

**UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO**

**ESTUDO E IMPLEMENTAÇÃO DE UM
MODELO DE FLUXO DE FLUÍDO
PARA GERAÇÃO DE FUNÇÃO DE
TRANSFERÊNCIA EM *DIRECT*
*VOLUME RENDERING***

TRABALHO DE GRADUAÇÃO

Luiz Felipe Netto

Santa Maria, RS, Brasil

2014

**ESTUDO E IMPLEMENTAÇÃO DE UM MODELO DE
FLUXO DE FLUÍDO PARA GERAÇÃO DE FUNÇÃO
DE TRANSFERÊNCIA EM *DIRECT VOLUME*
*RENDERING***

por

Luiz Felipe Netto

Trabalho de Graduação apresentado ao Curso de Ciência da Computação
da Universidade Federal de Santa Maria (UFSM, RS), como requisito
parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Prof. Dr. Cesar Tadeu Pozzer (UFSM)

Co-orientador: Prof. Dr. Lisandra Manzoni Fontoura (UFSM)

Trabalho de Graduação N. 370

Santa Maria, RS, Brasil

2014

**Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação**

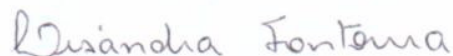
A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

**ESTUDO E IMPLEMENTAÇÃO DE UM MODELO DE FLUXO DE
FLUÍDO PARA GERAÇÃO DE FUNÇÃO DE TRANSFERÊNCIA
EM *DIRECT VOLUME RENDERING***

elaborado por
Luiz Felipe Netto

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

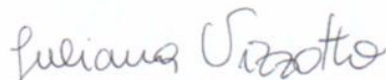
COMISSÃO EXAMINADORA:



Prof. Dr. Lisandra Manzoni Fontoura (UFSM)
(Presidente/Co-orientador)



Prof. Dr. Ana Trindade Winck (UFSM)



Prof. Dr. Juliana Kaizer Vizzotto (UFSM)

Santa Maria, 22 de Janeiro de 2014.

AGRADECIMENTOS

Agradeço aos meus pais e a minha irmã por sempre acreditarem e confiarem em mim, apoiando cada passo e cada decisão que tomei em direção ao meu futuro.

Agradeço a Darcielle, companheira, amiga e para sempre uma pessoa importante em minha vida.

Agradeço aos meus amigos, aqueles que convivi no dia-a-dia e aqueles que não consegui demonstrar o quanto foram importantes nesta trajetória.

Agradeço ao meu orientador, professor Cesar Pozzer, por acreditar em meu potencial e proporcionar a oportunidade de trabalhar nesta área incrível que é a Computação Gráfica.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

ESTUDO E IMPLEMENTAÇÃO DE UM MODELO DE FLUXO DE FLUÍDO PARA GERAÇÃO DE FUNÇÃO DE TRANSFERÊNCIA EM *DIRECT VOLUME RENDERING*

Autor: Luiz Felipe Netto

Orientador: Prof. Dr. Cesar Tadeu Pozzer (UFSM)

Co-orientador: Prof. Dr. Lisandra Manzoni Fontoura (UFSM)

Local e data da defesa: Santa Maria, 22 de Janeiro de 2014.

Dados obtidos através de tomografia computadorizada podem ser visualizados como dados volumétricos através do método Direct Volume Rendering (DVR). DVR faz uso de Funções de Transferência para atribuir opacidade e cor a cada ponto do conjunto de dados. Este trabalho apresenta o desenvolvimento de um método para geração da Função de Transferência utilizando um modelo de fluxo de fluido, o qual é responsável por atribuir a opacidade de cada ponto eficientemente sem a necessidade de vários parâmetros de inicialização.

Palavras-chave: Contour tree; direct volume rendering; função de transferência; visualização científica; imageamento médico.

LISTA DE FIGURAS

2.1	Exemplo do pipeline de <i>Volume Rendering</i>	15
2.2	Conjunto de imagens paralelas ao longo de um eixo z.	15
2.3	Exemplo do pipeline conceitual para <i>Direct Volume Rendering</i> descrito por (LJUNG, 2006)	16
2.4	Imagem resultante de DVR aplicado ao <i>dataset</i> bonsai.	16
2.5	Espaço de atributos (ZHOU; XIAO; TAKATSUKA, 2013)	20
2.6	Ilustração do experimento de Darcy (AUMANN; DAVID FORD, 2002)	21
2.7	Ilustração da opacidade fluindo em uma árvore de contornos (ZHOU; TAKATSUKA, 2009)	22
3.1	Diagrama de atividades para solução proposta	25
3.2	Diagrama da estrutura de dados <i>FeatureSet</i>	27
3.3	Diagrama do arquivo de cabeçalho <i>ctfunc.h</i>	27
4.1	Ilustração da árvore de contornos extraída do <i>dataset</i> Nucleon	34

LISTA DE TABELAS

4.1	Resultados obtidos para o <i>dataset Nucleon</i>	33
4.2	Resultados obtidos para o <i>dataset Hydrogen Atom</i>	34

LISTA DE CÓDIGOS

3.1	Função principal <i>int main()</i>	28
3.2	Classe VolHypervolAcc	29
3.3	Funções para o cálculo das medidas de importância	30
3.4	Implementação do modelo de fluxo de fluídos	31

LISTA DE ABREVIATURAS E SIGLAS

OpenGL	Open Graphics Library
MATLAB	Matrix Laboratory
DVR	Direct Volume Rendering
LaCA	Laboratório de Computação Aplicada
API	Application Programming Interface

SUMÁRIO

1	INTRODUÇÃO	11
1.1	Objetivos	12
1.1.1	Objetivos gerais	12
1.1.2	Objetivos específicos	12
2	REVISÃO BIBLIOGRÁFICA	14
2.1	<i>Volume Rendering</i>	14
2.1.1	<i>Direct Volume Rendering</i>	16
2.2	Árvore de Contornos	19
2.2.1	Definição	19
2.2.2	<i>Branch decomposition</i>	19
2.2.3	Simplificação	20
2.3	Modelo de Fluxo de Fluido em Meio Poroso	20
2.3.1	Lei de Darcy	21
2.3.2	Geração de Função de Transferência Baseada no Modelo de Fluxo de Fluido	21
3	PROJETO E IMPLEMENTAÇÃO	25
3.1	Obtenção e Decomposição	26
3.2	Simplificação	26
3.3	Modelo de Fluxo de Fluido	27
3.4	Implementação	28
3.4.1	Criação e Decomposição da Árvore de Contornos	28
3.4.2	Simplificação da Árvore de Contornos	29
3.4.3	Fluxo de fluido	30
4	RESULTADOS	33
5	CONCLUSÃO	36
5.1	Trabalhos futuros	36
	REFERÊNCIAS	38

1 INTRODUÇÃO

Grandes volumes de dados médicos são gerados diariamente durante a realização de exames em pacientes de diversos centros clínicos. Equipamentos avançados são necessários para a obtenção, análise e diagnóstico dos dados provenientes destes exames. Porém, mesmo com estes equipamentos cada etapa da análise de dados requer vários ajustes por parte do especialista, trazendo assim a necessidade de técnicas computacionais automatizadas para este fim. O tempo investido em imageamento médico é decorrência de vários fatores. Após a obtenção dos dados um aplicativo disponibiliza a visualização dos dados do paciente ao especialista em questão. Nestes aplicativos de visualização temos o tempo gasto com a renderização dos dados e o tempo gasto com a definição de parâmetros.

Diversas pesquisas são desenvolvidas buscando melhorar a visualização de dados volumétricos, obtidos através de tomografia computadorizada, fazendo uso de *Direct Volume Rendering* (DVR) (MAX, 1995). Porém, a grande maioria dos métodos de visualização que fazem uso do DVR, requerem vários ajustes manuais de parâmetros para visualização das áreas de interesse dos dados renderizados. Além disso, a qualidade de renderização está relacionada com a interatividade entre o usuário e o programa, o qual deve manter uma boa taxa de atualização de tela (FPS - *Frames-per-Second*) para garanti-la em detrimento de maiores detalhes (LJUNG, 2006).

