (3.10). O que resta fazer é estabelecer valores w'_i convenientes para obter um comportamento de dobra adequado. Pode-se observar que, incrementar e decrementar valores ao longo de uma seqüência de juntas entre dois pontos de controle produz parábolas de diferentes concavidades. A Figura 4.14 ilustra este efeito no modelo Homer. Por outro lado, o uso de pesos constantes numa seqüência de juntas não altera a forma do esqueleto, mas pode diminuir a influência da deformação nas outras juntas. Isto pode ser visto na Figura 4.15.



(a) (b) (c)

Figura 4.14: (a) Deformação sem controle de dobra; (b) utilizando pesos decrescentes (dos pés à cabeça) nas juntas , e (c) utilizando pesos crescentes.



(a)

(b)

Figura 4.15: Sem controle de dobra, o movimento da cabeça influencia os pés (a). O uso de pesos constantes isola as pernas do movimento do torso (b).

Capítulo 5

Conclusões

Este trabalho apresenta uma formulação prática para deformar modelos 3D usando transformações rígidas, as quais são computadas como soluções de um problema de otimização via mínimos quadrados móveis (MLS). O uso de transformações rígidas permite gerar resultados plausíveis, confirmando as observações de Schaefer et al. [SMW06]. Em algumas situações, ver Figura 3.13, estes são superiores aos obtidos por métodos baseados em minimização de energía variacional [BK04] ou baseados em RBFs [BK05a]. Não obstante, é possível obter deformações de menor qualidade do que os obtidos por métodos não lineares [BPWG07]. Isso pode ser observado quando são comparados os resultados da Figura 3.14 frente as deformações da Figura 2.8 (retirada do artigo de Botsch e Sorkine [BS08]). Com efeito, o método proposto se restringe a minimizar distâncias entre pontos, enquanto que técnicas não lineares podem ser mais flexíveis incorporando a otimização de outras propriedades tais como volume, curvatura, etc.

O posicionamento dos pontos de controle é crucial na determinação da aparência dos modelos deformados. Algumas poses não poderiam ser obtidas sem estabelecer a utilização de pontos de controle em lugares estratégicos. Veja, por exemplo, a importância dos pontos de controle nos quadris e no umbigo na deformação dos modelos "Girl" e "Homer" nas figuras 3.3 e 3.15.

Vários exemplos (ver figuras 3.9 e 3.10) mostram o efeito da presença/ausência da componente rotacional na formulação. Pode-se observar deformações bem mais plausíveis do que as obtidas quando somente é computada a componente de translação. Embora o custo computacional do algoritmo proposto seja da ordem $O(mn)$, onde m é o número de pontos de controle e n é o número de vértices do modelo, é possível observar

que as operações aritméticas necessárias para o cálculo de rotações ótimas dominam a complexidade computacional do algoritmo (ver Figura 3.11) e, portanto, devem ser computadas da maneira mais eficiente possível.

Experimentos mostram que a formulação proposta para computar a componente rotacional é mais eficiente (ver Figura 3.2) que formulações clássicas que computam uma matriz de rotação como sendo $\mathbf{M}^{\mathsf{T}}(\mathbf{M}\mathbf{M}^{\mathsf{T}})^{-1/2}$ [HHN88], e usa menos operações que representações baseadas em quatérnios como sugerido por Kanatani [Kan94]. Ademais, é ligeiramente mais eficiente que a solução por decomposição em valores singulares (SVD) [AHB87]. Uma desvantagem da técnica é que, diferente da solução baseada em SVD, o algoritmo funciona somente para o caso 3D. Essa mesma desvantagem é apresentada pelas formulações baseadas em matrizes ortonormais e quatérnios unitários.

A natureza fechada da formulação faz com que seja de simples implementação e adequada para processamento em arquiteturas paralelas. As taxas de quadros por segundo (FPS) mostradas na tabela da Figura 3.19 permitem verificar a alta interatividade alcançada pelas implementações em GPU. No pior caso, as implementações usando *shaders* se mostraram 20 vezes mais rápidas que a implementação convencional, enquanto que a implementação em CUDA mostrou-se 37 vezes mais eficiente. No melhor caso, as implementações com *shaders* e em CUDA são 97 e 129 vezes mais rápidas, respectivamente.

Uma outra questão interessante é o efeito da métrica de distância na formulação do método. Por exemplo, na Figura 4.2 contrasta-se a deformação obtida usando uma métrica euclidiana frente a outra que usa o algoritmo de Dijkstra para aproximar uma distância geodésica sobre a malha. Pode-se concluir que a métrica na superfície produz, ao menos em alguns casos, resultados mais plausíveis do que os obtidos usando a métrica euclidiana. Outros exemplos de deformações obtidas com este enfoque podem ser vistas na Figura 4.6.

