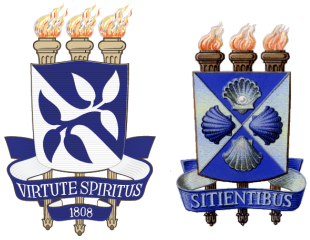

Universidade Federal da Bahia
Universidade Estadual de Feira de Santana

DISSERTAÇÃO DE MESTRADO

Deformação de Objetos Gráficos em GPU

Luciana Patricia das Virgens Silva

Mestrado em Ciência da Computação – MMCC

Salvador
28 de Novembro de 2014

MMCC-Msc-XXXX

LUCIANA PATRICIA DAS VIRGENS SILVA

DEFORMAÇÃO DE OBJETOS GRÁFICOS EM GPU

Dissertação apresentada ao Mestrado em Ciência da Computação da Universidade Federal da Bahia e Universidade Estadual de Feira de Santana, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Vinícius Moreira Mello

Salvador
28 de Novembro de 2014

Ficha catalográfica.

Silva, Luciana Patricia das Virgens

Deformação de Objetos Gráficos em GPU/ Luciana Patricia das Virgens Silva– Salvador, 28 de Novembro de 2014.

50p.: il.

Orientador: Prof. Dr. Vinícius Moreira Mello.
Dissertação (mestrado em Ciência da Computação) – Universidade Federal da Bahia, Instituto de Matemática, 28 de Novembro de 2014.

1. Deformação. 2. Deformação - Visualização Volumétrica. 3. Campos de Vetores..
I. Mello, Vinícius Moreira. II. Universidade Federal da Bahia. Instituto de Matemática. III Título.

<NUMERO CDD>

TERMO DE APROVAÇÃO

LUCIANA PATRICIA DAS VIRGENS SILVA

DEFORMAÇÃO DE OBJETOS GRÁFICOS EM GPU

Esta dissertação foi julgada adequada à obtenção do título de Mestre em Ciência da Computação e aprovada em sua forma final pelo Mestrado em Ciência da Computação da UFBA-UEFS.

Salvador, 28 de Novembro de 2014

Prof. Dr. Vinícius Moreira Mello
Universidade Federal da Bahia

Prof. Dr. Perfilino Eugênio Ferreira Jr.
Universidade Federal da Bahia

Prof. Dr. Dimas Martinez Morera
Universidade Federal de Alagoas

AGRADECIMENTOS

Agradeço primeiramente a Deus por minha vida, minha família e meus amigos. Aos meus pais Gildete e Evilásio pelo amor incondicional, pelo incentivo em todas as parte da minha existência neste mundo e por fazerem de tudo para que os meus sonhos se concretizassem. Ao meu irmão Luiz Claudio e minha irmã Eliana por serem meus maiores exemplos nesta vida de que se você realmente quer algo lute, nunca desista que com toda certeza você alcançará o seu objetivo. Ao meu orientador Vinícius pela paciência, pela oportunidade, o apoio e confiança depositados durante todo o período em que trabalhamos juntos. Aos meus familiares, minhas tias, tios, primos, primas, meu sobrinho e minha cunhada pelo amor, apoio, choros e risos que compartilhamos e continuaremos a compartilhar nesta vida e nas próximas. Aos meus amigos e companheiros do LASID pela importante companhia, por me aturarem por todos estes anos em que nos conhecemos e que com certeza continuarão presentes por muitos outros. À FAPESB pelo apoio financeiro. E por fim, agradeço a todos que direta ou indiretamente fizeram parte da minha formação. Muito obrigada!

RESUMO

O presente trabalho tem por objetivo desenvolver técnicas de deformação de objetos gráficos em GPU combinando *lançamento de raios* (*ray casting*) e campos de vetores. Na implementação computacional deste trabalho foi desenvolvida uma ferramenta para visualização e deformação de objetos gráficos.

Palavras-chave: deformação, objetos gráficos, campo de vetores.

ABSTRACT

This work aims to develop techniques for deformation of graphic objects in GPU combining ray casting and vector fields. In the computational implementation of this work we developed a tool for visualization and deformation of graphic objects.

Keywords: deformation, graphical objects, vector field.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1

INTRODUÇÃO

De acordo com a *International Organization for Standardization* (ISO), a Computação Gráfica é definida como o conjunto de métodos e técnicas utilizados para converter dados para um dispositivo gráfico, via computador. Estando presente em diversas áreas do conhecimento humano, a Computação Gráfica possui várias sub-áreas que representam conjuntos de técnicas com objetivos comuns.

A modelagem de objetos pode ser vista como uma sub-área da Computação Gráfica que engloba diversos aspectos relacionados com a criação, representação e manipulação de objetos gráficos no computador. A modelagem afeta praticamente todas as áreas da Computação Gráfica e, desta forma, na formulação das soluções para este problema devem ser levadas em consideração diversos fatores como, por exemplo, a escolha de um modelo matemático adequado para o problema e as estruturas de dados e métodos para implementar a representação no computador.

O processo de modelagem pode ser encarado sob dois aspectos: geométrico, que obedece às regras formais da Geometria clássica e utiliza recursos da topologia, e o procedural, que cria objetos a partir de regras que determinam a sua forma e evolução (Gomes; Velho, 2008).

Em diversas áreas científicas o uso de objetos é essencial para a representação de características de entidades concretas ou abstratas. Podemos citar como exemplo o modelo atômico de Bohr que estuda as estruturas do átomo e os modelos econômicos que descrevem aspectos de uma dada economia. Um dos principais objetivos da modelagem de um objeto ou fenômeno é permitir um melhor entendimento da estrutura dos mesmos. A visualização é um instrumento que pode ser usado para alcançar este objetivo.

Além de representar formas, os objetos também podem simbolizar comportamentos específicos aos mesmos ao serem sujeitos à manipulações. A este processo damos o nome de deformação. Uma deformação pode ser definida como uma transformação do espaço no espaço que é continuamente diferenciável e inversível. Os objetos visuais submetidos a este processo são denominados objetos deformáveis.

A deformação de objetos gráficos visa prover métodos eficientes para manipulação de objetos, tornando possível a realização de transformações geométricas que alteram

suas medidas e curvaturas. Esses métodos têm sido utilizados amplamente em diversas aplicações como animações, edição de imagens, efeitos especiais e simulações de cirurgias para representar tecidos deformados.

Os métodos desenvolvidos com o objetivo de gerar um objeto deformável podem ser separados nas seguintes categorias: geométricos, que se baseiam em fundamentos matemáticos e apresentam semelhanças com o processo de modelagem geométrica; físicos, que, além dos fundamentos matemáticos, adicionam princípios físicos que descrevem as características do material que constitui o objeto relacionando-se com a modelagem procedural; e, por fim, os híbridos que são combinações dos dois tipos anteriores (Gibson; Mirtich, 1997) (Moore; Molloy, 2007). Os métodos para deformação se encaixam como particularizações dos métodos para modelagem sendo destinados à geração de objetos dinâmicos.

1.1 OBJETIVO

O objetivo deste trabalho é desenvolver métodos de deformação de objetos gráficos integrando campos vetoriais à técnica de *lançamento de raios* (*ray casting*) de forma a permitir deformações locais nesses objetos, tomando como base as funções de deformação propostas por Barr (Barr, 1984).

Como contribuição, foi desenvolvida uma ferramenta que realiza a visualização de volumes utilizando *ray casting* e aplica as técnicas de deformação criadas neste trabalho. Essa ferramenta será melhor descrita no capítulo 5.

1.2 ESTRUTURA DO TRABALHO

No próximo capítulo deste trabalho (capítulo 2) apresentaremos uma visão geral de objetos gráficos. No capítulo 3, descreveremos conceitos e técnicas de deformação de objetos gráficos. Descrevemos as técnicas de deformação criadas neste trabalho no capítulo 4, deixando para o capítulo 5 a tarefa de descrever a ferramenta desenvolvida. Por fim, apresentamos nos capítulos 6 e 7, respectivamente, os resultados, as conclusões e propostas de trabalhos futuros.

Capítulo

2

OBJETOS GRÁFICOS

Devido a complexidade do mundo real, o processo de captura da realidade para efeito de modelagem envolve abstrações, generalizações e aproximações. Em computação Gráfica é aplicado um paradigma de abstração chamado *Paradigma dos Quatro Universos* (figura 2.1) (Gomes; Velho, 2008).

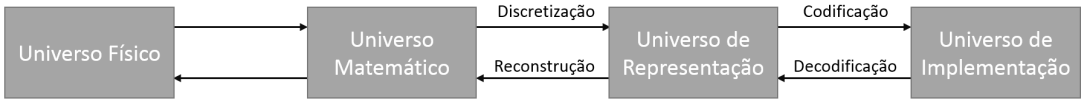

Figura 2.1 Paradigma dos Quatro Universos.

Essa abordagem divide o processo em vários níveis, ou universos, permitindo o encapsulamento dos problemas de cada nível e possibilitando um melhor entendimento dos problemas e de suas soluções. O universo Físico contém os objetos e fenômenos do mundo real que serão estudados. O universo Matemático contém descrições matemáticas abstratas dos objetos do universo Físico. O universo de Representação contém as descrições simbólicas e finitas que são associadas aos objetos do universo Matemático. E, por fim, o universo de Implementação que contém as estruturas de dados utilizadas na codificação da representação em uma determinada linguagem de programação.

Dentre estes níveis, os elementos manipulados pela Computação Gráfica são caracterizados como elementos do universo Matemático, onde são formuladas as descrições abstratas dos objetos do mundo real. Tais são conhecidos como Objetos Gráficos (Gomes; Velho, 2008).

O conceito de objeto gráfico (figura 2.2) é extremamente importante para a Computação Gráfica e áreas relacionadas. Um objeto gráfico é a representação da geometria e atributos de um objeto do mundo real. Matematicamente falando, temos a seguinte definição:

Definição 1. *Um* objeto gráfico *é um subconjunto* $\mathcal{O} \subset \mathbb{R}^m$ *e uma função* $\mathrm{a} : S \subset \mathbb{R}^m \rightarrow \mathbb{R}^n$

Onde $\mathcal{O}$ é chamado de *suporte geométrico*, a é chamada de *função de atributos* do objeto gráfico e $\mathbb{R}^m$ é o espaço de representação ou espaço ambiente. A dimensão de um objeto gráfico é dada pela dimensão do suporte geométrico. Na especificação de um objeto gráfico é necessário definir qual será a sua geometria e a topologia de seu suporte geométrico além de definir a sua função de atributos, que é responsável por determinar propriedades como cor, textura, brilho entre outros (Gomes; Velho, 2008).

Um exemplo de objeto gráfico é uma imagem. Tal é definida por uma função $a : U \subset \mathbb{R}^2 \to \mathbb{C}$. Desta forma, uma imagem é um objeto gráfico com suporte geométrico U e com uma função de atributos a que informa a cor em cada ponto do suporte.

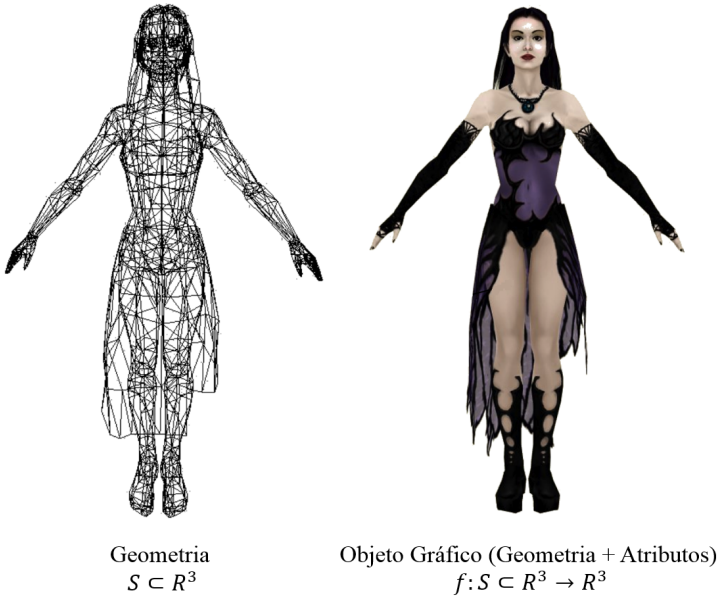

Figura 2.2 Objeto Gráfico. Adaptado de (Montenegro, 2008b)

Basicamente existem duas formas clássicas de representação de um objeto gráfico: a representação paramétrica e a implícita (Gomes; Velho, 2008). Intuitivamente, na representação paramétrica, o objeto $\mathcal{O}$ é a imagem de um conjunto por uma função, enquanto na representação implícita é o conjunto do pontos que satisfazem uma equação.

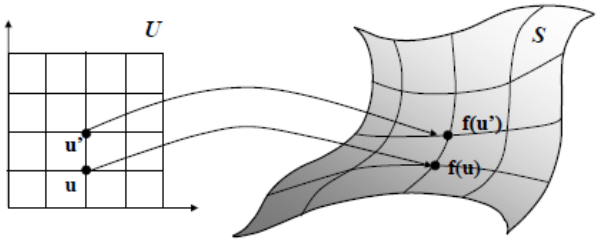

Figura 2.3 Correspondência um-para-um. Fonte (Montenegro, 2008a)

Mais formalmente, temos a seguinte definição para representação paramétrica:

Definição 2. *Seja* $\mathrm{F} : U \subset \mathbb{R}^2 \to \mathbb{R}^3$ *uma aplicação que atende as seguintes condições:*

1. F *é bijetiva em U, ou seja, existe uma correspondência um-para-um entre os pontos do domínio e do contra domínio, como mostra a figura 2.3;*

2. F *é continuamente diferenciável, ou seja, é possível definir um plano tangente em cada ponto do seu gráfico, como podemos ver na figura 2.4.*

Um subconjunto $\mathbb{R}^3$ *é um* objeto paramétrico *se* $\mathcal{O} = F(U)$

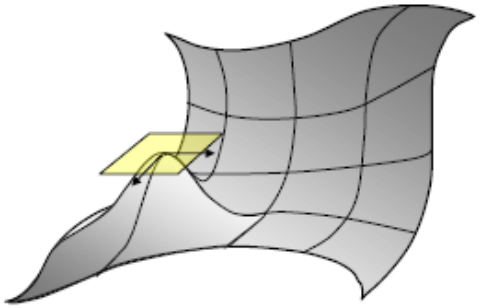

Figura 2.4 Plano tangente. Fonte (Montenegro, 2008a)

Nos universos de Representação e Implementação, normalmente, um objeto paramétrico é representado por uma malha de triângulos. O universo de Representação é responsável por trazer as descrições abstratas para o mundo digital, discretizando os sinais contínuos, enquanto o de Implementação realiza a codificação do sinal discretizado na memória do computador utilizando estrutura de dados (Gomes; Velho, 2008).

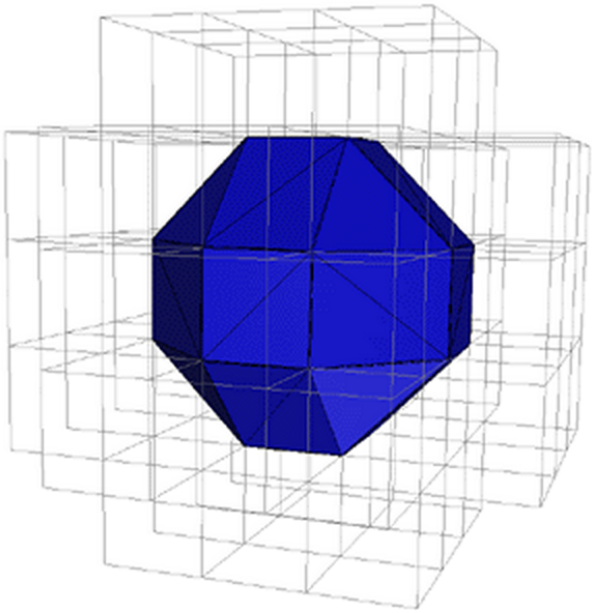

Figura 2.5 Objeto Implícito. Fonte (Bourke, 1994)

A representação implícita pode ser definida da seguinte forma:

Definição 3. *Um* objeto implícito $\mathcal{O} \subset \mathbb{R}^3$ *é definido como o conjunto das raízes de uma função continuamente diferenciável* $F : U \subset \mathbb{R}^3 \to \mathbb{R}$*, ou seja,* $\mathcal{O} = \{(x, y, z) \in U; F(x, y, z) = 0\}$.

