

UNIVERSIDADE FEDERAL DE JUIZ DE FORA  
PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM COMPUTACIONAL  
BACHARELADO EM MESTRADO EM MODELAGEM COMPUTACIONAL

# **Uma Ferramenta Computacional para Simulação de Escoamento Pulsátil em Modelos de Árvores Arteriais 1D**

**Igor Pires dos Santos**

JUIZ DE FORA  
SETEMBRO, 2021

# **Uma Ferramenta Computacional para Simulação de Escoamento Pulsátil em Modelos de Árvores Arteriais 1D**

IGOR PIRES DOS SANTOS

Universidade Federal de Juiz de Fora  
Programa de Pós-Graduação em Modelagem Computacional  
Departamento de Ciência da Computação  
Bacharelado em Mestrado em Modelagem Computacional

Orientador: Rafael Alves Bonfim de Queiroz

Coorientador: Ruy Freitas Reis

JUIZ DE FORA  
SETEMBRO, 2021

# UMA FERRAMENTA COMPUTACIONAL PARA SIMULAÇÃO DE ESCOAMENTO PULSÁTIL EM MODELOS DE ÁRVORES ARTERIAIS 1D

Igor Pires dos Santos

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO  
EM MODELAGEM COMPUTACIONAL DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA,  
COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU  
DE BACHAREL EM MESTRADO EM MODELAGEM COMPUTACIONAL.

Aprovada por:

Rafael Alves Bonfim de Queiroz  
Prof. Dr

Ruy Freitas Reis  
Prof. Dr

Nome Examinador 1  
Titulação Examinador 1

Nome Examinador 2  
Titulação Examinador 2

JUIZ DE FORA  
10 DE SETEMBRO, 2021

## Resumo

Neste trabalho, apresentam-se: (i) modelo matemático da literatura que descreve o escoamento sanguíneo pulsátil em modelos de árvores arteriais 1D, (ii) uma ferramenta computacional desenvolvida que calcula a pressão e fluxo em cada vaso a partir do modelo matemático . Os resultados obtidos neste trabalho estão condizentes com dados numéricos relatados na literatura.

**Palavras-chave:** Árvores arteriais. Escoamento pulsátil. Hemodinâmica Computacional.

## **Abstract**

In this work, the following are presented: (i) an analytical scheme based on physics and mathematical laws to calculate the local characteristics of the pressure and flux wave in 1D arterial tree's models, (ii) a computational environment developed to simulate and visualize the results of the model's construction and hemodynamic studies. The results produced in this work are consistent to real morphometric data and numeric data related in the literature.

**Keywords:** Arterial trees. Pulsatile flow. Computational hemodynamics.

# Agradecimentos

Agradecimentos

# Conteúdo

<b>Lista de Figuras</b>	<b>6</b>
<b>Lista de Tabelas</b>	<b>8</b>
<b>Lista de Abreviações</b>	<b>10</b>
<b>1 INTRODUÇÃO</b>	<b>12</b>
1.1 OBJETIVOS . . . . .	13
1.2 ORGANIZAÇÃO . . . . .	14
<b>2 MODELAGEM DO ESCOAMENTO SANGUÍNEO</b>	<b>15</b>
2.1 MODELO MATEMÁTICO . . . . .	15
2.1.1 CÁLCULO DA PRESSÃO E DO FLUXO SANGUÍNEO . . . . .	17
2.1.2 CÁLCULO DO COEFICIENTES DE REFLEXÃO E ADMITÂNCIA . . . . .	19
2.1.3 CÁLCULO DA IMPEDÂNCIA DE ENTRADA . . . . .	20
2.1.4 INCORPORAÇÃO DA VISCOSIDADE E VISCOELASTICIDADE NO MO- DELO . . . . .	21
2.2 MODELAGEM COMPUTACIONAL . . . . .	22
<b>3 FERRAMENTA COMPUTACIONAL</b>	<b>28</b>
3.1 ESTRUTURA DE DADOS . . . . .	29
3.1.1 ELEMENTO INTELIGENTE . . . . .	30
3.1.2 FÁBRICA . . . . .	36
3.1.3 OBJETO INTELIGENTE . . . . .	38
3.1.4 OBJETO GRÁFICO . . . . .	41
3.1.5 PROJETO INTELIGENTE . . . . .	44
3.1.6 FÁBRICA DE PROJETO . . . . .	45
3.1.7 THREADS INTELIGENTES . . . . .	47
3.2 INTERFACE DE USUÁRIO . . . . .	50
3.3 CONSOLE . . . . .	51
3.3.1 AJUDA . . . . .	52
3.3.2 LER ARQUIVO DE ENTRADA . . . . .	54
3.3.3 BATERIA DE TESTES . . . . .	54
3.3.4 LISTA DE TESTES . . . . .	55
3.3.5 CASO DE TESTE . . . . .	55
3.3.6 TESTAR EQUIDADE DE ARQUIVOS . . . . .	56
3.3.7 CRIAR PROJETO . . . . .	56
3.3.8 USAR PROJETO . . . . .	56
3.3.9 LISTAR PROJETOS . . . . .	57
3.3.10 IMPRIMIR PROJETO . . . . .	57
3.3.11 EXCLUIR PROJETO . . . . .	57
3.3.12 SALVAR PROJETO . . . . .	58
3.3.13 CARREGAR PROJETO . . . . .	58
3.3.14 CRIAR ELEMENTO . . . . .	58
3.3.15 CLONAR ELEMENTO . . . . .	59

3.3.16	LISTAR ELEMENTOS . . . . .	59
3.3.17	IMPRIMIR ELEMENTO . . . . .	60
3.3.18	EXCLUIR ELEMENTO . . . . .	60
3.3.19	SALVAR ELEMENTO . . . . .	60
3.3.20	CARREGAR ELEMENTO . . . . .	60
3.3.21	LISTAR EXPORTAÇÕES DO ELEMENTO . . . . .	61
3.3.22	EXPORTAR ELEMENTO . . . . .	61
3.3.23	ESCALAR ELEMENTO . . . . .	62
3.3.24	DEFINIR PARÂMETRO DE ELEMENTO . . . . .	62
3.3.25	DEFINIR TODOS OS PARÂMETROS DE ELEMENTO . . . . .	63
3.3.26	LISTAR FÁBRICAS DE ELEMENTO . . . . .	64
3.3.27	LISTAR EXEMPLOS DISPONÍVEIS DE ELEMENTO . . . . .	64
3.3.28	CRIAR OBJETO . . . . .	64
3.3.29	CLONAR OBJETO . . . . .	65
3.3.30	LISTAR OBJETOS . . . . .	65
3.3.31	IMPRIMIR OBJETO . . . . .	66
3.3.32	EXCLUIR OBJETO . . . . .	66
3.3.33	SALVAR OBJETO . . . . .	66
3.3.34	CARREGAR OBJETO . . . . .	67
3.3.35	EXPORTAR OBJETO . . . . .	67
3.3.36	DEFINIR PARÂMETROS DE OBJETO . . . . .	68
3.3.37	SETAR OBJETO . . . . .	68
3.3.38	ITERAR OBJETO . . . . .	69
3.3.39	LISTAR FÁBRICAS DE ITERAÇÃO DE OBJETO . . . . .	69
3.3.40	DEFINIR FÁBRICA DE ITERAÇÃO DE OBJETO . . . . .	69
3.3.41	LISTAR FÁBRICAS GRÁFICAS DE OBJETO . . . . .	70
3.3.42	DEFINIR FÁBRICA GRÁFICA DE OBJETO . . . . .	70
3.3.43	LISTAR OBJETOS GRÁFICOS . . . . .	70
3.3.44	LISTAR CANVAS . . . . .	71
3.3.45	LINK GRÁFICO . . . . .	71
3.3.46	TERMINAR LINK GRÁFICO . . . . .	72
3.4	JANELA . . . . .	72
<b>4</b>	<b>RESULTADOS NUMÉRICOS E DISCUSSÕES</b>	<b>73</b>
<b>5</b>	<b>CONCLUSÕES E TRABALHOS FUTUROS</b>	<b>82</b>
<b>A</b>	<b>PROCESSO DE COMPILAÇÃO</b>	<b>84</b>
<b>B</b>	<b>FORMATO DE ARQUIVO DE COMANDOS</b>	<b>85</b>
<b>C</b>	<b>FORMATO DE ARQUIVO DE ELEMENTO INTELIGENTE</b>	<b>87</b>
<b>D</b>	<b>FORMATO DE ARQUIVO DE OBJETO INTELIGENTE</b>	<b>88</b>



## Lista de Figuras

2.1	Notação usada para identificar cada segmento de vaso ( $k, j$ ) (figura adaptada de [6]). . . . .	17
3.1	Interface gráfica da ferramenta desenvolvida. . . . .	28
3.2	Representação de classes de um elemento inteligente. <i>WiseElement</i> a classe abstrata base e seus componentes: <i>WiseStructure</i> representa a estrutura contida em um arquivo VTK e <i>DataStructure</i> representa a estrutura de ponteiros e variáveis utilizadas na iteração. Tingido de azul as estruturas que nem sempre estão presentes. . . . .	31
3.3	Tipos de elementos inteligentes. <i>WiseGraphic</i> , um gráfico bidimensional. <i>WiseMesh</i> , uma malha bidimensional. <i>WisePoly</i> , uma malha tridimensional. <i>WiseArteryTree</i> , uma árvore arterial . . . . .	32
3.4	Pontos utilizados na especificação do modelo geométrico. Linhas utilizadas na especificação do modelo geométrico, através dos pontos previamente definidos. Células utilizadas na especificação do modelo geométrico, através dos pontos previamente definidos. . . . .	33
3.5	Máquina de Status que controla o funcionamento de um elemento inteligente. . . . .	34
3.6	Elemento inteligente enquanto no estado <i>Warming</i> . . . . .	34
3.7	Elemento inteligente enquanto no estado <i>Cold</i> . . . . .	35
3.8	Elemento inteligente enquanto no estado <i>Hot</i> . . . . .	36
3.9	Arquitetura de classes fábrica e fluxo de trabalho do elemento inteligente <i>WiseElement</i> . A fábrica <i>WiseElementFactory</i> é responsável por criar o elementos inteligentes, a fábrica <i>WiseIterationFactory</i> é responsável pela iteração do elemento inteligente e a fábrica <i>GraphicFactory</i> é responsável por criar as estruturas de visualização. . . . .	37
3.10	Objeto inteligente <i>WiseObject</i> e todos seu componentes: <i>WiseObjectFactory</i> , fábrica responsável pela criação de objetos inteligentes de um determinado tipo; <i>WiseIterationFactory</i> , fábrica de iteração; <i>WiseGraphicFactory</i> , fábrica gráfica; <i>WiseCollection</i> , coleção de elementos inteligentes; <i>GraphicModel</i> , coleção de objetos gráficos. . . . .	39
3.11	Modelo gráfico <i>GraphicModel</i> , contém uma coleção de objetos gráficos. . . . .	40
3.12	Modelo gráfico <i>GraphicModel</i> , contém uma coleção de objetos gráficos. . . . .	42
3.13	Tipos de objetos gráficos <i>GraphicObjects</i> . . . . .	43
3.14	Tipos de elementos gráficos <i>GraphicElements</i> . <i>Point</i> , um ponto. <i>Line</i> , uma linha. <i>Quad</i> , um quadrado. <i>Cube</i> , um cubo. <i>Cylinder</i> , um cilindro. <i>Sphere</i> , uma esfera. . . . .	43
3.15	Projeto inteligente <i>WiseProject</i> e seus componentes, uma lista de objetos inteligentes <i>WiseObject</i> e uma lista de elementos inteligentes <i>WiseElement</i> . . . . .	44
3.16	Fábrica de projetos inteligentes <i>WiseProjectFactory</i> e seus componentes, fábricas de elementos inteligentes <i>WiseElementFactories</i> , fábricas de objetos inteligentes <i>WiseObjectFactories</i> , fábrica de objetos gráficos <i>GraphicObjectFactories</i> , fábricas de elementos gráficos <i>GraphicElementFactories</i> e fábricas de iteração <i>WiseIterationFactories</i> . . . . .	45
3.17	Todas as fábricas que compõem uma fábrica de projeto. . . . .	46

3.18	Modelo de Threads. <i>WiseThreadPool</i> , responsável por orquestrar o funcionamento das demais threads, bem como os objetos contidos em um projeto <i>WiseProject</i> . <i>WiseIO</i> , thread responsável por processos de leitura e escrita. <i>WiseConsole</i> , thread responsável por interpretar os comandos de texto e os traduzir em métodos. <i>WiseProcessor</i> , thread responsável por realizar o método iterativo de um objeto inteligente <i>WiseObject</i> . . . . .	48
3.19	Modelo de Threads ao receber uma linha de comando da interface de usuário.	49
3.20	Modelo de Threads ao receber um comando de escrita/leitura. . . . .	50
3.21	Interface de usuário gráfica. . . . .	51
3.22	Estrutura do projeto que compõe o ambiente computacional InGU. . . . .	51
3.23	Captura de tela com a execução do ambiente computacional InGU em um console. . . . .	52
3.24	Captura de tela com a execução do ambiente computacional InGU em um console. . . . .	53
4.1	Representação do modelo de árvore arterial canina (figura adaptada de [6]). . .	73
4.2	Amplitude da pressão $ P $ ao longo da árvore arterial considerando diferentes viscosidade do fluido $\mu$ e frequências: (a) $f = 3,65$ Hz, (b) $f = 7,30$ Hz. . . . .	75
4.3	Amplitude da pressão $ P $ ao longo da árvore arterial considerando diferentes viscosidade do fluido $\mu$ e frequências: (a) $f = 10,95$ Hz, (b) $f = 14,60$ Hz. . . . .	76
4.4	Amplitude da pressão $ P $ ao longo da árvore arterial considerando diferentes valores de viscoelasticidade $\phi_0$ e frequências: (a) $f = 3,65$ Hz, (b) $f = 7,30$ Hz. .	78
4.5	Amplitude da pressão $ P $ ao longo da árvore arterial considerando diferentes valores de viscoelasticidade $\phi_0$ e frequências: (a) $f = 10,95$ Hz, (b) $f = 14,60$ Hz.	79
4.6	Amplitude da impedância de entrada $ Z $ em função da frequência $f$ : (a) efeito da viscosidade (b) impacto da viscoelasticidade. . . . .	80
4.7	Amplitude da pressão $ P $ ao longo da árvore arterial $X$ considerando viscosidade $\mu = 1,0\mu_0$ , viscoelasticidade $\phi_0 = 8^\circ$ . . . . .	81
4.8	Impedância de entrada $ Z $ em função da frequência $f$ considerando viscosidade $\mu = 1,0\mu_0$ , viscoelasticidade $\phi_0 = 8^\circ$ . . . . .	81

## Lista de Tabelas

2.1	Propriedades de cada vaso arterial do modelo. . . . .	23
3.1	Descrição do comando ajuda. . . . .	53
3.2	Descrição do comando ler arquivo de comando. . . . .	54
3.3	Descrição do comando bateria de testes. . . . .	55
3.4	Descrição do comando listar testes. . . . .	55
3.5	Descrição do comando executar caso de teste. . . . .	55
3.6	Descrição do comando para testar a equidade de dois arquivos. . . . .	56
3.7	Descrição do comando para criar projetos. . . . .	56
3.8	Descrição do comando para selecionar projetos. . . . .	57
3.9	Descrição do comando listar projetos. . . . .	57
3.10	Descrição do comando imprimir projetos. . . . .	57
3.11	Descrição do comando excluir projetos. . . . .	57
3.12	Descrição do comando para salvar projetos. . . . .	58
3.13	Descrição do comando para carregar projetos. . . . .	58
3.14	Descrição do comando para criar. . . . .	59
3.15	Descrição do comando para clonar elementos inteligentes. . . . .	59
3.16	Descrição do comando listar elementos. . . . .	59
3.17	Descrição do comando imprimir elementos inteligentes. . . . .	60
3.18	Descrição do comando excluir projetos. . . . .	60
3.19	Descrição do comando excluir projetos. . . . .	61
3.20	Descrição do comando para carregar elementos inteligentes. . . . .	61
3.21	Descrição do comando listar exportações de um elemento inteligente. . . . .	61
3.22	Descrição do comando para exportar elementos. . . . .	62
3.23	Descrição do comando para escalar elementos. . . . .	62
3.24	Descrição do comando para definir parâmetros de elementos. . . . .	63
3.25	Descrição do comando para definir parâmetros de elementos. . . . .	63
3.26	Descrição do comando listar fábricas de elemento. . . . .	64
3.27	Descrição do comando listar exemplos contidos em determinada fábrica de elemento. . . . .	64
3.28	Descrição do comando para criar objetos inteligentes. . . . .	65
3.29	Descrição do comando para clonar objetos inteligentes. . . . .	65
3.30	Descrição do comando listar elementos. . . . .	66
3.31	Descrição do comando imprimir objetos inteligentes. . . . .	66
3.32	Descrição do comando excluir objetos inteligentes. . . . .	66
3.33	Descrição do comando salvar objetos inteligentes. . . . .	67
3.34	Descrição do comando carregar objetos inteligentes. . . . .	67
3.35	Descrição dos comandos de exportação elementos contidos no <i>Forno</i> de objetos inteligentes. . . . .	67
3.36	Descrição dos comandos de exportação elementos contidos no <i>Forno</i> de objetos inteligentes. . . . .	68
3.37	Descrição do comando setar objetos inteligentes. . . . .	68
3.38	Descrição do comando iterar objetos inteligentes. . . . .	69
3.39	Descrição do comando listar fábricas de iteração. . . . .	69

3.40	Descrição do comando definir fábricas gráficas. . . . .	70
3.41	Descrição do comando listar fábricas gráficas. . . . .	70
3.42	Descrição do comando definir fábricas gráficas. . . . .	70
3.43	Descrição do comando listar objetos gráficos. . . . .	71
3.44	Descrição do comando listar elementos gráficos <i>Canvas</i> . . . . .	71
3.45	Descrição dos comandos que enviam um objeto gráfico para ser exibido em um elemento gráfico da interface de usuário. . . . .	72
3.46	Descrição do comando terminar link gráfico. . . . .	72
4.1	Propriedades dos segmentos do modelo de árvore arterial [6, 5] . . . . .	74

## **Lista de Abreviações**

PPGMC	Programa de Pós-Graduação em Modelagem Computacional
UFJF	Universidade Federal de Juiz de Fora

## Conteúdo

\*

# 1 INTRODUÇÃO

Estudos de simulação hemodinâmica têm sido frequentemente baseados em modelos de árvores arteriais para obter uma melhor compreensão de todos os aspectos relacionados ao escoamento sanguíneo, desde a propagação de ondas e análise do pulso de pressão, passando pelo diagnóstico e inclusive com aplicações no planejamento cirúrgico. Como a representação do sistema cardiovascular através de um modelo puramente 3D que leve em conta a estrutura geométrica exata de todos os vasos não é, no momento, viável computacionalmente, vêm sendo empregados modelos dimensionalmente heterogêneos conhecidos como 0D (zero-dimensional)–1D (unidimensional)–3D (tridimensional) [8].

Modelos 3D [18, 22] são utilizados para estudar em detalhe a hemodinâmica local de distritos arteriais de interesse, e a geometria destes modelos são provenientes de dados anatômicos obtidos normalmente via reconstrução de imagens médicas de pacientes específicos. Modelos 1D [3, 9, 21] são adotados para representar as artérias de maior calibre e a estrutura geométrica destes modelos pode ser construída a partir de dados anatômicos. Tais modelos são capazes de capturar os efeitos de propagação de ondas [2, 6], a interação das reflexões destas ondas e dar como resultado um pulso de pressão e vazão com significado fisiológico tanto em artérias centrais como periféricas. No entanto, um modelo 1D de toda a árvore arterial sistêmica não é possível devido à falta de dados anatômicos precisos das regiões periféricas. Portanto, a árvore tem que ser truncada em algum nível. Normalmente, este truncamento é feito empregando modelos 0D [15, 21] conhecidos por terminais Windkessel à jusante da posição distal do modelo 1D para representar o comportamento de distritos arteriais relacionados com o nível de arteríolas e capilares.

IGOR: os dois parágrafos acima estão legais no sentido da ideia para se colocar em uma introdução, mais necessitam ser um pouco reescritos e principalmente, citar referências mais recentes de estudos hemodinâmicas envolvendo modelos 3D, 1D, 0D, e modelos acoplados 0D-1D, 3D-1D-0D. Vale a pena buscar estas referências e ler a introdução deste artigos

A incidência maior de picos na onda de pressão ao percorrer a aorta já foi documen-

tada como evidência para os efeitos da reflexão em árvores vasculares [13, 14, 16]. Enquanto as áreas de reflexão não podem ser completamente conhecidas ou localizadas, é geralmente aceito que a forma da onda de pressão é modificada significativamente enquanto progride pela aorta, de uma forma que só pode ser explicada por reflexões de onda. Um entendimento mais claro da relação entre modificações e fatores de modificação motiva a busca e desenvolvimento de modelos matemáticos que determinam a forma da onda que o pulso de pressão toma em cada ponto ao percorrer uma árvore arterial.

Dentro deste contexto, adotou-se neste trabalho o modelo matemático de Duan e Zamir [6] que descreve escoamento sanguíneo pulsátil em árvores arteriais. Estes autores propuseram um modelo relativamente simples para representação da pressão sanguínea e do fluxo em um modelo de árvore arterial. Dentro de cada segmento de vaso, o escoamento sanguíneo foi calculado baseado em uma aproximação de Womersley, incluindo a elasticidade da parede, bem como a densidade do sangue e a viscosidade.

A capacidade de capturar o pico de pressão existente no escoamento sanguíneo justificativa a escolha do modelo matemático de Duan e Zamir para implementação e simulação computacional. Este modelo possibilita o cálculo correto das características locais das ondas de pressão e fluxo a medida que elas progridem ao longo de um modelo de árvore 1D e se tornam modificadas por reflexões de onda.

IGOR: escreva um parágrafo de trabalhos da literatura que citam e utilizam o modelo de Duan e Zamir. Busque referências na literatura. Lembrando que esta referência [6] está errada. A referência correta é de 1995. Principalmente, cite o seu trabalho publicado na revista Mundi deste parágrafo.

## 1.1 OBJETIVOS

Os objetivos que norteiam este trabalho são:

- desenvolver uma ferramenta computacional capaz de simular o modelo matemático de Duan e Zamir;
- aplicar a ferramenta desenvolvida considerando diferentes cenários hemodinâmicos para investigar os efeitos da viscosidade sanguínea e da viscoelasticidade da parede do



---

vaso no escoamento sanguíneo

## 1.2 ORGANIZAÇÃO

Os demais capítulos deste trabalho estão organizados como segue:

- Capítulo 2 -
- Capítulo 3 – Neste capítulo, apresenta-se a ferramenta computacional desenvolvida em C++, a qual contou com a utilização das bibliotecas Qt/OpenGL para ajudar na elaboração da interface gráfica
- Capítulo 4 —
- Capítulo 5

IGOR: complemente esta seção de organização. Recomendo fortemente ler a introdução da minha dissertação e tese para ter mais ideias de como montar uma introdução para sua dissertação.

## 2 MODELAGEM DO ESCOAMENTO SANGUÍNEO

IGOR: está errada a citação do trabalho do método de Duan e Zamir (1995), não é a citação que colocou [6].

Neste capítulo, apresenta-se em detalhe o modelo matemático de Duan e Zamir [7], para o escoamento sanguíneo pulsátil em árvores arteriais. Por fim, apresenta um algoritmo que sistematiza os passos dos cálculos realizados para obtenção da pressão e fluxo ao longo da árvore arterial.

### 2.1 MODELO MATEMÁTICO

A propagação de ondas em um tubo é governada pela equações da onda para a pressão  $p(x, t)$  e fluxo  $q(x, t)$  como seguem:

$$\frac{\partial q}{\partial t} = -cY \frac{\partial p}{\partial x}, \quad (2.1)$$

$$\frac{\partial p}{\partial t} = -\frac{c}{Y} \frac{\partial q}{\partial x}, \quad (2.2)$$

nos quais  $t$  é o tempo,  $x$  é a coordenada axial ao longo do tubo,  $c$  é a velocidade de onda,  $Y = \frac{A}{\rho c}$  é a admitância e  $A$  é a área da seção transversal do tubo, e  $\rho$  é a densidade do fluido. Estas equações são baseadas na linearização das equações de movimento do fluido [5, 14].