No pipeline do DVR, a qualidade e a eficiência do método estão intimamente ligadas com a Função de Transferência, que é responsável por atribuir a cor e a opacidade a cada ponto do conjunto de dados a ser renderizado na tela. Desta forma, este trabalho apresenta uma análise dos métodos de geração de Função de Transferência através de um modelo de fluxo de fluido. O modelo de fluxo de fluido é aplicado a uma estrutura de dados chamada Árvore de Contornos. Esta é gerada a partir dos dados volumétricos em uma etapa anterior, distribuindo o valor de opacidade para cada nó no sentido da raiz para as

folhas, o qual é necessário para geração da Função de Transferência.

Inicialmente os métodos de Direct Volume Rendering utilizavam Funções de Transferência unidimensionais para explorar os dados volumétricos. A Função de Transferência era gerada a partir do processamento de um conjunto de parâmetros pré-computados e selecionados pelo usuário, sendo aplicada de forma global ao conjunto de dados (ZHOU; TAKATSUKA, 2009). Novos métodos para geração de Função de Transferência Multidimensionais foram introduzidos fazendo uso de histogramas de volume para capturar as informações internas como bordas de objetos. Entretanto a geração desta ainda requer que o usuário defina vários parâmetros durante a execução do algoritmo (ZHOU; TAKATSUKA, 2009).

O foco deste trabalho é o desenvolvimento de um método que minimize a interação manual do usuário durante a definição da Função de Transferência. O aumento de eficiência, entendida como a facilidade de utilização e obtenção de informação, em aplicativos de visualização científica aplicado a imageamento médico é continuo alvo de pesquisa (LJUNG, 2006). Este trabalho visa integração com o trabalho de dissertação de mestrado do aluno Guilherme Gonçalves Schardong.

1.1 Objetivos

1.1.1 Objetivos gerais

Esta pesquisa tem como objetivo desenvolver uma aplicação para gerar a Função de Transferência utilizando um modelo de fluxo de fluido, no qual ocorre a distribuição da opacidade fazendo uso de uma estrutura de dados chamada Árvore de Contornos. A opacidade deverá fluir da raiz até os ramos, atribuindo a banda opacidade relativa a cada um. Este modelo tem como objetivo reduzir a necessidade de interação excessiva com o usuário na geração da Função de transferência.

1.1.2 Objetivos específicos

Define-se como objetivos específicos de desenvolvimento:

- Extrair informações topológicas do conjunto de dados volumétricos e obter a Árvore de Contornos;
- Implementar o modelo de fluxo de fluidos na linguagem de programação C++;

- Analisar se os resultados obtidos são compatíveis com o modelo escolhido.

2 REVISÃO BIBLIOGRÁFICA

Em Visualização Científica, a Visualização Volumétrica está encarregada de representar visualmente o conjunto de dados (*datasets*), o qual no contexto de imageamento médico consiste de uma pilha de imagens alinhadas obtidas através de tomografia computadorizada. A Visualização Volumétrica busca representar o *dataset* em sua totalidade, ou seja, todas imagens ao mesmo tempo. Os voxels, *volume elements*, do *dataset* devem ser selecionados, pesados, combinados e projetados no plano da imagem. O plano da imagem age literalmente como uma janela de dados, representando a posição e a direção do observador que visualiza o *dataset* (PREIM; BARTZ, 2007).

No presente capítulo serão apresentados conceitos relacionados a *Volume Rendering* e alguns trabalhos relacionados a aplicação de árvores de contornos no pré-processamento do conjunto de dados volumétricos durante a etapa de renderização. A seguir, será detalhado o modelo de fluxo de fluido e sua aplicabilidade em problemas com árvores de Contornos, mostrando o algoritmo a ser utilizado e os passos de execução. Por fim, são apresentadas algumas características relevantes ao trabalho.

2.1 *Volume Rendering*

O conjunto de dados volumétricos é constituído de um grande número de *voxels* individuais. Duas abordagens para extração de características desse conjunto de dados são obtidos através de *Direct Volume Rendering*(DVR) e *Indirect Volume Rendering*. A renderização destes dados passa por uma série de etapas, conforme podem ser visualizadas na figura 2.1. Como o foco deste trabalho é DVR, este será detalhado na subseção seguinte.

Matematicamente, o conjunto de dados volumétricos armazenado é representado como uma grade uniforme de posições bidimensionais (x, y) a partir da origem e seguindo ao longo do eixo z , como exemplificado na figura 2.2. Assim definimos a posição de cada

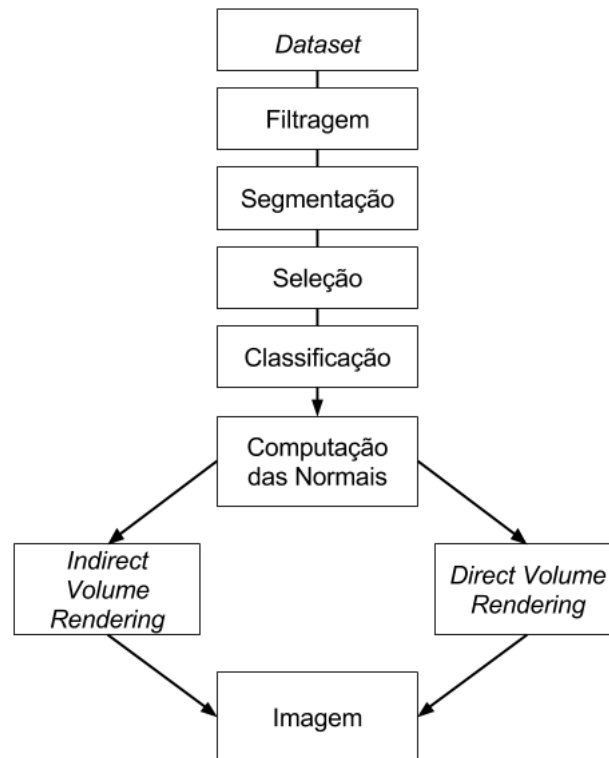


Figura 2.1: Exemplo do pipeline de *Volume Rendering*.

voxel que pode ser acessado pelo sistema de coordenadas $i - j - k$, representados pela relação única entre o vetor de indexação ξ , e a localização espacial P no R_3 . Podemos expressar $P = D\xi + m$ onde a matriz D expressa a distância entre as amostras e o vetor m define a origem espacial do volume (LJUNG, 2006).

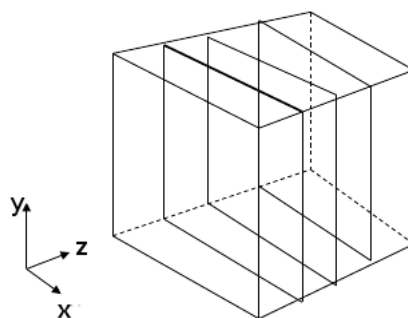


Figura 2.2: Conjunto de imagens paralelas ao longo de um eixo z.

Na prática, os *voxels* do volume podem ser armazenados através de estruturas de dados lineares como valores de inteiros ou ponto flutuantes, de acordo com a necessidade da aplicação e o tipo de dado de origem. Dados de imageamento médico são usualmente

armazenados no formato DICOM (*Digital Imaging and Communications in Medicine*), que provê uma sequência de imagens bidimensionais. Uma forma básica de acessar a posição de memória Q do *voxel* com vetor de indexação ξ em um volume de dimensões N_x, N_y, N_z , pode ser escrito como $Q = \xi_x + N_x(\xi_y + N_y\xi_z)$ (LJUNG, 2006).

2.1.1 *Direct Volume Rendering*

Direct Volume Rendering é um modelo de renderização para gerar imagens diretamente a partir do conjunto de dados volumétricos e na prática está estabelecido como uma técnica visualização de grande aceitação. Com a utilização de DVR diversas aplicações foram construídas para permitir ao usuário explorar e avaliar os dados interativamente. Um *pipeline* conceitual para DVR pode ser visualizado na figura 2.3 e um exemplo de imagem gerada por DVR na figura 2.4.