O conjunto das raízes, também indicado pela notação $F^{-1}(0)$, é chamado de imagem inversa do conjunto $\{0\} \subset \mathbb{R}$ pela função F e define uma *superfície de nível* da função F. O objeto implícito é, geralmente, representado por um volume (figura 2.5) nos universos de Representação e Implementação (Gomes; Velho, 2008).

2.1 ATRIBUTOS GEOMÉTRICOS DE OBJETOS GRÁFICOS

Os atributos geométricos definem a geometria dos objetos gráficos. Eles são: o *vetor tangente* (figura 2.6), o *plano tangente* e o *vetor normal*. Um vetor $v \in \mathbb{R}^3$ é tangente a uma superfície S em um ponto p se existe uma curva $\gamma(-\epsilon, \epsilon) \to \mathbb{R}^3$ tal que $\gamma(0) = p$ e $\gamma'(0) = v$ com $\gamma : (-\epsilon, \epsilon) \subset S$ para algum $\epsilon > 0$.

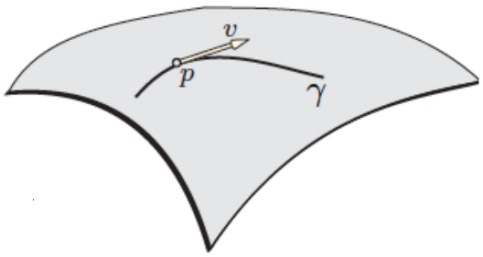

Figura 2.6 Vetor tangente. Fonte (Vieira, 2011)

O plano tangente (figura 2.7) de S em p é o conjunto de todos os vetores tangentes a S em p e é denominado T_pS.

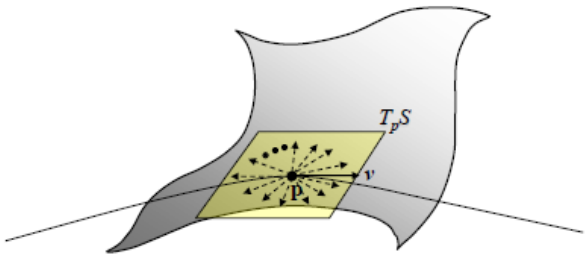

Figura 2.7 Plano tangente. Fonte (Montenegro, 2008a)

Um vetor $n \in \mathbb{R}^3$ é dito normal à superfície S em p se n é perpendicular a T_pS (como mostra a figura 2.8). A normal é essencial para calcular a iluminação dos objetos gráficos.

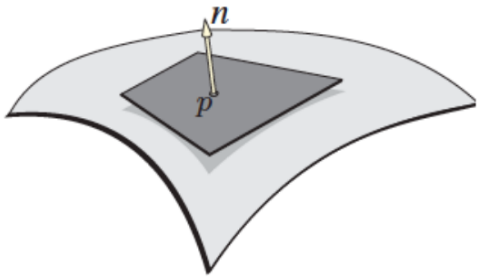

Figura 2.8 Vetor normal. Fonte (Vieira, 2011)

2.2 TÉCNICAS DE VISUALIZAÇÃO DE OBJETOS GRÁFICOS

Existem vários métodos de visualização de objetos gráficos. O *pipeline de visualização* é o conjunto de técnicas que permitirá a transformação das informações contidas em uma estrutura de dados em uma imagem visível pelo usuário. Dada a convergência tecnológica (diminuição do custo da memória RAM e o desenvolvimento do processamento paralelizado) alguns algoritmos se destacam mais que os outros. Dentre estas técnicas podemos destacar a técnica de *ray casting* utilizada para visualizar objetos implícitos e que será detalhada mais a frente.

2.2.1 Pipeline de Visualização

O *pipeline de visualização* (figura 2.9) é o processo básico de geração de imagens tridimensionais no computador e consiste em criar uma projeção bidimensional da cena levando em conta a iluminação e a perspectiva de visualização. Diversas *API*s (*Application Programming Interface*), como a OpenGL, disponibilizam bibliotecas gráficas que incluem funções específicas para a visualização de malhas de polígonos.

O processo é iniciado percorrendo-se a base de dados da cena. Tal base geralmente contém a especificação dos parâmetros de visualização, das fontes de luz, dos atributos físicos das superfícies, como cor e índices de reflexão, e os objetos descritos por polígonos. As atividades seguintes podem ser divididas em dois blocos: transformações geométricas e rasterização.

As atividades do bloco de transformações geométricas vão realizar a conversão das primitivas gráficas descritas em coordenadas do mundo para coordenadas da tela. Essas operações basicamente atuam sobre os vértices dos polígonos e incluem as operações de eliminação das faces escondidas (também conhecido como *back-faces culling*), recorte dos polígonos (ou *clipping*), aplicação do modelo de iluminação para determinar a luminosidade e a transformação de perspectiva.

O bloco de rasterização é responsável pela conversão das informações geométricas em *pixels* na tela com infomações de cor. Nas atividades realizadas neste bloco estão incluídas o cálculo dos *pixels* que devem ser ativados (processo de *scan-conversion*), o cálculo das cores de cada um dos *pixels*, a remoção das superfícies ocultas, implementação dos efeitos visuais (como mapeamento de texturas), aplicação do modelo de reflexão local de acordo com o algoritmo de sombreamento e a filtragem dos ruídos (processo conhecido como

anti-aliasing). Por fim, as informações referentes aos *pixels* são escritas na memória da imagem para visualização.

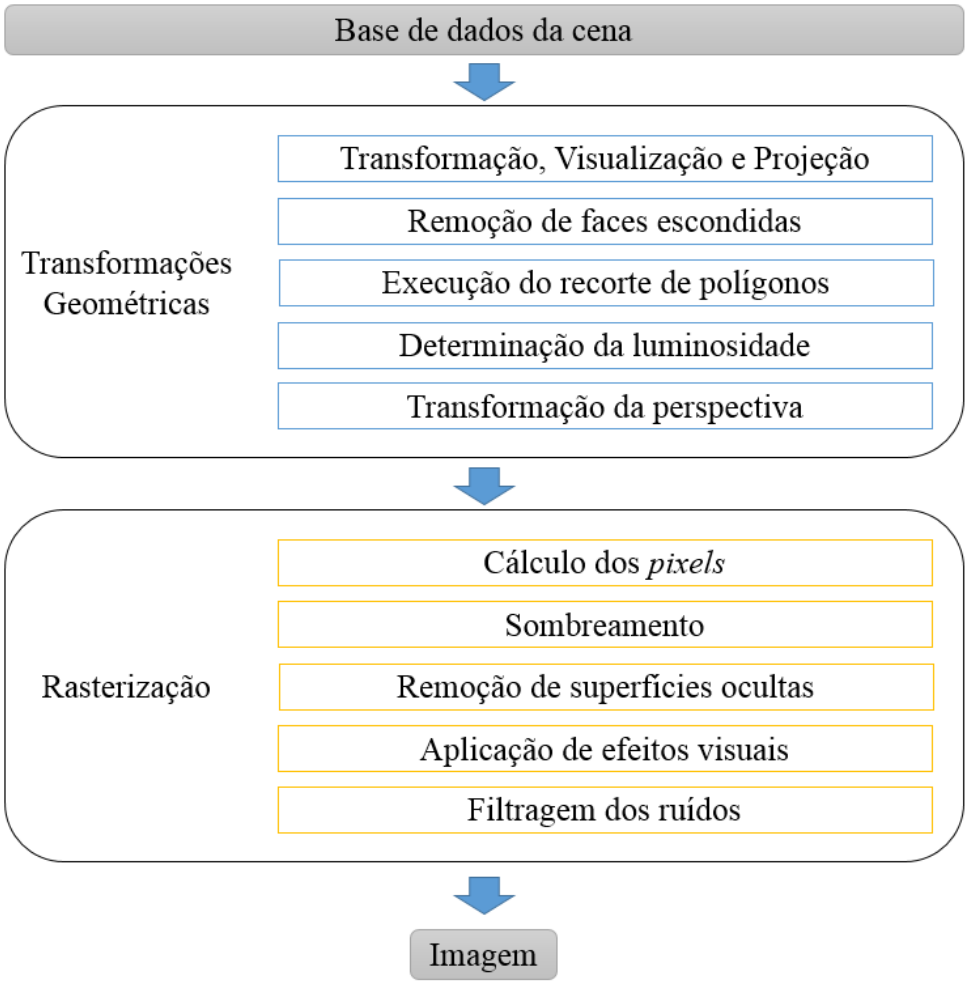

Figura 2.9 Pipeline de visualização tradicional.

2.2.2 Visualização Volumétrica

As principais abordagens de visualização de objetos implícitos podem ser classificadas a partir da primitiva de desenho aplicada. Uma dessas categorias é a de técnicas de visualização volumétrica. Estas podem ser usadas em situações em que outras não são adequadas como, por exemplo, quando o objeto é muito complexo e não há uma boa noção da superfície, ou caso seja necessário visualizar alguma propriedade espacial da função implícita.

Conceitualmente podemos entender esta categoria como o conjunto de técnicas utilizadas na visualização de dados associados à regiões de um volume, cujo principal objetivo é exibir o interior de objetos volumétricos explorando a sua estrutura e, consequentemente, facilitando a sua compreensão (Mccormick; Defanti; Brown, 1987). Os dados associados à regiões de volumes são chamados dados volumétricos.

Tradicionalmente, os dados volumétricos são tratados como uma matriz de elementos de volume. Tais são denominados *voxels* (*Volumetric Pixel* ou *Volumetric Picture Element*), análogos 3D do *pixel*. Enquanto o *pixel* é denominado o menor ponto *bidimensional* de uma imagem, o *voxel* é o menor ponto *tridimensional* de uma imagem digital [1].

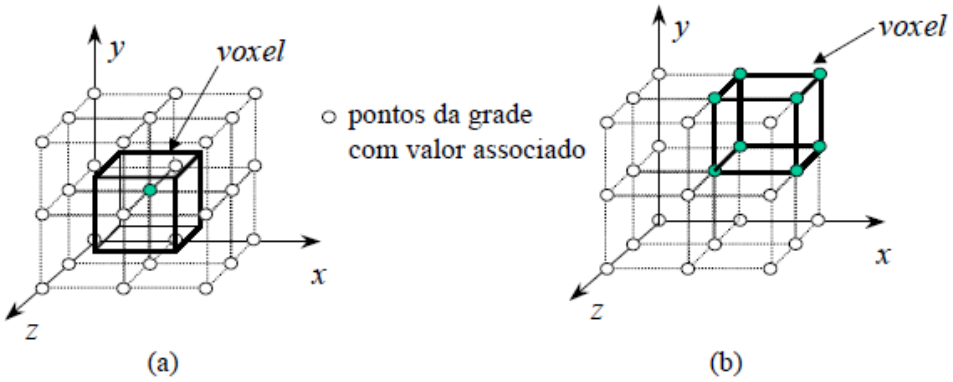

Figura 2.10 Definições de um *voxel*. Fonte (Paiva; Seixas; Gattass, 1999).

Na literatura não existe um consenso quanto à definição do que é um *voxel*. Em algumas publicações, um *voxel* é entendido como um hexaedro definido em torno do valor amostrado (figura 2.10 a). Outras entendem como sendo um hexaedro cujos vértices são os valores amostrados (figura 2.10 b).

Para (Paiva; Seixas; Gattass, 1999), os *voxels* são definidos como paralelepípedos agrupados formados pela divisão do espaço do objeto por um conjunto de planos paralelos aos eixos principais deste espaço. Já em (Kaufman, 1996) é dito que o *voxel* é a região de valor constante no entorno de cada amostra, sendo cada *voxel* um cubóide retangular com seis faces, doze arestas e oito vértices.

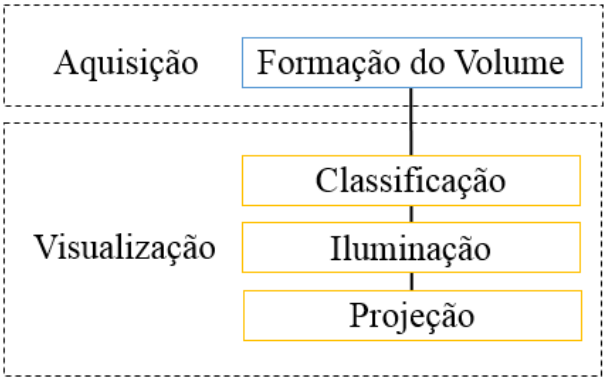

Figura 2.11 Pipeline de visualização volumétrica. Fonte (Paiva; Seixas; Gattass, 1999).

[1] "Representação discreta de informação gerada e processada por um computador, ou adquirida através de um dispositivo de captura e processada posteriormente através de um computador."

2.2.2.1 Pipeline de Visualização Volumétrica O processo de visualização volumétrica pode ser resumido em quatro etapas, como mostra a figura 2.11. A primeira, *formação do volume*, consiste na obtenção dos dados, o pré-processamento das fatias e a reconstrução do volume. A visualização propriamente dita é realizada a partir da implementação das três últimas etapas, uma vez que considera-se que o volume já foi pré-processado.

As principais fontes de dados volumétricos são dispositivos de medição ou simulações numéricas de fenômenos naturais. O primeiro produz dados volumétricos regulares como MRI (Ressonância Magnética), CT (Tomografia Computadorizada) e Tomografia Sísmica. O segundo produz tanto dados irregulares quanto regulares, como CFD (Computação Dinâmica de Fluídos).

Com os dados adquiridos é feito então o processamento para melhorar as características como contraste e a faixa de variação dos valores. Ao fim da primeira etapa, os dados são reconstruídos e um volume com dimensões proporcionais é gerado.

Logo em seguida é dado inicio ao processo de visualização. A etapa de classificação consiste na identificação do material representado em cada *voxel*. Para isso são selecionadas as características dos dados, baseando-se em um critério quantitativo, a partir da definição da região do volume escolhida para ser explorada. Esta etapa, nas técnicas de extração de superfícies, resume-se na definição do valor de limiarização (*threshold*), que é utilizado na identificação da superfície que deve ser poligonizada e em seguida visualizada. Na renderização direta, envolve a definição das funções de transferência que informa a relação entre os valores dos dados do volume e os valores de cor e opacidade que serão usados no algoritmo de exibição.

Na próxima etapa será aplicado o modelo de iluminação que irá calcular a tonalidade da cor em cada *voxel*, levando em consideração as propriedades do material de cada elemento do volume e as condições de iluminação externas. Nas técnicas de extração de superfície, aplica-se um dos modelos de iluminação tradicionais utilizados na Computação Gráfica. Geralmente, o modelo utilizado é o de Phong (Phong, 1975). Na renderização direta, adapta-se o modelo de Phong substituindo-se os vetores normais à superfície por gradientes do campo escalar no *voxel*.

Por último, a etapa que compreende a projeção dos *voxels* na superfície de visualização e, consequentemente, a composição que determinará a imagem que será visualizada. Nesta etapa, nas técnicas de extração de superfície, a visualização é realizada com o auxílio de efeitos de iluminação e remoção de áreas escondidas. Além disso, também é implementada a possibilidade do usuário definir o formato de visualização desejado realizando cortes, rotações e especificações do tipo de projeção, podendo ser ortográfica ou perspectiva. Nos algoritmos de renderização direta, são realizados a projeção dos *voxels* e o cálculo da composição destes sobre o plano da imagem.

2.2.2.2 Abordagens de Visualização Volumétrica As técnicas de visualização volumétrica podem ser classificadas de acordo com critérios bastante variados. Entretanto, existem duas abordagens clássicas que se destacam. A primeira é denominada extração de superfícies (ou *surface extraction*). A segunda é conhecida como renderização de volumes (ou *volume rendering*). Basicamente, as duas se distinguem no uso ou

não de representações intermediárias dos dados volumétricos na geração da visualização adequada a aplicação.