Adicionalmente, foi proposto um esquema de deformação guiado por esqueletos. O algoritmo original de deformação do espaço foi transformado em um que é mais sensível à forma do modelo. As poses são obtidas arrastando pontos de controle posicionados nas juntas de um esqueleto. O método emprega um procedimento de *rigging* que requer operações relativamente simples, a saber, a computação de distâncias de vértices a juntas e um processo de expansão progressiva no grafo da malha. Os resultados mostram

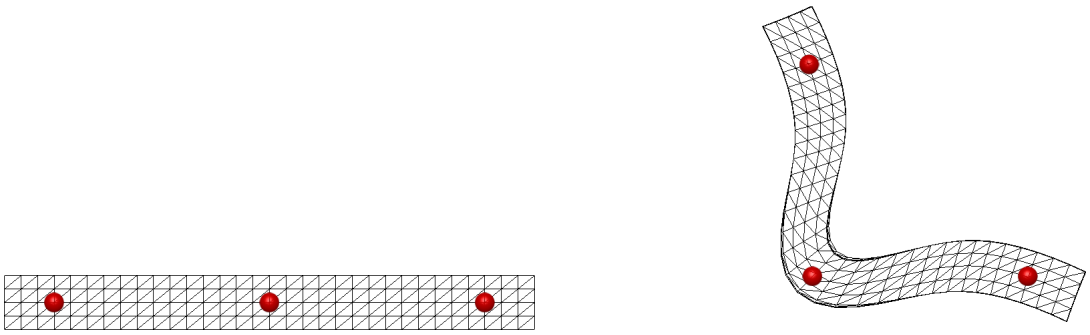


Figura 5.1: Uma barra com três pontos de controle. Deformação não natural obtida após movimentar um ponto de controle.

deformações mais plausíveis quando comparadas com as obtidas com a formulação original. O método proposto pode ainda ser compreendido como uma boa alternativa à cinemática inversa padrão, já que depende menos da estrutura dos esqueletos, tornando possível o emprego de esqueletos grosseiros ou construídos com processos automatizados.

Esta abordagem também é mais eficiente do que o esquema de deformação original, porque apenas as juntas dos esqueletos são diretamente transformadas pela formulação MLS, tornando possível deformar modelos compostos de até pouco mais de cem mil vértices interativamente.

Por último, é mostrado como um esquema de pesos pode ser usado para afetar o resultado da deformação. A idéia consiste na manipulação do posicionamento dos centróides $\mathbf{q}*$. Resultados preliminares de como obter maior controle nas dobras são apresentados nas figuras 4.14 e 4.15.

5.1 Trabalhos futuros

Uma primeira extensão refere-se à melhora da qualidade da deformação. Veja, por exemplo, na Figura 5.1 um resultado não natural obtido ao se tentar dobrar uma barra usando três pontos de controle. Nesse exemplo dois problemas podem ser observados. O primeiro são as dobras na parte interna da região côncava formada pela barra dobrada. Num resultado mais natural, as dobras seriam formadas em sentido contrário ou não seriam verificadas de todo. O segundo problema é o fato de que o movimento do ponto de controle da esquerda afeta vértices afastados, como os próximos do ponto de controle mais à direita. Numa deformação mais natural, os pontos de controle em movimento de-

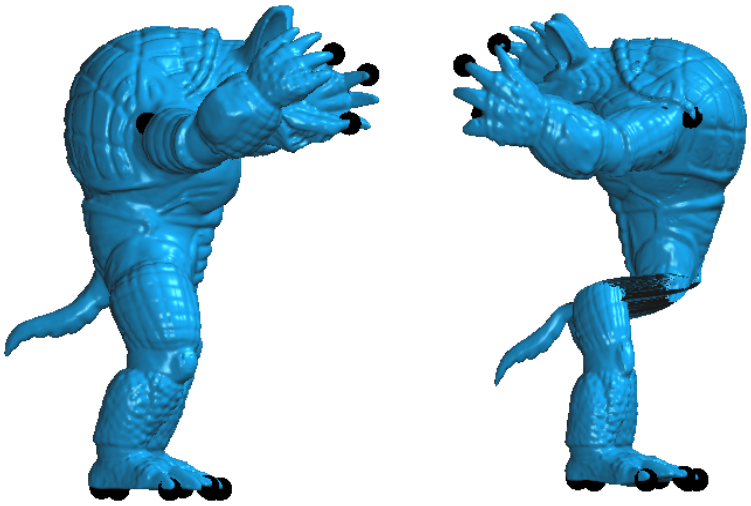


Figura 5.2: Descontinuidade nas deformações. Depois de rodar a parte superior do modelo, os vértices na cintura são deformados não suavemente.