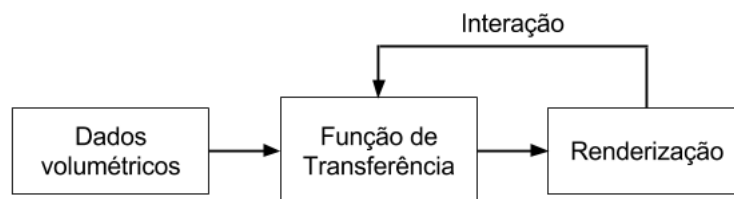


Figura 2.3: Exemplo do pipeline conceitual para *Direct Volume Rendering* descrito por (LJUNG, 2006)

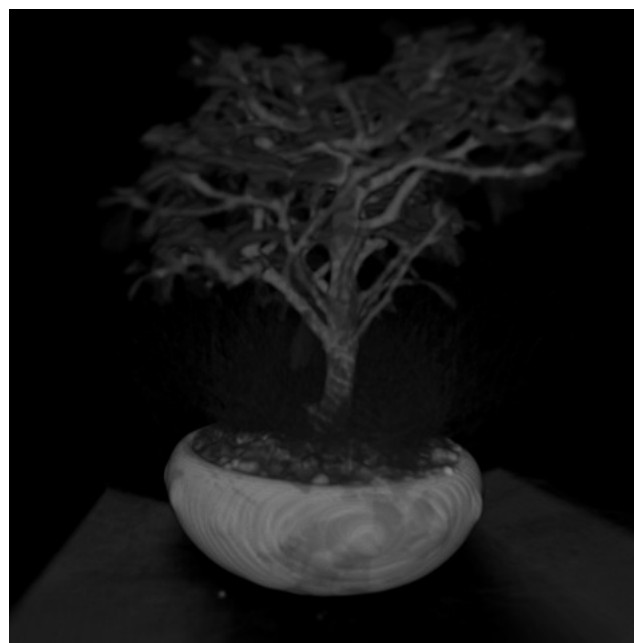


Figura 2.4: Imagem resultante de DVR aplicado ao *dataset* bonsai.

Para que ocorra a criação de imagens a partir de um modelo de renderização do volume de dados, é necessário que inicialmente defina-se um modelo matemático. Este modelo pode incluir diversas propriedades físicas para adequar-se as necessidades da renderização, melhorando assim a imagem resultante. O modelo ótico descrito na subseção 2.1.1.1 demonstra a utilização de técnicas de Emissão e Absorção de luz que foram inicialmente desenvolvidas por Blinn (1982) e depois aprimoradas por Kajiy, *et al.*(1984) . A seguir são apresentadas as propriedades físicas do modelo ótico e suas características.

2.1.1.1 Modelos óticos: absorção e emissão

A intensidade da luz em um ponto no espaço é determinado pela sua radiância, frequência e direção. Dispersão (*scattering*) descreve a direção e a frequência para a radiância. Absorção da luz é um processo onde a energia luminosa (radiância) é transformada em calor.

$$Absortion = pure absortion + scattering \quad (2.1)$$

Emissão da luz é onde a luz é introduzida na cena através de uma fonte externa.

$$Emission = source + scattering \quad (2.2)$$

As equações a seguir são descritas por Jonsson (2005) e são utilizadas para calcular as intensidades e propriedades óticas como escalares de imagens preto e branco. Para vários comprimentos de onda (i.e., vermelho, verde, e azul) em uma imagem colorida, as equações são repetidas para cada um dos canais, tornando estas quantidades vetoriais (MAX, 1995).

Para o caso de partículas de um volume com características de apenas absorção, sem emissão ou dispersão, temos a seguinte equação diferencial

$$\frac{dI}{ds} = -\tau(s)I(s) \quad (2.3)$$

onde s é a distância ao longo do raio e $I(s)$ é a intensidade da luz em s , $\tau(s)$ é a opacidade (ou coeficiente de extinção) em s . A solução desta equação é dada por

$$I(s) = I_0 e^{-\int_0^s \tau(s)dt} \quad (2.4)$$

onde I_0 é a intensidade em que o raio entra no volume. O mapeamento que designa o valor de τ para cada valor escalar f do volume de dados sendo visualizado é chamado de função de transferência (MAX, 1995).

Se considerarmos apenas o efeito de emissão, sem absorção ou dispersão, temos a seguinte equação diferencial

$$\frac{dI}{ds} = c(s)\tau(s) \quad (2.5)$$

onde $\tau(s)$ é uma porção infinitamente pequena de absorção e $c(s)$ é a intensidade em que a luz erradia. A solução para esta equação é dada por

$$I(s) = I_0 + \int_0^s c(t)\tau(t)dt \quad (2.6)$$

2.1.1.2 Função de transferência

O trabalho da função de transferência é mapear o valor dos dados de interesse com cor e opacidade, propriedades a serem utilizadas na renderização. O design da função de transferência é crucial em vários aspectos, dado que isto pode afetar a utilidade do programa de renderização. Usualmente a função de transferência é a única parte do programa em que o usuário pode modificar para ter controle sobre as características e informações que serão renderizadas (JONSSON, 2005). Porém, o tamanho e a complexidade dos dados volumétricos somados a um controle não intuitivo, que demanda tempo e experiência, não garante que o usuário vá extrair as informações desejadas.

Inicialmente as funções de transferência eram unidimensionais e aplicadas globalmente ao *dataset*, sendo necessário configurar um conjunto de parâmetros qualitativos e descritivos para visualizar as informações desejadas (FANG; BIDDLECOME; TUCERYAN, 1998). Novos métodos semiautomáticos para funções de transferência multidimensionais foram propostos fazendo uso de uma estrutura de dados conhecida como histograma de volume, mas ainda aplica a função de transferência globalmente (KINDLMANN; DURKIN, 1998).

A união de informações semânticas ao *dataset* fazendo uso de imagens como referência e uma lista das estruturas internas, permitiu uma abstração das configurações paramétricas das funções de transferência, tornando o conjunto de configurações mais intuitivo. Entretanto, esta abordagem ainda exige um alto grau de interatividade com o usuário para obter-se as informações desejadas (SALAMA; KELLER; KOHLMANN, 2006).

Informações topológicas foram introduzidas por Fujishiro et al. (1999) permitindo capturar características globais ao passo que possibilita a captura de características locais. Funções de transferência aplicadas a valores topológicos fixos, como profundidade, foram explorados por Takahashi et al. (2004) e Takeshima et al. (2005). Essas abordagens

não distinguem entre sub-regiões que possuem o mesmo valor escalar, limitando seu uso prático.

2.2 Árvore de Contornos

Com a introdução de atributos topológicos na especificação de funções de transferência foi possível obter uma nova abordagem na visualização do *dataset*. A abordagem de Weber et al. (2007) permite à árvore de contornos indexar sub-regiões (sub-volumes) e especificar funções de transferência para cada uma destas. Assim sendo, a estrutura de dados de árvore de contornos permite aos usuários mais experientes a extração de características do volume de dados ao passo que explora este. A subseção a seguir apresenta a definição formal de árvore de contorno.

2.2.1 Definição

Considerando um campo escalar contínuo F definido no domínio \mathbf{R}^d , $f : \mathbf{R}^d \rightarrow \mathbf{R}$. Assume-se \mathbf{R}^d como um complexo simplicial. Para um ponto dentro do simplexo, o valor de sua função é a interpolação linear dos valores dos vértices. A extensão funcional do campo F é o intervalo entre os valores mínimos e máximos da função f , $[f_{min}, f_{max}]$. Para um escalar $h \in [f_{min}, f_{max}]$, o conjunto de níveis do campo F no valor h é o subconjunto $L(h) = \{(x) | f(x) = h\}$. Enquanto h percorre monotonicamente por toda extensão de F de f_{min} a f_{max} , a topologia do conjunto de níveis muda apenas nos pontos críticos de F . A medida que h cresce no conjunto de níveis $L(h) = \{(x) | f(x) = h\}$, contornos aparecem no mínimo local de f , unindo ou separando nas selas, e desaparecendo no máximo local de f . Se cada contorno for representado como um nodo, a evolução do conjunto de níveis forma uma árvore chamada *árvore de contornos*. Esta árvore representa a relação aninhamento dos componentes conectados do contorno.