Os algoritmos de extração de superfície convertem os dados volumétricos para uma representação geométrica intermediária, normalmente polígonos, procurando isolar um determinado objeto que está representado nos dados, onde serão aplicados os métodos tradicionais de renderização de polígonos gerando a visualização. Desta forma, é possível diminuir a quantidade de dados manipulados quando na formação da imagem. Nestes algoritmos, na ocorrência de alterações nas características do volume visualizado é necessário a aplicação das técnicas de renderização de polígonos e de reprocessamento do volume para realizar novamente a extração do objeto. Alguns exemplos de algoritmos desta abordagem são o *Marching Cubes* (Lorensen; Cline, 1987) e o *Dividing Cubes* (Cline et al., 1988).

Já a renderização direta de volumes realiza a projeção da imagem diretamente a partir dos dados volumétricos, sem a necessidade de nenhuma representação intermediária. Como a quantidade de dados manipulados nesta abordagem é extensa, o esforço computacional envolvido na realização deste processo também será grande. Com estes algoritmos é possível visualizar mais de um objeto contido nos dados volumétricos, de forma a permitir a visualização de porções interiores dos volumes. As técnicas de *Ray Casting*, *Splatting* (Westover, 1990) e *V-Buffer* (Upson; Keeler, 1988), são exemplos deste tipo de abordagem.

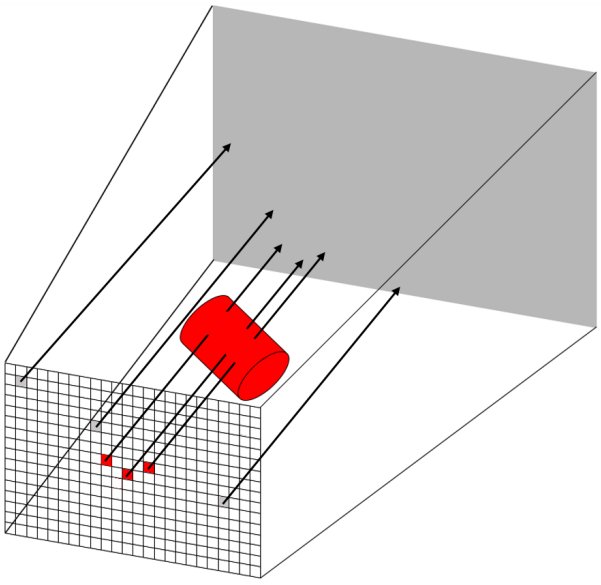

Figura 2.12 Descrição do algoritmo de *Ray Casting*. Os *pixels* cujos raios intersectam o volume são representados pelos quadrados vermelhos.

.

A técnica de ***Ray Casting*** é a abordagem de visualização volumétrica que optamos por utilizar neste trabalho. A ideia geral dessa técnica é de lançar "raios de visão" em direção aos objetos na cena, partindo do ponto de vista do observador. Cada raio é relacionado com um *pixel* da tela. Depois de lançados os raios, é preciso determinar a

primeira interseção entre cada raio e os objetos na cena e definir a cor correta do objeto naquele ponto. Esta cor será utilizada para pintar na tela os *pixels* que são relacionados com os raios que intersectam o objeto. A figura 2.12 exemplifica esta técnica. Cada raio é modelado como uma reta em forma paramétrica,

$$r(t) = \mathcal{O} + tD$$

onde $\mathcal{O} = (\mathcal{O}_x, \mathcal{O}_y, \mathcal{O}_z)$ representa o ponto de origem do raio no espaço da cena, $D = (D_x, D_y, D_z)$ representa a direção do raio, e t é o parâmetro utilizado para acessar todos os pontos sobre o raio. Podemos desmembrar a equação da seguinte forma:

$$x(t) = \mathcal{O}_x + tD_x$$

$$y(t) = \mathcal{O}_y + tD_y$$

$$z(t) = \mathcal{O}_z + tD_z$$

Tomamos como exemplo uma superfície implícita $F(x, y, z) = 0$, para calcular a interseção entre o raio e a superfície devemos determinar os pontos do espaço que pertencem ao raio e que, ao mesmo tempo, satisfaçam à equação. Para isso substituímos os argumentos na função da superfície pelas expressões em função de t, obtendo uma equação do tipo $F(x(t), y(t), z(t)) = 0$ ou, mais resumidamente:

$$g(t) = F(r(t)) = 0$$

A primeira interseção dos raios com o objeto fornecerá pontos sobre os quais serão aplicados a função a para obter suas cores originais e vetores normais, $C(i, j)$ e $N(i, j)$ respectivamente. Por fim, será aplicado um modelo de iluminação sobre cada um dos pontos utilizando as informações adquiridas (cores e vetores normais) para se definir a cor final de cada *pixel*.

Dentre as vantagens do processo de *ray casting* podemos destacar a geração de imagens mais realísticas e exatas e a economia de memória já que não há a necessidade de construção de grandes estruturas de dados.

Capítulo

3

DEFORMAÇÃO DE OBJETOS GRÁFICOS

Uma deformação pode ser entendida como uma transformação do espaço no espaço que é continuamente diferenciável e inversível. Neste processo um mesmo objeto sofre transformações geométricas e topológicas que irão modificar a sua forma. Tais transformações vem sendo cada vez mais estudadas e utilizadas na área de Computação Gráfica, especialmente em filmes, animações e jogos, com o objetivo de produzir transições na aparência de personagens.

Métodos computacionais voltados para a deformação de objetos gráficos originam-se, principalmente, da necessidade de se simular não só as formas dos objetos reais como também os seus comportamentos. Estes métodos podem ser classificados em três categorias: geométricos, físicos e híbridos.

Os métodos geométricos utilizam fundamentos matemáticos para a composição dos objetos, ou seja, neles, as deformações do objeto ou espaço ao redor do mesmo são obtidas com manipulações geométricas, como por exemplo vértices e pontos de controle. Apesar de geralmente serem eficientes computacionalmente e de relativamente fácil implementação, estes métodos resultam em movimentos pouco realistas uma vez que não permitem a simulação da mecânica envolvida nas deformações.

Os métodos físicos são aqueles que envolvem o tratamento geométrico das formas dos objetos com princípios físicos. A principal motivação para o desenvolvimento destes métodos vem da necessidade de se modelar ambientes e fenômenos naturais, tanto em termos de forma quanto de seu comportamento, que são praticamente impossíveis de se alcançar com métodos geométricos. Os métodos físicos consideram as leis da Mecânica e da Dinâmica no processo de modelagem para desenvolver o movimento e a forma dos objetos. Desta forma, podemos entender que os parâmetros usados na definição da modelagem dos objetos são de natureza puramente física, como por exemplo o torque. A modelagem também dependerá do tempo, uma vez que as equações que representam as leis do sistema físico devem ser assimiladas ao longo do tempo de forma que efeitos como torque possam atuar e produzir o efeito desejado (Gomes; Velho, 1990).

Os métodos híbridos são aqueles que associam características dos dois tipos anteriores buscando aproveitar ao máximo as potencialidades de cada um.

A maioria das aplicações que utilizam métodos de deformação funcionam com a intervenção do usuário permitindo, por exemplo, a definição de pontos e linhas de controle ou o ajuste da posição de partes da imagem (Jun; Xia; Tiangang, 2009). Existem vários modos de se especificar uma deformação genérica. A escolha do método influencia tanto na interface do usuário quanto no algoritmo de deformação utilizado.

3.1 TÉCNICAS DE DEFORMAÇÃO

Existem vários métodos voltados exclusivamente a deformação de objetos gráficos. Dentre estes métodos os que se destacam são o de deformação por malhas de triângulos, deformação por malhas de spline, deformação baseada em campos e deformação por formas livres. Alguns destes algoritmos podem ser adaptados e aplicados tanto em objetos bidimensionais quanto tridimensionais.

3.1.1 Deformação por Malhas de Triângulos

Este algoritmo descreve a deformação de um objeto bidimensional (uma imagem) através de duas malhas triangulares, a primeira representa o sistema de coordenadas original (não deformado), e a segunda representa a deformação deste sistema, como pode ser visto na figura 3.1. As duas devem possuir a mesma estrutura combinatória, de modo que a correspondência entre os pares de triângulos esteja bem definida.

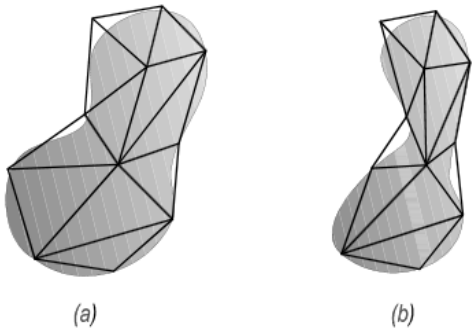

Figura 3.1 Malha de triângulos: (a) Original; (b) Deformada. Fonte (Darsa, 1994)

A aquisição das malhas de triângulos que serão utilizadas como entradas do algoritmo pode ser feita de diversas maneiras. Uma dessas formas é construir diretamente ambas as malhas garantindo a sua consistência. Entretanto, a probabilidade de erros e o trabalho exercido neste processo podem ser muito grandes. Por isso, normalmente, utiliza-se uma forma de especificação por características, onde tais características são definidas com pontos ou segmentos de retas, e aplicados algoritmos de triangulação automática para a construção da malha. Esta técnica pode ser utilizada para deformar imagens digitais a partir da rasterização de cada triângulo no estado deformado. e aplicando-se um mapeamento inverso a cada um dos *pixels* assim obtidos determinando suas áreas de origem (Foley et al., 1990).

A deformação por malhas de triângulos também pode ser aplicada para objetos tridimensionais. Nesse caso a deformação é descrita a partir de uma malha de tetraedros dada em dois estados distintos, o original e sua deformação. Para cada tetraedro correspondente, os pontos em seu interior podem ser mapeados muito facilmente utilizando coordenadas baricêntricas, tornando assim o algoritmo apropriado para qualquer representação de objeto. Particularmente, para objetos volumétricos, é necessária a rasterização 3D do interior dos tetraedros. Assim como na versão 2D, a triangulação automática de pontos usados para indicar as características é uma alternativa simples além de também poder ser utilizada para gerar malhas.

3.1.2 Deformação por Malhas de Splines em Dois Passos

Específico para a utilização com imagens digitais, recebe como entrada duas malhas de splines indicando os estados normal e deformado da imagem (ver figura 3.2). Neste processo uma imagem intermediária é criada para ser deformado logo em seguida evitando assim a transformação direta da imagem original. Desta forma, pode-se dividir o mapeamento em dois passos independentes:

- Horizontal: onde a imagem intermediária horizontalmente deformada é produzida realizando-se somente os deslocamentos horizontais da malha de origem para a de destino;
- Vertical: utiliza a malha e imagem intermediárias criadas no processo anterior para produzir a saída final. A malha de origem e a imagem original não serão utilizadas.

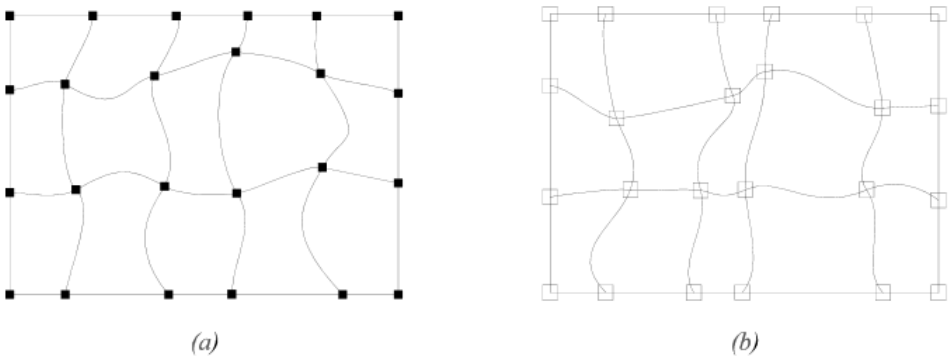

Figura 3.2 Malhas de splines. Fonte (Darsa, 1994)

Esta técnica é bastante eficiente e versátil, entretanto, possui uma grande desvantagem que é a necessidade de uma entrada de informação muito extensa por parte do usuário para poder definir a deformação.

3.1.3 Deformação Baseado em Campos

Tal técnica se resume na utilização de campos de influência ao redor das principais características de uma imagem. Características como nariz, boca ou contorno da face, que

são consideradas importantes, são marcadas por vetores (segmentos de retas orientadas) que indicarão o estado original (não deformado) e o estado da deformação daquela característica (Beier; Neely, 1992). Cada característica possui um campo de influência próprio. Na figura 3.3 vemos o sistema de coordenadas definido por um segmento de reta onde a coordenada v representa a distância perpendicular ao vetor e u representa a distância ao longo do vetor.

A utilização de mais de um segmento de reta pode produzir sistemas de coordenadas conflitantes. Esses sistemas devem ser combinados suavemente e de forma controlada. Para resolver essa questão, é proposto em (Beier; Neely, 1992) a atribuição de pesos à cada segmento. O peso será inversamente proporcional à distância do ponto para essa característica, ou seja, o peso será maior nos pontos sobre o segmento e decairá a medida em que os pontos forem se distanciando. Cada *pixel* da saída é mapeado inversamente para o sistema de coordenadas de origem de cada segmento de forma que seja produzido um conjunto de pontos mapeados distintos.

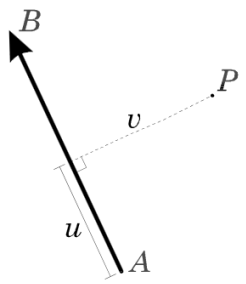

Figura 3.3 Sistema de coordenadas definido por um segmento de reta. Fonte (Darsa, 1994)

Esta definição de mapeamento de coordenadas, através de campos de influência, não depende de nenhuma estrutura ou particularidades existentes nas imagens digitais. Desta forma é possível combinar diferentes tipos de transformação de maneira inteligente e padronizada, realizando uma mudança global de coordenadas que pode ser utilizada para deformar objetos em qualquer representação.

Na abordagem 3D deste método as características importantes dos objetos gráficos também são marcadas com segmentos de retas orientados. Uma posição diferente de cada segmento irá definir a deformação da característica. Adicionalmente, existe um grau de liberdade maior resultando em um novo segmento no sistema de coordenadas que é definido pela característica (figura 3.4). Os dois vetores perpendiculares, ressaltados em negrito, são definidos pelo usuário e formam uma abstração de um osso (*bone*). Um conjunto de ossos, denominado esqueleto (*skeleton*), marca todas as características relevantes do objeto em questão.

O sistema de coordenadas global definido por dois esqueletos é resultado da combinação das mudanças das coordenadas individuais definidas por cada par de ossos. Assim como no caso 2D, esta combinação é a média ponderada dos mapeamentos de cada ponto levando em conta cada par de ossos.

O sistema de coordenadas cartesiano é definido através da obtenção de um eixo adicional a partir do produto vetorial entre os dois vetores em um osso. Cada ponto então

pode ser projetado em cada um dos eixos determinando, assim, suas coordenadas nesse sistema. Enquanto v e t referenciam distâncias absolutas, a coordenada u é normalizada de acordo com o comprimento do segmento.

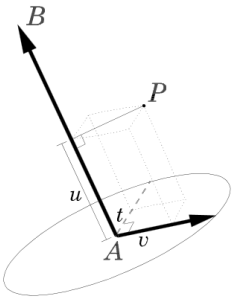

Figura 3.4 Sistema de coordenadas definido por uma característica 3D. Fonte (Darsa, 1994)

3.1.4 Deformações Por Formas Livres (*Free-Form Deformations*)

Este método, proposto em (Sederberg; Parry, 1986), foi desenvolvido especificamente para objetos tridimensionais. Nele o objeto gráfico é contido em uma grade (paralelepípedo repartido por um reticulado regular de pontos de controle) e deformado a medida em que a grade é deformada, ou seja, ao se alterar a posição dos pontos de controle da grade, define-se o sistema de coordenadas deformado, como mostra a figura 3.5.