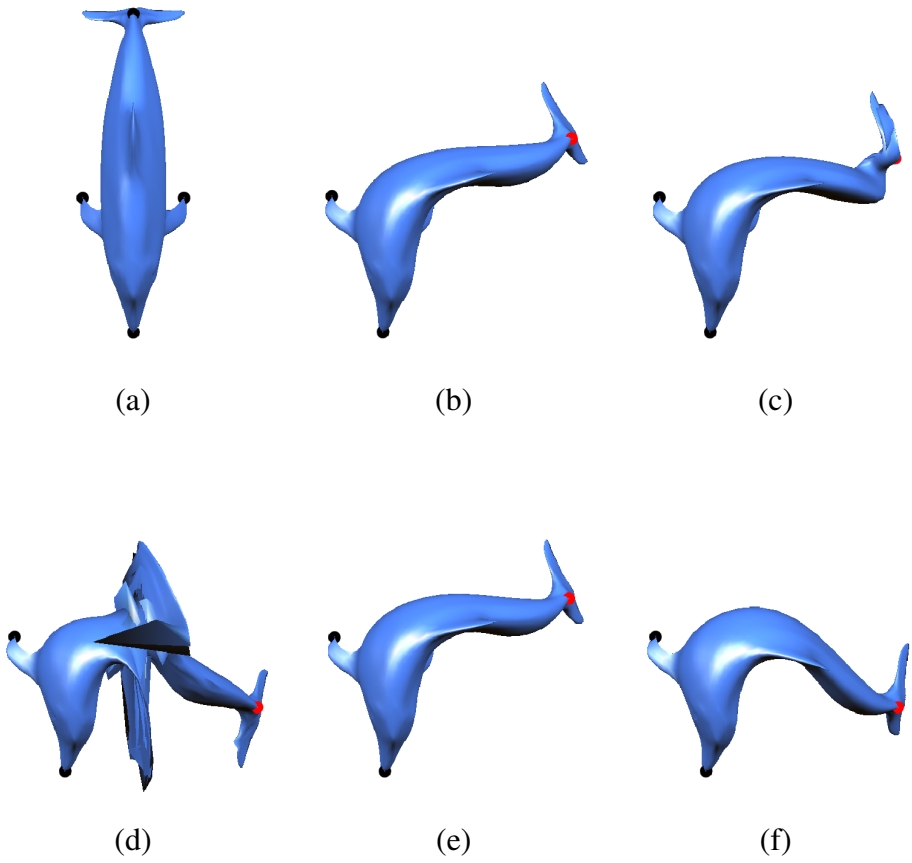
vem afetar somente vértices próximos. A penalização de outras propriedades além das distâncias entre pontos de controle e vértices, ajudaria a evitar estes problemas e melhoraria a qualidade dos resultados. Por exemplo, pode-se tentar otimizar a rigidez das faces dos triângulos ou a rigidez do conjunto de triângulos incidentes a um vértice. Uma idéia similar foi implementada por Sorkine e Alexa [SA07], obtendo resultados atraentes.

Uma segunda questão que pode ser abordada é a manutenção da consistência das transformações calculadas para vértices vizinhos. A abordagem proposta não garante que vértices vizinhos tenham soluções próximas. Por exemplo, na deformação da Figura 5.2 os vértices da cintura do Armadillo foram deformados bruscamente. Para lidar com isso, pode-se introduzir um regularizador que penalize a inconsistência de vizinhanças. Sumner et al. introduziram uma técnica para abordar questões semelhantes [SSP07].

O algoritmo para aproximar a métrica geodésica deve ser aprimorado. Uma implementação da proposta de Surazhsky et al. [SSK+05] pode ser uma ferramenta útil para futuras propostas de ferramentas de deformação.

Interfaces simples para o controle da deformação devem ser ainda pesquisadas. A especificação e manipulação podem ser feitas com outros paradigmas ou usando outros objetos de controle: segmentos de reta, curvas, traços, quadriláteros, etc.

Embora as implementações em GPU tenham conseguido diminuir o tempo de processamento, ainda é possível estudar outras questões relacionadas com eficiência e desempenho. Uma idéia possível consiste em escrever um esquema incremental para o cômputo



(a) (b) (c)

(d) (e) (f)

Figura 5.3: Reinicializando o processo. A aproximação trabalha corretamente se $\det(\mathrm{M}) > 0$ (b), mas produz distorções quando $\det(\mathrm{M}) \leq 0$ (veja (c) e (d)). A reinicialização do processo pode corrigir as distorções (veja (e) e (f)).

das rotações ótimas. Por exemplo, pode-se reescrever a Equação 3.30 de tal maneira que $\mathbf{u}$ chegue a ser a única incógnita:

$$(\mathrm{N} - (2\mathrm{E} - \mathrm{V}\mathbf{u}^{\mathsf{T}})\,\mathrm{I})\,\mathbf{u}^{\mathsf{T}} = \mathrm{V}^{\mathsf{T}}. \tag{5.1}$$

Seja k um índice que se refere ao k-ésimo quadro de uma sessão de deformação. Durante a manipulação interativa, onde a diferença entre as posições dos pontos de controle $\{\mathbf{q}_i\}_{k-1}$ e $\{\mathbf{q}_i\}_k$ é mínima, pode-se assumir que $\mathbf{u}_{k-1} \sim \mathbf{u}_k$. Isto sugere a linearização de (5.1), permitindo-nos obter $\mathbf{u}_k$ diretamente de

$$(\mathrm{N}_k - (2\mathrm{E}_k - \mathrm{V}_k\mathbf{u}_{k-1}^{\mathsf{T}})\,\mathrm{I})\,\mathbf{u}_k^{\mathsf{T}} = \mathrm{V}_k^{\mathsf{T}}, \tag{5.2}$$

o que evita resolver a equação de quarto grau. No entanto, reinicializações são necessárias nesta aproximação devido às descontinuidades da função de rotação. Tais descontinuidades produzem instabilidades no sistema (5.2) gerando efeitos desagradáveis conforme ilustrado na Figura 5.3.