Para uma onda harmônica simples, as equações (2.1) e (2.2) resultam em:

$$p = \bar{p}_0 \exp \left[ i\omega \left( t - \frac{x}{c} \right) \right] + R \bar{p}_0 \exp \left[ i\omega \left( t - \frac{2L}{c} + \frac{x}{c} \right) \right], \quad (2.3)$$

$$q = Y \left\{ \bar{p}_0 \exp \left[ i\omega \left( t - \frac{x}{c} \right) \right] - R \bar{p}_0 \exp \left[ i\omega \left( t - \frac{2L}{c} + \frac{x}{c} \right) \right] \right\}, \quad (2.4)$$

onde  $\omega = 2\pi f$  é a frequência angular,  $f$  é frequência em Hertz,  $L$  é o comprimento do tubo,  $\bar{p}_0$  é a amplitude da onda incidente,  $R$  é o coeficiente de reflexão definido pela razão entre as ondas refletidas pelas ondas que chegam no local de reflexão [5, 12] e  $i$  é a unidade imaginária ( $i^2 = -1$ ).

As equações (2.3) e (2.4) para pressão e fluxo são aplicadas em cada segmento de vaso do modelo de árvore arterial, tomando  $x = 0$  para o nó proximal e  $x = L$  para o nó distal do segmento. Um segmento de vaso é definido pelo intervalo vascular entre dois locais de ramificação [26]. No sistema arterial, as bifurcações são os locais de ramificação mais comuns [25].

Em [6], um segmento de vaso é identificado por  $(k, j)$ , onde o primeiro  $k$  representa o nível da geração e  $j$  representa a ordem do segmento naquela geração, como mostrado na Figura 2.1. Desta forma, a pressão e o fluxo ao longo de um segmento  $(k, j)$  do modelo de árvore arterial são dados por:

$$\begin{aligned} p(k, j) &= \bar{p}(k, j) \exp \left[ i\omega \left( t - \frac{x(k, j)}{c(k, j)} \right) \right] \\ &+ R(k, j) \bar{p}(k, j) \exp \left[ i\omega \left( t - \frac{2L(k, j)}{c(k, j)} + \frac{x(k, j)}{c(k, j)} \right) \right], \end{aligned} \quad (2.5)$$

$$\begin{aligned} q(k, j) &= Y(k, j) \left\{ \bar{p}(k, j) \exp \left[ i\omega \left( t - \frac{x(k, j)}{c(k, j)} \right) \right] \right. \\ &\left. - R(k, j) \bar{p}(k, j) \exp \left[ i\omega \left( t - \frac{2L(k, j)}{c(k, j)} + \frac{x(k, j)}{c(k, j)} \right) \right] \right\}, \end{aligned} \quad (2.6)$$

nos quais  $\bar{p}(k, j)$  é a amplitude combinada do grupo de ondas progressivas no segmento  $(k, j)$  e  $R(k, j)$  é o coeficiente de reflexão no final daquele segmento, como é chamado a razão das ondas progressivas pelas atrasadas avaliadas no nó distal  $x(k, j) = L(k, j)$ .

O grupo de ondas progressivas viaja no sentido positivo de  $x(k, j)$ , estas são compostas de ondas progressivas vindo de vasos acima deste, bem como, ondas refletidas na junção à montante  $x(k, j) = 0$ . O grupo de ondas atrasadas viaja no sentido oposto e é composto por ondas vindas de vasos à jusante como ondas refletidas na junção à jusante  $x(k, j) = L(k, j)$ .

As equações (2.5) e (2.6) descrevem, respectivamente, as ondas de pressão e de fluxo localmente em um segmento  $(k, j)$  do modelo de árvore, e localmente na posição  $x(k, j)$  dentro deste segmento de vaso. As duas variáveis desconhecidas são a amplitude da pressão  $\bar{p}(k, j)$  e o coeficiente de reflexão  $R(k, j)$ , que são detalhados na Seção 2.1.1.

A Figura 2.1 mostra a notação usada para identificar cada segmento de vaso  $(k, j)$ , onde  $k$  é a geração/nível do vaso e  $j$  é um número sequencial dentro daquela geração. Os nós proximal e distal do segmento  $(k, j)$  são denotados por  $A$  e  $B$ , respectivamente. O coeficiente

de reflexão  $R(k, j)$  do segmento  $(k, j)$  está associado ao nó distal  $B$ .

Na equação (2.6), tem-se a admitância característica para cada segmento dada por:

$$Y(k, j) = \frac{A(k, j)}{\rho(k, j) c(k, j)}, \quad (2.7)$$

nos quais  $A(k, j)$  é a área da seção transversal do segmento  $(k, j)$ ,  $\rho(k, j)$  é a densidade do fluido dentro do vaso e  $c(k, j)$  é a velocidade da onda correspondente. A admitância de um segmento é uma medida do quanto o segmento permite o fluxo.

Assumindo um segmento elástico de parede fina, a velocidade da onda  $c(k, j)$  é calculada por [5]:

$$c(k, j) = \sqrt{\frac{E(k, j) h(k, j)}{\rho(k, j) d(k, j)}}, \quad (2.8)$$

onde  $E(k, j)$  é o módulo de Young,  $d(k, j)$  é o diâmetro do segmento  $(k, j)$  e  $h(k, j)$  é a espessura da parede do segmento, a qual neste estudo é dada por [6]:

$$h(k, j) = 0,05d(k, j). \quad (2.9)$$

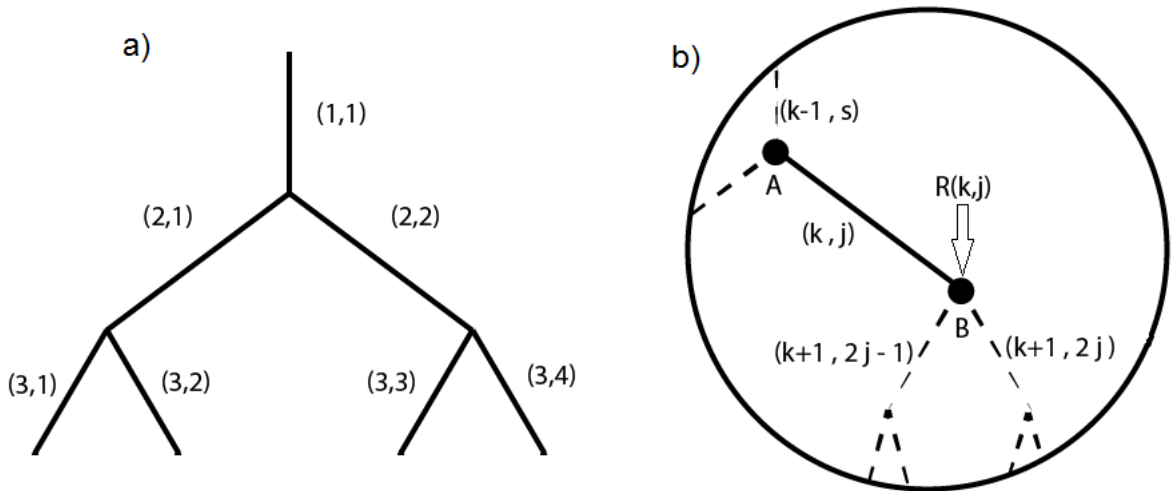


Figura 2.1: Notação usada para identificar cada segmento de vaso  $(k, j)$  (figura adaptada de [6]).

### 2.1.1 CÁLCULO DA PRESSÃO E DO FLUXO SANGUÍNEO

Para determinar a pressão  $\bar{p}(k, j)$  em um certo segmento  $(k, j)$ , aplica-se a condição de continuidade de pressão no nó proximal  $A$  (ver Figura 2.1). Escrevendo as componentes pro-

gressiva e atrasada da onda como  $p_f(k, j)$  e  $p_b(k, j)$  respectivamente, a pressão na posição proximal do segmento  $x(k, j) = 0$  é dada por:

$$[p(k, j)]_A = [p_f(k, j)]_A + [p_b(k, j)]_A, \quad (2.10)$$

nos quais as pressões  $[p_f(k, j)]_A$  e  $[p_b(k, j)]_A$  são expressas por:

$$[p_f(k, j)]_A = \bar{p}(k, j) \exp[i\omega t], \quad (2.11)$$

$$[p_b(k, j)]_A = R(k, j) \bar{p}(k, j) \exp\left[i\omega\left(t - \frac{2L(k, j)}{c(k, j)}\right)\right]. \quad (2.12)$$

Similarmente, a pressão no segmento pai ( $k - 1, s$ ) pode ser escrita como:

$$p(k - 1, s) = p_f(k - 1, s) + p_b(k - 1, s), \quad (2.13)$$

nos quais  $s$  é um número sequencial do segmento pai e as pressões  $p_f(k - 1, s)$  e  $p_b(k - 1, s)$  são dadas por:

$$p_f(k - 1, s) = \bar{p}(k - 1, s) \exp\left[i\omega\left(t - \frac{x(k - 1, s)}{c(k - 1, s)}\right)\right], \quad (2.14)$$

$$p_b(k - 1, s) = R(k - 1, s) \bar{p}(k - 1, s) \exp\left[i\omega\left(t - \frac{2L(k - 1, s)}{c(k - 1, s)} + \frac{x(k - 1, s)}{c(k - 1, s)}\right)\right].$$

No nó distal do vaso superior,  $x(k - 1, s) = L(k - 1, s)$ , a pressão é dada por:

$$[p(k - 1, s)]_A = [p_f(k - 1, s)]_A + [p_b(k - 1, s)]_A, \quad (2.15)$$

nos quais

$$[p_f(k - 1, s)]_A = \bar{p}(k - 1, s) \exp\left[i\omega\left(t - \frac{L(k - 1, s)}{c(k - 1, s)}\right)\right], \quad (2.16)$$

$$[p_b(k - 1, s)]_A = R(k - 1, s) \bar{p}(k - 1, s) \exp\left[i\omega\left(t - \frac{L(k - 1, s)}{c(k - 1, s)}\right)\right]. \quad (2.17)$$

A condição de continuidade da pressão exige que na junção ela assuma um único

valor, portanto

$$[p_f(k-1, s)]_A + [p_b(k-1, s)]_A = [p_f(k, j)]_A + [p_b(k, j)]_A. \quad (2.18)$$

Substituindo as equações (2.11), (2.12), (2.16) e (2.17) na equação (2.18) e resolvendo para  $\bar{p}(k, j)$ , resulta em:

$$\bar{p}(k, s) = \frac{\bar{p}(k-1, s) [1 + R(k-1, s)] \exp \left[ -\frac{i\omega L(k-1, s)}{c(k-1, s)} \right]}{1 + R(k, j) \exp \left[ -2i\omega \frac{L(k, j)}{c(k, j)} \right]}. \quad (2.19)$$

Conforme Duan e Zamir [6], para efeitos de cálculo da pressão e fluxo, adimensionalizam-se as pressões em (2.19) em termos da pressão de entrada  $p_0 = \bar{p}_0 \exp[i\omega t]$ .

Considerando  $P(k, j) = \frac{p(k, j)}{p_0}$  e  $\bar{P}(k, j) = \frac{\bar{p}(k, j)}{\bar{p}_0}$ , a equação (2.5) para o cálculo da pressão pode ser expressa de forma adimensionalizada por:

$$\begin{aligned} P(k, j) &= \bar{P}(k, j) \{ \exp[-i\beta(k, j)X(k, j)] \\ &\quad + R(k, j) \exp[-i2\beta(k, j)] \exp[i\beta(k, j)X(k, j)] \}, \end{aligned} \quad (2.20)$$

nos quais  $\beta(k, j) = \frac{\omega L(k, j)}{c(k, j)}$  e  $X = \frac{x(k, j)}{L(k, j)}$ . Similarmente, a equação (2.6) para o fluxo  $q(k, j)$  pode ser obtida de forma adimensionalizada por:

$$\begin{aligned} Q(k, j) &= M(k, j) \bar{P}(k, j) \{ \exp[-i\beta(k, j)X(k, j)] \\ &\quad - R(k, j) \exp[-i2\beta(k, j)] \exp[i\beta(k, j)X(k, j)] \}, \end{aligned} \quad (2.21)$$

nos quais  $Q(k, j) = \frac{q(k, j)}{q_0}$ ,  $M = \frac{Y(k, j)}{Y(1, 1)}$  e  $q_0 = Y(1, 1)p_0$ . O cálculo da admitância  $Y(1, 1)$  na posição proximal do segmento raiz, ou seja, da artéria de alimentação é apresentado na próxima seção.

### 2.1.2 CÁLCULO DO COEFICIENTES DE REFLEXÃO E ADMITÂNCIA

Para determinar os coeficientes de reflexão nas junções, consideram-se as duas junções  $A$  e  $B$  das extremidades de um segmento genérico  $(k, j)$  de um modelo de árvore arterial (ver

Figura 2.1). Na posição distal  $B$ , o coeficiente de reflexão é definido por [5, 14]:

$$R(k, j) = \frac{Y(k, j) - [Y_e(k+1, 2j) + Y_e(k+1, 2j-1)]}{Y(k, j) + [Y_e(k+1, 2j) + Y_e(k+1, 2j-1)]}, \quad (2.22)$$

nos quais  $Y_e(k+1, 2j-1)$  e  $Y_e(k+1, 2j)$  são admitâncias efetivas nos segmentos à jusante de  $B$ . Estas admitâncias são determinadas pela razão entre o fluxo e pressão naquela posição, que é dada por:

$$Y_e(k+1, s) = \frac{Y(k+1, s) \{1 - R(k+1, s) \exp[-i2\beta(k+1, s)]\}}{1 + R(k+1, s) \exp[-i2\beta(k+1, s)]}, \quad (2.23)$$

nos quais  $s = 2j-1$  e  $2j$  são os números sequenciais dos dois segmentos filhos e  $R(k+1, s)$  é o coeficiente de reflexão na posição distal de cada segmento. Similarmente,  $Y_e(k, j)$ , a admitância na posição proximal  $A$  do segmento  $(k, j)$  pode ser dada por:

$$Y_e(k, j) = \frac{Y(k, j) \{1 - R(k, j) \exp[-i2\beta(k, j)]\}}{1 + R(k, j) \exp[-i2\beta(k, j)]}. \quad (2.24)$$

Substituindo  $R(k, j)$  da equação (2.22) em (2.24), obtém-se uma equação para cálculo das admitâncias efetivas ao longo do modelo de árvore arterial:

$$Y_e(k, j) = \frac{Y(k, j)[Y_e(k+1, 2j) + Y_e(k+1, 2j-1) + iY(k, j) \tan \beta(k, j)]}{Y(k, j) + i[Y_e(k+1, 2j) + Y_e(k+1, 2j-1)] \tan \beta(k, j)}. \quad (2.25)$$

Em segmentos terminais, pode ser assumido que não ocorrem mais reflexões à jusante das posições distais destes segmentos, portanto a admitância efetiva destes segmentos é igual às suas admitâncias características. Adotando a equação (2.24), todas as admitâncias efetivas podem ser determinadas percorrendo a árvore a partir dos segmentos terminais até o segmento raiz.

### 2.1.3 CÁLCULO DA IMPEDÂNCIA DE ENTRADA

A impedância vascular de um modelo de árvore arterial é expresso por

$$z = \frac{p}{q}, \quad (2.26)$$

onde  $p$  e  $q$  podem ser definidos pelas equações (2.3) e (2.4), respectivamente. Dados que  $p(k, j) = P(k, j)p_0$ ,  $q(k, j) = Q(k, j)q_0$  e  $q_0 = Y(1, 1)p_0$ , em termos de variáveis adimensionais, a impedância pode ser reescrita por

$$Z = \frac{P(k, j)}{Y(1, 1)Q(k, j)}. \quad (2.27)$$

A impedância de entrada de um modelo de árvore arterial determina  $Z$  na posição proximal do vaso raiz, ou seja, em  $x(k, j) = 0$ .

Adotando as equações (2.20) e (2.21) com  $x(k, j) = 0$ , obtém-se a impedância de entrada:

$$Z = \frac{-1}{Y(1, 1)}. \quad (2.28)$$

Em suma, a impedância de entrada em módulo é o inverso da admitância característica da artéria de alimentação.

#### 2.1.4 INCORPORAÇÃO DA VISCOSIDADE E VISCOELASTICIDADE NO MODELO

A partir do modelo matemático aqui apresentado, os seguintes cenários podem ser investigados nas simulações hemodinâmicas:

- **cenário 1:** análise do impacto da viscosidade sanguínea ( $\mu(k, j)$ ).

Os efeitos da viscosidade sanguínea podem ser investigados por substituir a velocidade da onda  $c(k, j)$  por uma velocidade da onda complexa [7]:

$$c_v(k, j) = c(k, j)\sqrt{\epsilon}, \quad (2.29)$$

onde  $\epsilon$  é um fator viscoso que corresponde a um tubo elástico com restrições [7]. Seja  $\alpha$  o número de Womersley adimensional

$$\alpha = r(k, j)\sqrt{\frac{\omega\rho(k, j)}{\mu(k, j)}}, \quad (2.30)$$



o fator viscoso  $\epsilon$  é calculado por:

$$\epsilon = 1 - F_{10}(\alpha), \quad (2.31)$$

onde a função  $F_{10}$  é avaliada deste modo:

$$F_{10}(\alpha) = \frac{2J_1(i^{1,5}\alpha)}{\alpha i^{1,5}J_0(i^{1,5}\alpha)}, \quad (2.32)$$

onde  $J_p$  denota a função de Bessel de índice  $p$ .

- **cenário 2:** análise do impacto da viscoelasticidade da parede do vaso ( $\phi_0$ ).

A viscoelasticidade da parede do segmento é incorporado substituindo o módulo de Young estático  $E(k, j)$  por um módulo elástico complexo  $E_c(k, j)$  no cálculo da velocidade  $c(k, j)$  na equação (2.8) da seguinte forma [6]:

$$E_c(k, j) = |E_c(k, j)| \exp\{i\phi\}, \quad (2.33)$$

onde  $\phi$  é o ângulo de fase entre a pressão e o deslocamento da parede do segmento [23] expresso por  $\phi = \phi_0[1 - \exp(-\omega)]$  e  $|E_c(k, j)|$  corresponde ao módulo de Young fornecido para a simulação.

- **cenário 3:** efeitos da viscosidade sanguínea ( $\mu(k, j)$ ) e da viscoelasticidade da parede do segmento ( $\phi_0$ ) de forma combinada.

Neste último cenário, utiliza-se a equação (2.33) para determinar a velocidade da onda  $c(k, j)$  (2.8) no modelo. Com este resultado, calcula-se a equação (2.29) para determinar a velocidade complexa  $c_v(k, j)$  a ser considerada no modelo.

## 2.2 MODELAGEM COMPUTACIONAL

IGOR: fiz mais uma revisão nesta seção. Cheque se tudo está coerente com o quis dizer após minhas alterações no texto.

Nesta seção, detalham-se alguns aspectos da estrutura de dados e algoritmos que descrevem os passos para o cálculo das equações da pressão  $P$  e fluxo  $Q$  definidas em (2.20)

(2.21), respectivamente.

Inicialmente, salientam-se que as equações (2.22) e (2.23) expõem a necessidade de valores  $Y_e(k+1, 2j)$  e  $Y_e(k+1, 2j-1)$  atrelados aos vasos adjacentes na posição distal do vaso  $(k, j)$ . Por outro lado, o valor da pressão média do vaso  $(k, j)$  calculado usando a equação (2.19) depende de valores à montante desse vaso, ou seja, do valor  $\bar{p}(k-1, s)$ . Tendo isto em vista, a estrutura de dados desenvolvida utiliza de ponteiros para acessar as propriedades de artérias à montante e a jusante de um vaso do modelo.

Neste trabalho, adotou-se a linguagem de programação C++. Essa linguagem de programação escolhida permite que objetos **sejam passados por referência**. **Ao passar um objeto por referência, o endereço de memória é enviado, dando total acesso aos parâmetros do objeto, este endereço é chamado de ponteiro. Ao enviar o ponteiro do vaso raiz, é possível acessar todo o modelo de árvore arterial.** Cada vaso arterial do modelo contém as propriedades apresentadas na Tabela 2.1, além de ponteiros para os vasos adjacentes.

valores conhecidos	unidade	valores calculados	unidade
comprimento ( $L$ )	cm	velocidade da onda ( $c$ )	cm/s
densidade ( $\rho$ )	g/cm <sup>3</sup>	beta ( $\beta$ )	—
módulo de Young ( $E$ )	g/cms <sup>2</sup>	velocidade angular ( $\omega$ )	rad/s
viscosidade ( $\mu_0$ )	cm <sup>2</sup> /s	espessura da parede ( $h$ )	cm
raio ( $r$ )	cm	número de Womersley ( $\alpha$ )	—
		admitância característica ( $Y$ )	cm <sup>4</sup> s/g
		admitância efetiva ( $Y_e$ )	cm <sup>4</sup> s/g
		coeficiente de reflexão ( $R$ )	g/cm <sup>4</sup> s
		fator viscoso ( $\epsilon$ )	—
		pressão ( $P$ )	—
		pressão média ( $\bar{p}$ )	—
		fluxo ( $Q$ )	—

Tabela 2.1: Propriedades de cada vaso arterial do modelo.

Além das propriedades especificadas na Tabela 2.1, o vaso possui ainda três ponteiros para os seus vasos adjacentes. Um destes ponteiros, para o vaso adjacente ao seu nó proximal, ou seja, para o seu vaso pai  $(k-1, s)$ . Os demais ponteiros para os vasos adjacentes ao seu nó distal, ou seja, para seus vasos filhos à esquerda (*esq*) e à direita (*dir*) expressos por  $(k+1, 2j-1)$  e  $(k+1, 2j)$ , respectivamente. Este arranjo de ponteiros é equivalente a uma árvore duplamente encadeada, onde através de um único ponteiro para um vaso é possível trafegar a árvore nos dois sentidos. Isto é útil quando é necessário acessar propriedades de

outro vaso para se determinarem por exemplo, a pressão média, coeficientes de reflexão e admitâncias de um dado vaso.

Os ponteiros permitem também definir se o vaso é a raiz ou uma folha. Caso o ponteiro para o segmento superior  $(k - 1, s)$  seja igual ao valor da *flag* nula se trata de um vaso raiz. Caso ambos vasos  $(k + 1, 2j - 1)$  e  $(k + 1, 2j)$  sejam iguais ao valor da *flag* nula se trata de vaso folha, isto é, um vaso terminal.

Basicamente, os passos para determinar as ondas de pressão e fluxo que percorrem um modelo de árvore arterial usando o modelo de Duan e Zamir são dados por:

1. Cálculo da admitância característica ( $Y$ ) de cada vaso;
2. Cálculo da admitância efetiva ( $Y_e$ ) de cada vaso;
3. Cálculo do coeficiente de reflexão ( $R$ ) de cada vaso;
4. Armazenar valor da admitância característica ( $Y_r$ ) do vaso raiz, isto é, o valor de  $Y(1, 1)$ ;
5. Cálculo da pressão média ( $\bar{p}$ ) em cada vaso;
6. Cálculo das ondas de pressão e fluxo  $P$  e  $Q$ .

Os passos acima mencionados podem ser organizados no Algoritmo 1. Esse algoritmo tem como entrada: o modelo de árvore arterial ( $\mathcal{MAA}$ ) com seus vasos caracterizados por suas propriedades, a frequência ( $f$ ) em Hz, um fator de escala da viscosidade ( $\gamma_\mu$ ), o parâmetro da viscoelasticidade da parede ( $\phi$ ) e a quantidade de espaçamento para discretização de um vaso ( $N$ ).

O Algoritmo 1 proposto tem dois módulos, a saber:  $\mathcal{M}_1$  e  $\mathcal{M}_2$ . O módulo  $\mathcal{M}_1$  realiza o cálculo percorrendo o caminho a partir dos vasos finais até o vaso raiz (estratégia do tipo *bottom-up*). O módulo  $\mathcal{M}_2$  realiza os cálculos percorrendo o modelo a partir do vaso raiz até

os vasos terminais (abordagem do tipo *top-bottom*).