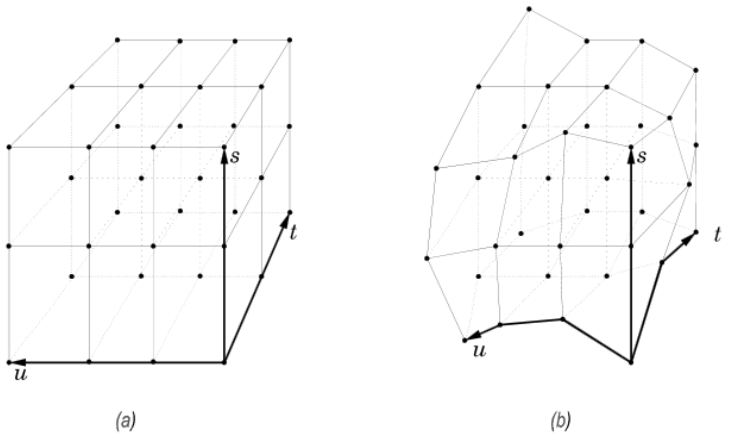

Figura 3.5 Sistema de coordenadas *free-form deformation*: (a) Original; (b) Deformado. Fonte (Darsa, 1994)

No mapeamento do sistema original para o deformado, cada ponto **X** é decomposto em três coordenadas (s, t, u) a partir da projeção sobre os três eixos da grade. A aquisição do ponto deformado **X'** é realizada pela transformação de (s, t, u) por um polinômio de Bernstein onde os coeficientes são definidos de acordo com os pontos de controle deformados. Desta forma é definido um mapeamento $F : \mathbb{R}^3 \rightarrow \mathbb{R}^3$, também conhecida como *warp* do espaço euclidiano tridimensional, deformação essa definida como uma transformação do espaço no espaço que é continuamente diferenciável e inversível.

Sendo esta técnica uma deformação do espaço ambiente, ela pode ser aplicada a diferentes representações de objetos.

3.2 DEFORMAÇÕES GLOBAIS DE BARR

Em (Barr, 1984), Barr introduz deformações com a criação de operações que irão torcer (*twist*), dobrar (*bend*) ou afunilar (*taper*) superfícies em torno de um eixo central (x, y ou z). Desta forma, operações que envolviam a movimentação de muitos pontos de controle agora poderiam ser realizadas com a alteração de um só parâmetro. Os métodos de deformação apresentados por Barr são matematicamente mais simples, reduzindo as necessidades computacionais, e amplamente aplicáveis não somente na animação quanto na modelagem de objetos. Barr também mostrou que é possivel realizar deformações por inversão de raio simplesmente aplicando as inversas das funções no raio. Nas primeiras etapas deste trabalho desenvolvemos um pequeno programa, o *MeshDeform*, para deformar objetos gráficos utilizando os métodos propostos por Barr.

3.2.1 *Tapering* global ao longo do eixo Z

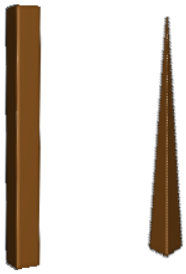

Figura 3.6 *Taper* aplicado em uma barra, do lado esquerdo temos o objeto original e do lado direito o deformado. Imagens geradas no programa *MeshDeform*.

É possível alcançar o afunilamento (figura 3.6) de um objeto modificando o comprimento de dois componentes globais sem modificar o terceiro. Neste caso serão modificados os valores das coordenadas x e y mantendo os valores de z. As equações de *tapering* são:

$$r = f(z),$$
$$X = rx,$$
$$Y = ry,$$
$$Z = z$$

Onde, $f(z)$ é uma função linear que decrementa a medida em que o valor de z aumenta. A matriz de transformação da normal é dada por:

$$r^2 \underline{\underline{\mathrm{J}}}^{-1T} = \begin{pmatrix} r & 0 & 0 \\ 0 & r & 0 \\ -rf'(z)x & -rf'(z)y & r^2 \end{pmatrix}$$

A função inversa, para transformação dos raios, é dada por:

$$r(Z) = f(Z),$$
$$x = X/r,$$
$$y = Y/r,$$
$$z = Z$$

3.2.2 *Twist axial global*

Figura 3.7 *Twist* aplicado em uma barra, do lado esquerdo temos o objeto original e do lado direito o deformado. Imagens geradas no programa *MeshDeform*.

Um *twist* (mostrado na figura 3.7) pode ser alcançado pela rotação de dois componentes globais, levando em consideração uma função de altura, e sem alterar o terceiro componente. O *twist* global ao longo do eixo z é produzido por:

$$\theta = f(z)$$
$$C_\theta = cos(\theta)$$
$$S_\theta = sin(\theta)$$
$$X = xC_\theta - yS_\theta$$
$$Y = xS_\theta + yC_\theta$$
$$Z = z$$

O *twist* procede ao longo do eixo z a uma taxa de $f'(z)$ radianos por unidade de comprimento na direção z. A matriz de transformação da normal é dada por:

$$\underline{\underline{\mathrm{J}}}^{-1T} = \begin{pmatrix} C_\theta & -S_\theta & 0 \\ S_\theta & C_\theta & 0 \\ yf'(z) & -xf'(z) & 1 \end{pmatrix}$$

A função inversa é dada por:

$$\theta = f(Z)$$
$$x = XC_\theta + YS_\theta$$
$$y = -XS_\theta + YC_\theta$$
$$z = Z$$

3.2.3 *Bend global linear* ao longo do eixo *Y*

Figura 3.8 *Bend* aplicado em uma barra, do lado esquerdo temos o objeto original e do lado direito o deformado. Imagens geradas no programa *MeshDeform*.

Estas operações levam em consideração a taxa de flexão (k), medida em radianos por unidade de comprimento que é uma constante, a região do objeto que será dobrada, delimitada pelos valores máximo (y_{max}) e mínimo (y_{min}), e o ponto nesta região que define o centro desta dobra (y_0). O raio de curvatura é dado por $\frac{1}{k}$. O ângulo da dobra θ é:

$$\theta = k(\hat{y} - y_0)$$

$$C_\theta = cos(\theta)$$

$$S_\theta = sin(\theta)$$

Onde:

$$\hat{y} = \begin{cases} y_{min}, & y \leq y_{min} \\ y, & y_{min} < y < y_{max} \\ y_{max}, & y \geq y_{max} \end{cases}$$

O *bend* ao longo do eixo y é produzido por:

$$X = x$$

$$Y = \begin{cases} -S_\theta(z - \frac{1}{k}) + y_0, & y_{min} \leq y \leq y_{max} \\ -S_\theta(z - \frac{1}{k}) + y_0 + C_\theta(y_0 - y_{min}), & y < y_{min} \\ -S_\theta(z - \frac{1}{k}) + y_0 + C_\theta(y_0 - y_{max}), & y > y_{max} \end{cases}$$

$$Z = \begin{cases} C_\theta(z - \frac{1}{k}) + \frac{1}{k}, & y_{min} \leq y \leq y_{max} \\ C_\theta(z - \frac{1}{k}) + \frac{1}{k} + S_\theta(y_0 - y_{min}), & y < y_{min} \\ C_\theta(z - \frac{1}{k}) + \frac{1}{k} + S_\theta(y_0 - y_{max}), & y > y_{max} \end{cases}$$

A matriz de transformação da normal é dada por:

$$(1-\hat{k}z)\underline{\underline{\mathrm{J}}}^{-1T} = \begin{pmatrix} 1-\hat{k}z & 0 & 0 \\ 0 & C_\theta & -S_\theta(1-\hat{k}z) \\ 0 & S_\theta & C_\theta(1-\hat{k}z) \end{pmatrix}$$

Onde:

$$\hat{k} = \begin{cases} k, & \hat{y} = y \\ 0, & \hat{y} \neq y \end{cases}$$

A função inversa é dada por:

$$\theta_{min} = k(y_{min} - y_0)$$

$$\theta_{max} = k(y_{max} - y_0)$$

$$\hat{\theta} = -tan^{-1}\left(\frac{Y - y_0}{Z - \frac{1}{k}}\right)$$

$$\theta = \begin{cases} \theta_{min}, & \hat{\theta} < \theta_{min} \\ \hat{\theta}, & \theta_{min} \leq \hat{\theta} \leq \theta_{max} \\ \theta_{max}, & \hat{\theta} > \theta_{max} \end{cases}$$

$$x = X$$

$$\hat{y} = \frac{\theta}{k} + y_0$$

$$y = \begin{cases} \hat{y}, & y_{min} < \hat{y} < y_{max} \\ (Y - y_0)C_\theta + (z - \frac{1}{k})S_\theta + \hat{y}, & \hat{y} = y_{min} \text{ ou } y_{max} \end{cases}$$

$$z = \begin{cases} \frac{1}{k} + ((Y - y_0)^2 + (Z - \frac{1}{k})^2)^{\frac{1}{2}}, & y_{min} < \hat{y} < y_{max} \\ -(Y - y_0)S_\theta + (Z - \frac{1}{k})C_\theta + \hat{y}, & \hat{y} = y_{min} \text{ ou } y_{max} \end{cases}$$

Na figura 3.8 vemos um exemplo desta deformação.

3.3 DEFORMAÇÃO POR INVERSÃO DE RAIOS

As abordagens de deformação mostradas também apresentam algumas desvantagens. Os métodos de Barr, por exemplo, definem deformações muito rígidas. No método *free-form* não é possível garantir que uma auto-interseção não ocorrerá quando o objeto for deformado. Como solução para essas questões propomos a utilização de transformações infinitesimais locais que serão integradas. Estas transformações serão então aplicadas utilizando a técnica de **deformação por inversão de raios**.

Geralmente quando se aplicam deformações em objetos gráficos, novos objetos deformados são criados, a partir do original, para serem então renderizados e visualizados pelo usuário, como podemos ver na figura 3.9.

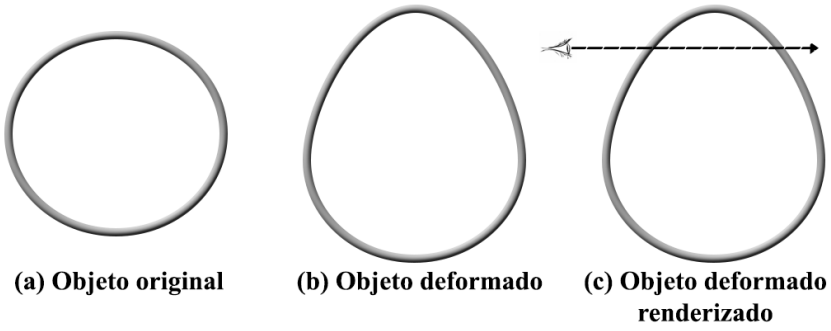

Figura 3.9 Processo de deformação normal.

Na abordagem de deformação por inversão de raios, o passo de criação de um novo objeto deformado pode ser eliminado uma vez que a deformação é aplicada nos raios lançados no objeto durante o *ray casting* tornando assim o processo mais eficiente e rápido. A figura 3.10 mostra esse processo.

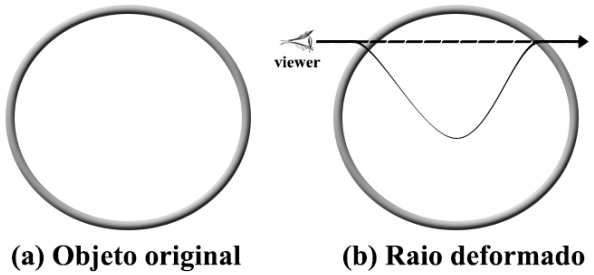

Figura 3.10 Deformação por inversão de raios.

No próximo capítulo apresentaremos o conceito de campos de vetores juntamente com a técnica de deformação desenvolvida neste trabalho.

Capítulo

4

DEFORMAÇÃO INFINITESIMAL DE OBJETOS GRÁFICOS

A teoria das deformações infinitesimais ou teoria das pequenas deformações pode ser vista como uma abordagem matemática que descreve a deformação de um corpo sólido onde os deslocamentos das partículas materiais assumem-se como infinitesimalmente pequenos. Baseando-se nessa teoria apresentamos uma proposta de deformação de objetos gráficos onde a posição de cada ponto pode ser especificada a cada momento de acordo com o campo de vetores que está sendo utilizado. O conceito de **campos de vetores** também é essencial para o bom entendimento do trabalho e será abordado com mais detalhes a seguir.

4.1 CAMPOS DE VETORES

Podemos definir um campo de vetores como uma função que associa um único vetor $\boldsymbol{F}(P)$ a cada ponto P do plano ou do espaço. Temos então a seguinte definição:

Definição 4. *Seja A um conjunto em $\mathbb{R}^n$. Um campo de vetores sobre $\mathbb{R}^n$ é uma aplicação $\boldsymbol{F} : A \subset \mathbb{R}^n \to \mathbb{R}^n$.*

Um exemplo de fácil visualização é o de um campo de velocidades de um fluido. Considere um fluido percorrendo um encanamento com fluxo constante. Associando a cada ponto a velocidade do fluido naquele local obtemos o seu campo de vetores $\boldsymbol{F}$ de velocidades, como mostra a figura 4.1.

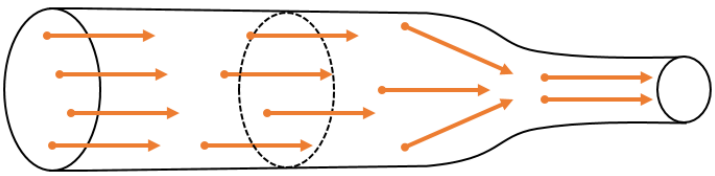

Figura 4.1 Campo de velocidade de um fluido.

Como o fluido é um meio contínuo, isso se refletirá na variação contínua dos valores da velocidade, ao se percorrer o fluido. Nas próximas seções explicaremos melhor o conceito de campos de vetores em um plano e no espaço.

4.1.1 Campos de vetores no plano

Uma função $\mathbf{F} : \mathbf{R}^2 \to \mathbf{R}^2$ pode ser visualizada como um campo de vetores em duas dimensões. Para cada ponto $(x, y)\, in\, \mathbb{R}^2$, avaliamos $\mathbf{F} : (x, y)$ e transladamos este para a posição original, como sugere a figura 4.2. O campo de vetores pode ser visualizado se repetirmos esta operação para cada ponto do domínio.

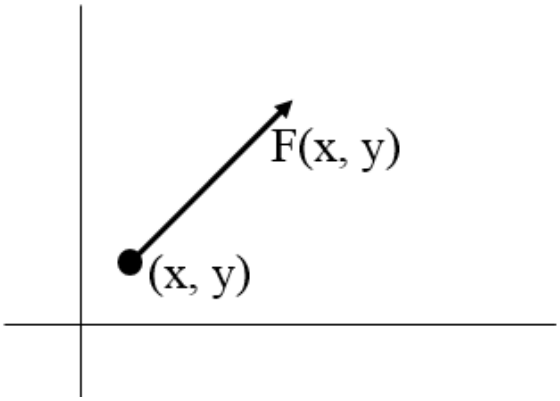

Figura 4.2 Vetor de um ponto.

Tomamos como exemplo a função $\mathbf{F}(x, y) = (-y, x)$ e calculamos os valores da função para um grupo de pontos como mostra a figura 4.3.

F(1, 0) = (0, 1)
F(0, 1) = (-1, 0)
F(2, 0) = (0, 2)
F(0, 2) = (-2, 0)
F(-2, 2) = (-2, -2)

Figura 4.3 Grupo de pontos.

Cada vetor do campo é tangente a uma circunferência centrada na origem, como mostra a figura 4.4. Quando marcamos os vetores a partir dos pontos correspondentes, começamos a visualizar a estrutura do campo de vetores (figura 4.5).