2.2.2 Branch decomposition

Branch decomposition foi inicialmente apresentada por (PASCUCCI; COLE-MCLAUGHLIN; SCORZELLI, 2004) e é uma alternativa para representação de árvores de contornos, onde ao invés de estruturar a árvore através de um grafo acíclico de nodos e arcos, utiliza-se o conceito de *branches* (ramos) que definem um caminho monótono, que sempre cresce ou sempre decresce, na árvore de contornos com valores escalares.

2.2.3 Simplificação

A simplificação da árvore de contornos se faz necessária devido a grande quantidade de ruídos topológicos que são gerados durante a sua criação, que além de interferir nos resultados, inviabilizam a aplicabilidade de algoritmos mais complexos. O artigo (ZHOU; XIAO; TAKATSUKA, 2013) propõe a definição de uma medida de importância para avaliar cada ramo da árvore de contornos. Esta medida faz uso dos atributos topológicos de cada ramo, a persistência p , o volume v e o hipervolume hv . Estes atributos são utilizados para montar um espaço de atributos de importância tridimensional, como ilustrado na figura 2.5.

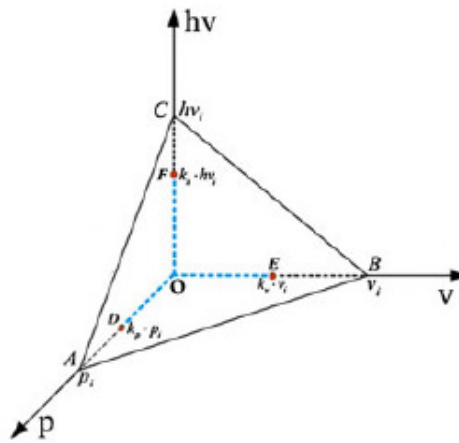


Figura 2.5: Espaço de atributos (ZHOU; XIAO; TAKATSUKA, 2013)

A medida de importância I_i é definida então como

$$I_i = g(p_i, v_i, hv_i) \quad (2.7)$$

onde

$$g(p_i, v_i, hv_i) = \frac{1}{2} \sqrt{(hv_i \cdot p_i)^2 + (v_i \cdot p_i)^2 + (hv_i \cdot v_i)^2} \quad (2.8)$$

é a área do triângulo definida pelos valores dos atributos.

2.3 Modelo de Fluxo de Fluido em Meio Poroso

O modelo de fluxo de fluido aplicado em árvores de contornos é explorado por (ZHOU; TAKATSUKA, 2009), onde é simulada a distribuição da opacidade como um sistema de escoamento de água. Quando a água passa por cada ramo da árvore de contornos,

parte dá água é redirecionada para cada ramo e uma quantidade residual é distribuída entre os ramos filhos no próximo nível de profundidade. A seguir é apresentada a Lei de Darcy, como uma teoria básica para distribuição dá água entre os ramos (AUMANN; DAVID FORD, 2002). Na subsecção 2.3.2 é apresentado o modelo de fluxo de fluído.

2.3.1 Lei de Darcy

Originalmente a Lei de Darcy foi proposta por Darcy (1856) utilizando o equipamento experimental ilustrado na figura 2.6, contendo um meio poroso inserido em um cilindro com área de secção transversal $A(m^2)$.

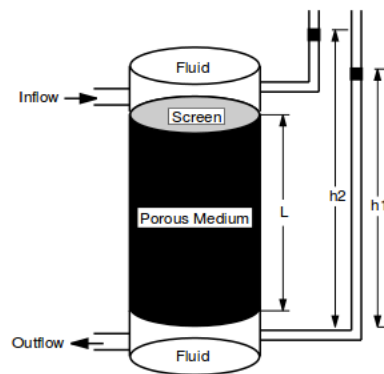


Figura 2.6: Ilustração do experimento de Darcy (AUMANN; DAVID FORD, 2002)

Os experimentos de Darcy mostraram que a taxa em que o fluido escoe, $Q(m^3s^{-1})$, por um meio poroso é proporcional a perda de altura entre as saídas do cilindro, $\Delta h = h_2 - h_1$, e a área de secção transversal do meio, $A(m^2)$, e inversamente proporcional ao comprimento do cilindro, $L(m)$:

$$Q = -KA \frac{\Delta h}{L}, \quad (2.9)$$

onde a constante de proporcionalidade $K(ms^{-1})$ é a condutividade hidráulica do meio (AUMANN; DAVID FORD, 2002).

2.3.2 Geração de Função de Transferência Baseada no Modelo de Fluxo de Fluído

O modelo de geração de função de transferência (opacidade) automática que faz uso da Lei de Darcy aplicada a árvore de contornos é descrita por ZHOU; TAKATSUKA (2009). As vantagens dessa abordagem refletem na necessidade mínima de interação com o usuário e na atribuição de diferentes funções de transferência a sub-regiões do *dataset* através da relação de inclusão na árvore de contornos.

Nesse modelo, a opacidade é distribuída entre os ramos como o fluxo da água em uma árvore. Assim, define-se uma banda de opacidade para cada ramo, α_{low} e α_{high} . Para exemplificar, imagina-se à estrutura como uma árvore de ponta-cabeça, onde a água entra pela raiz e assim flui para os ramos. Cada ramo é tratado como uma rede de canos porosos, a medida que a água flui pelos ramos, parte dela é retida no meio e parte é distribuída entre os filhos deste ramo, de forma que os níveis mais a baixo da árvore acumulem mais água e, portanto, mais opacidade. A figura 2.7 ilustra este comportamento.

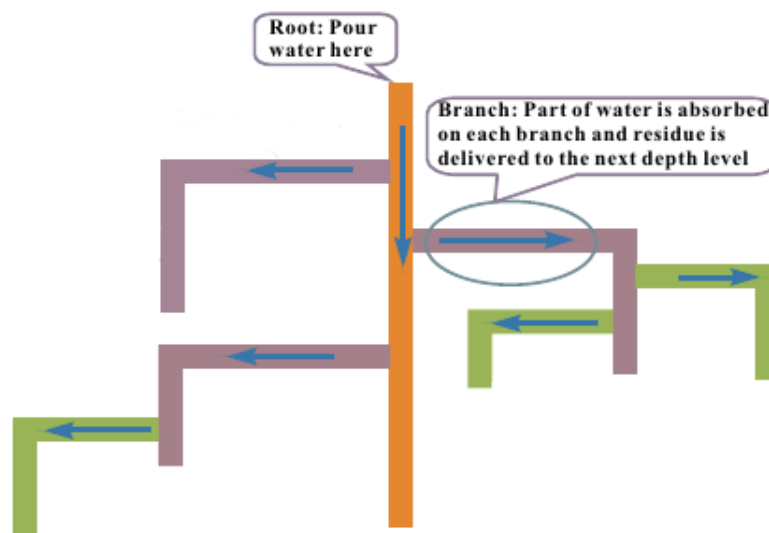


Figura 2.7: Ilustração da opacidade fluindo em uma árvore de contornos (ZHOU; TAKATSUKA, 2009)

Para implementação da Lei de Darcy, várias medidas foram adaptadas da equação e correlacionadas com as características dos ramos da árvore de contornos. O comprimento do cilindro L é modelado como a persistência p_i do ramo. A taxa em que o fluído esco Q é modelada como a taxa que a opacidade flui e é informada durante a execução da aplicação. A área de seção transversal do meio A é modelada como o número de ramos filhos do ramo atual. Δh é modelado como o fator residual de opacidade que é passado dos ramos pai ao filho. Este último pode ser obtido isolando-o na equação 2.10:

$$\Delta h_i = \frac{1}{k} \frac{Q \cdot p_i}{n_i^c} \quad (2.10)$$

O algoritmo a seguir é proposto por ZHOU; TAKATSUKA (2009) e ilustra os passos necessários para a obtenção da banda de opacidade de cada ramo.