---

**Algoritmo 1:** Cálculos hemodinâmicos do modelo de árvore arterial ( $\mathcal{MAA}$ ).

---

**Entrada:**  $\mathcal{MAA}, f, \gamma_\mu, \phi, \bar{p}_0, N$

---

```

1  início
2      inicializa vaso  $v$  a partir do vaso raiz;
3      se existe( $v$ ) então
4          | Chama  $\mathcal{M}_1(v, f, \gamma_\mu, \phi_0)$  e armazena  $Y_r$ ;
5      fim se
6      se existe( $Y(1, 1)$ ) então
7          | Chama  $\mathcal{M}_2(v, \bar{p}_0, Y_r, N)$ ;
8      fim se
9  fim

```

---

O módulo  $\mathcal{M}_1$  do Algoritmo 1 visa calcular os valores de admitância efetiva ( $Y_e$ ), admitância característica ( $Y$ ) e coeficiente de reflexão ( $R$ ) de cada vaso. Este módulo é definido pelo Algoritmo 2. Necessitam-se das admitâncias efetivas dos vasos à justante do vaso  $k$  ( $Y_e(k+1, 2j-1)$  e  $Y_e(k+1, 2j)$ ) estejam definidas. Em vista disso, verifica-se a existência dos vasos à justante do vaso atual tanto a esquerda (*esq*) quanto a direita (*dir*) e caso existam realiza-se uma chamada recursiva. Após o retorno das chamadas recursivas, calculam-se as propriedades do vaso atual. Assim, as propriedades do vasos à jusante do vaso atual serão conhecidas, ou seja, já terão sido determinadas  $Y_e(k+1, 2j-1)$  e  $Y_e(k+1, 2j)$ .

**Algoritmo 2:**  $\mathcal{M}_1$  – Cálculo das admitâncias e coeficiente de reflexão.

---

```

1   $\mathcal{M}_1(v, f, \gamma_\mu, \phi)$ 
2  início
3      se existe( $v \rightarrow esq$ ) então
4           $\mathcal{M}_1(v \rightarrow esq, f, \gamma_\mu, \phi);$ 
5      fim se
6      se existe( $v \rightarrow dir$ ) então
7           $\mathcal{M}_1(v \rightarrow dir, f, \gamma_\mu, \phi);$ 
8      fim se
9      Calcula as propriedades  $c, \omega, \beta;$ 
10     se ( $\gamma_\mu == 0$ ) e ( $\phi == 0$ ) então
11         Calcula  $Y;$ 
12     fim se
13     senão
14         se ( $\gamma_\mu \neq 0$ ) então
15             Calcula  $\alpha, \epsilon, E_v, c_v$  e  $Y_v;$ 
16             Atualiza  $E = E_v, c = c_v$  e  $Y = Y_v;$ 
17         fim se
18         se ( $\phi \neq 0$ ) então
19             Calcula  $E_c;$ 
20             Atualize  $E = E_c;$ 
21         fim se
22     fim se
23     se ( $v$  é um vaso terminal) então
24          $Y_e = Y$  e  $R = 0;$ 
25     fim se
26     senão
27         Calcula  $Y_e$  e  $R;$ 
28     fim se
29 fim

```

---

O módulo 2 ( $\mathcal{M}_2$ ) do Algoritmo 1 visa calcular o valor da pressão média, pressão e fluxo em cada vaso. Esse módulo é expresso no Algoritmo 3. Como expresso na equação (2.19), o valor da pressão média requer o valor da pressão média do vaso à montante  $(k-1, s)$ . Com isso, inicialmente, se o vaso atual é a artéria de alimentação (raiz), neste caso  $\bar{p} = \bar{p}_0$ . Caso contrário, calcula-se  $\bar{p}$  com a equação (2.19). Em seguida, o valor das ondas de pressão e fluxo são obtidas ao longo de cada vaso 1D, que são discretizados. Por fim, a recursão é enviada aos segmentos inferiores, desta forma se garante a existência de um valor  $\bar{p}(k-1, s)$ .

---

**Algoritmo 3:**  $\mathcal{M}_2$  – Cálculo da pressão e fluxo ao longo de cada vaso.

---

```

1   $\mathcal{M}_2(v, \bar{p}_0, Y_r, N)$ 
2  início
3      se  $(v \text{ é o vaso raiz})$  então
4           $\bar{p} = \bar{p}_0;$ 
5      fim se
6      senão
7          Calcula  $\bar{p}$ ;
8      fim se
9      Calcula  $P(X_i)$  e  $Q(X_i)$ , onde  $X_i \in [0, 1]$ ;
10     se  $\text{existe}(v \rightarrow \text{esq})$  então
11          $\mathcal{M}_2(v \rightarrow \text{esq}, \bar{p}_0, Y_r, N);$ 
12     fim se
13     se  $\text{existe}(v \rightarrow \text{dir})$  então
14          $\mathcal{M}_2(v \rightarrow \text{dir}, \bar{p}_0, Y_r, N);$ 
15     fim se
16 fim
```

---

No Algoritmo 3 tem-se que  $N$  denota a quantidade de espaçamento  $dX$  no intervalo  $[0, 1]$  para discretização de um vaso. Assim,  $dX = 1/N$  e  $X_i = i dX (i = 0, 1, \dots, N)$ .

### 3 FERRAMENTA COMPUTACIONAL

Nesta seção apresentam-se detalhes da ferramenta computacional desenvolvida para simulação do escoamento sanguíneo em modelos de árvores arteriais. A principal finalidade desta ferramenta é possibilitar a visualização de estruturas das árvores arteriais e após da simulação hemodinâmica, a visualização das curvas de distribuição do fluxo sanguíneo e pressão.

Esta ferramenta foi desenvolvida em C++ utilizando as bibliotecas comuns do Qt 5.15.0 [1] e OpenGL [10], que ajudam na construção da interface gráfica e na exibição de objetos, como árvores arteriais e gráficos. Ela foi nomeada de *Iterador Gráfico Universal* (IGU), pois em seu modelo de classes qualquer objeto que implemente a classe *WiseObject*, elucidada na Seção 3.1, está apto para realizar iterações e desenhar-se através de diretivas OpenGL em um elemento de interface gráfica. Buscou-se no desenvolvimento desta ferramenta alto grau de generalização para que seja possível analisar todos os parâmetros de um objeto facilmente e que ele possua grande versatilidade. Além de árvores arteriais a ferramenta possibilita que outros objetos sejam geradas para visualização.

A Figura 3.1 ilustra a ferramenta desenvolvida. À seguir, apresentam-se em detalhes a implementação computacional realizada.

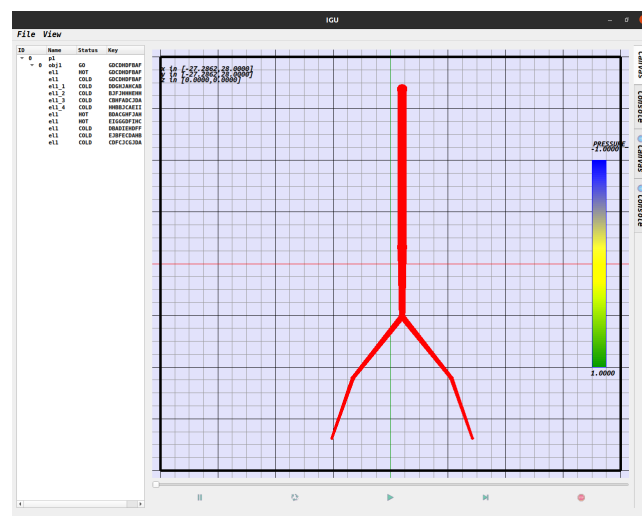


Figura 3.1: Interface gráfica da ferramenta desenvolvida.

Através da modelagem de classes no paradigma do C++ [17], foi possível realizar

diversas generalizações que ampliam a adaptabilidade dos objetos que podem ser inseridos neste modelo de classes. Na seção seguinte apresentamos um esquema de classes virtuais, ou abstratas, que foram criadas para facilitar o manuseio dos objetos dentro do ambiente computacional e prover funções básicas.

Uma classe virtual não possui construtor próprio, porque classes virtuais são incompletas. Estas classes possuem métodos virtuais, que por sua vez precisam ser definidos pela classe herdeira. Portanto uma classe virtual funciona como um conjunto de regras que classes herdeiras devem seguir, todas as classes que implementem esta classe virtual devem ter sua própria definição dos métodos virtuais, podendo ter funcionamentos distintos, entretanto recebendo os mesmo parâmetros. Utilizando este conceito a estrutura de dados, classes virtuais generalizaram os objetos e garantem o seu funcionamento, separando as estruturas de acordo com seu propósito em classes.

### 3.1 ESTRUTURA DE DADOS

Nesta seção apresentam-se detalhes da estrutura de dados adotada no funcionamento da ferramenta computacional. Como mencionado na Seção 2.2 é utilizada uma estrutura de ponteiros que é capaz de armazenar todas as informações do modelo geométrico da árvore arterial. A ferramenta computacional foi desenvolvida para ser capaz de armazenar, carregar e iterar o modelo matemático e gerar novas estruturas para visualização.

As seções à seguir descrevem a estrutura de dados genérica, sendo o principal objeto de estudo o fluxo pulsátil enquanto atravessa uma estrutura de árvore arterial e a visualização de seus resultados. Em versões anteriores, a ferramenta computacional definia seus modelo matemático, geométrico e de visualização em apenas uma estrutura abstrata, o que gerou classes grandes com difícil manutenção e pouca versatilidade. Isto porque o mesmo objeto ficava responsável por diversas funções, era capaz de se instanciar, iterar e desenhar na tela através de diretivas OpenGL. Portanto, nesta versão da ferramenta computacional as classes seguem o conceito de propósito único, cujo objetivo é garantir que as classe tenham uma função apenas para que sua implementação seja menor e mais facilmente interpretada.

Com a experiência adquirida previamente três principais fluxos foram identifica-



das: A iteração dos objetos, uma vez carregados estes objetos são duplicados e então atualizados por algoritmos, resultando em um objeto no estado original e um objeto atualizado pelo algoritmo; O armazenamento dos objetos, os elementos criados no método de iteração podem ser armazenados em um cache e então recuperados; E a exibição dos objetos, uma vez carregados os objetos podem ser exibidos na tela e visualizados. Entendendo o método iterativo como uma animação em que todos os quadros precisavam ser armazenados, o objeto inteligente *WiseObject* foi criado representando toda a animação e o elemento inteligente *WiseElement* como cada quadro da animação. Cada quadro antes de ser exibido precisa ser processado em diretivas OpenGL, o responsável sobre esse processo é o elemento gráfico *GraphicObject*. Igualmente quando se tratam de animações e reproduções de vídeo, somente quadros necessários estarão em memória.

Finalmente, estas estruturas são utilizadas pela ferramenta computacional possibilitando que modelos geométricos sejam carregados, iterados e então exibidos, sem que haja perda dos quadros ou de algum dado. Os elementos principais da estrutura de dados serão descritos nas seções que se seguem.

### 3.1.1 ELEMENTO INTELIGENTE

Elementos inteligentes são objetos que implementam a classe *WiseElement*, o principal objetivo de um elemento inteligente é manter os dados mais recentes de um modelo geométrico e os dados necessários para o método de iteração. No caso do estudo do fluxo pulsátil consideramos que uma iteração seja a execução completa do algoritmo apresentado na Seção 2.2 em toda a árvore. Portanto cada elemento inteligente possuirá todas as informações do modelo geométrico de uma árvore arterial, seus segmentos e suas propriedades.

Através das malhas estruturadas e desestruturadas presentes na biblioteca *VTK (Visualization ToolKit)* é possível descrever diversas estruturas de dados através de elementos padronizados, como pontos, linhas e células. Utilizando os elementos básicas contidas nestas malhas, variáveis e ferramentas da linguagem a estrutura básica da arquitetura foi criada, presente na Figura 3.2.

A Figura 3.2 mostra que um elemento inteligente é composto por duas outras estruturas: A primeira *WiseStructure*, utiliza pontos, linhas, células e campos para determinar

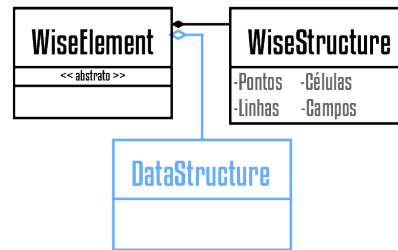


Figura 3.2: Representação de classes de um elemento inteligente. *WiseElement* a classe abstrata base e seus componentes: *WiseStructure* representa a estrutura contida em um arquivo VTK e *DataStructure* representa a estrutura de ponteiros e variáveis utilizadas na iteração. Tingido de azul as estruturas que nem sempre estão presentes.

estruturas geométricas; A segunda *DataStructure*, representa os dados abstratos específicos de cada elemento. Estas estruturas são equivalentes entre si, isto é feito para que a primeira estrutura siga um formato padrão de pontos, linhas, células e campos seja mantida. Enquanto dados abstratos equivalentes podem ser utilizados. Os dados mantidos na estrutura padrão *WiseStructure* são utilizados principalmente na leitura e escrita de objetos, portanto são mantidos como vetores que possuem cadeias de caracteres, os dados presentes nessa estrutura podem ser editados pelo usuário. Enquanto os dados abstratos contém a mesma informação contida em variáveis da linguagem, como números inteiros, vetores, ponteiros e outros.

Cada parâmetro de um elemento inteligente representa uma grandeza associada à um componente do modelo geométrico (pontos, linhas e células) ou ainda sobre todo o modelo (campos). Cada parâmetro irá possuir um nome e uma estrutura associada, os raios de um árvore arterial  $r$  são dados relacionados às linhas da estrutura e somente um valor será armazenado. Enquanto os valores da onda de uma onda de pressão  $P$  através de um segmento com  $X \in [0, 1]$  estão relacionados as mesmas linhas, mas são representados por uma sequência de valores. Ambos os valores são salvos na mesma estrutura e podem ser acessados da mesma forma.

Como demonstrado na Figura 3.3 um elemento inteligente é aquele que implementa a classe abstrata *WiseElement*, os dados abstratos *DataStructure* de cada classe podem ser salvos na estrutura padrão *WiseStructure* e utilizados quando necessário.

O modelo geométrico de uma árvore arterial foi traduzido para o elemento inteligente *WiseArteryTree*, a estrutura inteligente deste elemento conta com pontos e linhas que

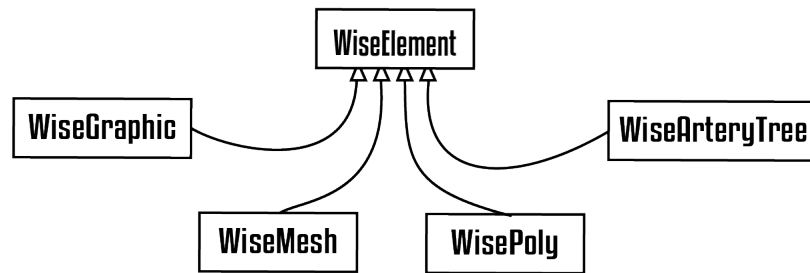


Figura 3.3: Tipos de elementos inteligentes. *WiseGraphic*, um gráfico bidimensional. *WiseMesh*, uma malha bidimensional. *WisePoly*, uma malha tridimensional. *WiseArteryTree*, uma árvore arterial

definem os parâmetros do modelo geométrico na estrutura *WiseStructure*, a estrutura abstrata *DataStructure* conta com ponteiros para cada segmento e grandezas físicas armazenadas em pontos flutuantes de precisão dupla.

O elemento inteligente *WiseGraphic* foi criado para armazenar dados ao longo de uma dimensão, como a pressão ao longo da árvore arterial. Os elementos *WiseMesh* e *WisePoly* foram criados para armazenar dados ao longo de duas dimensões e ao longo de três dimensões, estes dois elementos foram utilizados como exemplo de objetos que podem ser generalizados na ferramenta e visualizados, mudando pouco as estruturas já existentes. O elemento *WiseGraphic* é equivalente a um vetor de valores  $(x, v)$ , o elemento *WiseMesh* a uma matriz de valores  $(x, y, v)$  e o elemento *WisePoly* uma matriz tridimensional de valores  $(x, y, z, v)$ , onde  $v$  é o valor associado e  $(x, y, z)$  a posição. Podemos armazenar em um *WiseMesh* o valor  $v$  da pressão ao longo de uma árvore arterial na direção  $x$  variando a frequência no eixo  $y$ , por exemplo.

Os elementos inteligentes servem como estruturas de armazenamento padrão que compõem outros objetos. O ciclo de manipulação desses elementos se divide em três partes: A criação, aonde os objetos podem ser criados a partir de exemplos pré-definidos ou através de um arquivo de entrada *VTK* ou *XML* (*eXtensible Markup Language*); A iteração, processo em que o elemento inteligente com todas as estruturas definidas e consistentes é utilizado por um algoritmo; A exibição, este último ciclo utiliza a estrutura atualizada e gera um objeto

para visualização.

Todo elemento inteligente completo possui uma redundância de dados, podendo ser representado por qualquer uma das duas estruturas que o compõe. A estrutura *WiseStructure* utiliza componentes simples para descrever as estruturas e seus parâmetros, essa estrutura é a que mais se assemelha à encontrada na arquitetura *VTK*. Por utilizar uma quantidade limitada de diretivas o arquivo proveniente dessa estrutura pode ser rapidamente lido, interpretado e até mesmo exportado.

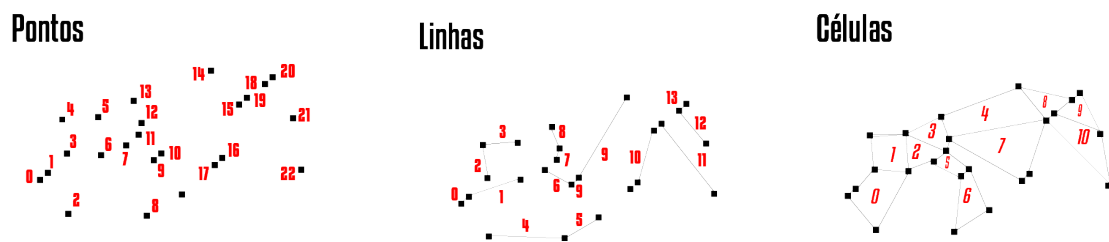


Figura 3.4: Pontos utilizados na especificação do modelo geométrico. Linhas utilizadas na especificação do modelo geométrico, através dos pontos previamente definidos. Células utilizadas na especificação do modelo geométrico, através dos pontos previamente definidos.

Os componentes de uma estrutura inteligente são pontos, células, linhas e campos. Pontos demarcam uma posição no espaço, células e linhas representam conjuntos de pontos e os campos são dados gerais da estrutura. Um modelo de pontos e linhas é capaz de representar o mesmo modelo geométrico de uma árvore arterial, basta considerar cada ponto uma bifurcação de vasos e cada segmento de vaso uma linha. Através destes componentes simples é possível armazenar e acessar dados sobre cada ponto e linha, armazenando dados de cada segmento pertencente à uma árvore arterial através da estrutura *WiseStructure*. Para dados gerais do modelo, como a frequência  $f$ , os campos da estrutura inteligente são utilizados.

A classe *WiseElement* foi desenvolvida para construir a estrutura abstrata (*DataStructure*) à partir das informações contidas na estrutura inteligente (*WiseStructure*), desta forma é importante que a estrutura inteligente possa rapidamente ser armazenado e recuperado. O elemento inteligente é o elemento básico do método de iteração, antes de ser iterado este elemento é duplicado, gerando estruturas inteligente e abstrata idênticas. Imediatamente a estrutura abstrata *DataStructure* é descartada e os dados contidos na estrutura padrão *WiseStructure* são armazenados. Portanto, a estrutura abstrata não estará sempre

presente. Seguindo este ciclo de vida, todos os elementos inteligentes obedecem à máquina de status contida na Figura 3.5.

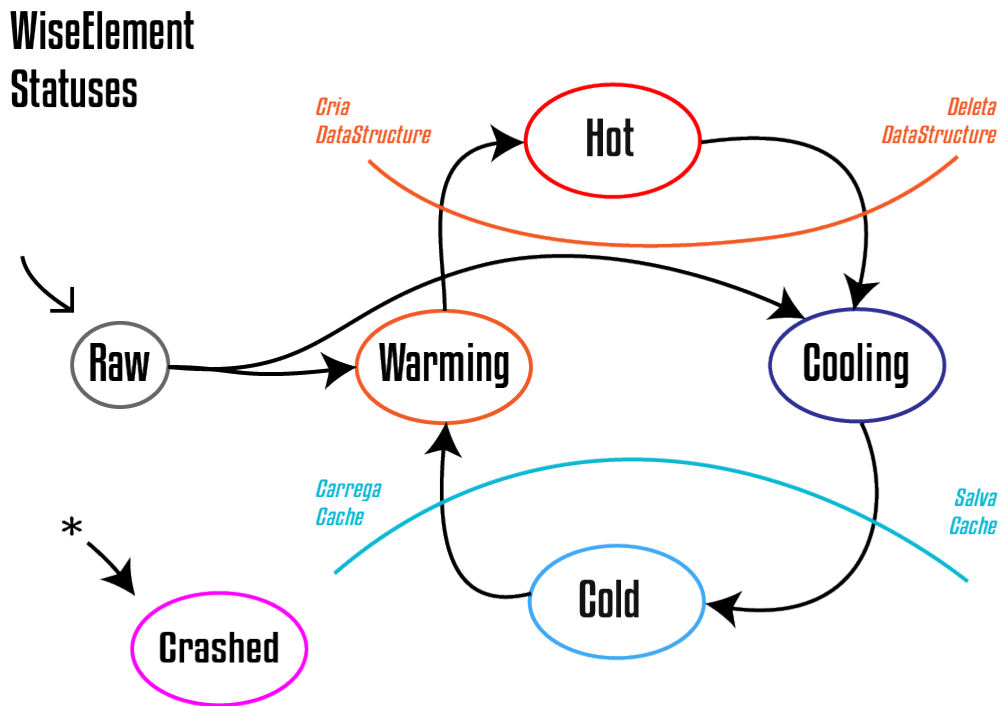


Figura 3.5: Máquina de Status que controla o funcionamento de um elemento inteligente.

A máquina de status dos elementos inteligentes gerencia o ciclo de vida de todos os elementos inteligentes, elementos inicialmente recebem o estado *Raw*, ou cru, que representa um elemento ainda sem estruturas carregadas ou sem consistência garantida. Uma vez que a estrutura inteligente é inserida e verificada o elemento muda para o status *Warming* ou *Cooling*, respectivamente esquentando ou esfriando, suas estruturas são representadas na Figura 3.6.

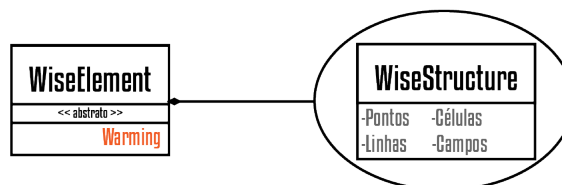


Figura 3.6: Elemento inteligente enquanto no estado *Warming*.

A figura 3.6 mostra as estruturas presentes em elementos no estado *Warming* que é a mesma dos estados *Raw* e *Cooling*. Cada estado possui uma finalidade diferente e representa em que estado estão as informações do elemento inteligente. Elementos no estado *Warming* estão aguardando a construção de sua estrutura abstrata, enquanto elementos no estado *Cooling* aguardam que seus dados sejam salvos em cache. Finalmente, elementos no estado *Raw* indicam que não é esperada a consistência dos dados.

Inicialmente os elementos inteligentes são criados no estado *Raw*, enquanto os dados do elemento são carregados ele permanecerá neste estado. Ao final do carregamento o objeto trocará de estado para *Warming* ou *Cooling*, que indicam o próximo passo do elemento. Estar no estado *Warming* indica que o objeto irá criar a sua estrutura de dados abstrata, enquanto o estado *Cooling* indica que a estrutura inteligente aguarda ser armazenada. Quando os dados abstratos são criados corretamente para a estrutura o estado é definido como *Hot*, neste estado todas as informações do objeto estão em memória, sendo este o estado mais pesado do elemento inteligente. Em contraponto, o estado *Cold* indica que a estrutura inteligente foi corretamente armazenada em disco, elementos inteligentes neste estado estão em seu estado mais leve, pois possuem em memória apenas o caminho para a estrutura inteligente armazenada.

Como demonstrado na figura 3.5, o estado frio (*Cold*) está associado com o uso de um cache para estruturas inteligentes. A estrutura do arquivo é equivalente à uma malha estruturada VTK, mas são efetivamente salvos em um arquivo *XML*. Caso sejam novamente carregados por uma mudança de estado ou deletados o arquivo armazenado é deletado. Quando um elemento inteligente está neste estado ele contém apenas o endereço para o arquivo em que foi armazenado.