Figura 5.4: Deformação de um modelo 3D usando uma abordagem guiada por esqueletos. O modelo é colocado numa pose onde é possível animá-lo usando movimento capturado.

A animação de personagens aparece como uma área candidata para fazer uso de uma ferramenta de deformação. Por exemplo, em muitas situações, é necessário deformar um modelo para colocá-lo numa pose (pose de referência) na qual seja possível aplicar dados de movimento capturado. Veja, por exemplo, na Figura 5.4 como um esquema de deformação guiado por esqueletos pode aliviar este problema.

Finalmente, é importante estudar como introduzir efeitos, tais como materiais heterogêneos ou deformações anisotrópicas, na formulação. Tais questões são importantes quando deseja-se simular características específicas da pele, por exemplo, músculos.

Por último, no processo de *skinning* a combinação das transformações pode ser efetuada usando quatérnios duais que não apresentam os problemas clássicos de métodos de combinação linear como o efeito conhecido como "papel de bala" *candy wrapper* decorrente de operações de torção ou o colapso de geometria durante operações de dobra.

Apêndice A

Processamento paralelo de baixo custo

Embora arquiteturas de processamento paralelo tenham sido introduzidas há décadas, elas ainda estão restritas a grandes instituições devido, principalmente, a seu alto preço. Recentemente, arquiteturas paralelas de baixo custo estão se tornando atrativas já que, além de oferecer um alto poder computacional, podem ainda ser adquiridas por um público mais amplo. Entre as alternativas com maior popularidade destacam-se: GPUs (*General Processing Units*) e processadores de múltiplos núcleos (*multi-core processors*).

A.1 GPUs

A potência, o baixo custo e a alta flexibilidade das GPUs modernas, tornou-as uma alternativa acessível para resolver diversos problemas que requerem a execução de um elevado volume de operações aritméticas em paralelo, que é o caso da deformação MLS. Uma GPU é um processador de múltiplos núcleos que pode ser usado como um co-processador da CPU, pois tem seu próprio espaço de memória e executa paralelamente um grande número de *threads*[1].

Tipicamente, o desenvolvimento de aplicações de caráter não gráfico (GPGPU[2]) é feito usando APIs gráficas (OpenGL ou DirectX) por meio da codificação de *shaders*[3]. Os *shaders* podem atuar em três partes do *pipeline* gráfico [SWND05], cuja funcionalidade pode ser resumida da seguinte forma:

[1]Neste trabalho, a palavra inglesa *thread* significa "linha de execução" e é responsável pela tarefa mais simples a ser computada em GPU.

[2]GPGPU é o acrônimo do termo em inglês *General-Purpose computation on GPUs*.

[3]A palavra *shader* refere-se a "programa de computador" no contexto de programação em placa gráfica.

- Primeiro, uma aplicação em CPU envia vértices de primitivas geométricas, isto é, triângulos, para a placa gráfica.

- Em seguida, diversas operações por vértice são realizadas em paralelo pelos processadores de vértices. Neste ponto, um **shader de vértice** pode ser utilizado substituindo a funcionalidade fixa da placa gráfica que realiza transformações geométricas, cálculos de iluminação, etc.

- Os vértices processados são então enviados para o **shader de geometria** que realiza a montagem de primitivas. Neste passo, primitivas podem ser geradas ou excluídas, por exemplo por operações de recorte.

- As primitivas são então *rasterizadas*[4], que é um processo que preenche cada primitiva gerando seus pixels, mais corretamente chamado de pré-pixels ou fragmentos, pois ainda não formam os pixels finais do *frame buffer*[5].

- Os fragmentos são enviados para os processadores de fragmentos onde um **shader de fragmento** pode ser utilizado. Neste ponto, a funcionalidade fixa realiza o mapeamento de textura, processo no qual *texels* (elementos de textura) são lidos da memória da placa gráfica e mapeados nos respectivos fragmentos de acordo com suas coordenadas de textura.

- Finalmente, os fragmentos coloridos, por textura ou simples atribuição de cor, são enviados para o processo de composição responsável por agregá-los em uma matriz de pixels, chamada de *frame buffer*. Os fragmentos que caem no mesmo pixel podem ser compostos por uma função de mistura ou descartados por uma função de profundidade. O conteúdo do *frame buffer* é, normalmente, mostrado na janela da aplicação.