Algoritmo 1: Fluxo residual na Árvore de Contornos

Data: Árvore de Contornos

Result: Banda de opacidade distribuída ao longo da Árvore de Contornos

Obtenha a banda de opacidade inicial α_d para cada nível de profundidade;

for cada nível de profundidade i **do**

 Obtenha o fator residual de opacidade Δh_i ;

 Obtenha a banda de opacidade básica α_i do nível de profundidade atual baseado no resíduo $\Delta\alpha_{i-1}$ do ramo pai;

for cada filho j **do**

 Distribua α_i baseado no valor escalar da sela s_{ij} e das medidas de importância;

 Obtenha a banda de opacidade α_{ij} para o ramo atual b_{ij} ;

end

 Obtenha o resíduo $\Delta\alpha_i$ do nível de profundidade atual;

end

O algoritmo inicia a partir do nível de profundidade $i = 0$, onde encontra-se a raiz, o ponto inicial de escoamento do fluido. Define-se então a banda de opacidade inicial α_d , dividindo igualmente a banda de opacidade total informada α_{max} entre os níveis de profundidade da árvore de contornos

$$\alpha_d = \frac{\alpha_{max}}{d_{max}} \quad (2.11)$$

onde d_{max} é o nível de profundidade máximo.

O escoamento do fluido é passado dos ramos pais para os ramos filhos. A quantidade de fluido absorvido pelo ramo é avaliada através do fator residual de opacidade Δh_i . α_i representa a opacidade básica aplicada ao ramo no nível de profundidade i e é dado pela equação 2.12

$$\alpha_i = (\alpha_d + \Delta\alpha_{i-1}) \cdot (1 - \Delta h_i) \quad (2.12)$$

O fluido residual da opacidade $\Delta\alpha_i$ que será entregue aos ramos filhos é calculada através da equação 2.13

$$\Delta\alpha_i = (\alpha_d + \Delta\alpha_{i-1}) \cdot (\Delta h_i) \quad (2.13)$$

A opacidade em cada nível de profundidade i é então distribuída entre os irmãos de acordo com a equação 2.14. Esta etapa do algoritmo tem por objetivo maximizar as diferenças entre cada ramo de acordo com as propriedades topológicas de cada um. Portanto, a opacidade final α_{ij} do ramo j no nível de profundidade i é definida por

$$\alpha_{ij} = \alpha_i \cdot g_{sb}(p_{ij}, v_{ij}, hv_{ij}) \cdot g_{sd}(s_{ij}) \quad (2.14)$$

onde $g_{sb}(p_{ij}, v_{ij}, hv_{ij})$ é a medida de importância do ramo. $g_{sd}(s_{ij})$ avalia o valor do *saddle* s_{ij} do ramo corrente, através da variação do valor do *saddle*, máximo e mínimo, nos ramos irmãos. Ambos são definidos pelas equações 2.15 e 2.16, respectivamente:

$$g_{sb}(p_{ij}, v_{ij}, hv_{ij}) = \frac{1}{2} \sqrt{(hv_i \cdot p_i)^2 + (v_i \cdot p_i)^2 + (hv_i \cdot v_i)^2} \quad (2.15)$$

$$g_{sd}(s_{ij}) = \text{frac}s_{ij} - s_i^{\text{min}} \Delta s_i \quad (2.16)$$

Por fim, a banda de opacidade inferior é definida por

$$\alpha_{low} = \alpha_{min} + \sum_{m=1}^{i-1} \alpha_m \quad (2.17)$$

e a banda de opacidade superior por

$$\alpha_{high} = \alpha_{min} + \sum_{m=1}^{i-1} \alpha_m + \alpha_{ij} \quad (2.18)$$

onde $\sum_{m=1}^{i-1} \alpha_m$ é a soma das opacidades finais α_{ij} do ramo raiz até o ramo pai.

Portanto, para aplicar este algoritmo, primeiramente é necessário extrair a árvore de contornos do *dataset* e, em seguida, obter a sua representação através de *branch decomposition*. Por fim, simplifica-se a árvore utilizando o algoritmo proposto por (ZHOU; XIAO; TAKATSUKA, 2013) descrito na seção 2.2.3 e somente então aplica-se o algoritmo 1.

A biblioteca *libtourtre* implementa algoritmos para criação de Árvores de Contornos baseados no artigo de (CARR; SNOEYINK; AXEN, 2000) assim como os algoritmos para *branch decomposition* baseados no artigo de (PASCUCCI; COLE-MCLAUGHLIN; SCORZELLI, 2004).

3 PROJETO E IMPLEMENTAÇÃO

Neste trabalho utilizou-se o algoritmo descrito por ZHOU; TAKATSUKA (2009) para geração das bandas de opacidade para função de transferência. Este algoritmo foi escolhido por apresentar resultados satisfatórios na visualização dos *datasets* volumétricos e por ser pioneiro na proposta de uma solução automática para geração de função de transferência locais, além de requerer interações mínimas do usuário. Entretanto, alguns passos do algoritmo não ficaram claros no artigo de ZHOU; TAKATSUKA (2009) e serão explorados ao longo da implementação, uma breve discussão sobre eles é abordada ao longo da seção 3.3 a seguir.

O algoritmo desenvolvido neste trabalho contém os passos para obtenção, decomposição (*branch decomposition*), simplificação e aplicação do modelo de fluxo de fluido na árvore de contornos. O diagrama de atividades na figura 3.1 ilustra as fases da solução proposta.

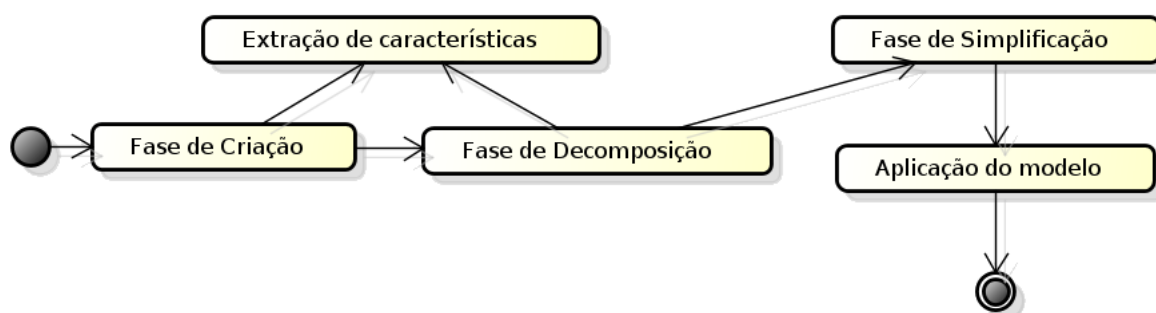


Figura 3.1: Diagrama de atividades para solução proposta

A simplificação da árvore de contornos foi baseada no artigo (ZHOU; XIAO; TAKATSUKA, 2013) e, como não faz parte do escopo deste trabalho, foi desenvolvida em colaboração com o aluno de mestrado Guilherme Gonçalves Schardong, no Laboratório de Computação Aplicada (LaCA), Universidade Federal de Santa Maria (UFSM).

3.1 Obtenção e Decomposição

A obtenção e decomposição da árvore de contornos se dá através das funções disponibilizadas pela biblioteca *libtourtre*, sendo necessário passar o caminho do *dataset* de interesse. Estes *datasets* são disponibilizados em formato binário, sendo necessário informar para aplicação desenvolvida o tipo de dado, usualmente *byte* ou *short*, e as dimensões do conjunto de dados.

3.2 Simplificação

A simplificação pode ser feita tanto com relação à árvore de contornos quanto a decomposição dela. Como a árvore de contornos é uma árvore sem raiz, a simplificação é realizada com base na decomposição.