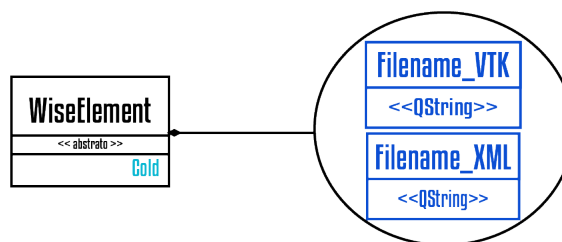


Figura 3.7: Elemento inteligente enquanto no estado *Cold*.

O estado *Crashed* serve para identificar objetos que não tem mais o funcionamento esperado. Durante a troca de estados do elemento é verificado se os elementos esperados estão presentes, caso não estejam o objeto é direcionado à este estado.

Finalmente, o estado *Hot* representa os elementos que possuem todas as estruturas presentes. Isto significa que a estrutura inteligente *WiseStructure* e a estrutura abstrata equivalente *DataStructure* estão presente e são consistentes.

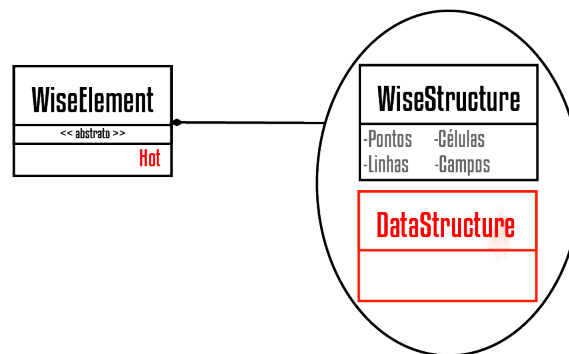


Figura 3.8: Elemento inteligente enquanto no estado *Hot*.

Para que um elemento possa ser iterado ele precisa estar no estado *Hot*, porque durante a iteração de um elemento inteligente seus dados abstratos são utilizados. A cada passo da iteração a estrutura *DataStructure* é atualizada, exigindo uma atualização da estrutura *WiseStructure*.

Seguindo o conceito de classes com propósito único, os elementos inteligentes são responsáveis apenas por gerir os dados contidos na estrutura inteligente e abstrata. Portanto as funções de criar, iterar e visualizar estas estruturas foram divididas em classes.

### 3.1.2 FÁBRICA

Como mencionado anteriormente três principais barreiras foram identificadas armazenamento, iteração e visualização. Foi idealizada uma arquitetura de classes que permite a execução de cada passo através do paradigma de Fábricas Dinâmicas [24]. Uma arquitetura com fábricas permite a criação de instâncias com definições concretas, armazenadas como metadados. Isso facilita a adição de novos objetos que podem ser interpretados sem modificar o código da fábrica em si. Sendo a fábrica também uma classe virtual, três principais

tipos de fábricas foram idealizadas, a primeira responsável por criar elementos inteligentes, a segunda responsável por iterar o elemento e a terceira que cria um elemento gráfico a partir de um elemento inteligente.

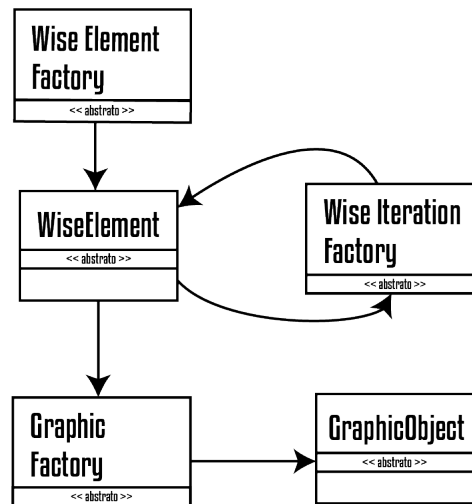


Figura 3.9: Arquitetura de classes fábrica e fluxo de trabalho do elemento inteligente Wise-Element. A fábrica WiseElementFactory é responsável por criar o elementos inteligentes, a fábrica WiseIterationFactory é responsável pela iteração do elemento inteligente e a fábrica GraphicFactory é responsável por criar as estruturas de visualização.

Primeiramente, a fábrica de elementos inteligentes *WiseElementFactory* é utilizada para a criação de elementos inteligentes, essa fábrica é utilizada para criar um elemento inteligente à partir de parâmetros. Esse tipo de fábrica é também uma classe virtual, seus métodos definem que fábricas de elementos inteligentes possuem três maneiras de executar: Um dos métodos permite a criação de elementos à partir de uma lista de exemplos; Outro método permite que um arquivo *VTK* ; Com um arquivo *XML*.

Os métodos de criação de elementos são selecionados através do nome da fábrica e de um dos três métodos de criação. A ferramenta computacional possui uma lista com todas as fábricas disponíveis e as utiliza quando necessário. Devido à forma como os dados são carregados para cada elemento, é necessário que haja uma fábrica para cada tipo de elemento inteligente.



A fábrica *WiseIterationFactory* só pode operar com um tipo específico de elemento inteligente, entretanto é possível que haja mais de uma fábrica de iteração disponível por tipo de elemento inteligente. Desta forma uma árvore arterial pode ser iterada por diferentes algoritmos de iteração e o mesmo ocorre com os outros tipos de elementos. Estas fábricas são responsáveis por executar algum algoritmo que utilize o tipo de dados do elemento inteligente. No caso de uma árvore arterial é possível utilizar uma fábrica que irá executar o modelo matemático descrito na seção 2.2 utilizando os ponteiros para segmentos disponíveis em uma *WiseArteryTree*.

Por último, a fábrica *GraphicFactory* irá criar o objeto gráfico correspondente ao elemento inteligente. Assim como um elemento inteligente um objeto gráfico pode ser salvo em cache. Um elemento inteligente é composto por todas as linhas, pontos, células e seus valores associados, enquanto um objeto gráfico contém as representações gráficas destes elementos e para cada um armazena um valor. Os elementos estruturais de uma árvore arterial, que são seus pontos e linhas, são traduzidos em círculos e cilindros como visto na Figura 3.1. Sobre estes elementos gráficos apenas um valor pode ser exibido por vez ao longo da escala de cores, como o fluxo  $Q$  ou a pressão  $P$ . Dessa forma os elementos gráficos só exigem que um parâmetro por vez sejam armazenados.

### 3.1.3 OBJETO INTELIGENTE

A classe que combina todas as estruturas utilizadas no método de iteração da ferramenta computacional foi nomeada de objeto inteligente *WiseObject*, este objeto preserva todos os passos de iteração e poupa a quantidade de recursos mantida em memória. A classe é composta por uma coleção de elementos inteligentes e objetos gráficos equivalentes entre si. A presença de uma fábrica gráfica é opcional, possibilitando que objetos sejam iterados sem que alguma estrutura seja disponibilizada para visualização. Utilizando a mesma separação de classes com propósito único, fábricas dinâmicas garantem que um objeto inteligente seja criado corretamente. As fábricas presentes na Seção ?? foram incluídas como propriedades de um objeto inteligente, desta forma estes objetos serão compostos por três fábricas.

O ciclo de vida de um objeto inteligente consiste na sua criação, a iteração de um modelo matemático e, opcionalmente, a exibição de um modelo gráfico. Objetos intelligen-

tes são criados com seu primeiro elemento inteligente. Ao criar um objeto inteligente, sua fábrica adiciona em sua estrutura a fábrica de elementos inteligentes, desta forma objetos inteligentes são capazes de replicar seus elementos inteligentes. No momento da criação, o primeiro elemento é duplicado e uma instância segue para o *Forno*, enquanto a outra segue para o *Freezer*.

O elemento inteligente guardado no *Forno* será o objeto utilizado pelo algoritmo a cada iteração, sendo o mais atual. A cada ciclo o elemento contido no *Forno* é duplicado e uma nova instância é adicionada ao *Freezer*, estes elementos serão armazenados.

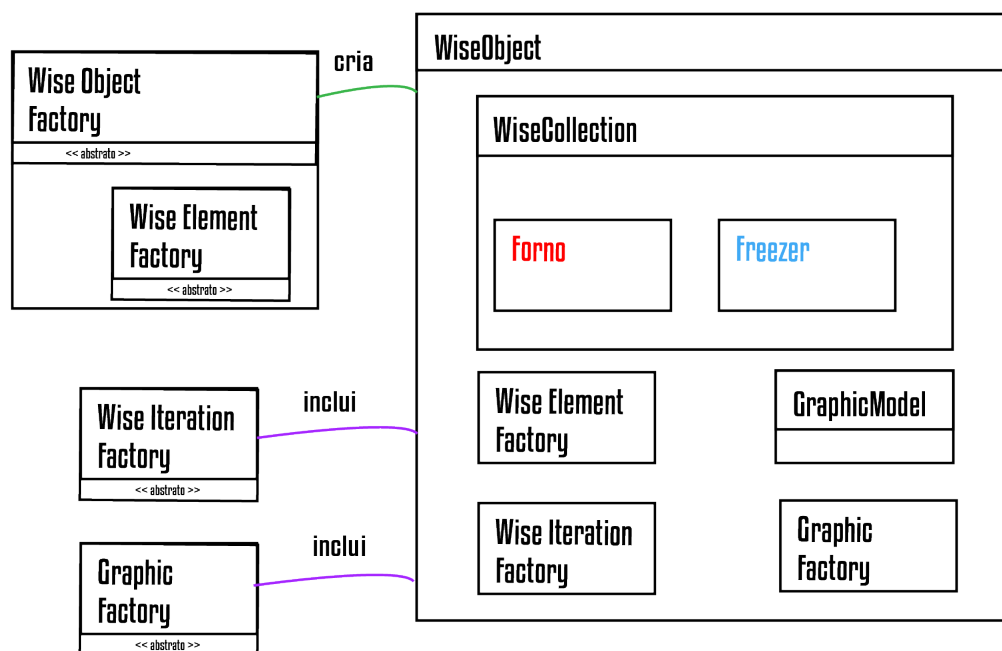


Figura 3.10: Objeto inteligente *WiseObject* e todos seu componentes: *WiseObjectFactory*, fábrica responsável pela criação de objetos inteligentes de um determinado tipo; *WiseIterationFactory*, fábrica de iteração; *WiseGraphicFactory*, fábrica gráfica; *WiseCollection*, coleção de elementos inteligentes; *GraphicModel*, coleção de objetos gráficos.

Através do modelo de classes de um objeto inteligente presente na Figura 3.10 é possível identificar todos os componentes presentes em um objeto inteligente. O objeto inteligente troca seus elementos de estado automaticamente, a coleção de elementos inteligentes *WiseCollection* irá manter apenas um elemento em memória, o elemento contido no *Forno* que deve permanecer no estado *Hot*. Ao mesmo tempo a coleção irá manter um históricos de elementos armazenados no *Freezer* que devem permanecer no estado *Cold*. É possível que o objeto inteligente volte à um estado anterior, substituindo o elemento pre-

sente no *Forno* com algum estado anterior armazenado no *Freezer*, dessa forma o elemento inteligente utilizado no método iterativo será o objeto recuperado. Ao realizar essa operação, a coleção de elementos inteligentes irá recuperar a estrutura inteligente do elemento, alterando seu estado para *Warming*. Em seguida irá recriar os seus dados abstratos utilizando uma das fábricas disponíveis, alterando seu estado para *Hot*.

Objetos inteligentes também tem seu funcionamento descrito por uma máquina de estados. Para definir corretamente os resultados do método de iteração utilizando a estrutura do objeto inteligente, uma sequência de passos foi criada: os dados são instanciados em objetos inteligentes, os dados são ajustados para cada experimento, uma sequência de iterações é executada e finalmente os objetos são descartados.

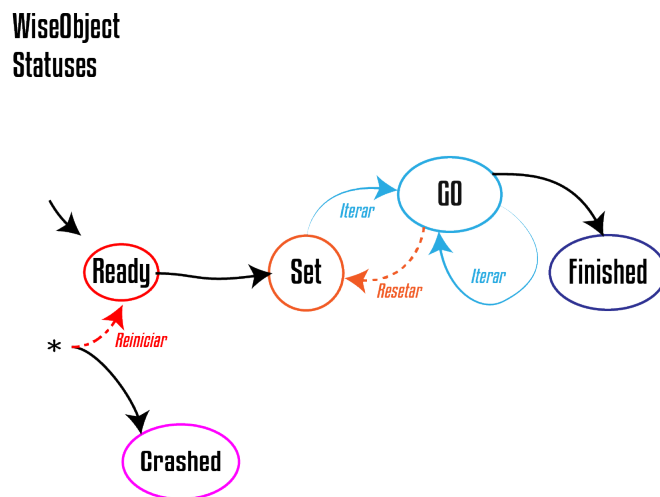


Figura 3.11: Modelo gráfico *GraphicModel*, contém uma coleção de objetos gráficos.

As trocas de estado dos objetos inteligentes são causadas por operações do usuário para preparar o objeto para iteração e para executar o método de iteração. Desta forma o usuário pode definir quais fábricas serão inseridas no objeto, parâmetros do modelo geométrico e do experimento e quais trocas de estados devem ser executadas.

Como objetos inteligentes são criados à partir de um elemento inteligente, eles possuem os mesmos métodos de criação. As fábricas de objetos inteligentes *WiseObjectFactory* são compostas por fábricas de elementos inteligentes *WiseElementFactory*. É possível criar objetos inteligentes de duas formas, utilizando um elemento inteligente já existente ou utilizar os métodos de criação de elementos disponíveis na fábrica de elementos inteligentes.

Em seguida é necessário definir qual será a lógica de iteração do objeto, adicio-

nando uma fábrica de iteração *WiseIterationFactory* compatível. Para o caso de escoamento pulsátil através de uma árvore arterial é necessário se adicionar a fábrica de iteração correspondente ao algoritmo da Seção 2.2 e em seguida definir os parâmetros desejados, como a frequência  $f$ , a viscosidade  $\mu$  e o ângulo de fase  $\phi$ . Com as alterações concluídas o objeto passa a poder ser iterado, opcionalmente a fábrica gráfica *GraphicFactory* pode ser inserida. Caso seja incluída, o objeto inteligente passará a gerar objetos gráficos *GraphicObject* a cada iteração.

Inicialmente, um objeto é criado no estado *Ready* com somente seu elemento inicial e uma fábrica do tipo *WiseElementFactory*. Neste estado é esperada a inclusão das fábricas de iteração e gráficas. Uma vez que elas estejam corretamente acopladas ao objeto inteligente, é possível fazer a troca do estado *Ready* para o estado *Set*. Com a mudança de estado, é adicionado à estrutura inteligente *WiseStructure* todos os parâmetros disponibilizados pela fábrica de iteração e gráfica. Um objeto no estado *Set* indica que o objeto foi corretamente criado, uma fábrica de iteração foi adicionada, possivelmente uma fábrica gráfica e agora aguarda alterações nestes parâmetros ou execução do método iterativo.

Com os parâmetros definidos e as fábricas devidamente acopladas o objeto está pronto para a iteração. O método iterativo de um objeto inteligente é representado na transição para o estado *Go*. Uma iteração de uma *WiseArteryTree* representa o cálculo dos valores de pressão e fluxo em toda a árvore arterial. Caso algum erro ocorra durante o processamento de dados o objeto se desloca para o estado *Crashed*, assim como elementos inteligentes. É possível também finalizar a execução de um objeto inteligente o enviando para o estado *Finished*, neste estado o objeto não poderá ser iterado novamente. O objeto inteligente também pode ser reiniciado à partir de qualquer estado, o que significa que todas as alterações feitas na estrutura do objeto inteligente serão apagadas. Para que parâmetros da iteração possam ser alterados sem que se perda elementos inteligentes até então definidos, é possível que um objeto no estado *Go* seja resetado e retorne para o estado *Set*.

### 3.1.4 OBJETO GRÁFICO

Quando o objeto inteligente *WiseObject* estiver corretamente carregado e uma fábrica gráfica *GraphicFactory* for adicionada à sua estrutura o primeiro objeto gráfico *GraphicObject*

de sua coleção será criado. Assim como o elemento inteligente representa uma iteração, um objeto gráfico representa um ciclo iterativo. Enquanto a estrutura responsável por elementos inteligentes, *WiseCollection*, é responsável por manter os últimos dados de iteração aquecidos, a estrutura *GraphicModel* mantém o objeto que está sendo exibido por alguma tela e elementos próximos. As coleções tem como objetivo manter seus elementos no estado necessário e realizar trocas de estados quando necessário. No caso dos elementos inteligentes, somente o último objeto é necessário e isso não muda, no contexto dos objetos gráficos apenas o objeto exibido em um elemento da interface de usuário e seus vizinhos serão mantidos em memória.

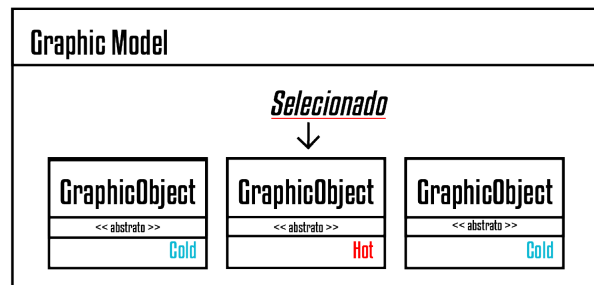


Figura 3.12: Modelo gráfico *GraphicModel*, contém uma coleção de objetos gráficos.

Presente na Figura 3.12 estão os componentes que permitem armazenar todos os quadros da animação final. Quando um objeto *WiseObject* está corretamente configurado com uma instância de fábrica gráfica *GraphicFactory*, seu método iterativo é atrelado à criação de objetos gráficos. Isso permite que o objeto iterado crie um elemento inteligente *WiseElement* e um objeto gráfico *GraphicObject* à cada iteração.

Cada objeto gráfico se desenha através de diretivas OpenGL, o formato do modelo geométrico é apresentado ao usuário através das formas desenhadas, enquanto algum parâmetro do elemento inteligente *WiseElement* é visualizado através de uma escala de cores.

Os objetos gráficos contidos na Figura 3.13 possuem métodos específicos para se desenhar utilizando um gradiente de cores e formas geométricas padrão, bidimensionais ou tridimensionais. Cada forma geométrica utilizada é representada por elementos gráficos *GraphicElement*, elementos que implementam a classe virtual *GraphicElement*. Cada elemento gráfico é atrelado a um valor ou a uma lista de valores, estes valores são selecionados pela fábrica gráfica e adicionados a cada iteração, retirando a informação do elemento

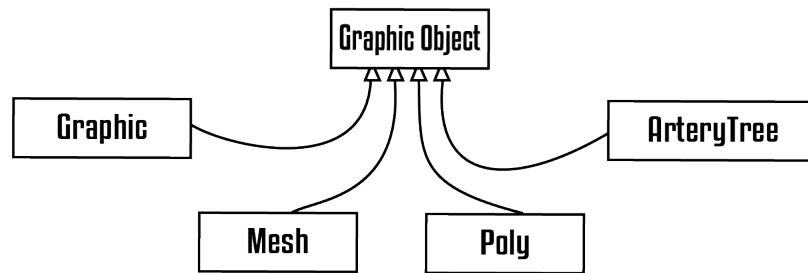


Figura 3.13: Tipos de objetos gráficos *GraphicObjects*.

inteligente atual. Cada objeto gráfico possui um valor máximo e mínimo que é vinculado ao gradiente cores, com isso as cores representam os valores armazenados em cada objeto gráfico.

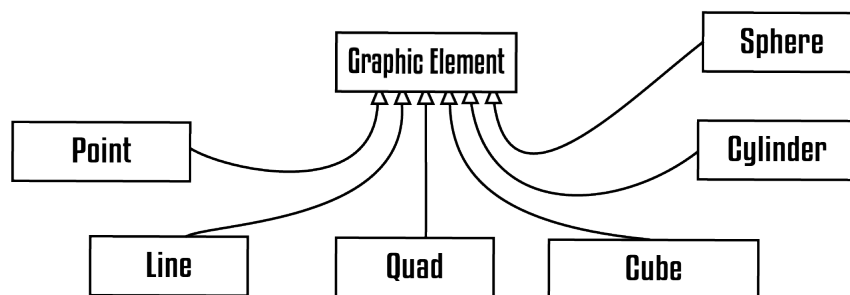


Figura 3.14: Tipos de elementos gráficos *GraphicElements*. *Point*, um ponto. *Line*, uma linha. *Quad*, um quadrado. *Cube*, um cubo. *Cylinder*, um cilindro. *Sphere*, uma esfera.

Os elementos gráficos podem ser utilizados por qualquer tipo de objeto gráfico. A representação geométrica de uma árvore arterial é construída utilizando cilindros e esferas, que representam segmentos de vaso e terminais, respectivamente. Por padrão, os cilindros receberão uma lista de valores da pressão  $P(X) \forall X \in [0, 1]$ , enquanto as esferas receberão os valores da pressão  $P(X)$  quando  $X = 0$  ou  $X = 1$ , condicionado a escolha do nó distal ( $X = 1$ ) ou o nó proximal ( $X = 0$ ). Assim como os objetos inteligentes *WiseObject* têm seus tipos

definidos pelo tipo de elemento inteligente que o compõe, o objeto gráfico *GraphicObject* tem seu tipo definido pelo elemento inteligente que representa, com sua fábrica própria.

Apesar do objeto gráfico *GraphicObject* ser uma redundância dos dados armazenados em um elemento inteligente *WiseElement*, ele é menor, podendo ser carregado e armazenado mais rapidamente. O objetivo das estruturas nesta seção é permitir que diversos objetos gráficos *GraphicObjects* possam ser armazenados em memória. Os objetos gráficos em sequência representam uma animação que pode ser visualizada através da interface gráfica.

### 3.1.5 PROJETO INTELIGENTE

Os projetos inteligentes *WiseProject* são os exopos de trabalho da ferramenta computacional, o primeiro passo para se utilizar a ferramenta é criar um projeto inteligente.

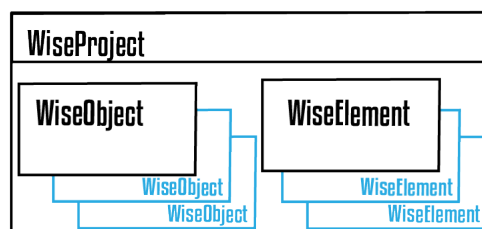


Figura 3.15: Projeto inteligente *WiseProject* e seus componentes, uma lista de objetos inteligentes *WiseObject* e uma lista de elementos inteligentes *WiseElement*.

Como representado na Figura 3.15, os projetos inteligentes são representados por duas coleções, uma de elementos inteligentes *WiseElement* e outra de objetos inteligentes *WiseObject*. Os comandos recebidos pela ferramenta computacional terão efeito sobre apenas um projeto inteligente. Por exemplo, quando um objeto inteligente for criado utilizando um elemento inteligente como parâmetro de entrada eles devem estar no mesmo projeto inteligente.

Os projetos inteligentes são estrutura organizacionais, uma vez que um comando é executado sobre objetos de um projeto, esses objetos são bloqueados pelo projeto. Ao excluir um projeto, todas as demandas devem ser finalizadas. Finalmente, as estruturas dos projetos podem ser salvas e carregadas, portanto estas estruturas servem também para armazenar experimentos inteiros e seus resultados.

### 3.1.6 FÁBRICA DE PROJETO

O conceito de fábrica foi adicionado ao projeto para facilitar que as estruturas sejam construídas através de comandos simples de forma padronizada, inclusive neste conceito, a fábrica de projetos *WiseProjectFactory* é responsável pela construção de projetos. Diferentemente das outras fábricas, a fábrica de projetos inteligentes contém todas as fábricas suportadas pela ferramenta computacional.