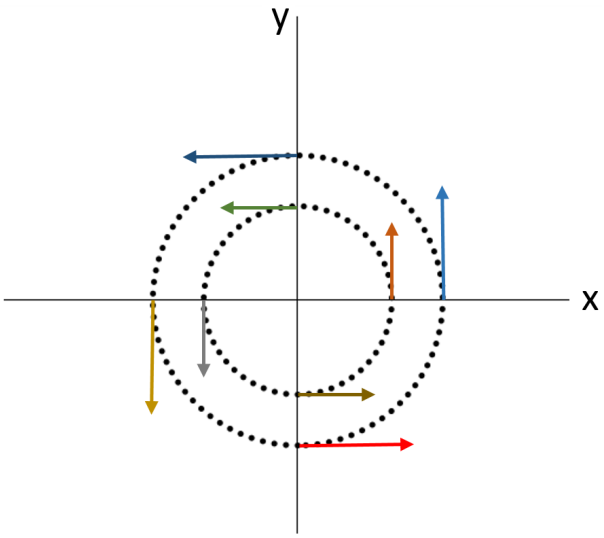

Figura 4.4 Vetores tangentes a circunferências centradas na origem.

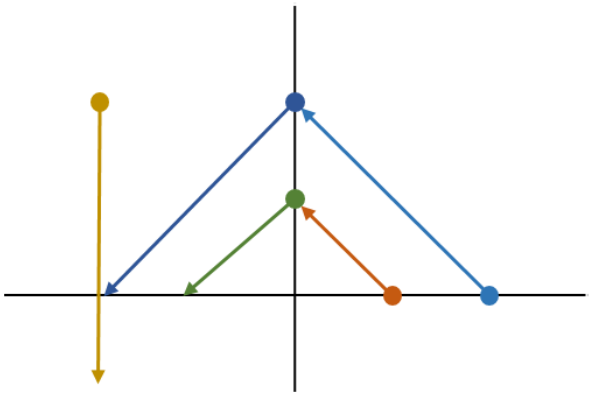

Figura 4.5 Parte da estrutura de um campo de vetores.

Entretanto, se continuarmos a marcar os pontos eles começarão a se sobrepor tornando difícil a visualização do campo. Na figura 4.6 vemos o campo $\mathbf{F}(x, y) = (-y, x)$, com os vetores desenhados em menor escala em relação ao seu tamanho real para melhor entendimento, e percebemos que ele parece rodar na direção anti-horária.

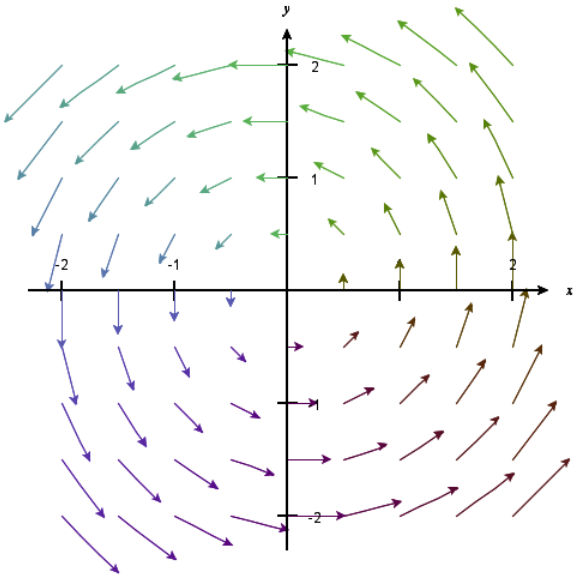

Figura 4.6 Campo de vetores bidimensional $\mathbf{F}(x, y) = (-y, x)$.

Outro exemplo de um campo de vetores pode ser visto na figura 4.7. Neste campo, em cada ponto $(x,\ y)$, o vetor $\mathbf{F}(x, y)$ aponta para fora da origem. Desta forma ficamos com a impressão de que uma explosão emana da origem.

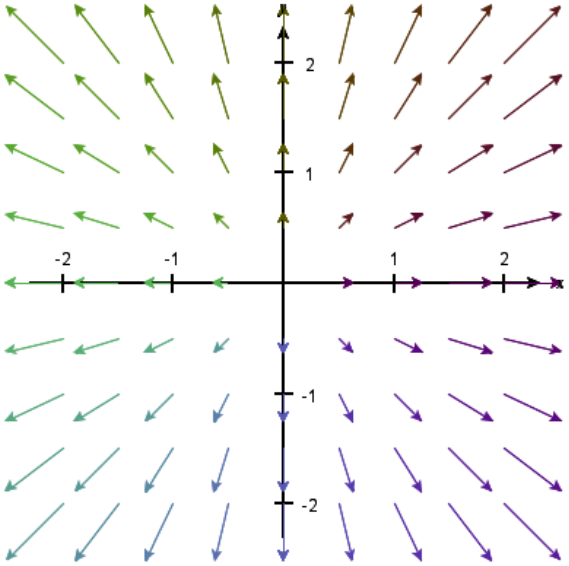

Figura 4.7 Campo de vetores bidimensional $\mathbf{F}(x, y) = (x, y)$.

4.1.2 Campos de vetores no espaço

O princípio de campos de vetores em três dimensões é o mesmo do de duas só que desta vez marcamos os vetores com comprimento proporcional a $\mathbf{F}(x, y, z)$ a partir do ponto

$(x,\ y,\ z)$.

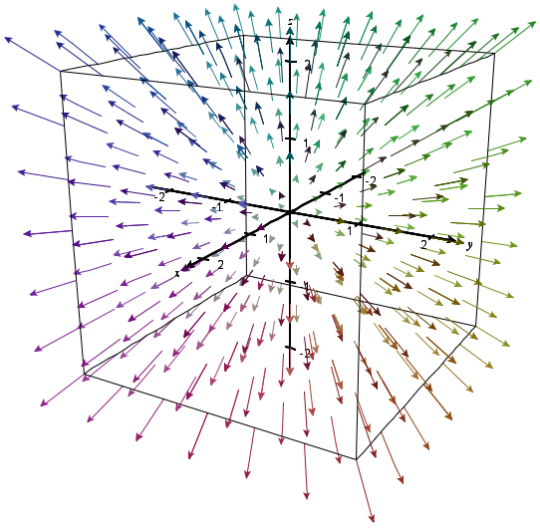

Figura 4.8 Campo de vetores tridimensional $\mathbf{F}(x, y, z) = (x, y, z)$.

Na figura 4.8 mostramos o análogo tridimensional do segundo exemplo de campos de vetores mostrado na seção anterior. Este campo é $\mathbf{F}(x, y, z) = (x, y, z)$. Assim como no caso bidimensional, este campo de vetores corresponde a uma explosão partindo da origem.

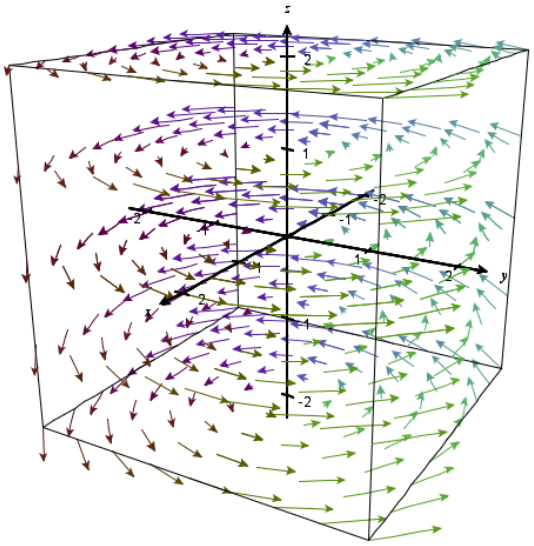

Figura 4.9 Campo de vetores tridimensional $\mathbf{F}(x, y, z) = (-y, x, 0)$.

Já o campo $\mathbf{F}(x, y, z) = (-y, x, 0)$ corresponde a uma rotação em três dimensões, onde o vetor gira ao redor do eixo z, como podemos observar na figura 4.9. Este campo é similar ao mostrado na figura 4.6.

4.2 DEFORMAÇÃO LOCAL COM CAMPOS DE VETORES

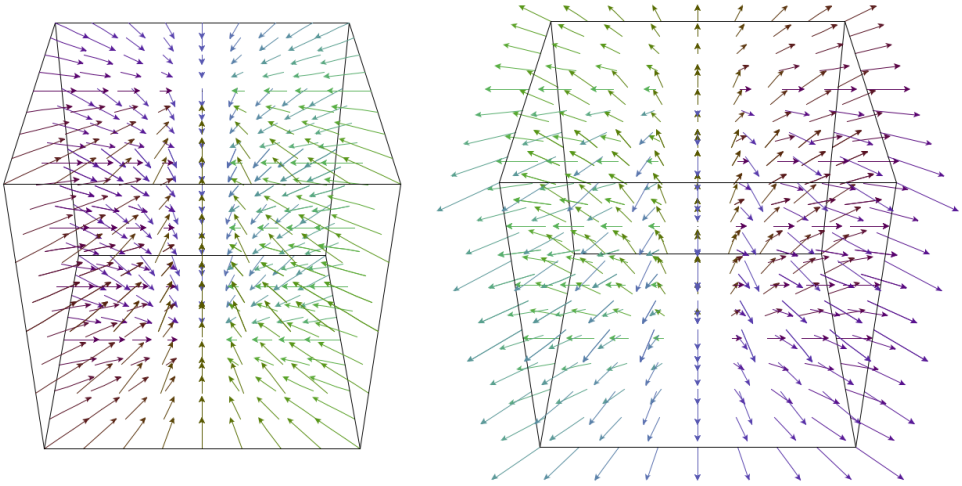

Figura 4.10 Campos de vetores de contração, $\mathbf{F}(x, y, z) = (-x, -y, 0)$, e expansão, $\mathbf{F}(x, y, z) = (x, y, 0)$.

O objetivo deste trabalho foi desenvolver uma nova técnica de deformação de objetos gráficos que realiza o processo de deformação juntamente com o processo de *ray casting*. Esta técnica define que os raios lançados no objeto seguirão caminhos ou fluxos delimitados por campos de vetores predefinidos. Estes campos de vetores serão adaptações dos métodos propostos por Barr e mostrados na seção 3.2. Para realizar estas adaptações utilizaremos o que chamamos de campos de vetores básicos. Tais campos representam movimentos simples como rotação (figura 4.9), expansão e contração (figura 4.10) e translação, que quando aplicados uma função escalar, que depende do valor de z, criarão os campos de deformação de *twist*, *taper* e *bend*.

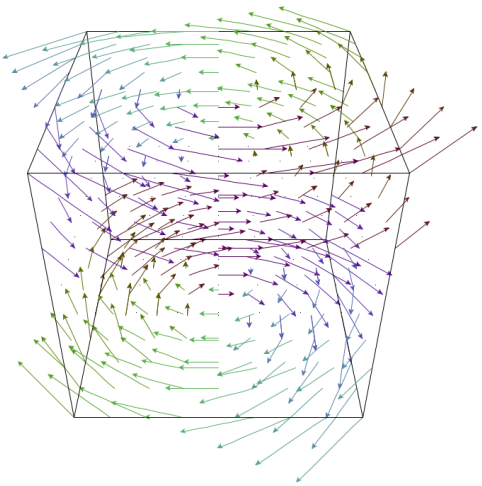

Figura 4.11 Campo de vetores do *twist*, $\mathbf{F}(x, y, z) = (-y, x, 0) * h(z)$.

A deformação *twist*, mostrada na figura 4.11, é definida aplicando ao campo de rotação uma função escalar *h(z)*. A deformação *taper* (figura 4.12) é alcançada com os campos de contração e expansão. Com a aplicação da função escalar, nos valores negativos de z a direção do campo será de contração enquanto nos positivos estará se expandindo.

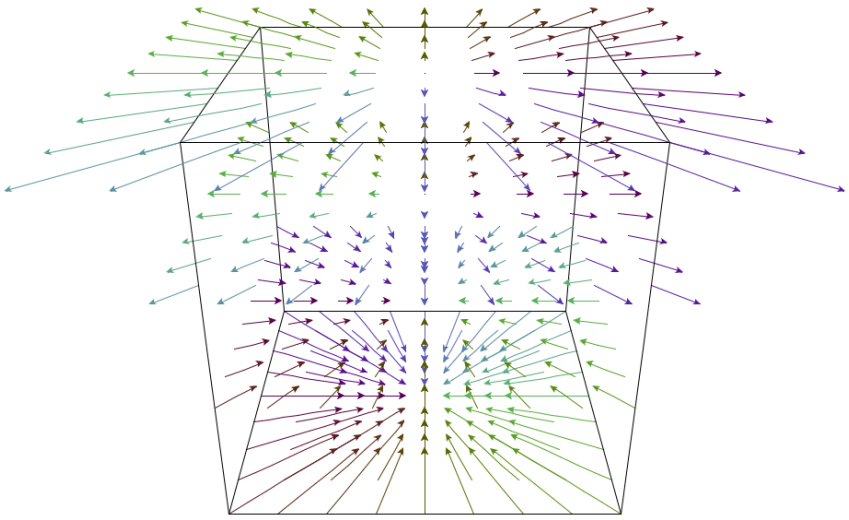

Figura 4.12 Campo de vetores do *taper*, $\mathbf{F}(x, y, z) = (x, y, 0) * h(z)$.

A deformação *bend* não é exatamente igual a deformação de Barr. Nesta adaptação é utilizado o campo básico de rotação ao redor do eixo x (figura 4.13).

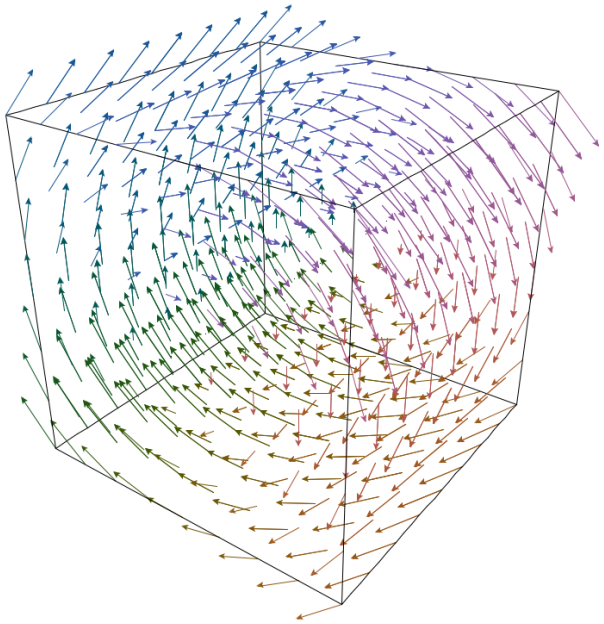

Figura 4.13 Campo de vetores de rotação, $\mathbf{F}(x, y, z) = (0, -z, y)$.

A função escalar aplicada neste campo pode ser:

$$F(x, y, z) = (0, -z, y) * (\arctan(10 * z)/(\pi/2))$$
ou
$$F(x, y, z) = (0, -z, y) * (\text{sign}(z)))$$

A função *sign(z)* retorna -1 caso z seja negativo, 0 se z for 0 e 1 nos casos de z positivo. Assim os valores do campo se inclinarão para o centro dando a impressão de que o campo está se curvando, como mostra a figura 4.14.

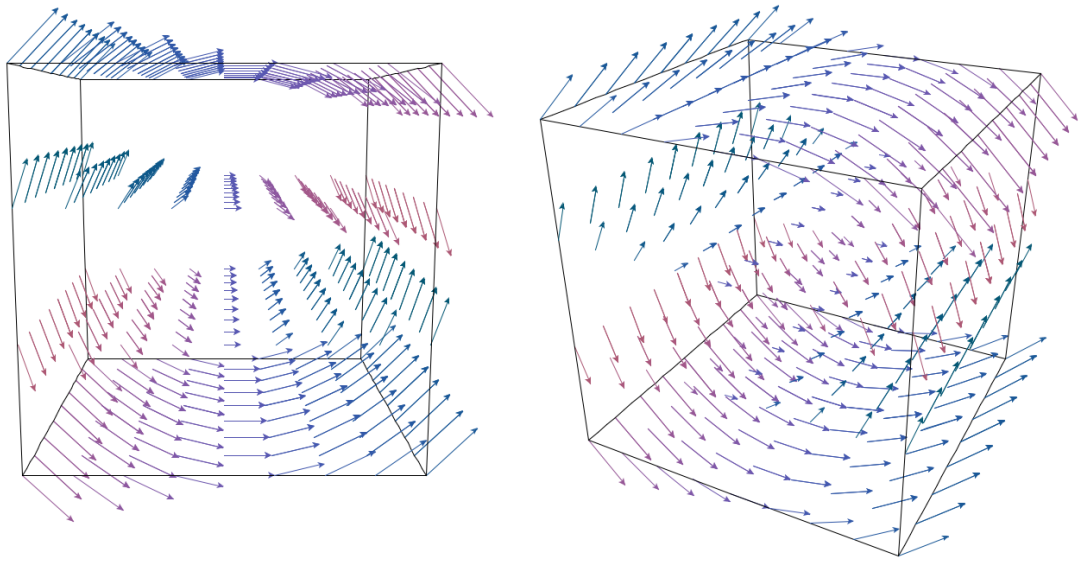

Figura 4.14 Campo de vetores do *bend*, $\mathbf{F}(x, y, z) = (0, -z, y) * h(z)$.