Antes da simplificação propriamente dita, três características são medidas, o volume, o hipervolume e a persistência de cada ramo. O volume é a contagem de *voxels* pertencentes ao ramo. Como a biblioteca fornece um mapa vértice-ramo, basta iterar no mapa e contar o número de vértices pertencentes a cada ramo e armazenar os valores no próprio ramo.

O hipervolume é definido como a integral do campo escalar na região definida. E ele é calculado de forma semelhante ao volume, porém ao invés de uma contagem do número de vértices, acumulam-se as intensidades de cada um, e estas são armazenadas nos galhos correspondentes.

A persistência é definida como a diferença absoluta entre as intensidades de dois pontos críticos. Como os trabalhos relacionados não especificam quais pontos críticos são utilizados, no presente trabalho foram utilizados o ponto extremo e o *saddle* de cada ramo.

Essas medidas são calculadas para cada ramo da decomposição e com base em uma combinação delas, descrita por Zhou, et al. (2013), a importância de cada galho é calculada de acordo com a equação 2.7.

Após o cálculo da importância de cada ramo, basta iterar sobre a decomposição e eliminar os ramos que tiverem a importância menor que um limiar definido. O limiar escolhido é igual à média das importâncias de todos os galhos.

3.3 Modelo de Fluxo de Flúido

A implementação do modelo de fluxo de flúido é composta de uma busca em largura na árvore de contornos, percorrendo assim o nível de profundidade dos pais antes dos filhos. A cada nó da árvore de contornos é anexada uma estrutura de dados com o papel de armazenar as variáveis utilizadas durante o seu processamento. Um diagrama para esta estrutura de dados pode ser visualizado na figura 3.2 a seguir:

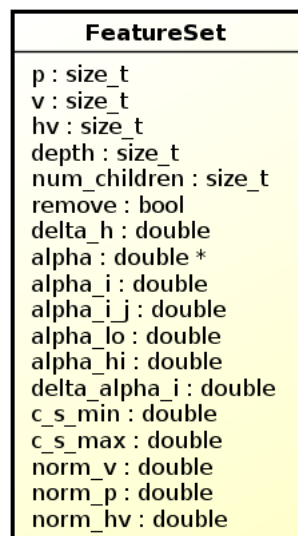


Figura 3.2: Diagrama da estrutura de dados *FeatureSet*

Os métodos utilizados pelo modelo de fluxo de flúido são ilustrados no diagrama do arquivo de cabeçalho *ctfunc.h* na figura 3.3 a baixo:

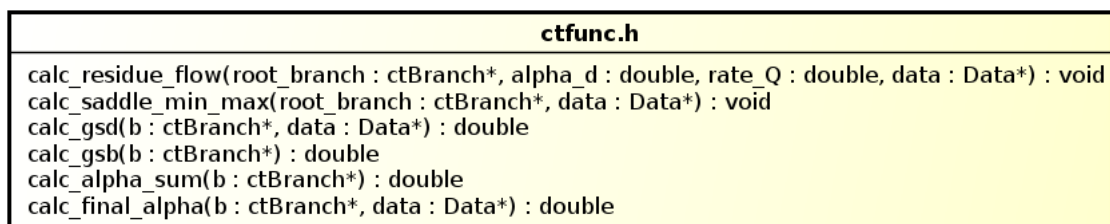


Figura 3.3: Diagrama do arquivo de cabeçalho *ctfunc.h*

O artigo proposto por ZHOU; TAKATSUKA (2009) não especifica como definir a opacidade α_i no nível de profundidade i para nós folhas, os quais não possuem nós filhos, que tendo $n_i^c = 0$ inviabiliza a aplicação da equação 2.10 e, conseqüentemente, da equação 2.12. Neste caso, este trabalho propõe definir α_i dos nós filhos apenas com base na opacidade inicial α_d e na opacidade residual do nó pai $\Delta\alpha_{i-1}$, assim

$$\alpha_i = \alpha_d + \Delta\alpha_{i-1} \quad (3.1)$$

e como este nó não possui folhas, naturalmente, não é necessário calcular a sua opacidade residual $\Delta\alpha_i$ que seria utilizado apenas por nós filhos. O algoritmo, descrito no artigo de ZHOU; TAKATSUKA (2009), também não especifica como definir a banda de opacidade final do ramo α_{ij} quando este não possui nós irmãos. Como esta etapa tem como objetivo redistribuir a opacidade α_i no nível de profundidade i entre os nós irmãos de acordo com a equação 2.14, assume-se então que

$$\alpha_{ij} = \alpha_i \quad (3.2)$$

por não necessitar redistribuição.

3.4 Implementação

Neste capítulo são discutidos os aspectos da implementação desenvolvida. O trecho de código 3.1 apresenta a implementação da função principal *int main(void)* em C++, na qual pode-se observar as funções de cada etapa do algoritmo que serão exploradas ao longo das seções seguintes.

```

1 ctContext* ctx = ct_init (...);
2 ...
3 ct_sweepAndMerge ( ctx );
4 ctBranch* root_branch = ct_decompose ( ctx );
5 ctBranch** branch_map = ct_branchMap ( ctx );
6 calc_branch_depth (...);
7 simplify_tree_dfs (...);
8 ...
9 calc_residue_flow (...);

```

Código 3.1: Função principal

3.4.1 Criação e Decomposição da Árvore de Contornos

A inicialização da aplicação ocorre nesta fase através da biblioteca *libtourtre*. Uma variável de contexto é alocada fazendo uso da função *ct_init()* e na sequência a função *ct_sweepAndMerge()* é chamada para criar a árvore de contornos, um ponteiro para a raiz da árvore de contornos é então armazenada na variável de contexto.

A função *ct_decompose()* aplica o método de *branch decomposition* na árvore de contornos, retornando a forma decomposta desta e armazenando seu endereço na variável *root_branch*.

A função `ct_branchMap()` tem como objetivo mapear os *voxels* correspondentes de cada *branch*, provendo acesso a esses através de ponteiros para ponteiros pela variável `branch_map`.

3.4.2 Simplificação da Árvore de Contornos

A simplificação da árvore de contornos é efetuada através da função `simplify_tree_dfs(...)` e é precedida pelos cálculos das medidas de importância de cada ramo. A classe `VolHypervolAcc` a seguir foi implementada para realização dessa etapa.

```

1  class VolHypervolAcc
2  {
3      ctBranch** b_map;
4      ctBranch* b;
5      Data* data;
6  public:
7      size_t v[2];
8
9      VolHypervolAcc(ctBranch* _b, ctBranch** _b_map, Data* _data)
10         : b_map(_b_map), b(_b), data(_data) {
11         v[0] = v[1] = 0;
12     }
13
14     VolHypervolAcc(VolHypervolAcc& rhs, tbb::split)
15         : b_map(rhs.b_map), b(rhs.b), data(rhs.data) {
16         v[0] = v[1] = 0;
17     }
18
19     void operator()(const tbb::blocked_range<size_t>& r) {
20         size_t begin = r.begin();
21         size_t end = r.end();
22         for(size_t i = begin; i < end; i++) {
23             if(b_map[i]->extremum == b->extremum
24                 && b_map[i]->saddle == b->saddle) {
25                 v[0]++;
26                 v[1] += data->data[i];
27             }
28         }
29     }
30
31     void join(const VolHypervolAcc& rhs) {
32         v[0] += rhs.v[0];
33         v[1] += rhs.v[1];
34     }
35 };