A memória da placa gráfica, também chamada de memória de textura, pode ser acessada por qualquer um dos três *shaders*. O acesso à memória de textura pelos *shaders* é restrito a leitura somente ou só gravação e incorre numa latência maior que no acesso à

[4]O verbo *rasterizar* é um estrangeirismo derivado da palavra inglesa *rasterize*, que significa converter representações vetoriais em matriciais.

[5]Um *frame buffer* é uma porção de memória da placa gráfica usada como repositôrio de resultados produzidos por um *fragment shader*.

memória RAM na CPU, pois a arquitetura da placa gráfica é dedicada para maximizar o desempenho de cálculos aritméticos em detrimento do acesso à memória [PF05]. Por este motivo, a intensidade aritmética, conceito definido pela razão de operações aritméticas por quantidade de acesso à memória, deve ser maximizada para obter um desempenho maior ao utilizar programação em GPU.

Embora linguagens de programação de alto nível tenham sido desenvolvidas (Cg, HLSL e GLSL), a adaptação de um problema arbitrário num contexto gráfico dificulta a adoção em larga escala desta tecnologia. As primeiras propostas que buscavam desvincular a programação de GPUs de definições usadas no contexto gráfico foram: BrookGPU [BFH+04], Sh [MTP+04] e Accelerator [TPO06]. Recentemente, propostas que permitem um entendimento menos restrito de conceitos da computação gráfica e que procuram o desenvolvimento de software comercial têm aparecido, por exemplo: RapidMind [MD06], PeakStream [Pap07], além de CTM e CUDA das companhias AMD© e NVIDIA©, respectivamente.

No final de 2006, a NVIDIA introduziu o chamado CUDA [NVI07] (*Compute Unified Device Architecure*), que é uma tecnologia para programação em GPU, a qual oferecer uma plataforma que faz uso eficiente dos recursos da placa gráfica e que permite implementações independentes do pipeline gráfico. Em CUDA as implementações são feitas através de *kernels*, os quais são executados por múltiplas *threads*. As *threads* são agrupadas em blocos, onde compartilham os recursos de um multiprocessador da GPU. Cada multi-processador possui uma arquitetura SIMD (*Single Instruction Multiple Data*) que executa as mesmas instruções do *kernel*, porém em dados diferentes.

A.2 CPUs de múltiplos núcleos

Processadores de múltiplos núcleos são chips que contêm dois ou mais processadores no mesmo circuito integrado. Embora independentes, foram projetados para processar múltiplas instruções simultaneamente e para fazer uso compartilhado da memória . Normalmente, cada processador possui uma área de memória reservada ou *cache* nível 1. Adicionalmente, uma área de memória compartilhada conecta múltiplos núcleos através de uma *cache* de nível 2.

Um dos principais objetivos para o desenvolvimento desta arquitetura foi melhorar a

relação entre energia utilizada e desempenho, o que é alcançado com núcleos rodando em baixas freqüências de trabalho. Tal estratégia divide a energia gasta por um só processador entre os diversos núcleos.

Existem várias bibliotecas projetadas para facilitar o desenvolvimento de aplicações *multi-core*. Entre as mais usadas temos: Pthreads [Law95], TBB (*Threading Building Blocks*) [INT08], STAPL (*Standard Template Adaptive Parallel Library*) [RAO98], MC-STL (*The Multi-Core Standard Template Library*) [PKT06], etc.

Referências Bibliográficas

[AB97] Fabrice Aubert and Dominique Bechmann. Volume-preserving space deformation. *Computer Graphics*, 21(5):625–639, 1997.

[ACOL00] M. Alexa, D. Cohen-Or, and D. Levin. As-rigid-as-possible shape interpolation. *Proceedings of SIGGRAPH 2000*, pages 157–164, July 2000. New Orleans, Louisianna USA, July 23-28.

[ACWK06] Alexis Angelidis, Marie-Paule Cani, Geoff Wyvill, and Scott King. Swirling-sweepers: constant-volume modeling. *Graphics Models*, 68(4):324–332, 2006.

[AHB87] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(5):698–700, 1987.

[Ale06] Marc Alexa. Interactive shape editing. ACM SIGGRAPH Courses, 2006.

[Bar84] Alan H. Barr. Global and local deformations of solid primitives. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 21–30, New York, NY, USA, 1984. ACM Press.

[BFH+04] Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, and Pat Hanrahan. Brook for gpus: stream computing on graphics hardware. *ACM Trans. Graph.*, 23(3):777–786, 2004.

[BK03] G. H. Bendels and R. Klein. Mesh forging: editing of 3d-meshes using implicitly defined occluders. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 207–217, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[BK04] Mario Botsch and Leif Kobbelt. An intuitive framework for real-time freeform modeling. *ACM Trans. Graph.*, 23(3):630–634, 2004.

[BK05a] Mario Botsch and Leif Kobbelt. Real-time shape editing using radial basis functions. *Computer Graphics Forum*, 24(3):611–621, 2005.

[BK05b] Mario Botsch and Leif Kobbelt. Real-time shape editing using radial basis functions. *Computer Graphics Forum*, 24(3):611–621, 2005.