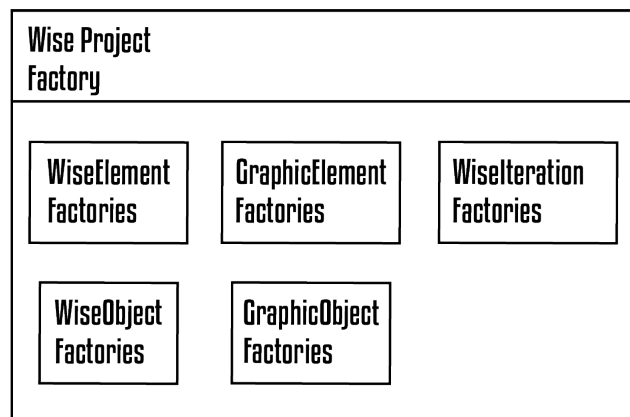
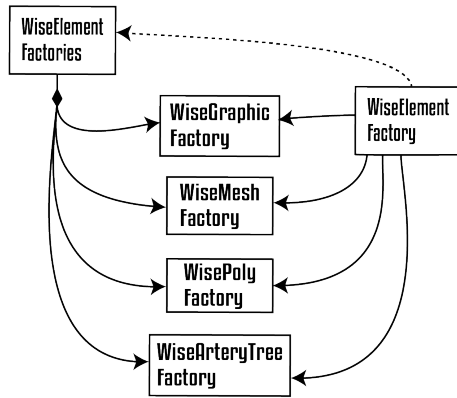


Figura 3.16: Fábrica de projetos inteligentes *WiseProjectFactory* e seus componentes, fábricas de elementos inteligentes *WiseElementFactories*, fábricas de objetos inteligentes *WiseObjectFactories*, fábrica de objetos gráficos *GraphicObjectFactories*, fábricas de elementos gráficos *GraphicElementFactories* e fábricas de iteração *WiseIterationFactories*.

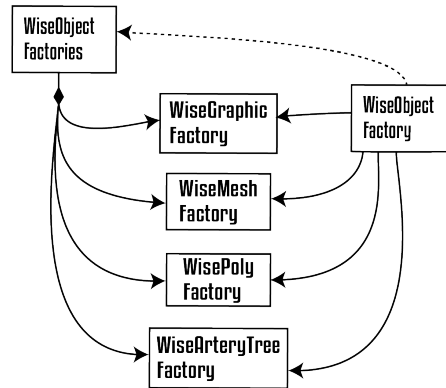
Como é possível observar na Figura 3.16, a fábrica de projetos inteligentes é composta de outras coleções de fábricas, agrupadas pelo tipo de estrutura que criam. Nesta classe estão as fábricas de elementos inteligentes *WiseElement*, objetos inteligentes *WiseObject*, objetos gráficos *GraphicObject*, elementos gráficos *GraphicElement* e iteração *WiseIterationFactory*. Por exemplo, quando um projeto contendo diversas árvores arteriais e gráficos é carregado ele é construído pela fábrica de projetos. A fábrica de projetos irá reconhecer o formato de cada estrutura e encaminhar para a fábrica correspondente.

Para cada tipo de elemento inteligente, objeto inteligente, objeto gráfico e elemento gráfico existe uma fábrica. Uma fábrica de iteração só pode ser utilizada por um tipo de objeto inteligente específico, entretanto um objeto inteligente pode ter mais de uma fábrica de iteração disponível. As fábricas de iteração descritas na Figura 3.17 foram criadas para resolver o modelo matemático descrito na Seção 2.1 e extrair o seu resultado. A fábrica de iteração estática *StaticIterationFactory* é uma fábrica que não altera o objeto inteligente no

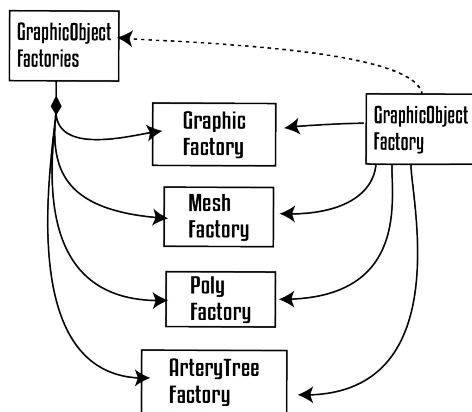




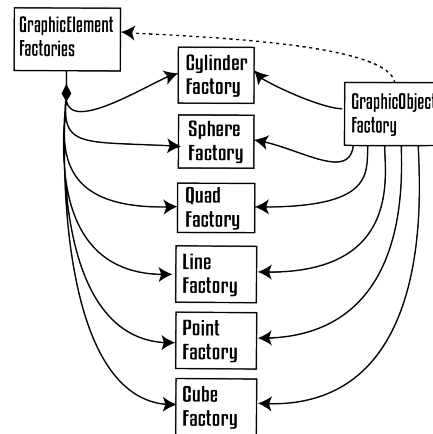
(a) Fábricas de elementos inteligentes



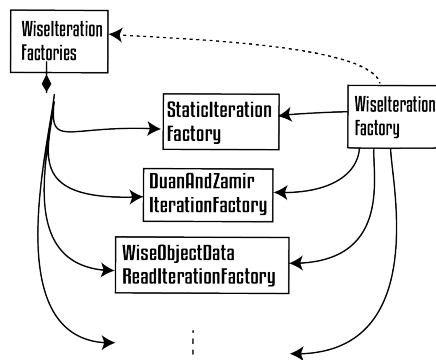
(b) Fábricas de objetos inteligentes



(c) Fábricas de objetos gráficos



(d) Fábricas de elementos gráficos



(e) Fábricas de iteração

Figura 3.17: Todas as fábricas que compõem uma fábrica de projeto.

processo de iteração, esta é a única classe de iteração que pode ser utilizada em mais de um tipo de objeto, podendo ser executada em qualquer tipo de objeto inteligente.

O modelo do escoamento pulsátil proposto está presente na fábrica de iteração *DuanAndZamirIterationFactory*, esta fábrica é responsável por utilizar a estrutura de uma árvore arterial *WiseArteryTree* e aplicar o modelo matemático. Finalmente, a fábrica de iteração *WiseObjectDataReadIterationFactory* é uma fábrica responsável por armazenar os resultados obtidos a cada iteração e armazenar em uma estrutura do tipo *WiseMesh*.

### 3.1.7 THREADS INTELIGENTES

Até o momento as estruturas foram construídas utilizando conceitos padrões da linguagem C++, herança de propriedades através do polimorfismo, ponteiros, classes virtuais e fábricas dinâmicas. Até aqui o conceito de programação paralela não foi adicionado. Observou-se que as estruturas possuem comportamentos padronizados e que um objeto inteligente é autônomo, ele sozinho é capaz de se armazenar, reconstruir, iterar e ocasionalmente se desenhar. Imaginando um cenário em que diversos objetos inteligente estão iterando, as tarefas foram divididas em *Threads*, primeiramente trabalhos de leitura e escrita, em seguida a iteração dos objetos. Desta forma computadores *multi-thread* podem fazer uso de suas threads e permitir que mais de um objeto realize suas tarefas por vez.

Ao longo da pesquisa esta estrutura foi agregada as bibliotecas comuns do Qt 5.15.0 [1], em um primeiro momento para facilitar a visualização dos resultados através dos elementos gráficos de interface de usuários disponibilizados. Em seguida, permitiu o processamento de elementos de forma paralela. Com isso um modelo que suportasse o processamento distribuído utilizando classes *QThreads* que possuem tarefas concorrentes foi construído.

As tarefas se dividem em três principais grupos: tarefas auxiliares, tarefas de leitura/escrita e trabalhos de iteração. O principal trabalho auxiliar é interpretar os comandos recebidos pelo programa e então criar a instância de trabalho *WiseJob*. Sendo os grupos de tarefas mais custosos os de tarefas de escrita/leitura e iteração, se dividiram em threads específicas, *WiseIO* e *WiseProcessor* respectivamente.

Os objetos distribuídos threads contidos na Figura 3.18 possuem um ciclo próprio,

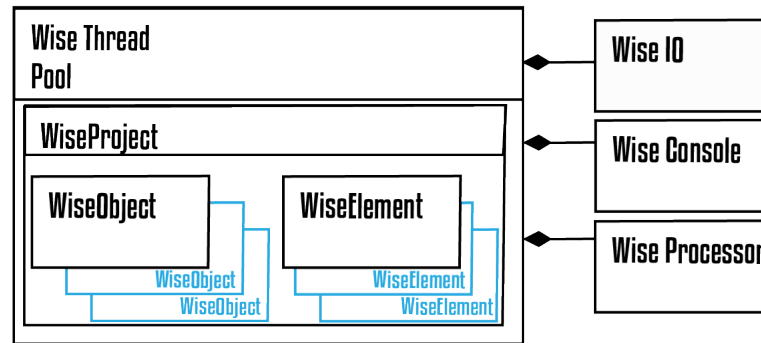


Figura 3.18: Modelo de Threads. *WiseThreadPool*, responsável por orquestrar o funcionamento das demais threads, bem como os objetos contidos em um projeto *WiseProject*. *WiseIO*, thread responsável por processos de leitura e escrita. *WiseConsole*, thread responsável por interpretar os comandos de texto e os traduzir em métodos. *WiseProcessor*, thread responsável por realizar o método iterativo de um objeto inteligente *WiseObject*

portanto executam tarefas assíncronas e precisam de tratamento adequado. O sistema de sinais e fendas disponibilizado pelas bibliotecas comuns do Qt permitem que as threads se comuniquem assincronamente através do envio de mensagens. Estas mensagens são chamadas de trabalhos inteligentes *WiseJobs*, cada uma possuindo sua atividade relacionada e objeto relacionado. Uma vez criados, os trabalhos inteligentes são alocados à sua respectiva categoria passando por um balanceamento executado pelo gerenciador de threads. Enquanto o processamento é feito, todos os dados relativos ao processamento são bloqueados para escrita de outras threads, isso previne a sobrescrita de dados quando há mais de uma thread trabalhando.

O gerenciador de threads *WiseThreadPool* é composto por threads que executam os trabalhos e por projetos inteligentes *WiseProject* que criam trabalhos. Ao executar um comando de texto o projeto deve ser selecionado para que os elementos e objetos certos sejam selecionados. Primeiramente, o comando será recebido pelo gerenciador e o primeiro trabalho inteligente *WiseJob* será criado. Em seguida será interpretado por threads do grupo de tarefas auxiliares *WiseConsole*, caso seja um trabalho de leitura, escrita ou iteração um trabalho inteligente associado será criado e enviado ao gerenciador de threads. Caso não seja um trabalho desses tipos ele será executado na thread atual e o trabalho inteligente finalizado. O console inteligente *WiseConsole* funciona como interpretador principal dos comando e das mensagens, ao receber uma linha de comando este objeto irá analisar seu conteúdo e executar a ação correspondente.

Os elementos inteligentes e objetos inteligentes associados à um trabalho são enviados sempre como referência e protegidos no gerenciador de threads, desta forma é possível que todas as threads trabalhem com a mesma informação na memória sem que ela seja utilizada por outra thread ao mesmo tempo.

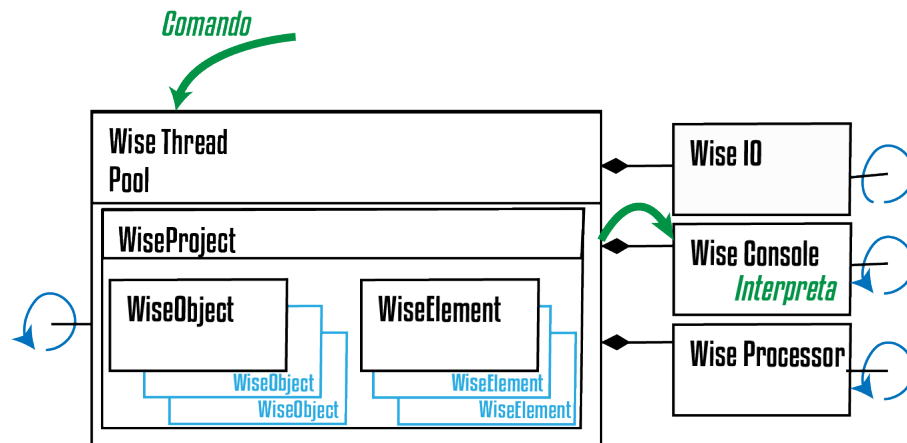


Figura 3.19: Modelo de Threads ao receber uma linha de comando da interface de usuário.

Quando se trata de um comando de escrita ou leitura, a thread *WiseIO* é utilizada, ao receber esse tipo de comando o console inteligente irá listar o trabalho no gerenciador de threads *WiseThreadPool*. O gerenciador irá balancear as requisições entre as threads, em seguida o gerenciador de threads irá aguardar uma mensagem indicando o final da execução do trabalho. Quando um objeto inteligente *WiseObject* é iterado um elemento inteligente é criado na estrutura *Freezer*, ao ser criado o objeto inteligente irá requisitar através de um trabalho inteligente que esse elemento seja armazenado. Portanto a thread *WiseIO* é muito utilizada no ciclo de vida de um elemento inteligente, gerenciando o processo de armazenamento e reconstrução do elemento. É esta estrutura que efetivamente aquece e resfria elementos inteligentes.

Os trabalhos de iteração funcionam da mesma forma que as de leitura ou escrita, entretanto são enviadas a threads *WiseProcessor*. Essas threads não possuem lógicas muito complexas, elas são responsáveis apenas por gerenciar os objetos e chamar os métodos equivalentes, por este motivo os objetos inteligentes e elementos inteligentes possuem métodos abstratos e seguem esses conjuntos de regras, para que uma estrutura que desconheça o seu funcionamento ou suas estruturas internas seja capaz de executar métodos padrões.

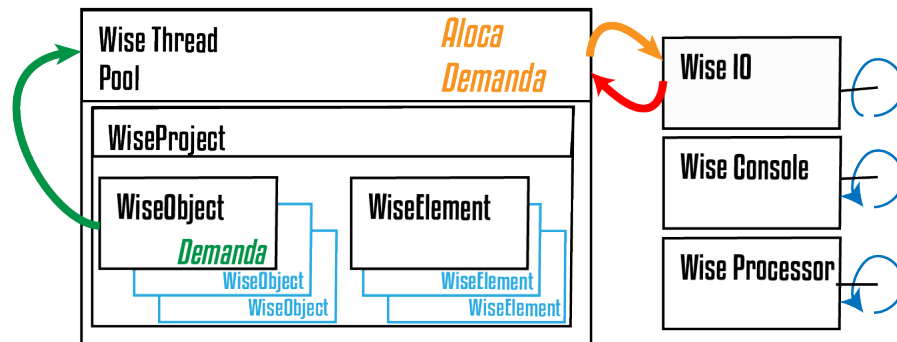


Figura 3.20: Modelo de Threads ao receber um comando de escrita/leitura.

Mesmo com a arquitetura de threads inclusa na ferramenta computacional ainda é possível que ela seja executada em plataformas de thread única, neste caso mais threads inteligentes não farão diferença. Pois apesar dos trabalhos ainda serem divididos em threads diferentes, elas serão executadas no mesmo núcleo de processamento. As threads inteligentes têm um impacto maior nas atividades de escrita e leitura, isto porque o objeto inteligente está dividido em várias estruturas menores que são armazenadas e acessadas constantemente, principalmente no caso de uma animação gráfica.

## 3.2 INTERFACE DE USUÁRIO

Nesta seção apresentam-se detalhes da interface escolhida para facilitar o uso da ferramenta computacional. A interface de usuário foi concebida para permitir o controle de todas as estruturas descritas na Seção 3.1 com todos os seus recursos gráficos extraídos. A interface se divide em duas partes principais: Um console, que permite uma interação textual e acesso as mesmas funcionalidades de iteração; E uma janela com elementos gráficos capazes de exibir os resultados gráficos obtidos.

Ambas as interfaces foram construídas para manipular objetos inteligentes e suas estruturas, por isto ambas possuem instâncias do gerenciador de threads inteligentes descritos na Seção 3.1. O console se trata de um envio direto de mensagens para a thread *WiseConsole*, que é efetivamente o console. Nas próximas seções os diferentes ambientes da

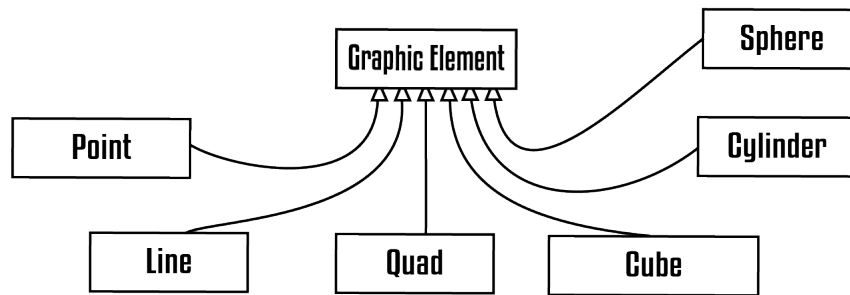


Figura 3.21: Interface de usuário gráfica.

ferramenta computacional serão descritos. Cada ambiente representa um projeto *Qt/C++* distinto.

### 3.3 CONSOLE

O ambiente de console da ferramenta computacional consiste em um projeto *Qt/C++*. O projeto foi intitulado *InGU* ou **I**terador **n**ão-**G**ráfico **U**niversal, porque o ambiente apesar de não conter os elementos para a visualização dos elementos gráficos, ele é capaz de criar as estruturas gráficas para visualização futura. No console as bibliotecas gráficas do OpenGL não foram incluídas no processo de compilação, portanto ambientes sem recursos gráficos são capazes de compilar e executar as ferramentas computacionais.

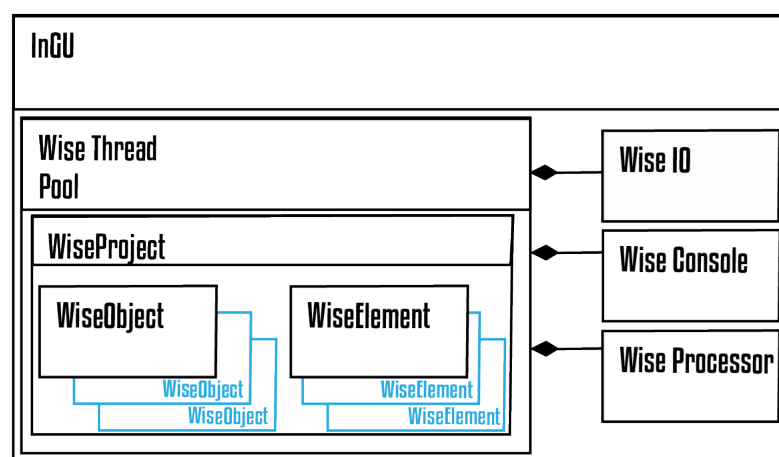


Figura 3.22: Estrutura do projeto que compõe o ambiente computacional InGU.

No anexo estão as sequências de instruções necessárias para compilar ambas as partes da ferramenta computacional. Ao ser compilado, um executável InGU contendo o projeto console é gerado, o processo de compilação da ferramenta é multiplataforma, tendo sido testado nos sistemas operacionais *Windows* e *Linux*. Entretanto, recomenda-se a utilização da ferramenta computacional no *Linux*. Ao ser executado, o console corresponde do sistema operacional irá ser aberto com o cabeçalho da ferramenta e aguardará entrada de texto.

```

. /=====
. /====/      IGU (Iterador Gráfico Universal)      /=====
. /====/      (or, Universal Graphic Iterator)      /=====
. /=====
. /1.1=====
. /=====
. segunda-feira 09/08/2021 19:19:08 313
. /=====
. [WISE CONSOLE] LOAD invoked
. <p1:WISE_PROJECT> 'WISE_ID [0] (NAME: p1) WISE PROJECT CREATED'
. <el1:WISE_ELEMENT> 'WISE_ID [0] (NAME: el1) WISE ELEMENT CREATED'
. <el1:WISE_OBJECT> 'WISE_ID [0] (NAME: obj1) WISE OBJECT CREATED'
. <el1:WISE_ELEMENT> 'WISE_ID [1] (NAME: el1) WISE ELEMENT CREATED'
. <el1:WISE_ELEMENT> 'WISE_ID [2] (NAME: el1) WISE ELEMENT CREATED'
. <el1:WISE_ELEMENT> 'WISE_ID [3] (NAME: el1) WISE ELEMENT CREATED'
. <el1:WISE_ELEMENT> 'WISE_ID [4] (NAME: el1) WISE ELEMENT CREATED'
. <el1:WISE_ELEMENT> 'WISE_ID [5] (NAME: el1) WISE ELEMENT CREATED'
. <el1:WISE_ELEMENT> 'WISE_ID [6] (NAME: el1) WISE ELEMENT CREATED'
. <el1:WISE_ELEMENT> 'WISE_ID [7] (NAME: el1) WISE ELEMENT CREATED'
. <el1:WISE_ELEMENT> 'WISE_ID [8] (NAME: el1) WISE ELEMENT CREATED'
. <el1:WISE_ELEMENT> 'WISE_ID [9] (NAME: el1) WISE ELEMENT CREATED'
. <el1:WISE_ELEMENT> 'WISE_ID [10] (NAME: el1) WISE ELEMENT CREATED'
. <el1:WISE_ELEMENT> 'WISE_ID [11] (NAME: el1) WISE ELEMENT CREATED'
. <el1:WISE_ELEMENT> 'WISE_ID [12] (NAME: el1) WISE ELEMENT CREATED'
. <el1:WISE_ELEMENT> 'WISE_ID [13] (NAME: el1) WISE ELEMENT CREATED'

```

Figura 3.23: Captura de tela com a execução do ambiente computacional InGU em um console.

Para executar um comando, basta inserir uma linha de texto e apertar a tecla *Enter*. Ao capturar a linha de texto, o programa de console irá na verdade redirecionar o comando para a estrutura *WiseThreadPool*, que por sua vez irá alocar uma thread do tipo *WiseConsole* para interpretar a mensagem. Este comportamento é o mesmo apresentado na Seção 3.1.7. Uma lista de comandos foi disponibilizada e está acessível através do comando *help*. Ao enviar este comando para o console uma lista com todas as possíveis entradas será exibida. Nas próximas seções estes comandos, suas entradas, seu escopo e thread responsável serão descritos em profundidade. A thread responsável indica qual thread irá executar a tarefa.

### 3.3.1 AJUDA

O comando de ajuda é o primeiro comando da interface e foi feito para listar todas as entradas possíveis do programa. Ao receber este comando a thread *WiseConsole* envia o texto

pré-definido com todos os comandos.

<b>Linha de Comando</b>	help
<b>Escopo</b>	nenhum
<b>Thread Responsável</b>	WiseConsole
<b>Entrada</b>	Nenhuma

Tabela 3.1: Descrição do comando ajuda.

Ao executar o comando, o usuário receberá uma lista de comandos divididos em escopos específicos. Os escopos foram criados para agrupar comandos por área de atuação, comandos auxiliares, como o comando de ajuda, são os comandos que não alteram as estruturas e exibem informações auxiliares ao usuário.

### 3.3.2 LER ARQUIVO DE ENTRADA

O comando ler arquivo de comandos irá receber o endereço de um arquivo local, este arquivo deve conter um comando por linha, um exemplo pode ser encontrado no Anexo B. Para este tipo de comando um *WiseJob* contendo a linha de comando e qual a sequência de trabalhos que deve ser executada. Por se tratar de uma leitura de arquivo de comando, a thread *WiseIO* será responsável por ler o arquivo e criar os trabalhos com os comandos subsequentes. Estes novos comandos irão passar novamente pela interpretação de uma thread do tipo *WiseConsole* e em seguida executados.

<b>Linha de Comando</b>	read cmds <file>	
<b>Escopo</b>	READ	
<b>Thread Responsável</b>	WiseIO	
<b>Entrada</b>	<file>	Caminho para arquivo contendo sequência de comandos.

Tabela 3.2: Descrição do comando ler arquivo de comando.