Passamos agora a descrever como funciona o processo de deformação local a partir dos campos de vetores. A ideia básica é transformar cada ponto infinitesimalmente segundo a equação

$$\text{step}(p) = p + dt \cdot \vec{v}(p),$$

onde assumimos que dt é bem pequeno (infinitesimal), e iteramos o processo n vezes por um tempo t, de modo que $dt = t/n$. Ao final do processo, o ponto p é levado em um ponto $I_t(p)$. A função I_t assim definida é denominada *fluxo* associado ao campo $\vec{v}$.

É interessante notar que o mesmo processo pode ser efetuado contra o sentido do campo, ou seja, no sentido do campo $-\vec{v}$, dando origem ao fluxo I_{-t}. Se queremos deformar uma malha de triângulos, basta aplicar o fluxo I_t a cada vértice da malha. Agora suponha que queiramos deformar um objeto implícito $\mathcal{O} : f(q) = 0$. O objeto deformado é o conjunto dos pontos que satisfaz uma certa equação da $\mathcal{O}' : g(p) = 0$. Para saber se um ponto p de fato pertence a $\mathcal{O}'$, temos de saber se p é a imagem de algum ponto q em $\mathcal{O}$ pelo fluxo I_t, ou seja, $p = I_t(q)$. Mas como $I_t^{-1} = I_{-t}$, ou seja, para inverter o fluxo basta andar no sentido contrário do campo, temos que $q = I_{-t}(p)$, e

$$g(p) = f(q) = f(I_{-t}(p)).$$

Por exemplo, a figura 4.15 mostra o trajeto percorrido, destacado em vermelho, por um ponto quando aplicado um campo similar ao mostrado na figura 4.6. O objeto original

f(p) = 0 tem o contorno pontilhado enquanto o do deformado, $g(p) = f(I_{-t}(p)) = 0$, é completo.

Para adicionar uma deformação infinitesimal ao processo de *ray casting* usual, basta incluir o *loop* exibido na figura 4.16 dentro do *loop* executado para cada raio. Claramente, isso encarece bastante o processo.

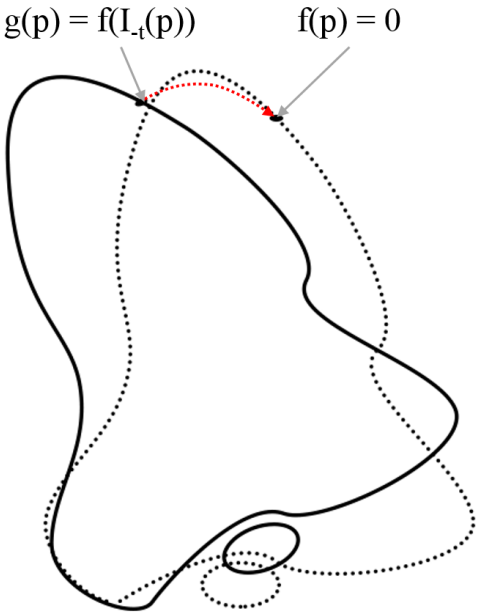

Figura 4.15 Trajeto percorrido por um ponto.

```
1  while(t ≥ dt){
2      p = p + dt · (-v(p));
3      t = t - dt;
4  }
5  return p;
```

Figura 4.16 Cálculo de $I_{-t}(p)$.

Se for necessário calcular o gradiente da função, para cálculo de iluminação local, por exemplo, podemos fazê-lo de duas maneiras: tomando a média das amostragens próximas ao ponto desejado ou através da interpolação tricúbica. A interpolação tricúbica seria a melhor solução para realizar este cálculo uma vez que os resultados alcançados com a média das amostragens não ficam muito perfeitos. O modo de interpolação tricúbica foi utilizado no trabalho (Cabral, 2010). Com a introdução da deformação, é também necessário corrigir o gradiente sobre o objeto deformado, através de um processo de acumulação, como descrito na figura 4.17.

```
1  D = Id
2  while(t ≥ dt){
3     p' = p + dt · (-v(p));
4     D = (Id + dt · (-Dv(p)) · D;
5     t = t - dt;
6     p = p';
7  }
8  return p; D;
```

Figura 4.17 Cálculo de $DI_{-t}(p)$.

A matemática utilizada para calcular a integração do raio ao caminhar no campo é a seguinte: o ponto q é obtido por n iterações da função step, ou seja,

$$q = \text{step}^n(p),$$

com $n \cdot dt = t$. Mas

$$\text{step}^n(p) = \text{step}(\text{step}^{n-1}(p)).$$

Diferenciando e usando a regra da cadeia, segue que

$$D\,\text{step}^n(p) = D\,\text{step}(step^{n-1}(p)) \cdot D\,\text{step}^{n-1}(p).$$

Por outro lado, como

$$\text{step}(p) = p + dt \cdot \vec{v}(p),$$

temos que

$$D\,\text{step}(p) = Id + dt \cdot D\vec{v}(p).$$

Portanto

$$D\,\text{step}^n(p) = (Id + dt \cdot D\vec{v}(\text{step}^{n-1}(p))) \cdot D\,\text{step}^{n-1}(p),$$

o que justifica o algoritmo em 4.17.

Finalmente, uma curiosidade. Como step(p) pode ser escrita como

$$\text{step}(p) = (Id(\cdot) + \frac{t}{n} \cdot \vec{v}(\cdot))(p),$$

segue que

$$q = \text{step}^n(p) = (Id(\cdot) + \frac{t}{n} \cdot \vec{v}(\cdot))^n(p),$$

o que lembra a definição de exponencial, portanto, ao menos formalmente,

$$q \approx e^{t \cdot \vec{v}(\cdot)}(p).$$

4.2.1 *Shaders*

Nesta seção apresentamos os *shaders* que realizarão a deformação. *Shaders* são definidos como pedaços de código que são executados na GPU para manipular imagens antes delas serem desenhadas na tela. Os *shaders* utilizados neste trabalho foram codificados com GLSL (*OpenGL Shading Language*). Na figura 4.18, apresentamos o código do *shader* responsável pelo processo de deformação. As partes mais relevantes do código serão discutidas aqui.

Da linha 5 até a 8 são encontrados os pontos de entrada e saída dos raios e suas profundidades. Em seguida é inicializada a variável que guardará os resultados da composição. A direção e o comprimento do raio são calculados nas linhas 11 e 12. Na 13 o raio é escalado para dar passos menores.

No inicio do *loop* principal do *ray casting*, linha 19, é definida a localização do ponto. Na linha 21 é feita a normalização do mesmo para valores no intervalo de 0 a 1. Em seguida é feita, se necessária, a mudança do eixo principal onde será aplicada a deformação, na linha 22.

O *loop* secundário na linha 25 calcula o caminho percorrido pelo raio no campo de vetores. Lembramos que esse processo é realizado para cada ponto individualmente o que torna o processo não tão rápido já que é um *loop* dentro de um *loop*.

Depois disso os eixos originais são retornados e o valor do ponto volta para intervalos entre -1 e 1, nas linhas 30 e 31. São acessados os valores da amostra no volume e a cor daquela amostra de acordo com a função de transferência. A composição é atualizada na linha 37. O *loop* principal será interrompido caso o raio esteja fora do volume ou se o valor do *alpha* é muito pequeno. Passamos o resultado para a variável *gl_FragColor* na linha 47. Nas linhas 48 a 51 é definido o valor de *gl_FragDepth*. As funções auxiliares deste *shader* podem ser vistas na figura 4.19.

```
void main(void) {
    float t = 0.0;
    float tdepth = -1.0;

    vec3 entry = texture2D(front, texCoord).xyz;
    vec3 exit = texture2D(back, texCoord).xyz;
    float entryDepth = texture2D(frontDepth, texCoord).z;
    float exitDepth = texture2D(backDepth, texCoord).z;

    vec4 result = vec4(0.0);
    vec3 direction = exit - entry;
    float len = length(direction);
    direction = 1.0 / len * direction;

    if(len > 0.0) {
        float acu_tau = 1.0;

        for (int loop = 0; loop < 50000; loop++) {
            vec3 p = entry + t * direction;

            p = 2.0 * p - 1.0;
            p = swizzle_axis(p, true);

            float t_unit = 1.0;
            while(t_unit > Step){
                p = p + Step * apply_vector_field(p);
                t_unit = t_unit - Step;
            }

            p = swizzle_axis(p, false);
            p = (p + 1.0)/2.0;

            float v = texture3D(volume, p).r;
            vec4 color = texture1D(transfer, v);
            float tau = 1.0 - color.a;

            result = result + (1.0 - tau) * (color) * acu_tau * Step;
            acu_tau = acu_tau * pow(tau + 0.000001, Step);
            t = t + Step;

            if(tdepth < 0.0 && acu_tau < 0.9) tdepth = t;
            if(acu_tau < 0.001 || t >= len) exit;
        }
        result.a = 1.0 - acu_tau;
    }

    gl_FragColor = result;
    if(tdepth > 0.0)
        gl_FragDepth = mix(entryDepth, exitDepth, tdepth / len);
    else
        gl_FragDepth = 1.0;
}
```

Figura 4.18 *Main* do *shader* de deformação.

```
float getValueFactor(float z){
    float value = 0.0;

    if(min <= z && z <= max){
        float mid_point = (max + min)/2;
        float range = max - mid_point;
        value = factor * z;

        if(z <= mid_point && min != -1){
            value = ((z - min) * (value))/range;
        } else if(z > mid_point && max != 1){
            value = ((max - z) * (value))/range;
        }
    }

    return value;
}

vec3 apply_vector_field(vec3 p){
    vec3 result = vec3(0.0);
    float value = getValueFactor(p.z);

    if(field == 1){
        //Rotation In Three Dimensions Vector Field
        result = value * vec3(p.y/p.z, -p.x/p.z, 0.0);
    } else if(field == 2){
        //Rotation Vector Field
        result = value * vec3(-p.y, p.x - p.z, p.y);
    } else if(field == 3){
        //Taper Vector Field
        result = value * vec3(p.x, p.y, 0);
    } else if(field == 4){
        //Twist Vector Field
        result = value * vec3(-p.y, p.x, 0.0);
    } else if(field == 5){
        //Bend Vector Field
        float PI = 3.1415;
        float h = atan(10 * p.z)/(PI/2);
        //float h = sign(p.z);
        if(value < 0) value = value * -1;
        result = value * (h * vec3(0, -p.z, p.y));
    }

    return result;
}

vec3 swizzle_axis(vec3 p, bool ch){
    if(axis == 1)
        p = p.zyx;
    else if(axis == 2) {
        if(ch){
            p = p.zxy;
        } else {
            p = p.yzx;
        }
    }

    return p;
}
```

Figura 4.19 Funções auxiliares do *shader* de deformação.

Capítulo 5

VOLREND

Neste capítulo descreveremos os aspectos relevantes na criação da ferramenta *Volrend*. Esta ferramenta permite a visualização e deformação de objetos gráficos utilizando as técnicas apresentadas no capítulo anterior.

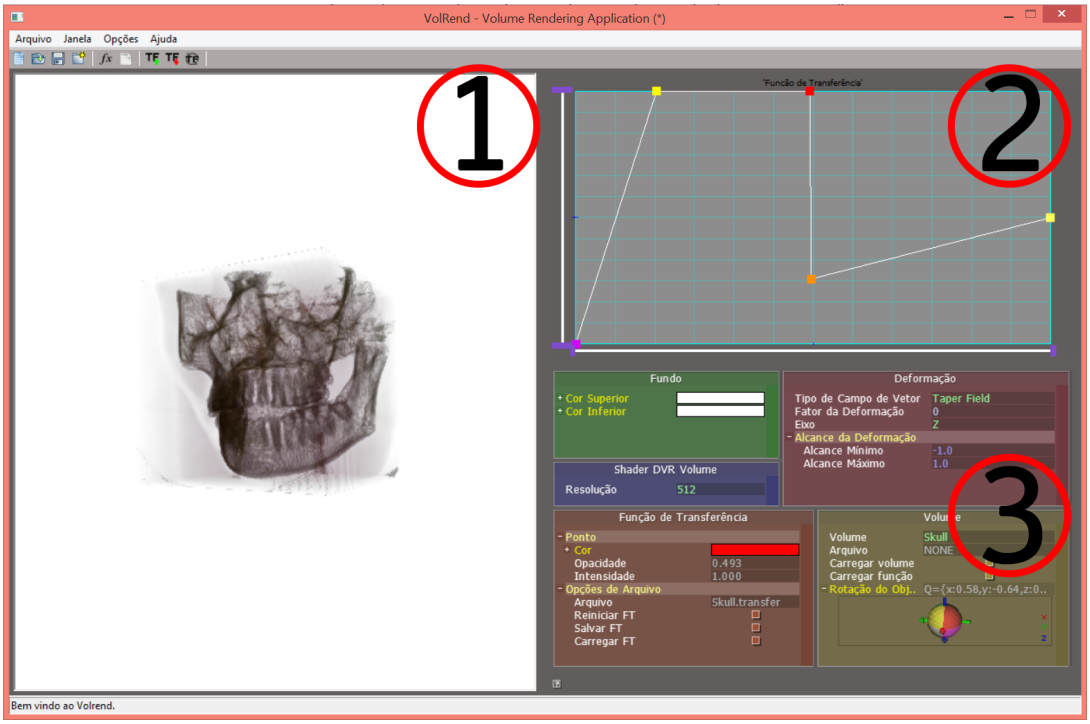

Figura 5.1 Interface gráfica do *Volrend*

5.1 ESTRUTURA

A ferramenta foi desenvolvida utilizando o conceito de redes de fluxo de dados. Em resumo, estas redes são formadas por unidades modulares, também conhecidas como

processadores, que encapsulam os algoritmos de renderização e processamento de dados. Estas unidades possuem portas de entrada (*inports*) e saída (*outports*) por onde recebem dados e enviam os resultados processados na unidade, respectivamente. Cada porta tem um tipo específico, ou seja, somente poderão ser transmitidos dados que condizem com o tipo especificado na porta. A conexão entre *outports* e *inports* constitui o fluxo de dados entre os processadores.

Uma das vantagens desta abordagem é a flexibilidade de arranjo e de criação de novos componentes. Diferentes redes de fluxo de dados podem ser criadas somente levando em conta os dados de entrada e saída trocados pelos processadores uma vez que o comportamento de cada unidade é determinado pela sua configuração.

A figura 5.1 mostra a interface do *Volrend*. A tela que mostra os resultados produzidos na ferramenta é indicada pelo número 1. Na parte 2 está o painel de configuração da função de transferência. E, finalmente, na parte 3 estão os menus de configuração dos processadores utilizados.