[Bla94] Caroline Blanc. A generic implementation of axial procedural deformation techniques. In *Graphics Gems*, pages 249–256, New York, 1994. Academic Press.

[Bli82] James F. Blinn. A generalization of algebraic surface drawing. *ACM Trans. Graph.*, 1(3):235–256, 1982.

[Bli06] James F. Blinn. How to solve a cubic equation, part 1: The shape of the discriminant. *IEEE Comput. Graph. Appl.*, 26(3):84–93, 2006.

[BM92] P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and machine Intelligence*, 14(2):239–258, February 1992.

[BN92] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 35–42, New York, NY, USA, 1992. ACM Press.

[BO08] Fausto Richetti Blanco and Manuel M. Oliveira. Instant mesh deformation. In *SI3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pages 71–78, New York, NY, USA, 2008. ACM.

[Boo89] F. L. Bookstein. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(6):567–585, 1989.

[BP07] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3d characters. *ACM Trans. Graph.*, 26(3):72, 2007.

[BPGK06] Mario Botsch, Mark Pauly, Markus Gross, and Leif Kobbelt. Primo: Coupled prisms for intuitive surface modeling. In *Eurographics Symposium on Geometry Processing*, pages 11–20. Eurographics, 2006.

[BPK+08] Mario Botsch, Mark Pauly, Leif Kobbelt, Pierre Alliez, and Bruno Levy. Geometric modeling based on polygonal meshes. In *EUROGRAPHICS 2008: Tutorials*, 2008.

[BPWG07] Mario Botsch, Mark Pauly, Martin Wicke, and Markus Gross. Adaptive space deformations based on rigid cells. *Computer Graphics Forum*, 26(3):339–347, September 2007.

[BS08] Mario Botsch and Olga Sorkine. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics*, 14(1):213–230, 2008.

[Buh03] M. D. Buhmann. *Radial Basis Functions: Theory and Implementations*. Cambridge University Press, United Kingdom, 2003.

[CBC+01] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 67–76, New York, NY, USA, 2001. ACM Press.

[CG91] George Celniker and Dave Gossard. Deformable curve and surface finite-elements for free-form shape design. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 257–266, New York, NY, USA, 1991. ACM.

[CH90] Jindong Chen and Yijie Han. Shortest paths on a polyhedron. In *SCG '90: Proceedings of the sixth annual symposium on Computational geometry*, pages 360–369, New York, NY, USA, 1990. ACM.

[CIJ+05] Min Chen, Shoukat Islam, Mark W. Jones, Peiyi Shen, Deborah Silver, Simon J. Walton, and Phil J. Willis. Deforming and animating discretely sam-

pled object representations. In *Eurographics 2005, STAR Reports*, pages 113–140, 2005.

[CJ91] Sabine Coquillart and Pierre Jancene. Animated free-form deformation: an interactive animation technique. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 23–26, New York, NY, USA, 1991. ACM Press.

[CM07] Nicu D. Cornea and Patrick Min. Curve-skeleton properties, applications, and algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):530–548, 2007.

[Coq90] Sabine Coquillart. Extended free-form deformation: a sculpturing tool for 3d geometric modeling. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 187–196, New York, NY, USA, 1990. ACM Press.

[Duc77] J. Duchon. Spline minimizing rotation-invariant semi-norms in sobolev spaces. *Constructive Theory of Functions of Several Variables*, pages 85–100, 1977.

[ELF97] D. W. Eggert, A. Lorusso, and R. B. Fisher. Estimating 3-d rigid body transformations: a comparison of four major algorithms. *Mach. Vision Appl.*, 9(5-6):272–290, 1997.

[FÔ3] Martin Fêdor. Application of inverse kinematics for skeleton manipulation in real-time. In *SCCG '03: Proceedings of the 19th spring conference on Computer graphics*, pages 203–212, New York, NY, USA, 2003. ACM.

[FAT06] Hongbo Fu, Oscar Kin-Chung Au, and Chiew-Lan Tai. Effective derivation of similarity transformations for implicit laplacian mesh editing. *Computer Graphics Forum (to appear)*, 2006.

[FO06] S. Forstamann and J. Ohya. Fast skeletal animation by skinned arc-spline based deformation. In *EuroGraphics 2006 Short Papers*, 2006.

[FOKGM07] Sven Forstmann, Jun Ohya, Artus Krohn-Grimberghe, and Ryan McDougall. Deformation styles for spline-based skeletal animation. In *SCA*

'07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation, pages 141–150, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[GL89] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 1989.

[HHK92] William M. Hsu, John F. Hughes, and Henry Kaufman. Direct manipulation of free-form deformations. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 177–184, New York, NY, USA, 1992. ACM Press.

[HHN88] Berthold K. P. Horn, Hugh M. Hilden, and Shahriar Negahdaripour. Closed form solutions of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America*, 5(7):1127–1135, 1988.

[HML99] Gentaro Hirota, Renee Maheshwari, and Ming C. Lin. Fast volume-preserving free form deformation using multi-level optimization. In *SMA '99: Proceedings of the fifth ACM symposium on Solid modeling and applications*, pages 234–245, New York, NY, USA, 1999. ACM Press.