### 3.3.3 BATERIA DE TESTES

O ciclo principal de testes da ferramenta se baseia em verificar a consistência de todas as fábricas e do fluxo principal da ferramenta computacional, exceto detalhes gráficos e do



```

. /=====
. /===/          AJUDA          /=====
. /=====
. |help.....
. |      Lista todos os comandos da plataforma.
. |SCOPE READ.....
. |read cmds <file>.....
. |      <file> Caminho para arquivo de entrada (XML ou VTK).
. |      Lê todas as linhas do arquivo e executa a sequência de comandos.
. |SCOPE TEST.....
. |test.....
. |      Realiza todas as baterias de teste.
. |test.list.....
. |      Realiza todas as baterias de teste.
. |test.<test_id>.....
. |      Realiza um teste específico.
. |test file <test_id> <file1> <file2>.....
. |      Testa a equidade de dois arquivos.
. |.....
. |SCOPE PROJECT.....
. |project.....
. |project create <project_name>
. |      <project_name> Nome do projeto à ser criado.
. |      Cria projeto.
. |project use <project_name>
. |      <project_name> Nome do projeto à ser selecionado.
. |      Seleciona projeto para execução dos próximos comandos.
. |project list
. |      Lista os projetos disponíveis.
. |project print
. |      Imprime o projeto selecionado.
. |project delete
. |      Deleta o projeto selecionado.
. |project save <filename>
. |      <filename> Arquivo de saída aonde o projeto será salvo.
. |      Salva projeto em um arquivo XML.
. |project load <filename>
. |      <filename> Arquivo de entrada de onde o projeto será reconstruído.
. |      Carrega um arquivo XML.
. |.....
. |SCOPE ELEMENT.....
. |element.....
. |element create <type> <example> <name> [ARGS]
. |      <type> Tipo de elemento à ser criado.
. |      <example> Nome do exemplo à ser utilizado.
. |      <name> Nome do elemento à ser criado.
. |      [ARGS] Argumentos extras do método de criação específico.
. |      Cria elemento inteligente à partir de um exemplo da fábrica.
. |element clone <element_name> <name>
. |      <element_name> Nome do elemento à ser clonado.
. |      <name> Nome do elemento clone.
. |      Clona um elemento inteligente.
. |element list
. |      Lista os elementos do projeto selecionado.

```

Figura 3.24: Captura de tela com a execução do ambiente computacional InGU em um console.

processo de iteração. Ao executar a bateria de testes todos os elementos inteligentes *WiseElement* e objetos inteligentes *WiseObject* disponíveis serão testados individualmente. Para isto, uma fábrica de projeto *WiseProjectFactory* irá ser encarregada de criar todos os elementos e objetos inteligentes disponíveis.

Estes testes foram utilizados principalmente no momento do desenvolvimento da ferramenta e garantem que as fábricas de elementos e objetos inteligentes estão funcionando corretamente. Futuramente, caso novas estruturas sejam incluídas, estes testes garantem que todas as classes do projeto foram projetadas corretamente, com isso todas as estruturas que são carregadas pela ferramenta não perdem informação ao serem armazenadas e recuperadas, processo recorrente na ferramenta computacional. Os comandos deste tipo são de responsabilidade da thread *WiseConsole*, que irá criar e executar subcomando, trabalhos *WiseJob* relacionados à uma atividade superior.

<b>Linha de Comando</b>	test
<b>Escopo</b>	TEST
<b>Thread Responsável</b>	WiseConsole
<b>Entrada</b>	Nenhuma

Tabela 3.3: Descrição do comando bateria de testes.

### 3.3.4 LISTA DE TESTES

O comando de listar testes irá enumerar todos os teste possíveis de serem executados pela ferramenta computacional. Assim como o comando de ajuda, este comando irá imprimir no console todas os resultados encontrados.

<b>Linha de Comando</b>	test list
<b>Escopo</b>	TEST
<b>Thread Responsável</b>	WiseConsole
<b>Entrada</b>	Nenhuma

Tabela 3.4: Descrição do comando listar testes.

### 3.3.5 CASO DE TESTE

Como visto na Seção 3.3.4, os testes são enumerados. Baseando-se nessa lista, é possível selecionar um teste pelo seu número correspondente. Os casos de testes irão gerar uma sequência de comandos à serem interpretados pelo *WiseConsole*, os testes são compostos de comandos que irão criar e deletar estruturas que são finalizados na própria thread *WiseConsole*, entretanto estas estruturas são enviadas para um arquivo externo e então lidas. Estes subcomandos irão ser executados pela thread *WiseIO*. O caso de teste só é dado como concludido quando todos os subcomandos terminam sua execução.

Linha de Comando	test <test_id>	
Escopo	TEST	
Thread Responsável	WiseConsole	
Entrada	<test_id>	Número do teste a ser executado

Tabela 3.5: Descrição do comando executar caso de teste.

### 3.3.6 TESTAR EQUIDADE DE ARQUIVOS

Outro comando de teste disponibilizado pela ferramenta computacional é o teste de equidade de arquivos. Este comando utiliza uma chave numérica *test\_id* para salvar o resultado do teste, tanto o resultado quanto a quantidade de testes executados com a mesma chave são salvos. Por se tratar da leitura de dois arquivos diferentes esse comando é executado por uma thread do tipo *WiseIO*.

Linha de Comando	test file <test_id> <file1> <file2>	
Escopo	TEST	
Thread Responsável	WiseIO	
Entrada	<test_id> <file1> <file2>	Chave numérica para armazenar resultado Caminho para o primeiro arquivo Caminho para o segundo arquivo

Tabela 3.6: Descrição do comando para testar a equidade de dois arquivos.

### 3.3.7 CRIAR PROJETO

Para criar um projeto inteligente vazio este comando é utilizado, como entrada ele recebe o nome do projeto. Este comando irá utilizar a fábrica de projetos inteligentes para criar a estrutura vazia do projeto. Ao receber um comando deste a thread *WiseConsole* irá retornar um projeto em branco para ser acoplado à *WiseThreadPool*, isto é feito para que outras threads possam receber o mesmo projeto de forma separada.

<b>Linha de Comando</b>	project create <name>	
<b>Escopo</b>	PROJECT	
<b>Thread Responsável</b>	WiseConsole	
<b>Entrada</b>	<name>	Nome do projeto a ser criado

Tabela 3.7: Descrição do comando para criar projetos.

### 3.3.8 USAR PROJETO

Para que a maioria dos comandos funcione é necessário que um projeto esteja selecionado, pois a estrutura do projeto que disponibiliza elementos e objetos inteligentes. Uma vez que projetos tenham sido criados eles podem ser selecionados utilizando o comando de usar projetos

<b>Linha de Comando</b>	project use <project_name>	
<b>Escopo</b>	PROJECT	
<b>Thread Responsável</b>	WiseConsole	
<b>Entrada</b>	<project_name>	Nome do projeto a ser selecionado

Tabela 3.8: Descrição do comando para selecionar projetos.

### 3.3.9 LISTAR PROJETOS

É possível verificar todos os projetos do ambiente utilizando o comando de listar projetos. Todos os projetos carregados no momento da execução serão exibidos na listagem.

<b>Linha de Comando</b>	project list
<b>Escopo</b>	PROJECT
<b>Thread Responsável</b>	WiseConsole
<b>Entrada</b>	Nenhuma

Tabela 3.9: Descrição do comando listar projetos.

### 3.3.10 IMPRIMIR PROJETO

O comando de imprimir projetos irá extrair do projeto selecionado sua representação em um arquivo *XML*, este arquivo será impresso como resultado no console. Caso o projeto possua elementos e objetos inteligentes eles também serão impressos na estrutura de saída.

<b>Linha de Comando</b>	project print
<b>Escopo</b>	PROJECT
<b>Thread Responsável</b>	WiseConsole
<b>Entrada</b>	Nenhuma

Tabela 3.10: Descrição do comando imprimir projetos.

### 3.3.11 EXCLUIR PROJETO

Com um projeto selecionado, o comando de exclusão de elementos irá excluir o elemento inteligente do projeto recebendo o nome do elemento como parâmetro de entrada. Este comando será interpretado pela thread *WiseConsole*, que notifica o gerenciador de threads *WiseThreadPool*. O gerenciador só irá aguardar até que o projeto não possua trabalhos em execução para que posso excluí-lo.

<b>Linha de Comando</b>	project delete
<b>Escopo</b>	PROJECT
<b>Thread Responsável</b>	WiseConsole
<b>Entrada</b>	Nenhuma

Tabela 3.11: Descrição do comando excluir projetos.

### 3.3.12 SALVAR PROJETO

O comando de salvar projetos irá extrair do projeto selecionado sua representação em um arquivo *XML*, assim como no processo de impressão do projeto. Este arquivo será impresso em um arquivo externo e por isto é executado pela thread *WiseIO*.

<b>Linha de Comando</b>	project save <file_name>	
<b>Escopo</b>	PROJECT	
<b>Thread Responsável</b>	WiseIO	
<b>Entrada</b>	<file_name>	Caminho do arquivo de saída.

Tabela 3.12: Descrição do comando para salvar projetos.

### 3.3.13 CARREGAR PROJETO

O comando de carregar projetos irá extrair de um arquivo de entrada no formato *XML* a estrutura de um projeto inteligente. A ferramenta computacional irá utilizar os arquivos de entrada diretamente nas estruturas de Fábrica relacionadas. Caso o projeto possua elementos e objetos inteligentes eles também serão reconstruídos no projeto reconstruído.

<b>Linha de Comando</b>	project load <filename>	
<b>Escopo</b>	PROJECT	
<b>Thread Responsável</b>	WiseIO	
<b>Entrada</b>	<filename>	Caminho do arquivo de entrada.

Tabela 3.13: Descrição do comando para carregar projetos.

### 3.3.14 CRIAR ELEMENTO

As fábricas de elementos inteligentes descritas na Seção ?? são acessadas pelo comando de criar elementos. Este comando recebe como entrada o tipo de elemento inteligente, o nome e o exemplo à ser utilizado. Cada fábrica de elementos possui uma lista de exemplos disponíveis, essa lista é visualizada através do comando de listar exemplos de uma fábrica de elementos.

Linha de Comando	element create <type> <example> <name> [ARGS]	
Escopo	ELEMENT	
Thread Responsável	WiseConsole	
Entrada	<type> <example> <name> [ARGS]	Tipo de elemento à ser criado, seleciona a fábrica de elemento à ser utilizada. Caminho para o primeiro arquivo. Caminho para o segundo arquivo. Individualmente, as fábricas podem receber parâmetros para a criação de elementos.

Tabela 3.14: Descrição do comando para criar.

### 3.3.15 CLONAR ELEMENTO

Assim como o comando de criação de elementos descrito na Seção 3.3.14, a clonagem de elementos também irá acessar as fábricas de elementos. Ao clonar um elemento o seu tipo é selecionado juntamente com a fábrica correspondente, uma vez selecionados a fábrica receberá como parâmetro o elemento inteligente e o nome do novo elemento à ser criado. Portanto o comando de clonagem de elementos poderá ser utilizando com a entrada do nome do elemento inteligente à ser clonado e o nome do novo elemento.

Linha de Comando	element clone <element_name> <name>	
Escopo	ELEMENT	
Thread Responsável	WiseConsole	
Entrada	<element_name> <name>	Nome do elemento à ser clonado. Nome do novo elemento.

Tabela 3.15: Descrição do comando para clonar elementos inteligentes.

### 3.3.16 LISTAR ELEMENTOS

Com um projeto já selecionada, o comando de listar elementos pode ser utilizado para que o console imprima uma lista com o nome de todos os elementos inteligentes presentes no projeto.

<b>Linha de Comando</b>	element list
<b>Escopo</b>	ELEMENT
<b>Thread Responsável</b>	WiseConsole
<b>Entrada</b>	Nenhuma

Tabela 3.16: Descrição do comando listar elementos.

### 3.3.17 IMPRIMIR ELEMENTO

O comando de imprimir elementos irá extrair do projeto selecionado a representação em um arquivo *XML*, a informação textual deste arquivo será impressa como resultado no console. Para que o comando funcione é necessário que um projeto esteja selecionado e que o nome do elemento à ser impresso seja informado.

<b>Linha de Comando</b>	element print <element_name>	
<b>Escopo</b>	ELEMENT	
<b>Thread Responsável</b>	WiseConsole	
<b>Entrada</b>	<element_name>	Nome do elemento à ser impresso.

Tabela 3.17: Descrição do comando imprimir elementos inteligentes.

### 3.3.18 EXCLUIR ELEMENTO

O comando de excluir elementos irá remover os dados do elemento da memória.

<b>Linha de Comando</b>	element delete <element_name>	
<b>Escopo</b>	ELEMENT	
<b>Thread Responsável</b>	WiseConsole	
<b>Entrada</b>	<element_name>	Nome do elemento à ser excluído.

Tabela 3.18: Descrição do comando excluir projetos.

### 3.3.19 SALVAR ELEMENTO

Os elementos inteligentes podem ser exportados em dois formatos, um arquivo *XML* e um arquivo *VTK*. Ambos futuramente podem ser utilizados na reconstrução do elemento inteligente. Como entrada o comando receberá o nome do elemento, o tipo de arquivo à ser



escrito e o caminho onde ele deve ser salvo. A saída do comando será o arquivo determinado, à partir deste arquivo é possível reconstruir o elemento completo.

Linha de Comando	element save <element_name> <save_type> <filename>	
Escopo	ELEMENT	
Thread Responsável	WiseIO	
Entrada	<element_name> <save_type> <filename>	Nome do elemento inteligente à ser salvo. Tipo de arquivo à ser exportando, podendo ser <i>XML</i> ou <i>VTK</i> . Caminho para o arquivo à ser exportado.

Tabela 3.19: Descrição do comando excluir projetos.

### 3.3.20 CARREGAR ELEMENTO

O comando de carregar elementos irá extrair de um arquivo de entrada no formato *XML* ou *VTK* a estrutura de um elementos inteligente. A ferramenta computacional irá utilizar os arquivos de entrada diretamente na fábrica de elemento inteligente relacionada. Caso seja um arquivo *XML* não será necessário informar o tipo de elemento inteligente à ser construído. Caso contrário o tipo e o nome serão recebidos como parâmetros de entrada.

Linha de Comando	element load <load_type> element load <VTK> <filename> <type> <name> element load <XML> <filename>	
Escopo	ELEMENT	
Thread Responsável	WiseIO	
Entrada	<filename> <load_type> <type> <name>	Caminho para o arquivo de entrada. Tipo de arquivo à ser carregado, podendo ser <i>XML</i> ou <i>VTK</i> . Tipo de elemento inteligente à ser criado. Nome do elemento inteligente à ser criado.

Tabela 3.20: Descrição do comando para carregar elementos inteligentes.

### 3.3.21 LISTAR EXPORTAÇÕES DO ELEMENTO

Os elementos inteligentes podem ser exportados outros formatos de arquivo. Cada tipo de elemento inteligente possui uma lista de exportações disponíveis, um comando foi inserido

para acessar a lista de exportações de um certo objeto.

<b>Linha de Comando</b>	element export_list <element_name>	
<b>Escopo</b>	ELEMENT	
<b>Thread Responsável</b>	WiseConsole	
<b>Entrada</b>	<element_name>	Nome do elemento inteligente à analisado.

Tabela 3.21: Descrição do comando listar exportações de um elemento inteligente.

### 3.3.22 EXPORTAR ELEMENTO

A estrutura abstrata dos elementos inteligentes requerem que cada um deles possua o método de exportação, os métodos de exportação podem receber parâmetros diferentes. A principal exportação utilizada foi disponibilizada na classe *WiseGraphic*, este tipo de elemento possui implementada a exportação dos gráficos em arquivos de imagem. Desta forma, possibilitou-se a criação de imagens em um ambiente que não possui interface gráfica.

<b>Linha de Comando</b>	element export <element_name> <export_name> [ARGS]	
<b>Escopo</b>	ELEMENT	
<b>Thread Responsável</b>	WiseIO	
<b>Entrada</b>	<element_name> <export_name> [ARGS]	Nome do elemento à ser exportado. Tipo de exportação à ser realizada. Os elementos podem receber parâmetros para sua exportação.

Tabela 3.22: Descrição do comando para exportar elementos.

### 3.3.23 ESCALAR ELEMENTO

Os dados inseridos em um elemento inteligente podem ser escalados. As informações contidas em pontos, linhas, células e campos podem ser multiplicados por um escalar  $f$ . Isso permite modelos geométricos que possuam parâmetros em unidades fora do padrão possam ser escaladas. Utilizando o nome do elemento, o tipo e o nome do parâmetro à ser escalado e o valor do escalar  $f$  o comando de escalar elementos permite ao usuário dimensionar cada parâmetro individualmente.

Linha de Comando	element scale <element_name> <cell_type> <field> <scale>	
Escopo	ELEMENT	
Thread Responsável	WiseConsole	
<b>Entrada</b>	<type>	Tipo de elemento à ser criado, seleciona a fábrica de elemento à ser utilizada.
	<element_name>	Nome do elemento à ser exportado.
	<cell_type>	Tipo de parâmetro à ser escalado (Ponto,Célula,Linha e Campos).
	<field>	Nome do campo à ser escalado.
	<scale>	Escalar $f$ utilizado ao escalar elemento.

Tabela 3.23: Descrição do comando para escalar elementos.

### 3.3.24 DEFINIR PARÂMETRO DE ELEMENTO

Os dados inseridos em um elemento inteligente podem ser alterados e receber um valor. As informações contidas em pontos, linhas, células e campos podem ser substituídas por um valor de entrada. Isso permite modelos geométricos tenham seus parâmetros editados. Para alterar o parâmetro de um elemento inteligente é necessário adicionar à linha de comando o nome do elemento, o tipo e nome do parâmetro à ser alterado, a posição do valor no vetor do parâmetro e o valor à ser inserido.

Linha de Comando	element set_field <element_name> <cell_type> <field_name> <id> <data>	
Escopo	ELEMENT	
Thread Responsável	WiseConsole	
<b>Entrada</b>	<type>	Tipo de elemento à ser criado, seleciona a fábrica de elemento à ser utilizada.
	<element_name>	Nome do elemento à ser definido.
	<cell_type>	Tipo de parâmetro à ser escalado (Ponto,Célula,Linha ou Campo).
	<field_name>	Nome do campo à ser escalado.
	<id>	Posição do valor no vetor do parâmetro, 0 caso valor único.
	<data>	Valor à ser inserido como parâmetro do elemento inteligente.

Tabela 3.24: Descrição do comando para definir parâmetros de elementos.

### 3.3.25 DEFINIR TODOS OS PARÂMETROS DE ELEMENTO

O comando de definição de todos os valores pertencentes à um parâmetro auxilia no manuseio de parâmetros que possuem múltiplos valores. O comando define todas as posições

encontradas no vetor do parâmetro e os altera para o valor de entrada. Para alterar todos os valores de um determinado parâmetro de um elemento inteligente é necessário adicionar à linha de comando o nome do elemento, o tipo e nome do parâmetro à ser alterado e o valor à ser inserido.

<b>Linha de Comando</b>	element set_all_field <element_name> <cell_type> <field_name> <data>	
<b>Escopo</b>	ELEMENT	
<b>Thread Responsável</b>	WiseConsole	
<b>Entrada</b>	<type> <element_name> <cell_type> <field_name> <data>	Tipo de elemento à ser criado, seleciona a fábrica de elemento à ser utilizada. Nome do elemento à ser exportado. Tipo de parâmetro à ser escalado (POINT,LINE,CELL ou FIELD). Nome do campo à ser escalado. Valor à ser inserido como parâmetro do elemento inteligente.

Tabela 3.25: Descrição do comando para definir parâmetros de elementos.

### 3.3.26 LISTAR FÁBRICAS DE ELEMENTO

Para listar todas as fábricas de elemento inteligente *WiseElementFactory* o comando desta seção foi criada. O resultado impresso ao executar este comando é a lista de fábricas disponíveis na classe *WiseElementFactories* acoplada na ferramenta computacional.

Consequentemente os resultados impressos por este comando representam os tipos de elementos inteligentes suportados pela ferramenta computacional.

<b>Linha de Comando</b>	element factories list
<b>Escopo</b>	ELEMENT
<b>Thread Responsável</b>	WiseConsole
<b>Entrada</b>	Nenhuma

Tabela 3.26: Descrição do comando listar fábricas de elemento.

### 3.3.27 LISTAR EXEMPLOS DISPONÍVEIS DE ELEMENTO

Para cada tipo de elemento inteligente, ou fábrica inteligente listada pelo comando descrito na Seção 3.3.27, existirá uma lista de exemplos disponíveis. Exemplos são elementos inteligentes pré-definidos que podem ser criados a partir do comando de criar elementos descrito na Seção 3.3.14. Para listar os exemplos disponíveis o comando recebe como parâmetro de entrada a fábrica à ser analisada.

<b>Linha de Comando</b>	element factories examples <factory>	
<b>Escopo</b>	ELEMENT	
<b>Thread Responsável</b>	WiseConsole	
<b>Entrada</b>	<factory>	Nome da fábrica de elementos inteligente à analisada.

Tabela 3.27: Descrição do comando listar exemplos contidos em determinada fábrica de elemento.

### 3.3.28 CRIAR OBJETO

Assim como o comando de criação de elementos inteligentes, o comando de criação de objetos irá acessar a fábrica de elementos inteligentes *WiseElementFactory*. É possível criar objetos inteligentes de duas formas: A primeira, utilizando um elemento inteligente; A segunda utilizando os exemplos disponibilizados pela fábrica de elementos inteligentes.

Assim como descrito na Seção 3.1, ao criar um objeto inteligente, um elemento inteligente é adicionado à estrutura do *Forno*, enquanto um Clone é acoplado ao *Freezer*;

<b>Linha de Comando</b>	object create <object_name> <element_name> object create <type> <example> <name> <element_name> [ARGS]	
<b>Escopo</b>	OBJECT	
<b>Thread Responsável</b>	WiseConsole	
<b>Entrada</b>	<object_name> <element_name> <type> <name> [ARGS]	Nome do objeto à ser criado. Nome do elemento à ser utilizado na criação ou do elemento à ser criado a partir do exemplo. Tipo de elemento inteligente à ser criado. Nome do exemplo de elemento inteligente à ser criado. Individualmente, as fábricas podem receber parâmetros para a criação de elementos.

Tabela 3.28: Descrição do comando para criar objetos inteligentes.

### 3.3.29 CLONAR OBJETO

Assim como o elemento inteligente, o objeto inteligente pode ser clonado em uma fábrica própria. Através do comando de clonar objetos estes métodos podem ser acessados, basta inserir o nome do objeto à ser clonado. A fábrica correta é acessada verificando o tipo do objeto à ser clonado.

<b>Linha de Comando</b>	object clonet <object_name> <name>	
<b>Escopo</b>	OBJECT	
<b>Thread Responsável</b>	WiseConsole	
<b>Entrada</b>	<object_name> <name>	Nome do objeto à ser clonado. Nome do novo elemento.

Tabela 3.29: Descrição do comando para clonar objetos inteligentes.

### 3.3.30 LISTAR OBJETOS

Com um projeto já selecionada, o comando de listar objetos pode ser utilizado para que o console imprima uma lista com o nome de todos os objetos inteligentes presentes no projeto.

<b>Linha de Comando</b>	object list
<b>Escopo</b>	OBJECT
<b>Thread Responsável</b>	WiseConsole
<b>Entrada</b>	Nenhuma

Tabela 3.30: Descrição do comando listar elementos.

### 3.3.31 IMPRIMIR OBJETO

O comando de imprimir elementos irá extrair do projeto selecionado a representação em um arquivo *XML*, a informação textual deste arquivo será impressa como resultado no console. Para que o comando funcione é necessário que um projeto esteja selecionado e que o nome do objeto à ser impresso seja informado.

Dentro deste arquivo *XML* estarão todas as estruturas descritas na Seção 3.1.3, todos os elementos inteligentes que compõe as estruturas do *Forno* e *Freezer* de detalhes do objeto inteligente, bem como seu tipo e fábricas utilizadas.

<b>Linha de Comando</b>	object print <object_name>	
<b>Escopo</b>	OBJECT	
<b>Thread Responsável</b>	WiseConsole	
<b>Entrada</b>	<object_name>	Nome do objeto à ser impresso.

Tabela 3.31: Descrição do comando imprimir objetos inteligentes.

### 3.3.32 EXCLUIR OBJETO

Com um projeto selecionado, o comando de exclusão de objetos irá excluir o objeto inteligente do projeto recebendo o nome do objeto como parâmetro de entrada.

<b>Linha de Comando</b>	object delete <object_name>	
<b>Escopo</b>	OBJECT	
<b>Thread Responsável</b>	WiseConsole	
<b>Entrada</b>	<object_name>	Nome do objeto à ser excluído.

Tabela 3.32: Descrição do comando excluir objetos inteligentes.

### 3.3.33 SALVAR OBJETO

O comando de salvar objetos possibilita que a estrutura complexa de um objeto inteligente, bem como seus componentes, sejam arquivados em um arquivo *XML*. Os objetos salvos podem em seguida ser recuperados. Como entrada o comando de salvar objetos irá receber o nome do objeto à ser salvo e o caminho para o arquivo de saída.