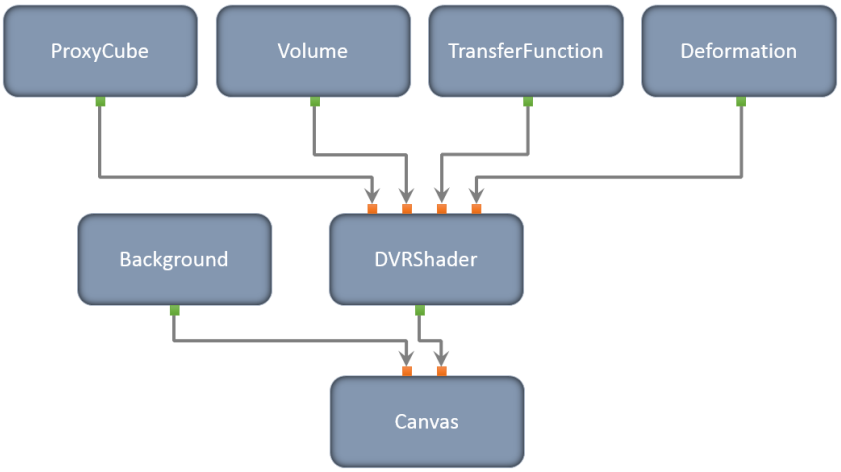

Figura 5.2 Rede de fluxo de dados do *Volrend*

Atualmente a rede de fluxo de dados do *Volrend* é composta por 7 processadores, como mostra a figura 5.2. Os processadores foram desenvolvidos com a linguagem de programação Lua.

5.1.1 Processador *ProxyCube*

Responsável pela criação das texturas com os pontos de entrada e saída armazenados como cores. Estas texturas, mostradas na figura 5.3, são utilizadas no modelo de *ray casting* empregadas neste trabalho para definir os pontos de entrada e saída dos raios nos objetos.

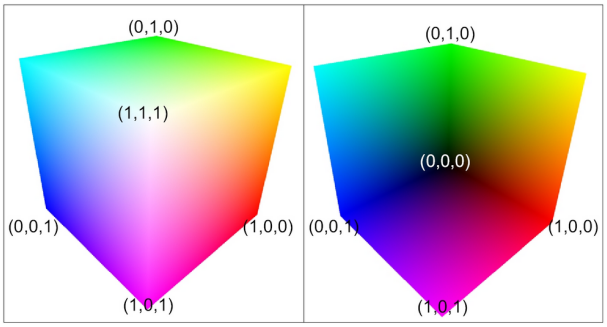

Figura 5.3 Texturas de entrada e saída

Elas formam um cubo que envolve o objeto e possui aresta de comprimento 1 (de *(0, 0, 0)* a *(1, 1, 1)*). A cor de cada vértice é definida de acordo com suas coordenadas no modelo de cores *RGB*, por exemplo, o vértice da textura na posição *(1, 1, 1)* será de cor branca já que tal cor no modelo *RGB*, utilizando o sistema binário, é representada por *RGB(1, 1, 1)*.

5.1.2 Processador *Volume*

Processador responsável por carregar o volume que será visualizado. Pelo menu de configuração do processador, mostrado na figura 5.4, é possível escolher entre os volumes existentes na ferramenta, carregar o arquivo de volume de uma fonte externa, definir um novo volume a partir de uma função e rotacionar o objeto na tela principal.

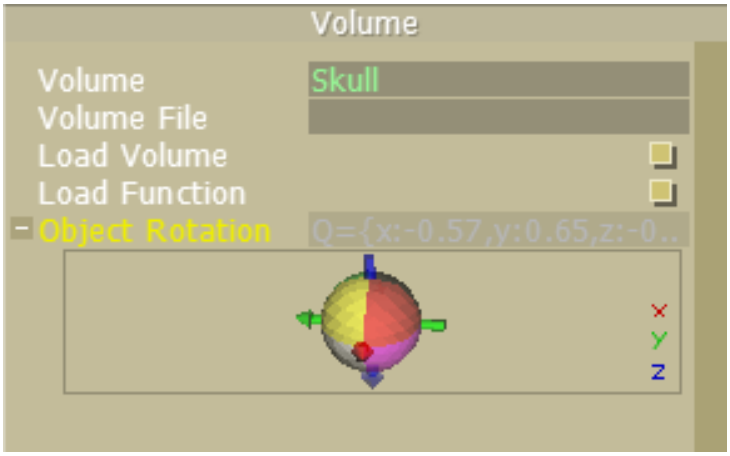

Figura 5.4 Menu de configuração do processador *Volume*.

5.1.3 Processador *TransferFunction*

Responsável pela função de transferência. Durante o processo de visualização, a função de transferência é utilizada para definir os valores de cor e transparência associados a cada ponto do volume, ou seja, será feito o mapeamento dos valores contidos no volume em cores *RGB* características e em transparências que são normalmente representadas por

um número real no intervalo de 0.0 (para totalmente transparente) a 1.0 (para totalmente opaco).

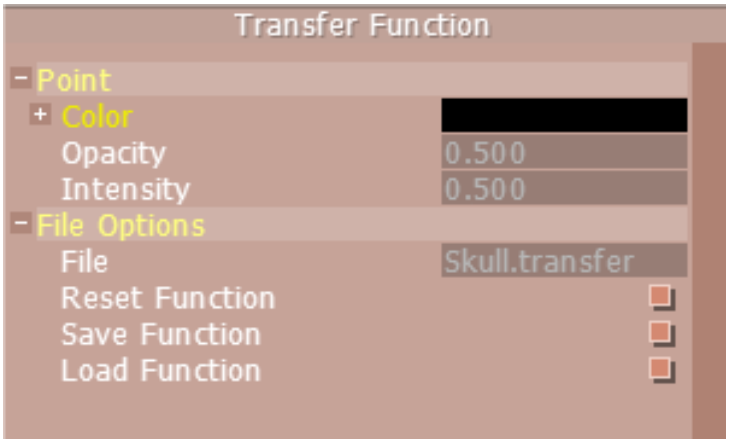

Figura 5.5 Menu de configuração do processador *TransferFunction*.

Utilizando o menu, figura 5.5, é possível salvar e carregar uma função de transferência além de realizar a mudança das cores dos picos da função.

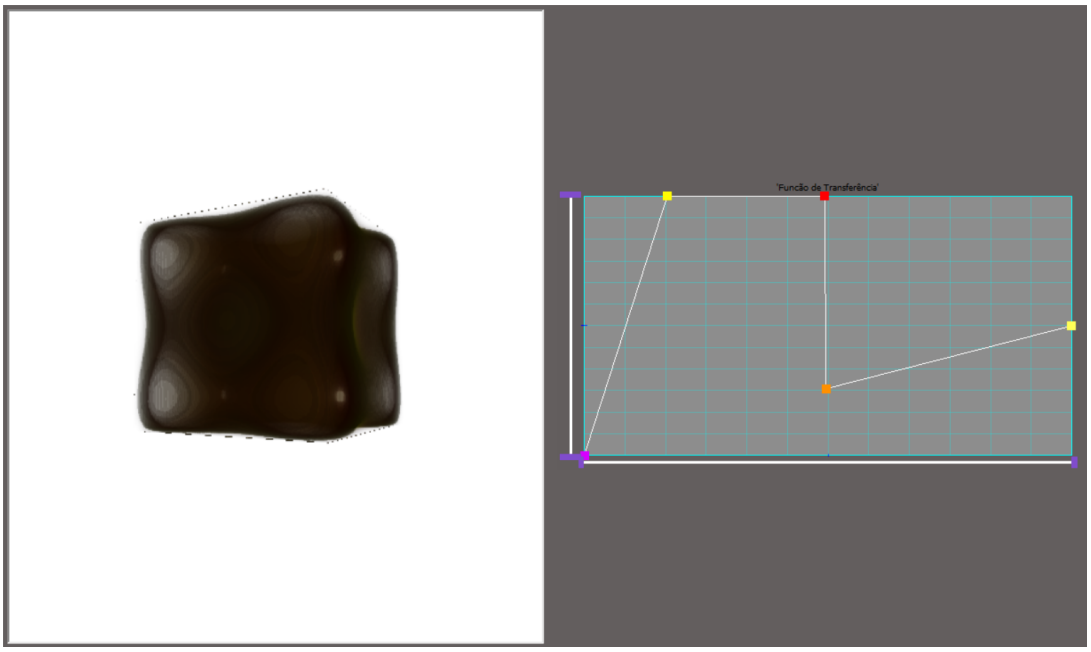

Figura 5.6 Aplicação da função de transferência em um volume.

As figuras 5.6 e 5.7 mostram os resultados da aplicação de diferentes funções de transferência em um volume de um cubo quártico. O painel no lado direito das imagens permite a edição da função que será aplicada no volume. Para mapear os valores do volume em cores e transparência é necessário marcar os "picos" da função no painel.

Estes picos são representados por pontos. Dependendo de sua posição horizontal e de sua cor, um ponto pode definir uma determinada cor *RGB* para uma faixa de intensidade do volume. Desta forma, as estruturas internas e externas do volume podem ser diferenciadas de acordo com a cor. Já a posição vertical do ponto define o valor da transparência de uma determinada camada do volume. Quanto mais alto seja a posição do ponto no painel maior será o seu valor e menos transparente aquela camada será. Podemos assim definir, por exemplo, os valores 0.0 para as camadas externas e 1.0 para as internas, tornando possível uma melhor visualização da estrutura interna do volume.

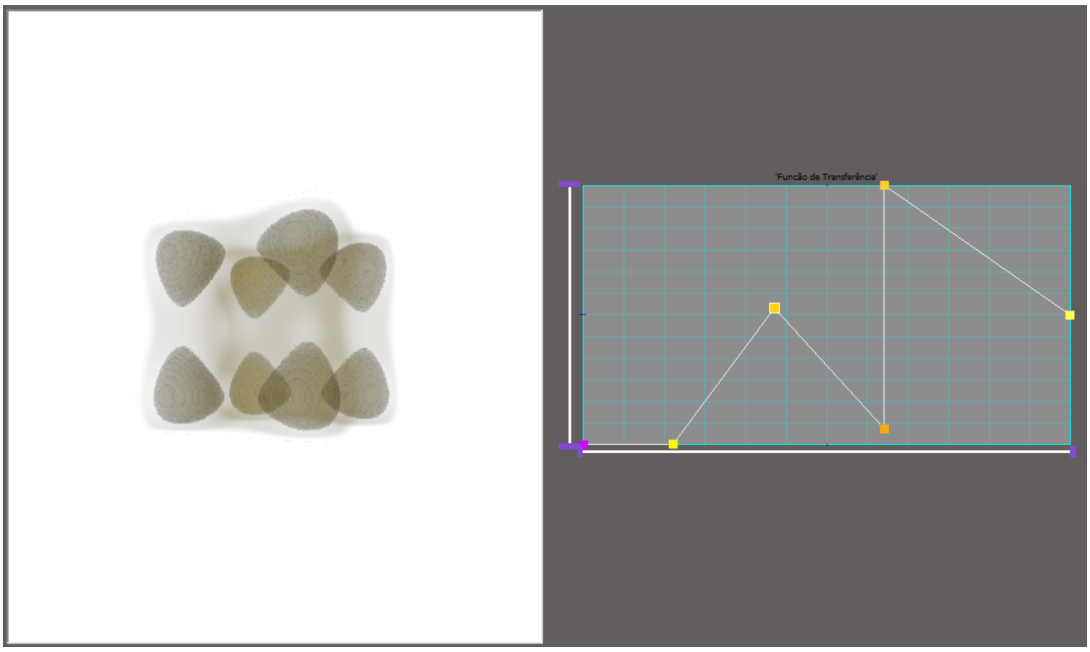

Figura 5.7 Aplicação da função de transferência em um volume.

Podemos ver que a função de transferência do exemplo mostrado na figura 5.6 foi definida de tal forma que a estrutura interna do volume não pode ser visualizada, já no exemplo da figura 5.7 os valores de transparência das camadas externas são menores que os das internas deixando o interior do volume visível para o usuário.

5.1.4 Processador *Deformation*

Neste processador são configurados os parâmetros da deformação. Como podemos ver na figura 5.8, é possível escolher qual a deformação que será aplicada no volume, o eixo, o alcance (qual parte do volume será afetada) e, por fim, o fator que vai determinar a quantidade de deformação aplicada no objeto.

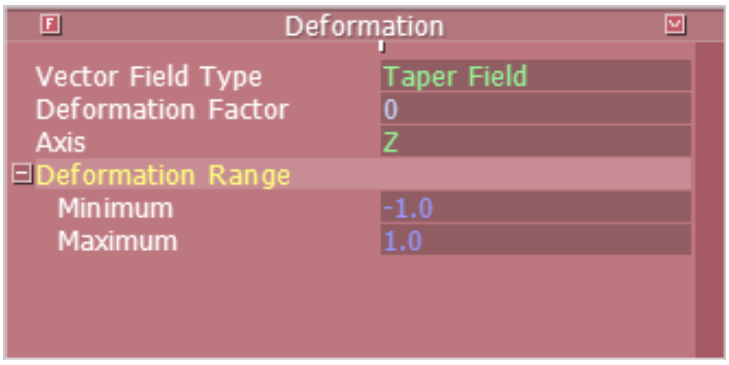

Figura 5.8 Menu de configuração do processador *Deformation*.

5.1.5 Processador *DVRShader*

Recebe como entrada dados vindos dos processadores *ProxyCube*, *Volume*, *TransferFunction* e *Deformation*. Aqui são aplicados os *shaders* responsáveis pela renderização e deformação do volume. Pelo menu deste componente, mostrado na figura 5.9, o usuário pode escolher entre três tipos de resolução: *128*, *256* e *512*. Os shaders utilizados neste processador são descritos na seção 4.2.1

Figura 5.9 Menu de configuração do processador *DVRShader*.

5.1.6 Processador *Background*

Neste processador são definidas as cores do plano de fundo na tela de resultados. O usuário pode modificar as cores da parte superior e inferior da tela utilizando o menu indicado na figura 5.10.

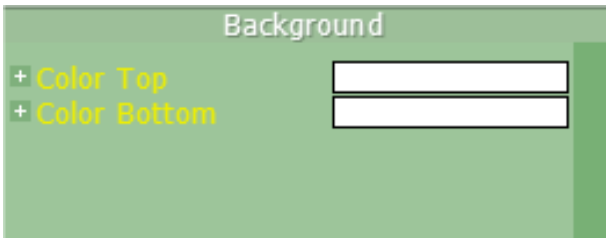

Figura 5.10 Menu de configuração da cor do *background*.

5.1.7 Processador *Canvas*

Recebe dados dos processadores *Background* e *DVRShader*. Com estes dados este processador mostra os resultados na tela.

Capítulo

6

RESULTADOS

Este capítulo será voltado a descrição dos resultados obtidos ao utilizar a ferramenta *Volrend* e ao aplicar os métodos de deformação implementados neste trabalho.

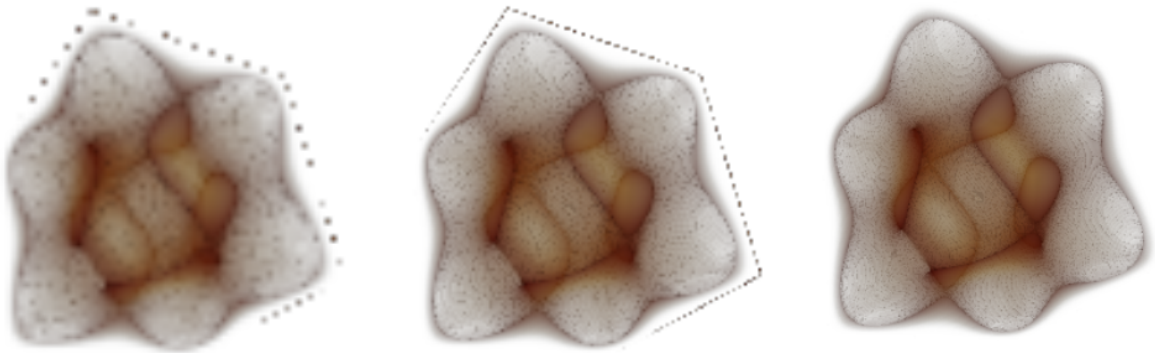

Figura 6.1 Resultados obtidos utilizando diferentes resoluções no cubo quártico.

No capítulo 5 descrevemos os processadores que compõem a ferramenta. Um destes processadores permite a modificação da resolução do volume. As imagens 6.1 e 6.2 mostram a visualização de volumes no *Volrend* em diferentes resoluções. A resolução mínima é de 128 (figuras 6.1(a) e 6.2(a)) e a média é de 256 (figuras 6.1(b) e 6.2(b)).