```

Código 3.2: Classe `VolHypervolAcc`

A classe `VolHypervolAcc` é usada como um *functor* pela função `tbb::parallel_reduce` para acumular os valores de volume e hipervolume de cada ramo. Onde a operação de *parallel reduce* é implementada pela biblioteca *Intel Threading Building Blocks*, a paralelização destes cálculos se fez necessária devido ao tempo de processamento que esta

função consumia. O cálculo das medidas de importância é realizado pelo seguinte conjunto de funções:

```

1 size_t* parallel_calc_vol_hypervol_branch(ctBranch* b,
2   ctBranch** b_map, Data* data)
3 {
4   VolHypervolAcc v(b, b_map, data);
5   size_t* fs = (size_t*) calloc(2, sizeof(size_t));
6   tbb::parallel_reduce(tbb::blocked_range<size_t>
7     (0, data->totalSize), v);
8   std::memcpy(fs, v.v, 2 * sizeof(size_t));
9   return fs;
10 }
11
12 size_t calc_persistence_branch(ctBranch* b, Data* data)
13 {
14   return std::abs(data->data[b->extremum] - data->data[b->saddle]);
15 }
16
17 void calc_branch_features(ctBranch* root, ctBranch** b_map, Data* data)
18 {
19   if(root == NULL) return;
20   if(root->data == NULL)
21     root->data = (FeatureSet*) calloc(1, sizeof(FeatureSet));
22
23   FeatureSet* bdata = (FeatureSet*) root->data;
24   size_t* f = parallel_calc_vol_hypervol_branch(root, b_map, data);
25   bdata->v = f[0];
26   bdata->hv = f[1];
27   bdata->p = calc_persistence_branch(root, data);
28   bdata->c_s_min = 10000.0; //since maximum value is 255
29   bdata->c_s_max = 0;
30   free(f); f = NULL;
31
32   for(ctBranch* c = root->children.head; c!= NULL; c = c->nextChild)
33     {
34       calc_branch_features(c, b_map, data);
35     }

```

Código 3.3: Funções para o cálculo das medidas de importância

3.4.3 Fluxo de fluido

A implementação do modelo de fluxo de fluido se dá através da função *calc_residue_flow()*, que é responsável por alocar as bandas de opacidade para cada ramo da árvore de contornos *root_branch*. O algoritmo desenvolvido foi levemente adaptado para realizar as proposições apontadas na seção 3.3 e realizar busca em largura para percorrer a árvore. Antes de processar cada nodo é realizado um teste condicional para verificar se este válido e não foi previamente removido pela etapa de simplificação. Durante o processamento da função *calc_residue_flow()*, destacam-se duas funções utilizadas:

- A função *calc_alpha_sum()* é responsável por realizar a soma da opacidade α_{ij} dos nós pais, soma que é utilizada para definir a banda de opacidade inferior α_{low} e de opacidade superior α_{high} de acordo com as equações 2.17 e 2.18;
- A função *calc_final_alpha()* utiliza a banda de opacidade para calcular e retornar um ponteiro para um vetor que representa a função de transferência do ramo. Esta função de transferência é responsável por atribuir a opacidade α para cada sub-região do ramo. Uma ilustração dos resultados obtidos através desta função são discutidos no capítulo 4.

```

1 void calc_residue_flow(ctBranch* root, double alpha_d,
2                       double rate_Q, Data* data)
3 {
4     if(root == NULL) return;
5     if(root->data == NULL)
6         root->data = (FeatureSet*)
7             calloc(1, sizeof(FeatureSet));
8
9     std::bqueue<ctBranch*> bqueue;
10    bqueue.push(root);
11
12    do {
13        ctBranch* cb = bqueue.front();
14        bqueue.pop();
15
16        if(cb->data == NULL) {
17            cb->data = (FeatureSet*)
18                calloc(1, sizeof(FeatureSet));
19        }
20
21        FeatureSet* bdata = (FeatureSet*) cb->data;
22        if(!bdata->remove) {
23            if (bdata->num_children != 0) {
24                bdata->delta_h =
25                    (1.0*rate_Q*(((double)bdata->p)
26                    /255.0))/(300.0*(((double)bdata->num_children));
27                if(bdata->depth == 0) { //root node
28                    bdata->alpha_i = alpha_d
29                        *(1-bdata->delta_h);
30                    bdata->delta_alpha_i = alpha_d
31                        - bdata->alpha_i;
32                } else {
33                    ctBranch* pb = cb->parent;
34                    FeatureSet* pdata = (FeatureSet*) pb->data;
35                    bdata->alpha_i = (alpha_d
36                        + pdata->delta_alpha_i)
37                        *(1.0-bdata->delta_h);
38                    bdata->delta_alpha_i = (alpha_d
39                        + pdata->delta_alpha_i)*bdata->delta_h;
40                }
41            } else { // leaf node
42                ctBranch* pb = cb->parent;
43                FeatureSet* pdata = (FeatureSet*) pb->data;

```

```

44         bdata->alpha_i = (alpha_d
45             + pdata->delta_alpha_i)*(1.0 - 0);
46         bdata->delta_alpha_i = 0.0;
47     }
48     bdata->alpha_i_j = bdata->alpha_i
49         *calc_gsb(cb)*calc_gsd(cb, data);
50     bdata->alpha_lo = calc_alpha_sum(cb);
51     bdata->alpha_hi = calc_alpha_sum(cb)
52         + bdata->alpha_i_j;
53     bdata->alpha = calc_final_alpha(cb, data);
54 }
55
56 for(ctBranch* c = cb->children.head; c != NULL;
57     c = c->nextChild) {
58     FeatureSet* cdata = (FeatureSet*) c->data;
59     if(!cdata->remove) {
60         bqueue.push(c);
61     }
62 }
63 } while(!bqueue.empty());
64 }

```

Código 3.4: Implementação do modelo de fluxo de fluídos

4 RESULTADOS

Os resultados obtidos se mostraram semelhantes com aqueles propostos no artigo de ZHOU; TAKATSUKA (2009). A análise dos resultados é limitada por não ser possível integrá-lo a uma ferramenta de *DVR* até a data de finalização deste trabalho. Testes iniciais foram realizados com o *dataset Nucleon* de dimensões 41x41x41 e logo após com o *dataset Hydrogen Atom* de dimensões 128x128x128. Os resultados são apresentados em tabelas para cada *dataset*. Cada linha representa um ramo e nas colunas temos o nível de profundidade, o valor da opacidade final e a banda opacidade, respectivamente.

Para o *dataset Nucleon*, a árvore de contornos sem simplificação possui 439 ramos e profundidade 4. Após a simplificação obtêm-se uma árvore de contornos com 6 ramos e profundidade 2. A tabela a seguir apresenta os dados obtidos para este *dataset* após a aplicação do algoritmo de fluxo de fluido desenvolvido.

i	α_{ij}	α_{low}	α_{high}
0	0.303	0.000	0.304
1	0.021	0.304	0.325
1	0.508	0.304	0.812
1	0.126	0.304	0.430
2	0.354	0.430	0.783
2	0.385	0.430	0.815

Tabela 4.1: Resultados obtidos para o *dataset Nucleon*

A figura 4.1 representa a árvore de contornos extraída. Em cada nó da árvore, um retângulo ilustra a variação de opacidade do ramo.