### 3.3.34 CARREGAR OBJETO

O comando de carregar objetos irá enviar um arquivo de entrada a fábrica de objetos inteligentes *WiseObjectFactory*, que por sua vez irá reconstruir cada componente do objeto com

<b>Linha de Comando</b>	object save <object_name> <filename>	
<b>Escopo</b>	OBJECT	
<b>Thread Responsável</b>	WiseIO	
<b>Entrada</b>	<object_name> <filename>	Nome do objeto à ser salvo. Caminho para o arquivo de saída.

Tabela 3.33: Descrição do comando salvar objetos inteligentes.

sua fábrica adequada. Isto significa que a coleção de elementos inteligentes e objetos gráficos irá reconstruir cada objeto. O comando recebe como entrada o nome do arquivo apenas, no Anexo C há um exemplo de arquivo *XML* contendo um objeto inteligente válido.

<b>Linha de Comando</b>	object load <filename>	
<b>Escopo</b>	OBJECT	
<b>Thread Responsável</b>	WiseIO	
<b>Entrada</b>	<filename>	Caminho para o arquivo de entrada.

Tabela 3.34: Descrição do comando carregar objetos inteligentes.

### 3.3.35 EXPORTAR OBJETO

O comando de exportar objetos funciona da mesma forma que o comando de exportar elementos, isto porque o comando irá exportar o elemento contido na estrutura do *Forno*. Portanto os comandos de listar exportações e o de exportar elemento foram adicionados ao objeto.

<b>Linha de Comando</b>	object export_list <object_name> object export <object_name> <export_type> [ARGS]	
<b>Escopo</b>	OBJECT	
<b>Thread Responsável</b>	WiseConsole	
<b>Entrada</b>	<object_name> <export_type> [ARGS]	Nome do objeto à ser analisado ou exportado. Tipo de exportação à ser realizada. Argumentos de entrada, específicos de cada exportação.

Tabela 3.35: Descrição dos comandos de exportação elementos contidos no *Forno* de objetos inteligentes.



### 3.3.36 DEFINIR PARÂMETROS DE OBJETO

Assim como o comando de exportar objeto atua sobre o elemento contido na estrutura do *Forno*, ao realizar comandos de definição de parâmetros num objeto inteligente o elemento inteligente contido na estrutura do *Forno* que será afetado. Portanto, caso algum ajuste seja feito no modelo geométrico as iterações passadas não sofrerão mudanças, enquanto qualquer iteração nova irá receber a nova informação definida.

Linha de Comando	object set_field <object_name> <cell_type> <field_name> <id> <data> object set_all_field <object_name> <cell_type> <field_name> <data>	
Escopo	OBJECT	
Thread Responsável	WiseConsole	
Entrada	<object_name> <cell_type> <field_name> <id> <data>	Nome do elemento à ser definido. Tipo de parâmetro à ser escalado (Ponto,Célula,Linha ou Campo). Nome do campo à ser escalado. Posição do valor no vetor do parâmetro, 0 caso valor único. Valor à ser inserido como parâmetro do elemento inteligente.

Tabela 3.36: Descrição dos comandos de exportação elementos contidos no *Forno* de objetos inteligentes.

### 3.3.37 SETAR OBJETO

O comando de setar um objeto inteligente tem ligação direta com o seu processo de iteração. Como mencionado na Seção 3.1, os objetos são criados por padrão no estado *Ready*, tendo sido corretamente criados e acoplados de fábricas de iteração e, opcionalmente, gráficas, ele pode ser setado. O comando para setar um objeto inteligente recebe apenas o nome do objeto inteligente e retorna uma mensagem de conclusão.

Linha de Comando	object set <object_name>	
Escopo	OBJECT	
Thread Responsável	WiseConsole	
Entrada	<object_name>	Nome do objeto à ser setado.

Tabela 3.37: Descrição do comando setar objetos inteligentes.

### 3.3.38 ITERAR OBJETO

O comando de iterar objetos irá executar o ciclo de iteração de um objeto que tenha sido corretamente setado. Os objetos inteligentes no estado *Set* podem ter seus parâmetros alterados e corretamente configurados, em seguida passam pelo ciclo de iteração. Ao terminarem o ciclo os objetos inteligentes mudam para o estado *Go*.

<b>Linha de Comando</b>	object go <object_name>	
<b>Escopo</b>	OBJECT	
<b>Thread Responsável</b>	WiseProcessor	
<b>Entrada</b>	<object_name>	Nome do objeto à ser iterado.

Tabela 3.38: Descrição do comando iterar objetos inteligentes.

### 3.3.39 LISTAR FÁBRICAS DE ITERAÇÃO DE OBJETO

Cada tipo de objeto inteligente possuirá uma lista de fábricas de iteração *WiseIterationFactory* que podem ser acopladas à ele. O comando de listar fábricas de iteração irá receber o nome do objeto inteligente à ser analisado, as fábricas de iteração que forem compatíveis com o objeto serão escritas no console.

<b>Linha de Comando</b>	object iteration_factories list <object_name>	
<b>Escopo</b>	OBJECT	
<b>Thread Responsável</b>	WiseConsole	
<b>Entrada</b>	<object_name>	Nome do objeto à ser analisado.

Tabela 3.39: Descrição do comando listar fábricas de iteração.

### 3.3.40 DEFINIR FÁBRICA DE ITERAÇÃO DE OBJETO

Antes que um objeto possa passar para o estado *Set* ele precisa ter em sua composição uma fábrica de iteração, caso deseje-se um objeto estático, que não mude durante o ciclo iterativo, a fábrica *StaticIterationFactory* deve ser acoplada. O comando para definir a fábrica de iteração irá receber o nome do objeto que irá receber a fábrica e o nome da fábrica à ser adicionada.

<b>Linha de Comando</b>	object iteration_factories set <object_name> <factory_name>	
<b>Escopo</b>	OBJECT	
<b>Thread Responsável</b>	WiseConsole	
<b>Entrada</b>	<object_name> <factory_name>	Nome do objeto inteligente que irá receber a fábrica. Fábrica de iteração à ser inserido no objeto.

Tabela 3.40: Descrição do comando definir fábricas gráficas.

### 3.3.41 LISTAR FÁBRICAS GRÁFICAS DE OBJETO

Além de fábricas de iteração, objetos inteligentes podem ser compostos por fábricas gráficas. O comando de listar fábricas gráficas irá imprimir no console as fábricas gráficas disponíveis para um determinado objeto inteligente, como dado de entrada o comando irá receber o nome deste objeto.

<b>Linha de Comando</b>	object graphic_factories list <object_name>	
<b>Escopo</b>	OBJECT	
<b>Thread Responsável</b>	WiseConsole	
<b>Entrada</b>	<object_name>	Nome do objeto à ser analisado.

Tabela 3.41: Descrição do comando listar fábricas gráficas.

### 3.3.42 DEFINIR FÁBRICA GRÁFICA DE OBJETO

Uma vez que o objeto inteligente tenha sido criado corretamente ele pode ser acoplado de fábricas de iteração ou gráficas. O comando de definir fábrica gráfica irá acoplar uma fábrica gráfica disponível à um objeto inteligente compatível. Como dados de entrada o comando receberá o nome do objeto inteligente à ser alterado e o nome da fábrica gráfica à ser adicionada.

### 3.3.43 LISTAR OBJETOS GRÁFICOS

Uma vez que o objeto gráfico tenha sido corretamente criado e acoplado à uma fábrica gráfica, o seu modelo é disponibilizado numa lista de objetos gráficos. Essa lista irá exibir o

<b>Linha de Comando</b>	object graphic_factories set <object_name> <factory_name>	
<b>Escopo</b>	OBJECT	
<b>Thread Responsável</b>	WiseConsole	
<b>Entrada</b>	<object_name> <factory_name>	Nome do objeto inteligente que irá receber a fábrica. Fábrica gráfica à ser inserido no objeto.

Tabela 3.42: Descrição do comando definir fábricas gráficas.

nome de todos os objetos que podem ser visualizados e está disponível para visualização através de um comando.

<b>Linha de Comando</b>	graphic list
<b>Escopo</b> OBJECT	
<b>Thread Responsável</b> WiseConsole	
<b>Entrada</b>	Nenhuma.

Tabela 3.43: Descrição do comando listar objetos gráficos.

### 3.3.44 LISTAR CANVAS

O comando de listar *Canvas* irá nomear todas os elementos gráficos disponíveis. Quando o ambiente *InGU* está sendo executado não existirão elementos gráficos disponíveis, entretanto como veremos à frente o ambiente computacional *IGU* apresentar exatamente as mesmas funcionalidades com elementos gráficos. Ao executar este comando no ambiente computacional com elementos gráficos, cada objeto e seu nome será enviado como resultado.

<b>Linha de Comando</b>	canvas list
<b>Escopo</b> GRAPHIC	
<b>Thread Responsável</b> WiseConsole	
<b>Entrada</b>	Nenhuma.

Tabela 3.44: Descrição do comando listar elementos gráficos *Canvas*.

### 3.3.45 LINK GRÁFICO

Tendo o conhecimento dos elementos gráficos disponíveis para desenho e das estruturas aptas à serem desenhadas, é possível que estas estruturas sejam conectadas para que funcionem em conjunto. Ao realizar esta conexão, nomeada de link gráfico, o elemento gráfico passará a exibir o elemento gráfico.

Linha de Comando	graphic link <graphic_name> <canvas_name> canvas link <canvas_name> <graphic_name>	
Escopo	GRAPHIC	
Thread Responsável	WiseConsole	
Entrada	<graphic_name> <canvas_name>	Nome do objeto gráfico à ser linkado com elemento gráfico <i>Canvas</i> . Nome do <i>Canvas</i> à receber objeto gráfico para desenho.

Tabela 3.45: Descrição dos comandos que enviam um objeto gráfico para ser exibido em um elemento gráfico da interface de usuário.

### 3.3.46 TERMINAR LINK GRÁFICO

Uma vez que os elementos gráficos começam a ser exibidos em elementos de interface gráfica eles são desenhados continuamente. Para liberar o elemento de interface gráfica e deixá-lo vazio o comando de terminar link gráfico foi utilizado. Como dado de entrada este comando recebe o nome do elemento de interface gráfica.

Linha de Comando	canvas purge <canvas_name>	
Escopo	GRAPHIC	
Thread Responsável	WiseConsole	
Entrada	<canvas_name>	Canvas à ser limpo.

Tabela 3.46: Descrição do comando terminar link gráfico.

## 3.4 JANELA

]

O ambiente com interface gráfica foi nomeado de **Iterador Gráfico Universal**, que também está presente como parte de um projeto *Qt/C++*. Diferentemente dos outro ambiente, este possui elementos gráficos. O principal elemento gráfico disponibilizado pela interface gráfico é a janela principal, que é um objeto do tipo *QMainWindow*. Estes objetos são disponibilizados pela biblioteca *Qt* e permitem que um projeto *C++* possua uma interface gráfica utilizando diretivas OpenGL e GLUT (The OpenGL Utility Kit).

## 4 RESULTADOS NUMÉRICOS E DISCUSSÕES

- IGOR: já fiz uma primeira revisão deste capítulo e indico em vermelho o que deve fazer para melhorar este capítulo. Tudo que acrescentar ou mudar coloque em azul.
- IGOR: não está aparecendo a numeração das Figuras no caption. Algo de errado está acontecendo no modelo que está utilizando.
- IGOR: está errada a citação do trabalho do método de Duan e Zamir (1995), não é a citação que colocou [6].OK

Nesta seção, apresentam-se resultados obtidos com a implementação computacional e simulação do modelo matemático de Duan e Zamir [7]. As simulações realizadas aqui tratam da propagação de uma onda harmônica simples ao longo de uma árvore, onde reflexões de onda modificam a amplitude da onda de pressão enquanto ela avança. A escolha de uma onda harmônica simples neste estudo possibilita investigar os efeitos da frequência, fluido viscoso e viscoelasticidade da parede do segmento de vaso.

Considerou-se neste estudo um modelo de árvore arterial canina como ilustrado na Figura 4.1. As propriedades dos segmentos foram escolhidas oriundas dos dados de Fung [5] e são descritas na Tabela 4.1.

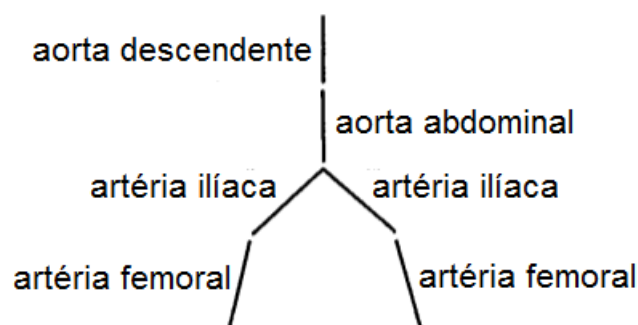


Figura 4.1: Representação do modelo de árvore arterial canina (figura adaptada de [6]).

Nas simulações aqui realizadas, calculou-se a distribuição de amplitude de pressão ao longo da árvore arterial (Figura 4.1). Os resultados foram obtidos para quatro diferentes frequências e três diferentes cenários de escoamento/segmento: (i) escoamento viscoso em

Tabela 4.1: Propriedades dos segmentos do modelo de árvore arterial [6, 5]

Artéria	Comprimento (cm)	Densidade $\rho$ (g/cm <sup>3</sup> )	Viscosidade $\mu_0$ (g/cms)	Diâmetro (cm)	Módulo de Young (dyn/cm <sup>2</sup> )
Aorta Descendente	25	0,960	0,0385	1,3	$4,8 \times 10^6$
Aorta Abdominal	11	1,134	0,0449	0,9	$1,0 \times 10^7$
Íliaca	12	1,172	0,0472	0,6	$1,0 \times 10^7$
Femoral	10	1,235	0,0494	0,4	$1,0 \times 10^7$

segmento puramente elástico (cenário 1 da Seção 2.1.4), (ii) escoamento invíscido em segmento viscoelástico (cenário 2) e (iii) escoamento viscoso em segmento viscoelástico (cenário 3).

Os resultados obtidos nas simulações são mostrados nas Figuras 4.2, 4.3, 4.4, 4.5, 4.7 envolvendo a amplitude da pressão ao longo do modelo de árvore arterial. Nestas figuras, o comprimento de cada segmento arterial foi dimensionado para 1,0, de modo que o comprimento adimensional total da árvore é 4,0. O comprimento real é 58 cm. A amplitude da pressão também foi escalada pela pressão de entrada  $P_o$ , e os resultados finais são portanto mostrados em termos de amplitude de pressão adimensional  $|P|$  versus a distância adimensional  $X$  do início da árvore.

Nas Figuras 4.2 e 4.3, o efeito da viscosidade do fluido é examinado separadamente considerando-se o escoamento em segmentos puramente elásticos com quatro valores diferentes de viscosidade do fluido, ou seja,  $\mu = 0; 0,5\mu_0; 1,0\mu_0$  e  $1,5\mu_0$ , onde  $\mu_0$  é o valor base da viscosidade da Tabela 4.1. Observa-se que o efeito da viscosidade do fluido é reduzir o aumento global na amplitude da onda de pressão causada pelas reflexões das ondas à medida que a onda se desloca na direção à jusante. Além disso, modera os picos locais na distribuição de pressão.

Nas Figuras 4.4, 4.5, o efeito da viscoelasticidade da parede do segmento de vaso é considerado separadamente considerando-se o escoamento invíscido e tomando-se quatro valores diferentes da viscoelasticidade da parede do segmento. O modelo viscoelástico proposto utilizado para fins destes cálculos é apresentado no cenário 2 da Seção 2.1.4, no qual a viscoelasticidade da parede do vaso é representada por um módulo de Young complexo. Estas figuras mostram os resultados para  $\phi_0 = 0^\circ, 4^\circ, 8^\circ$  e  $12^\circ$ . Quando  $\phi_0 = 0^\circ$  tem-se um



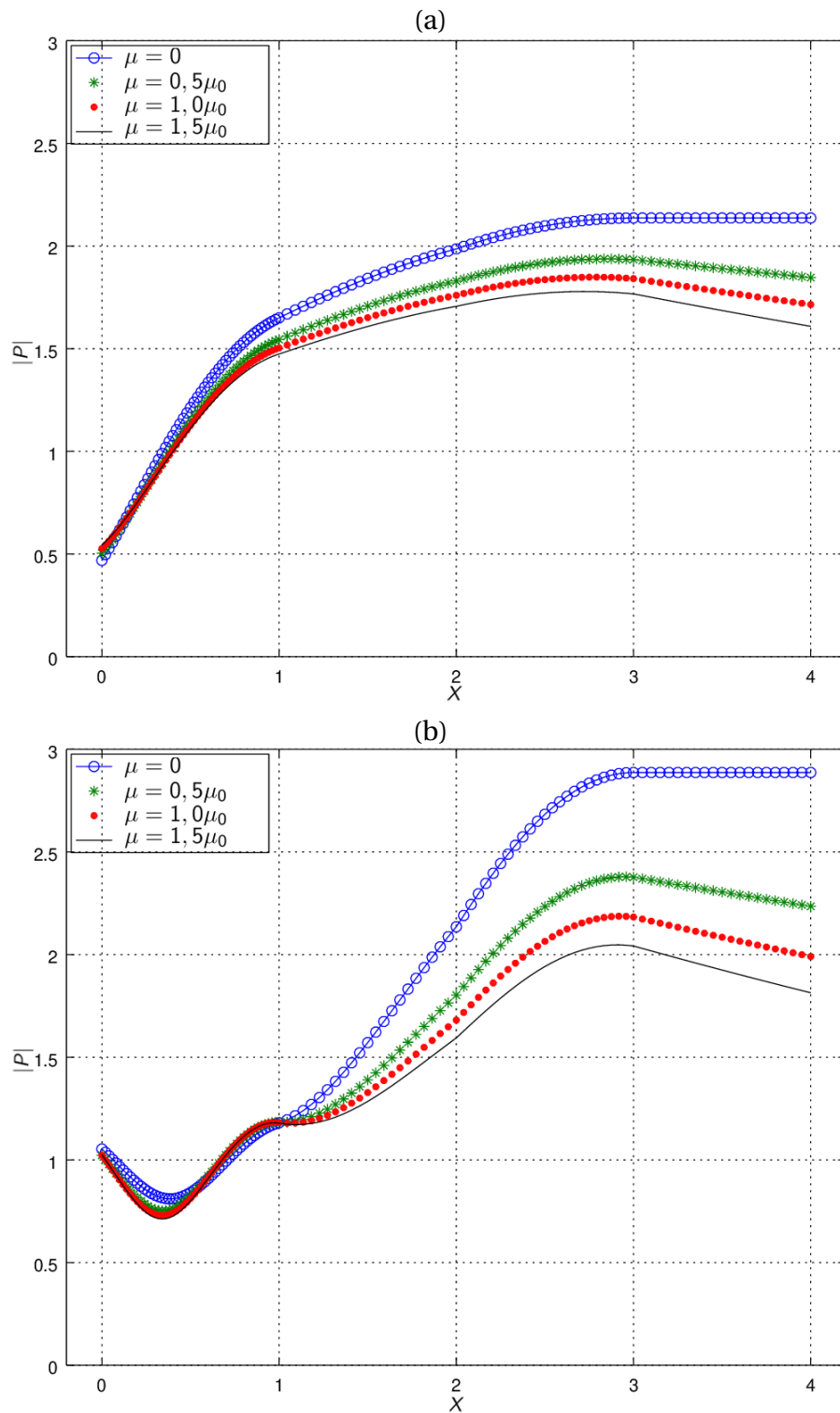


Figura 4.2: Amplitude da pressão  $|P|$  ao longo da árvore arterial considerando diferentes viscosidade do fluido  $\mu$  e frequências: (a)  $f = 3,65$  Hz, (b)  $f = 7,30$  Hz.

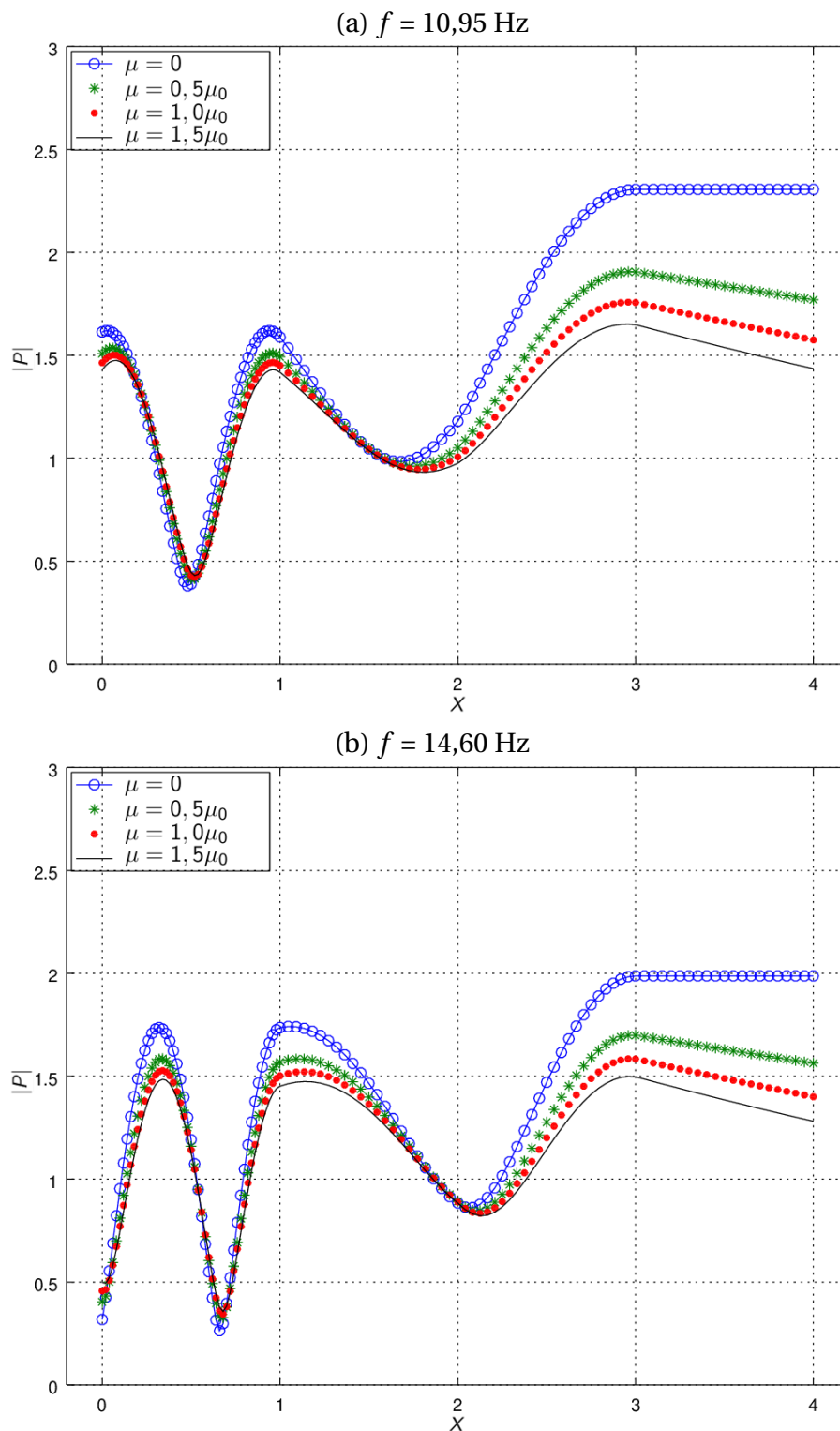


Figura 4.3: Amplitude da pressão  $|P|$  ao longo da árvore arterial considerando diferentes viscosidade do fluido  $\mu$  e frequências: (a)  $f = 10,95$  Hz, (b)  $f = 14,60$  Hz.

valor representando uma parede puramente elástica e para  $\phi_0 > 0$  tem-se a representação da viscoelasticidade. Nota-se a partir destas figuras que o efeito da viscoelasticidade, como o da viscosidade do fluido, é amortecer o aumento global da amplitude da onda de pressão causada pelas reflexões das ondas à medida que a onda se desloca na direção à jusante, bem como moderar os picos locais na distribuição de pressão.

A Figura 4.6 apresenta o comportamento da impedância de entrada  $|Z|$  em função da frequência  $f$ , que está coerente com dados experimentais [?] da aorta. Na Figura 4.6a, nota-se que, quando se considera a viscosidade ( $\mu \neq 0$ ) na simulação do método, isso afeta a impedância de entrada para frequências que se aproximam de 0 Hz e atenua o pico que surge na curva entre 10 Hz e 12,5 Hz. Além disso, neste intervalo de frequência, observa-se o aparecimento do pico mencionado em uma frequência ligeiramente menor com o aumento de  $\mu$ .