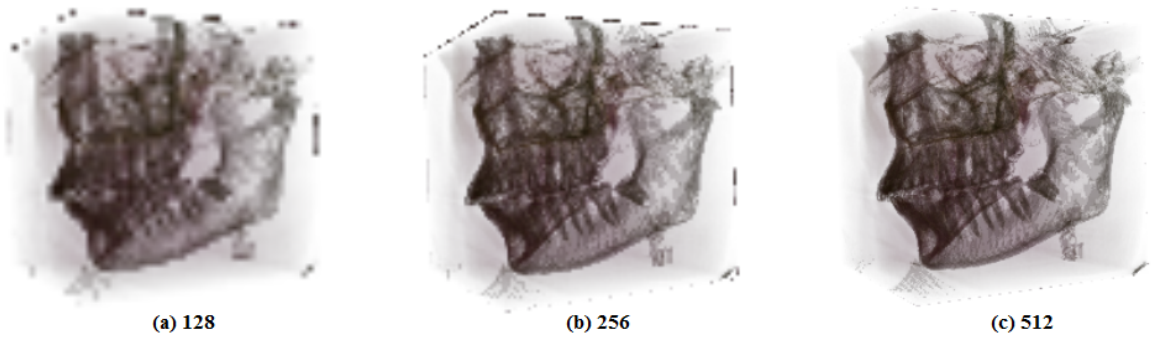

Figura 6.2 Resultados obtidos utilizando diferentes resoluções em um volume.

Os melhores resultados ocorrem quando utilizamos a resolução alta, de 512, mostrada nas figuras 6.1(c) e 6.2(c).

Na figura 6.3 podemos observar os efeitos da deformação *Taper* aplicada no volume. As imagens mostram o volume original (6.3(a)), e o deformado em três posições: lateral (6.3(b)) visão frontal (6.3(c)) e aérea (6.3(d)). Nessas imagens a deformação foi aplicada ao longo do eixo y e com força 40.

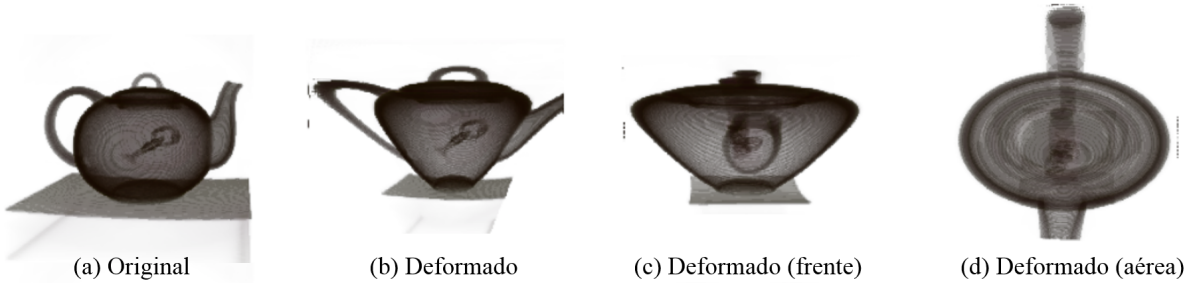

Figura 6.3 Resultados obtidos com o campo de vetores *Taper* no volume de bule de chá.

A imagem 6.4 mostra imagens dos resultados adquiridos com a aplicação da deformação *Twist* ao longo do eixo y com força 60.

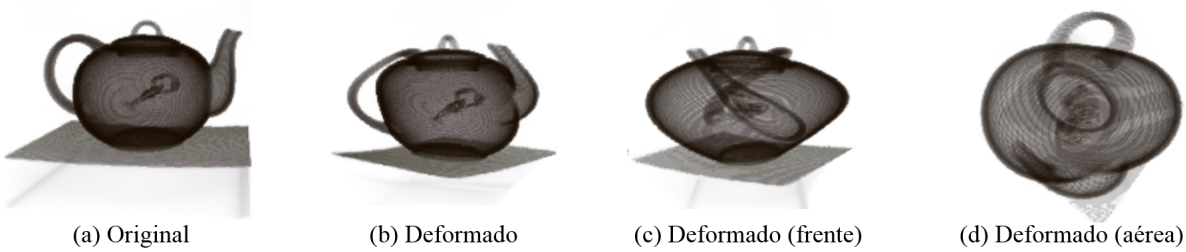

Figura 6.4 Resultados obtidos com o campo de vetores *Twist* no volume de bule de chá.

A figura 6.5 exemplifica os resultados alcançados com a deformação *Bend* aplicada ao longo do eixo y com força 40.

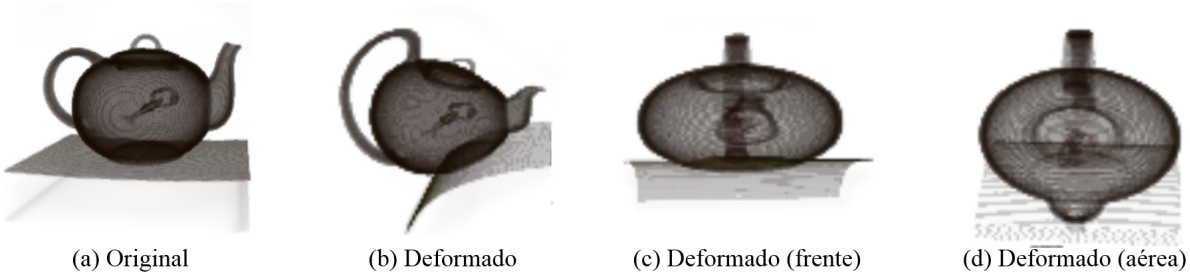

Figura 6.5 Resultados obtidos com o campo de vetores *Bend* no volume de bule de chá.

Também foi mostrado no capítulo 5 que é possível definir a região do volume que será afetada pela deformação. Na imagem 6.6 podemos observar as diferenças de aplicações totais e parciais do *Taper*. Na figura 6.6(a) vemos a deformação em todo o volume. Já em 6.6(b) a deformação é aplicada somente em 75% do volume. E, por fim, em 6.6(c) visualizamos o volume com a deformação aplicada somente em sua metade inferior.

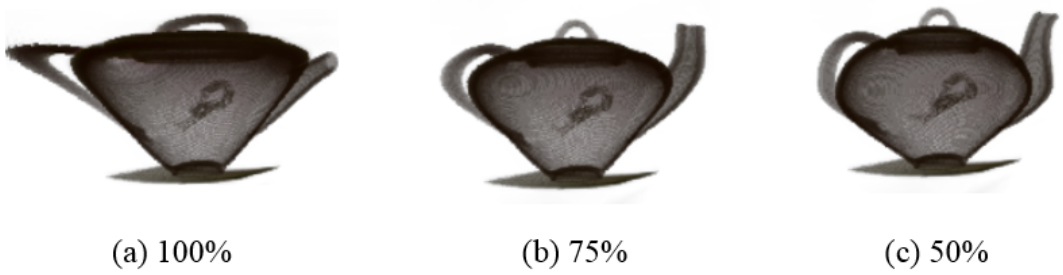

Figura 6.6 Aplicação da deformação *Taper* em determinadas regiões do volume. As legendas informam a porcentagem do volume que está sendo afetada pela deformação.

As imagens 6.7 e 6.8 também mostram resultados das aplicações das deformações em regiões definidas no volume mas desta vez com as deformações *Twist* e *Bend*, respectivamente.

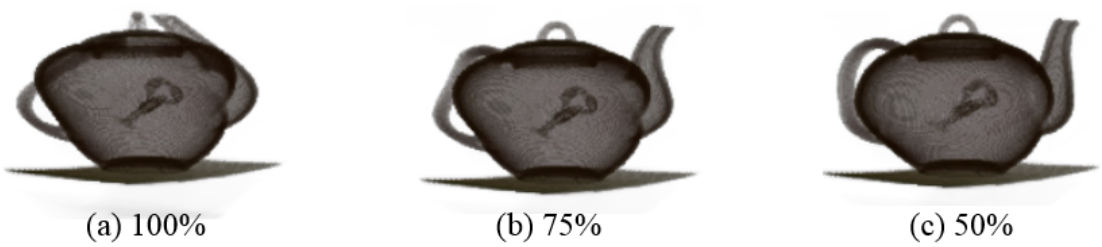

Figura 6.7 Aplicação da deformação *Twist* em determinadas regiões do volume.

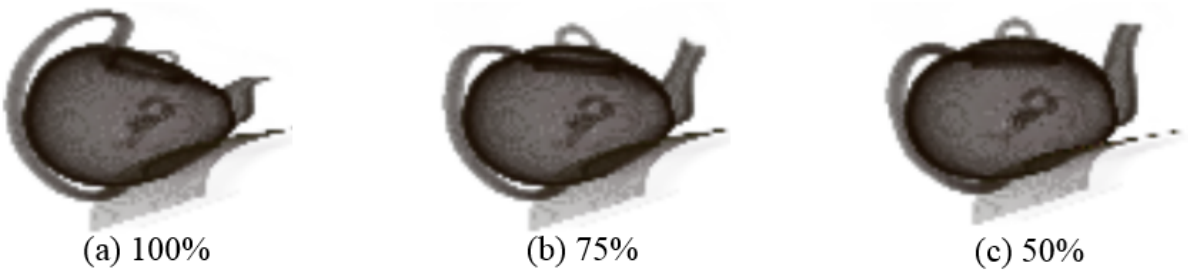

Figura 6.8 Aplicação da deformação *Bend* em determinadas regiões do volume.

Para finalizar, a Tabela 6.1 apresenta a porcentagem de *frames* por segundo (*FPS*) durante o funcionamento da ferramenta. Comparamos os *FPSs* do *ray casting* com o *loop* tradicional e do *ray casting* com deformação, que adiciona mais um *loop* ao processo, nas resoluções mínima, média e alta.

Tabela 6.1 Quantidade de Frames Por Segundo

Ray Casting	**Resolução**	**Frames Por Segundo (FPS)**
Loop tradicional	128	12.11
	256	11.50
	512	10.2
Loop tradicional + *Loop* de deformação	128	5.89
	256	4.12
	512	3.32

Notamos que com o processamento adicional e a medida que a resolução vai aumentando, a taxa de *FPS* diminui. A partir destes resultados podemos perceber que, além deste *loop* adicional, se calcularmos também as normais com a interpolação tricúbica e realizarmos a acumulação das matrizes das normais, para realizar a iluminação, o tempo de processamento iria aumentar mais.

Capítulo 7

CONSIDERAÇÕES FINAIS

Neste trabalho estudamos técnicas de deformação de objetos gráficos utilizando campos de vetores. Para tanto implementamos um método de deformação por inversão de raios combinando *ray-casting* e campos de vetores. Os campos de vetores foram adaptados dos métodos de deformação propostos por Barr (Barr, 1984). Aqui aplicamos o método de deformação para objetos implícitos mas este também pode ser utilizado para objetos paramétricos como as malhas de triângulos.

A ferramenta *Volrend* desenvolvida neste trabalho foi implementada com a linguagem de programação Lua. Sua estrutura é baseada no conceito de redes de fluxos de dados e é composta por 7 processadores, cada um com uma função específica. O *ProxyCube* cria as texturas que serão utilizadas para definir os pontos de entrada e saída dos raios no processo de *ray-casting*. O processador *Volume* carrega o volume escolhido pelo usuário para a aplicação. Os processadores *Deformation* e *TransferFunction* permitem configurar os parâmetros da deformação e da função de transferência, respectivamente. No processador *DVRShader* são aplicados os *shaders* que realizam a renderização e a deformação do objeto. O *Background* é reservado para modificar a cor de fundo da tela de resultados. Por fim, o *Canvas* desenha os resultados do processo de renderização na tela. Os algoritmos dos *shaders* utilizados na ferramenta foram implementados em *GLSL*.

Como trabalhos futuros, destacamos a implementação de novos campos de vetores com outros tipos de deformação, campos de vetores obtidos a partir de experimentos ou simulações. A aplicação de um processo de iluminação também é um possível trabalho futuro. Este processo não foi implementado pois seria necessário calcular o gradiente, com o método de interpolação tricúbica, e também fazer a acumulação das normais. Tudo isso iria encarecer ainda mais o processo de deformação. Mas esperamos que com o avanço da tecnologia o processo de iluminação possa ser realizado em tempo real e de forma satisfatória.

REFERÊNCIAS BIBLIOGRÁFICAS

Barr, A. H. Global and local deformations of solid primitives. *ACM Computer Graphics*, v. 18, p. 21–30, 1984.

Beier, T.; Neely, S. Feature-based image metamorphosis. *Proceedings of SIGGRAPH'92*, v. 26, n. 2, p. 35–42, 1992.

Bourke, P. *Polygonising a scalar field*. 1994. Disponível em: <`http://paulbourke.net/geometry/polygonise/`>.

Cabral, A. N. T. *Visualização da Curvatura de Objetos Implícitos em um Sistema Extensível*. Dissertação (Mestrado) — Universidade Federal de Alagoas, 2010.

Cline, H. et al. Two algorithms for three-dimensional reconstruction of tomograms. *Medical Physics*, v. 15, n. 3, p. 320–327, 1988.

Darsa, L. *Deformação e Metamorfose de Objetos Gráficos*. Dissertação (Mestrado) — PUC-Rio, 1994.

Foley, J. et al. *Computer Graphics: Principles and Practice*. Second edition. [S.l.]: Addison-Wesley, 1990.

Gibson, S.; Mirtich, B. *A Survey of Deformable Modeling in Computer Graphics Technical Report*. [S.l.], 1997.

Gomes, J.; Velho, L. *Fundamentos da Computação Gráfica*. [S.l.]: IMPA, 2008.

Gomes, J. M.; Velho, L. C. *Conceitos Básicos de Computação Gráfica*. [S.l.]: Sao Paulo, IME-USP, 1990.

Jun, Z.; Xia, D.; Tiangang, D. A non-linear warping method for face hallucination based-on subdivision mesh. *2nd International Congress on Image and Signal Processing*, p. 01–05, 2009.

Kaufman, A. E. Volume visualization. *ACM Computing Surveys*, v. 28, n. 1, p. 165–167, 1996.

Lorensen, W. E.; Cline, H. E. Marching cubes: a high resolution 3d surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, v. 21, p. 163–169, 1987.

Mccormick, B.; Defanti, T.; Brown, M. Visualization in scientific computing. *Computer Graphics*, v. 21, n. 6, p. 144–153, November 1987.

Montenegro, A. *Computação Gráfica I - Objetos Gráficos Espaciais*. 2008. Disponível em: <http://www2.ic.uff.br/~anselmo/cursos/CGI/slidesGrad/CG_aula9%28objetosgraficos3D%29.pdf>.

Montenegro, A. *Computação Gráfica I - Objetos Gráficos Planares*. 2008. Disponível em: <http://www2.ic.uff.br/~anselmo/cursos/CGI/slidesGrad/CG_aula2%28objetosgraficos2D%29.pdf>.

Moore, P.; Molloy, D. A survey of computer-based deformable models. *International Machine Vision and Image Processing Conference (IMVIP 2007)*, p. 55–66, 2007.

Paiva, A. C.; Seixas, R. B.; Gattass, M. *Introdução a Visualização Volumétrica*. Rio de Janeiro: PUC-RIo, Inf. MCC03/99, 1999.

Phong, T. B. Illumination for computer generated pictures. *Communications of the ACM*, v. 18, n. 6, p. 311–317, 1975.

Sederberg, T.; Parry, S. Free-from deformation od solid geometric models. *Proceedings of SIGGRAPH'86*, v. 20, n. 4, p. 151–160, 1986.

Upson, C.; Keeler, M. V-buffer - visible volume rendering. In: *Computer Graphics*. [S.l.: s.n.], 1988. v. 22, n. 4.

Vieira, T. *Objetos Gráficos Espaciais*. 2011. Disponível em: <http://www.im.ufal.br/professor/thales/icg/Aula5.pdf>.

Westover, L. Footprint evaluation for volume rendering. *Computer Graphics*, v. 24, n. 4, p. 367–376, 1990.