[Hor87] B. K. P. Horn. Closed form solutions of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4(4):629–642, April 1987.

[IMH05] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. As-rigid-as-possible shape manipulation. *ACM Trans. Graph.*, 24(3):1134–1141, 2005.

[INT08] INTEL. Intel Threading Building Blocks for Open Source. `http://www.threadingbuildingblocks.org/`, 2008.

[Kan94] K. Kanatani. Analysis of 3-d rotation fitting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(5):543–549, 1994.

[Kan00] Takashi Kanai. Approximate shortest path on a polyhedral surface and its applications. In *Computer-Aided Design*, pages 801–811, 2000.

[Kap99] Sanjiv Kapoor. Efficient computation of geodesic shortest paths. In *STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 770–779, New York, NY, USA, 1999. ACM.

[KCVS98] Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 105–114, New York, NY, USA, 1998. ACM Press.

[Kob97] Leif Kobbelt. Discrete fairing. In *Proceedings on 7th IMA Conference on the Mathematics of Surfaces*, pages 101–137, New York, NY, USA, 1997. ACM Press.

[KVS99] L. Kobbelt, J. Vorsatz, and H.-P. Seidel. Multiresolution hierarchies on structured triangle mesh. *Computational Geometry Journal: Theory and Applications*, 14:5–24, 1999.

[LAP90] LAPACK. Linear Algebra PACKage. `http://www.netlib.org/lapack/`, 1990.

[Law95] Lawrence Livermore National Laboratory. POSIX Threads Programming. `https://computing.llnl.gov/tutorials/pthreads/`, 1995.

[LCOGL06] Yaron Lipman, Daniel Cohen-Or, Ran Gal, and David Levin. Volume and shape preservation via moving frame manipulation. *Technical report*, 2006.

[Lev98] David Levin. The approximation power of moving least-squares. *Math. Comput.*, 67(224):1517–1531, 1998.

[LS81] P. Lancaster and K. Salkauskas. Surfaces generated by moving least squares methods. *Mathematics of Computation*, 37(155):141–158, 1981.

[LSCO+04] Yaron Lipman, Olga Sorkine, Daniel Cohen-Or, David Levin, Christian Rössl, and Hans-Peter Seidel. Differential coordinates for interactive mesh editing. In Franca Giannini and Alexander Pasko, editors, *Shape Modeling International 2004 (SMI 2004)*, pages 181–190, Genova, Italy, 2004. IEEE.

[LSLCO05] Yaron Lipman, Olga Sorkine, David Levin, and Daniel Cohen-Or. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph.*, 24(3):479–487, 2005.

[MB03] Leif Kobbelt Mario Botsch. Multiresolution surface representation based on displacement volumes. *Computer Graphics Forum*, 22(3):483–491, 2003.

[MD06] Michael D. McCool and Bruce D'Amora. Programming using rapidmind on the cell be. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 222, New York, NY, USA, 2006. ACM.

[MJ96] Ron MacCracken and Kenneth I. Joy. Free-form deformations with lattices of arbitrary topology. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 181–188, New York, NY, USA, 1996. ACM Press.

[MMP87] Joseph S. B. Mitchell, David M. Mount, and Christos H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16(4):647–668, 1987.

[MS92] Henry P. Moreton and Carlo H. Sequin. Functional optimization for fair surface design. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 167–176, New York, NY, USA, 1992. ACM Press.

[MTP+04] Michael McCool, Stefanus Du Toit, Tiberiu Popa, Bryan Chan, and Kevin Moule. Shader algebra. *ACM Trans. Graph.*, 23(3):787–795, 2004.

[MTPS08] Johannes Mezger, Bernhard Thomaszewski, Simon Pabst, and Wolfgang Strasser. Interactive physically-based shape editing. In *Proceedings of ACM Solid and Physical Modeling Symposium*, 2008.

[MVC04] Dimas Martinez, Luiz Velho, and Paulo Cezar Carvalho. Geodesic paths on triangular meshes. In *SIBGRAPI '04: Proceedings of the Computer Graphics and Image Processing, XVII Brazilian Symposium*, pages 210–217, Washington, DC, USA, 2004. IEEE Computer Society.

[MYC+01] Bryan S. Morse, Terry S. Yoo, David T. Chen, Penny Rheingans, and K. R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *SMI '01: Proceedings of the International Conference on Shape Modeling & Applications*, page 89, Washington, DC, USA, 2001. IEEE Computer Society.

[NMK+05] Andrew Nealen, Matthias Miller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics. In *Eurographics 2005, STAR Reports*, 2005.

[NR92] NR. Numerical Recipes. `http://www.nr.com/`, 1992.

[NVI07] NVIDIA Corporation. CUDA Environment - Compute Unified Device Architecture. `http://www.nvidia.com/object/cuda_home.html`, 2007.