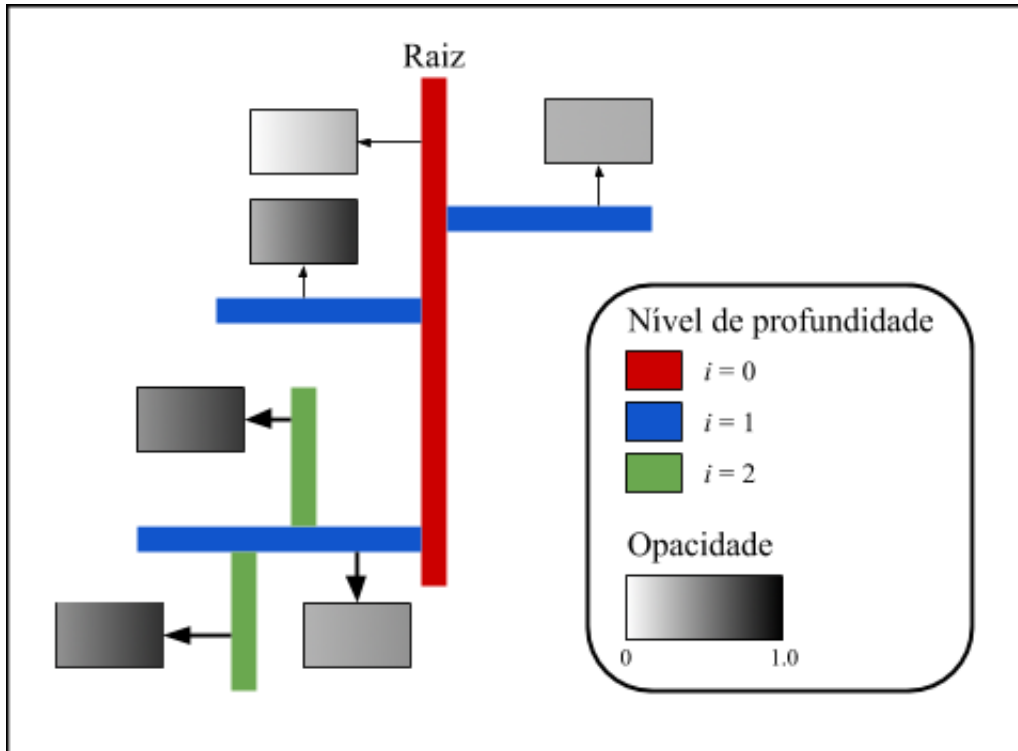


Figura 4.1: Ilustração da árvore de contornos extraída do *dataset Nucleon*

Para o *dataset Hydrogen Atom*, a árvore de contornos sem simplificação possui 6360 ramos e profundidade 9. Após a simplificação obtêm-se uma árvore de contornos com 14 ramos e profundidade 2. A tabela a seguir apresenta os dados obtidos para este *dataset* após a aplicação do algoritmo de fluxo de fluido desenvolvido.

i	α_{ij}	α_{low}	α_{high}
0	0.434	0.000	0.434
1	0.226	0.434	0.660
1	0.000	0.434	0.434
1	0.000	0.434	0.434
1	0.000	0.434	0.434
1	0.000	0.434	0.434
1	0.000	0.434	0.434
1	0.000	0.434	0.434
1	0.000	0.434	0.434
1	0.000	0.434	0.434
1	0.182	0.434	0.616
2	0.236	0.660	0.896
2	0.001	0.660	0.662
2	0.000	0.660	0.661
2	0.574	0.617	1.000

Tabela 4.2: Resultados obtidos para o *dataset Hydrogen Atom*

Para ambos *datasets* é possível verificar que a opacidade cresce de acordo com o nível

de profundidade, assim como ocorre a redistribuição da opacidade entre os nós irmãos de acordo com a medida de importância de cada ramo.

5 CONCLUSÃO

Este trabalho teve como objetivo o estudo e a implementação do Modelo de Fluxo de Fluído para distribuição de opacidade proposto por ZHOU; TAKATSUKA (2009). Também foram objetivos de estudo os conceitos que formam a base deste trabalho, como as etapas para o processamento de *Direct Volume Rendering*, a criação e manipulação da Árvore de Contornos e seus aspectos topológicos.

O objetivo de realizar a implementação do Modelo de Fluxo de Fluído foi alcançado, obtendo-se resultados semelhantes aos propostos no artigo de ZHOU; TAKATSUKA (2009) para diferentes *datasets*. As bandas de opacidades se mostraram crescentes de acordo com o nível de profundidade da árvore de contornos, de forma que percorrendo o volume de dados de sub-regiões externas para sub-regiões internas, verifica-se que estas se tornam mais opacas. A etapa de simplificação da árvore de contornos se mostrou imprescindível para o desenvolvimento deste trabalho, tornando possível a análise dos dados gerados e o encapsulamento das regiões topológicas de uma forma intuitiva.

Neste trabalho também foi possível discutir a etapa de Simplificação e o tratamento dos nós filhos, quando únicos, não especificadas no algoritmo proposto por ZHOU; TAKATSUKA (2009), facilitando futuras implementações.

5.1 Trabalhos futuros

A integração do método desenvolvido com uma ferramenta de DVR é o primeiro passo para dar continuidade a este trabalho, verificando assim sua eficiência em um aplicativo de tempo real. A integração com o método de *Color Harmonics* proposto por ZHOU; TAKATSUKA (2009) poderia prover resultados estéticos, facilitando a análise de *datasets* complexos, assim como, criar uma interface para interação como as zonas topológicas, permitindo escolher a visualização de campos escalares em ângulos específicos.

A implementação de uma biblioteca para criação e decomposição de uma árvore de contornos, bem documentada e planejada, pode ser utilizada para fortalecer o conhecimento sobre o campo de estudo.

Também pode-se considerar uma maneira de automatizar a escolha da velocidade em que o fluido escoe, fazendo uso de atributos topológicos, de forma que permita uma visualização perfeita do *dataset*.

REFERÊNCIAS

AUMANN, C. A.; DAVID FORD, E. Modeling tree water flow as an unsaturated flow through a porous medium. **Journal of theoretical biology**, [S.l.], v.219, n.4, p.415–429, 2002.

BLINN, J. F. Light reflection functions for simulation of clouds and dusty surfaces. **ACM SIGGRAPH Computer Graphics**, [S.l.], v.16, n.3, p.21–29, 1982.

CARR, H.; SNOEYINK, J.; AXEN, U. Computing contour trees in all dimensions. In: ACM-SIAM SYMPOSIUM ON DISCRETE ALGORITHMS, 2000. **Proceedings...** [S.l.: s.n.], 2000. p.918–926.

CORPORATION, I. **Intel Threading Buildin Blocks**.

FANG, S.; BIDDLECOME, T.; TUCERYAN, M. Image-based transfer function design for data exploration in volume visualization. In: VISUALIZATION'98, 1998. **Proceedings...** [S.l.: s.n.], 1998. p.319–326.

JONSSON, M. **Volume rendering**. 2005. Tese (Doutorado) — Master's Thesis in Computing Science.

KAJIYA, J. T.; VON HERZEN, B. P. Ray tracing volume densities. In: ACM SIGGRAPH COMPUTER GRAPHICS, 1984. **Anais...** [S.l.: s.n.], 1984. v.18, n.3, p.165–174.

KINDLMANN, G.; DURKIN, J. W. Semi-automatic generation of transfer functions for direct volume rendering. In: IEEE SYMPOSIUM ON VOLUME VISUALIZATION, 1998., 1998. **Proceedings...** [S.l.: s.n.], 1998. p.79–86.

LJUNG, P. **Efficient methods for direct volume rendering of large data sets**. 2006. Tese (Doutorado) — Linköping.

MAX, N. Optical models for direct volume rendering. **Visualization and Computer Graphics, IEEE Transactions on**, [S.l.], v.1, n.2, p.99–108, 1995.

PASCUCCI, V.; COLE-MCLAUGHLIN, K.; SCORZELLI, G. Multi-resolution computation and presentation of contour trees. In: IASTED CONFERENCE ON VISUALIZATION, IMAGING, AND IMAGE PROCESSING, 2004. **Proceedings...** [S.l.: s.n.], 2004. p.452–290.

PREIM, B.; BARTZ, D. **Visualization in medicine: theory, algorithms, and applications**. [S.l.]: Morgan Kaufmann, 2007.

SALAMA, C. R.; KELLER, M.; KOHLMANN, P. High-level user interfaces for transfer function design with semantics. **Visualization and Computer Graphics, IEEE Transactions on**, [S.l.], v.12, n.5, p.1021–1028, 2006.

ZHOU, J.; TAKATSUKA, M. Automatic transfer function generation using contour tree controlled residue flow model and color harmonics. **Visualization and Computer Graphics, IEEE Transactions on**, [S.l.], v.15, n.6, p.1481–1488, 2009.

ZHOU, J.; XIAO, C.; TAKATSUKA, M. A multi-dimensional importance metric for contour tree simplification. **Journal of Visualization**, [S.l.], v.16, n.4, p.341–349, 2013.