[OBS04] Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. 3d scattered data approximation with adaptive compactly supported radial basis functions. In Franca Giannini and Alexander Pasko, editors, *Shape Modeling International 2004 (SMI 2004)*, pages 31–39, Genova, Italy, June 2004. IEEE.

[Pap07] Matthew Papakipos. The peakstream platform: High-productivity software development for gpus. In *Proceedings of LCI International Conference on High-Performance Clustered Computing*, 2007.

[PF05] Matt Pharr and Randima Fernando. *GPUGems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley, USA, 2005.

[PHYH06] Helmut Pottmann, Qi-Xing Huang, Yong-Liang Yang, and Shi-Min Hu. Geometry and convergence analysis of algorithms for registration of 3d shapes. *Int. J. Comput. Vision*, 67(3):277–296, 2006.

[PKKG03] Mark Pauly, Richard Keiser, Leif P. Kobbelt, and Markus Gross. Shape modeling with point-sampled geometry. *ACM Trans. Graph.*, 22(3):641–650, 2003.

[PKT06] Felix Putze, Universität Karlsruhe, and Thread Sun T. Mcstl: The multi-core standard template library. `http://algo2.iti.uni-karlsruhe.de/singler/mcstl/`, 2006.

[RAO98] Lawrence Rauchwerger, Francisco Arzu, and Koji Ouchi. Standard templates adaptive parallel library. In *In Proc. of the 4th International Workshop on*

Languages, Compilers and Run-Time Systems for Scalable Computers (LCR, pages 402–409. Springer-Verlag, 1998.

[RM93] D. Ruprecht and H. Mueller. Free form deformation with scattered data interpolation methods. *Springer Computing Supplementum*, 8:267–281, 1993.

[RM95] Detlef Ruprecht and Heinrich Mueller. Image warping with scattered data interpolation. *IEEE Comput. Graph. Appl.*, 15(2):37–43, 1995.

[RSB95] Ari Rappoport, Alla Sheffer, and Michel Bercovier. Volume-preserving free-form solid. In *SMA '95: Proceedings of the third ACM symposium on Solid modeling and applications*, pages 361–372, New York, NY, USA, 1995. ACM Press.

[SA07] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *SGP' 07: Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 109–116, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[SCOL+04] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel. Laplacian surface editing. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 175–184, New York, NY, USA, 2004. ACM Press.

[SD92] Ken Shoemake and Tom Duff. Matrix animation and polar decomposition. In *Proceedings of the conference on Graphics interface '92*, pages 258–264, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[She68] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524, New York, NY, USA, 1968. ACM Press.

[SMW06] Scott Schaefer, Travis McPhail, and Joe Warren. Image deformation using moving least squares. *ACM Trans. Graph.*, 25(3):533–540, 2006.

[SP86] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph.*, 20(4):151–160, 1986.

[SP04] Robert W. Sumner and Jovan Popovic. Deformation transfer for triangle meshes. *ACM Trans. Graph.*, 23(3):399–405, 2004.

[SSK+05] Vitaly Surazhsky, Tatiana Surazhsky, Danil Kirsanov, Steven J. Gortler, and Hugues Hoppe. Fast exact and approximate geodesics on meshes. *ACM Trans. Graph.*, 24(3):553–560, 2005.

[SSP07] Robert W. Sumner, Johannes Schmid, and Mark Pauly. Embedded deformation for shape manipulation. *ACM Transactions on Graphics*, 26(3):80:1–80:7, July 2007.

[SWND05] Dave Shreiner, Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2.0*. Addison-Wesley Professional, Boston, MA, USA, 2005.

[TPBF87] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 205–214, New York, NY, USA, 1987. ACM.

[TPO06] David Tarditi, Sidd Puri, and Jose Oglesby. Accelerator: using data parallelism to program gpus for general-purpose uses. In *ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, pages 325–335, New York, NY, USA, 2006. ACM.

[vFTS06] Wolfram von Funck, Holger Theisel, and Hans-Peter Seidel. Vector field based shape deformations. *ACM Trans. Graph.*, 25(3):1118–1125, 2006.

[WSV91] M. W. Walker, L. Shao, and R. A. Volz. Estimating 3-D location parameters using dual number quaternions. *CVGIP: Image Understanding*, 54(3):358–367, November 1991.

[WW92] William Welch and Andrew Witkin. Variational surface modeling. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 157–166, New York, NY, USA, 1992. ACM Press.

[YZX+04] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph.*, 23(3):644–651, 2004.

[ZHS+05] Kun Zhou, Jin Huang, John Snyder, Xinguo Liu, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Large mesh deformation using the volumetric graph laplacian. *ACM Trans. Graph.*, 24(3):496–503, 2005.

[ZRKS05] Rhaleb Zayer, Christian Rössl, Zachi Karni, and Hans-Peter Seidel. Harmonic guidance for surface deformation. *Computer Graphics Forum*, 24(3):601–610, September 2005.

[ZT00] O. C. Zienkiewicz and Robert Leroy Taylor. *The Finite Element Method, volume 1 and 2*. Butterworth-Heinemann, 2000.