No tocante ao impacto da viscoelasticidade da parede na impedância de entrada, o aumento do parâmetro  $\phi$  reduz o pico que aparece na curva de impedância entre 10 Hz e 12,5 Hz como pode ser visto na Figura 4.6b. Diferente da Figura 4.6a, o pico destacado surge na mesma frequência independente do  $\phi$ .

Os efeitos combinados da viscosidade do fluido e viscoelasticidade da parede do vaso na amplitude de pressão  $P$  e impedância de entrada  $Z$  são mostrados nas Figuras 4.7 e 4.8. Os valores  $\mu = 1,0\mu_0$  e  $\phi_0 = 8^\circ$  foram utilizados para este propósito, para serem comparados com os mesmos valores dos resultados anteriores. A observação mais importante a ser feita é que os dois efeitos, quando combinados, não se somam. Por outro lado, combinam-se de uma maneira não linear que não é prontamente previsível.

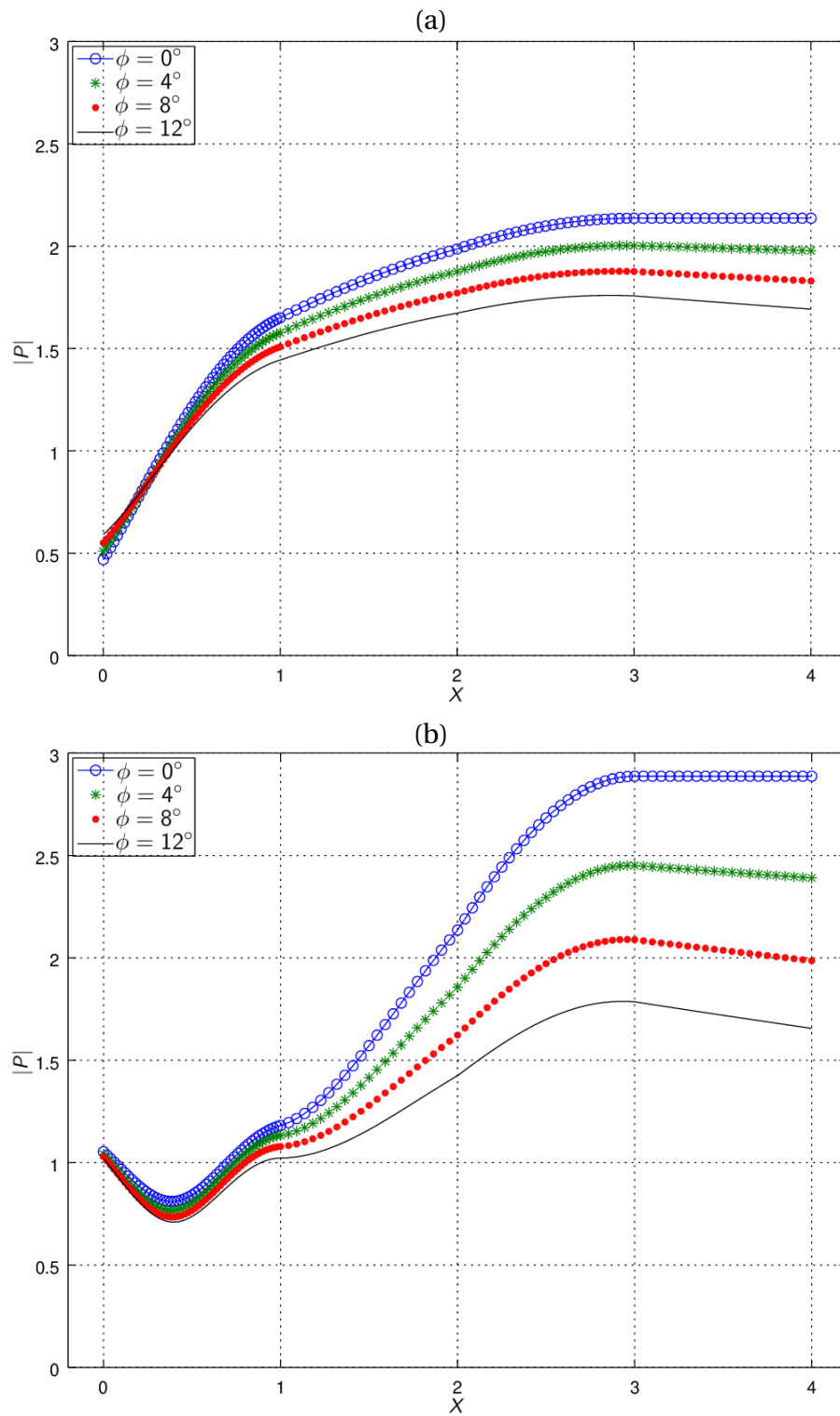


Figura 4.4: Amplitude da pressão  $|P|$  ao longo da árvore arterial considerando diferentes valores de viscoelasticidade  $\phi_0$  e frequências: (a)  $f = 3,65$  Hz, (b)  $f = 7,30$  Hz.

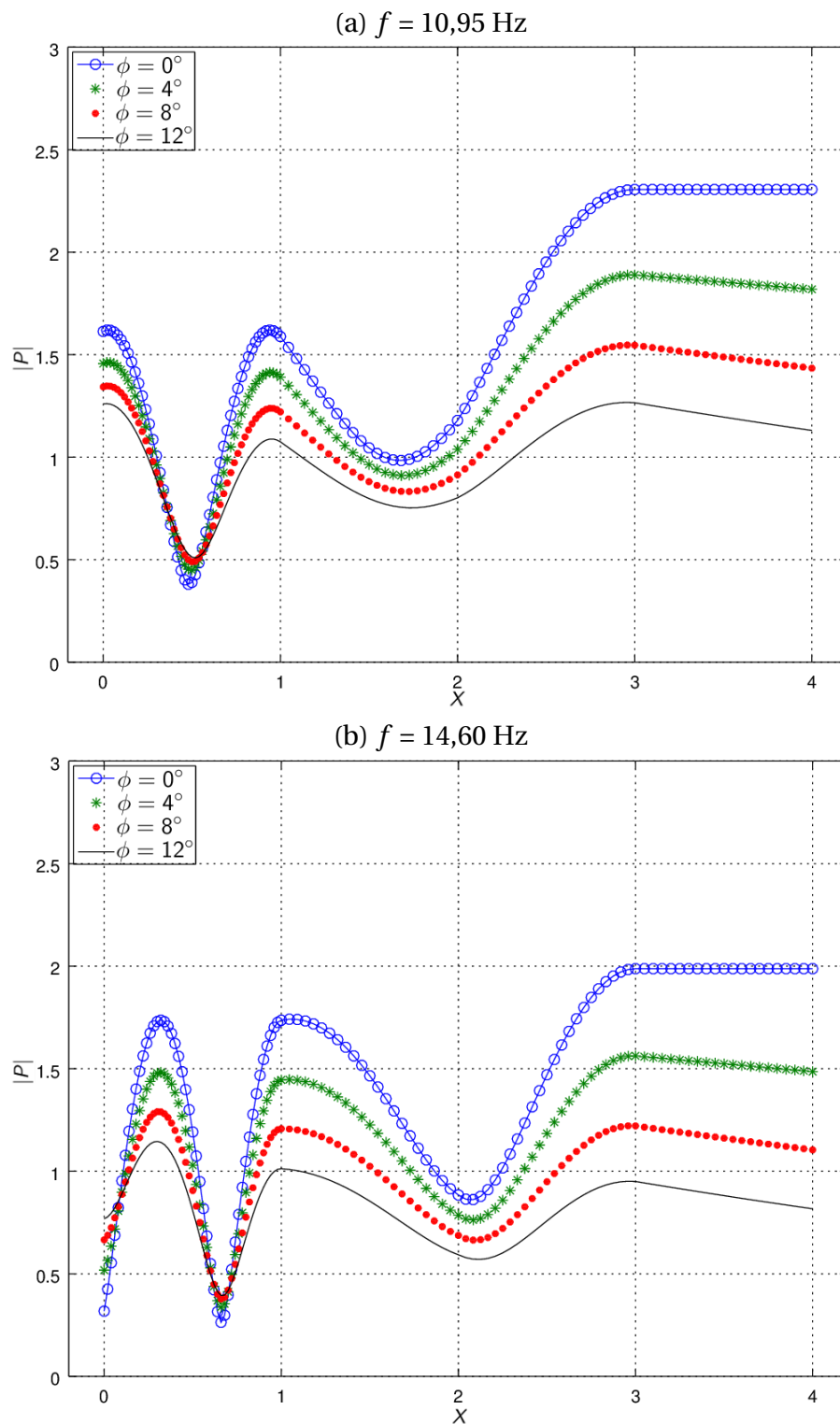


Figura 4.5: Amplitude da pressão  $|P|$  ao longo da árvore arterial considerando diferentes valores de viscoelasticidade  $\phi_0$  e frequências: (a)  $f = 10,95$  Hz, (b)  $f = 14,60$  Hz.

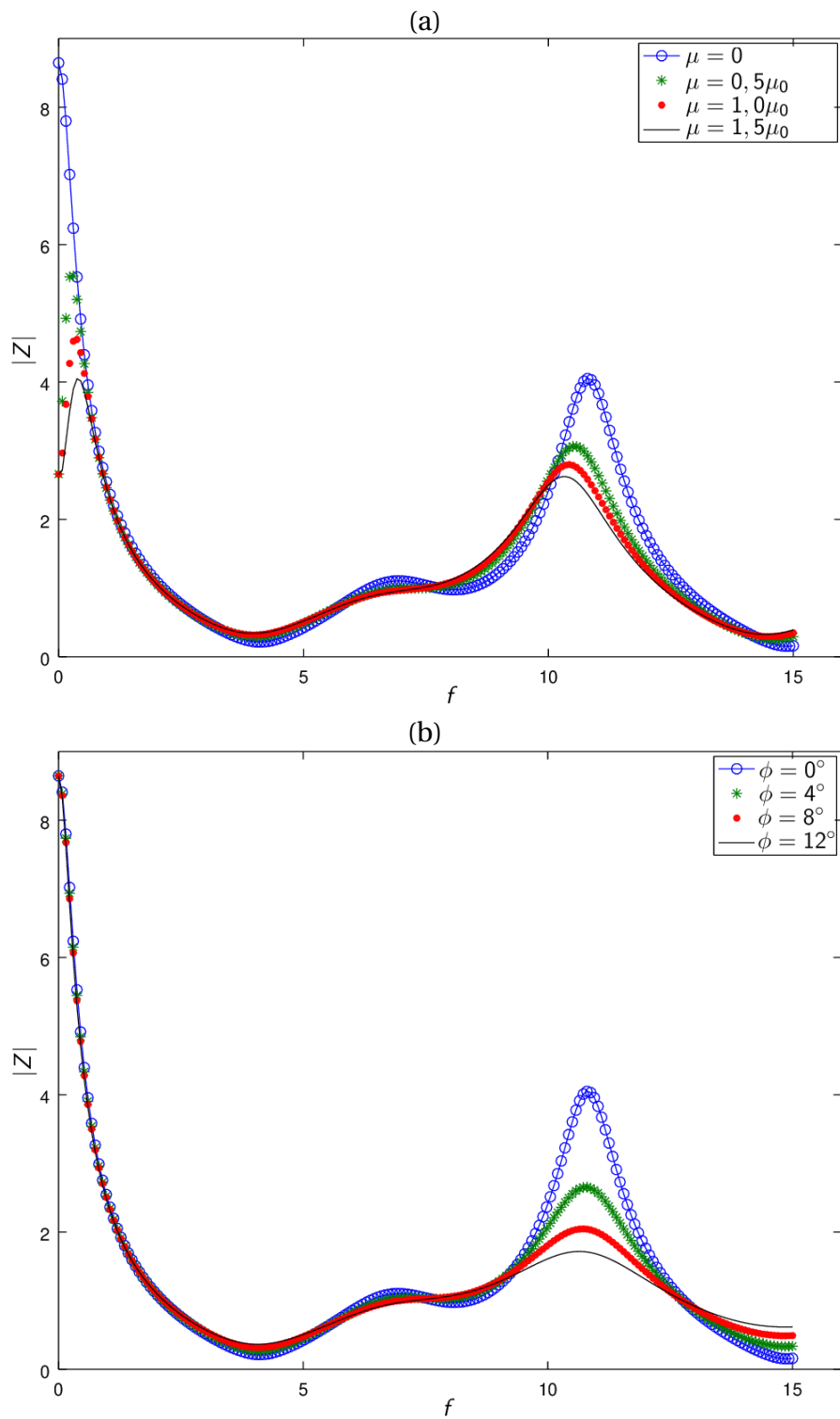


Figura 4.6: Amplitude da impedância de entrada  $|Z|$  em função da frequência  $f$ : (a) efeito da viscosidade (b) impacto da viscoelasticidade.

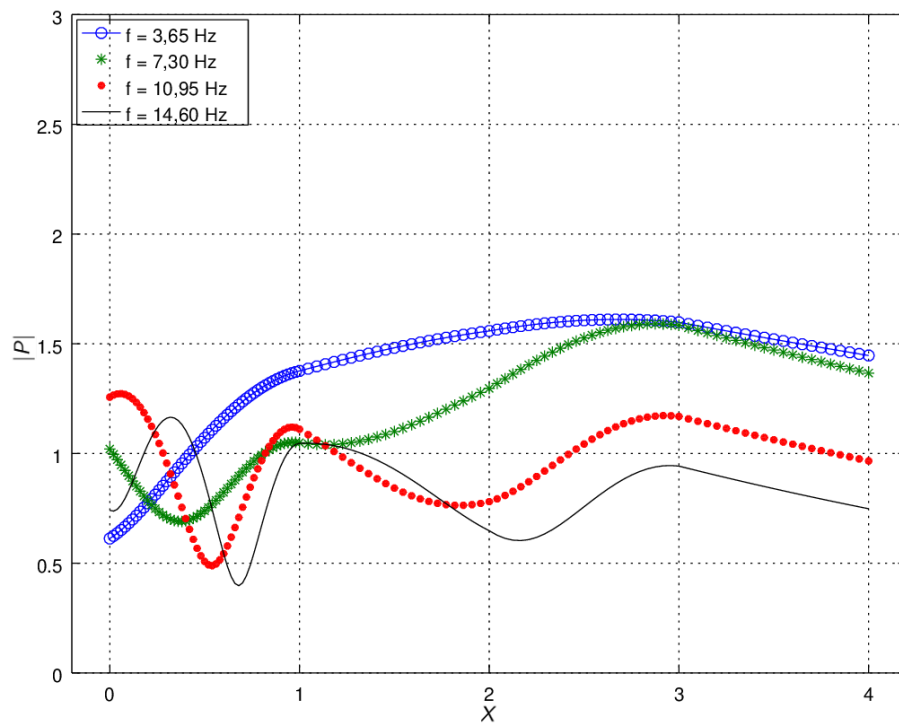


Figura 4.7: Amplitude da pressão  $|P|$  ao longo da árvore arterial  $X$  considerando viscosidade  $\mu = 1,0\mu_0$ , viscoelasticidade  $\phi_0 = 8^\circ$ .

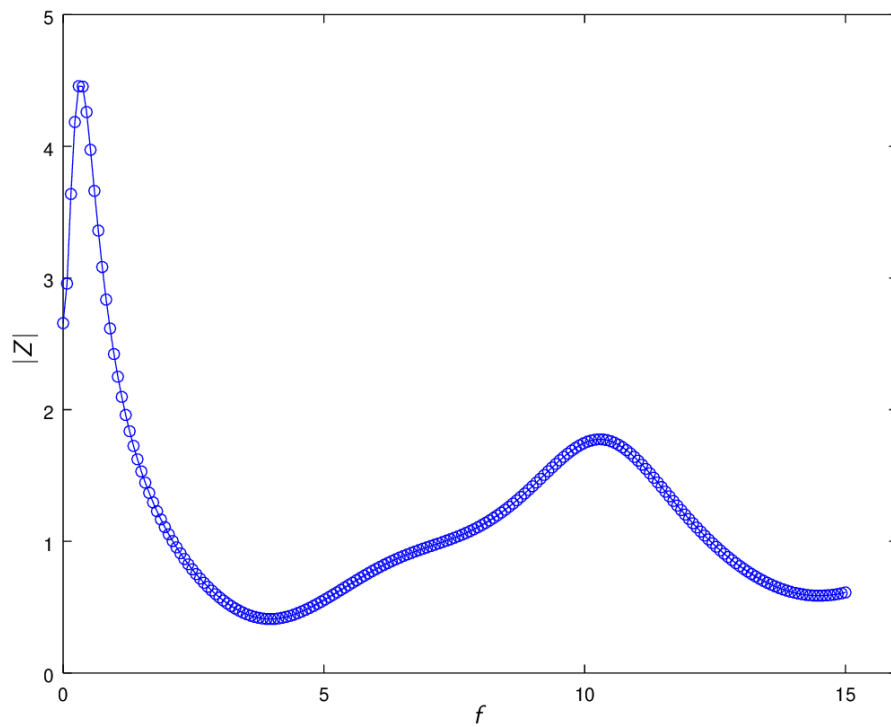


Figura 4.8: Impedância de entrada  $|Z|$  em função da frequência  $f$  considerando viscosidade  $\mu = 1,0\mu_0$ , viscoelasticidade  $\phi_0 = 8^\circ$ .

## 5 CONCLUSÕES E TRABALHOS FUTUROS

IGOR: já fiz uma primeira revisão deste capítulo e indico em vermelho o que deve fazer para melhorar este capítulo. Tudo que acrescentar ou mudar coloque em azul. XXXXXXXXXXXXX.

Em relação a hemodinâmica, os resultados obtidos neste trabalho estão de acordo com aqueles obtidos por Duan e Zamir[7] considerando a propagação de uma onda harmônica simples nos três cenários abordados nas simulações.

Em destaque, visando contribuir na investigação do método desenvolvido por Duan e Zamir, curvas de impedância de entrada do modelo de árvore canina aqui considerada foram apresentadas. Estas curvas apresentam comportamento que pode ser observado em dados experimentais.

A realização deste trabalho resultou no desenvolvimento de uma nova ferramenta computacional descrita no Capítulo ??, que permite a simulação de escoamento sanguíneo pulsátil em modelos de árvores arteriais no contexto de Duan e Zamir.

IGOR: escreva um parágrafo destacando propriedades/características e potencialidades da sua ferramenta computacional. Talvez você poderá recuperar ou utilizar algo do parágrafo abaixo.

A ferramenta simula e analisa modelos de árvores arteriais. Ela pode ser utilizada nos sistemas Operacionais: Windows e o Ubuntu (Unix). Os modelos podem ter sua análise e gráficos gerados automaticamente através dos comandos da ferramenta computacional. O código possibilita o processamento concorrente de diversos modelos em um mesmo ambiente. Cada uma destas contribuições resultou numa ferramenta robusta, ferramenta esta que além de analisar corretamente as variações de fluxo e pressão através de um modelo de árvore arterial, possibilita que as simulações sejam facilmente ajustadas e analisadas, com ou sem interface gráfica.

IGOR: o parágrafo acima descreve a ferramenta descrita no Capítulo 3? Este parágrafo representa a descrição da ferramenta.

Como trabalhos futuros, destacam-se:

- Analisar hemodinamicamente modelos de árvores gerados no contexto do método



CCO (Constrained Constructive Optimization) [11, 19, 20, 4];

- Investigar a influência da escolha parâmetros na resposta do método, tais como: módulo de Young e espessura do vaso;
- IGOR: descreva algo que poderia ser interessante agregar na ferramenta, pode ser mais de uma coisa (coloque em item).

## **A PROCESSO DE COMPILAÇÃO**

## B FORMATO DE ARQUIVO DE COMANDOS

---

**Algoritmo 4:** Cálculos hemodinâmicos do modelo de árvore arterial (*MAA*)

---

em comandos que a ferramenta computacional é capaz de processar.

---

```

1 object create artery_tree DUAN_AND_ZAMIR obj_t5 el_t5
2 object iteration_factories set obj_t5 DUAN_AND_ZAMIR
3 object set obj_t5
4 object set_field obj_t5 FIELD FREQUENCY 0 3.65
5 object set_field obj_t5 FIELD ARTERY_CHOOSE_MODE 0 HIGHEST
6 object go obj_t5
7 object create graphic BLANK obj_t6 el_t6
8 object iteration_factories set obj_t6 OBJECT_READ
9 object set obj_t6
10 object set_field obj_t6 FIELD READ_ELEMENT_NAME 0 obj_t5
11 object set_field obj_t6 FIELD READ_FIELD 0 PRESSURE_WAVE
12 object set_field obj_t6 FIELD READ_CELL_TYPE 0 FIELD
13 object go obj_t6
14 object export obj_t6 IMAGE_PNG pressure_3_65.png
15 object set_field obj_t6 FIELD READ_FIELD 0 FLOW_WAVE
16 object go obj_t6
17 object export obj_t6 IMAGE_PNG flow_3_65.png
18 object set_field obj_t5 FIELD FREQUENCY 0 7.30
19 object go obj_t5
20 object set_field obj_t6 FIELD READ_FIELD 0 PRESSURE_WAVE
21 object set_field obj_t6 FIELD READ_CELL_TYPE 0 FIELD
22 object go obj_t6
23 object export obj_t6 IMAGE_PNG pressure_7_30.png
24 object set_field obj_t6 FIELD READ_FIELD 0 FLOW_WAVE
25 object go obj_t6

```

---

---

**Algoritmo 5:** Cálculos hemodinâmicos do modelo de árvore arterial (*MAA*)

---

em comandos que a ferramenta computacional é capaz de processar.

---

```
1 object export obj_t6 IMAGE_PNG flow_7_30.png
2 object set_field obj_t5 FIELD FREQUENCY 0 10.95
3 object go obj_t5
4 object set_field obj_t6 FIELD READ_FIELD 0 PRESSURE_WAVE
5 object set_field obj_t6 FIELD READ_CELL_TYPE 0 FIELD
6 object go obj_t6
7 object export obj_t6 IMAGE_PNG pressure_10_95.png
8 object set_field obj_t6 FIELD READ_FIELD 0 FLOW_WAVE
9 object go obj_t6
10 object export obj_t6 IMAGE_PNG flow_10_95.png
11 object set_field obj_t5 FIELD FREQUENCY 0 14.60
12 object go obj_t5
13 object set_field obj_t6 FIELD READ_FIELD 0 PRESSURE_WAVE
14 object set_field obj_t6 FIELD READ_CELL_TYPE 0 FIELD
15 object go obj_t6
16 object export obj_t6 IMAGE_PNG pressure_14_60.png
17 object set_field obj_t6 FIELD READ_FIELD 0 FLOW_WAVE
18 object go obj_t6
19 object export obj_t6 IMAGE_PNG flow_14_60.png
20 object delete obj_t6
21 object delete obj_t5
```

---

## **C FORMATO DE ARQUIVO DE ELEMENTO INTELIGENTE**

# **D FORMATO DE ARQUIVO DE OBJETO INTELIGENTE**

## Bibliografia

- [1] Qt, 2019.
- [2] M. Anliker, R. Rockwell, and E. Odgen. Nonlinear analysis of flow pulses and shock waves in arteries. ZAMP, (22):217–246, 1971.
- [3] A. Avolio. Multi-branched model of the human arterial system. Med. Biol. Eng. Comput, (18):709–718, 1980.
- [4] P. Brito, L. Meneses, R. Santos, and R. Queiroz. Automatic construction of 3d models of arterial tree incorporating the fahraeus-lindqvist effect. Revista Eletrônica Paulista de Matemática, (10):38–49, 2017.
- [5] B. Duan and M. Zamir. Biodynamics: Circulation. New York: Springer-Verlag, 1984.
- [6] B. Duan and M. Zamir. Effect of dispersion of vessel diameters and lengths in stochastic networks. i. modeling of microcirculatory flow. Microvascular Research, (31):203–222, 1986.
- [7] B. Duan and M. Zamir. Viscous damping in one dimensional wave transmission. J. Acoust. Soc. Am., (92):3358–3363, 1995.
- [8] L. Formaggia, J. Gerbeau, F. Nobile, and A. Quarteroni. Computer methods in applied mechanics and engineering. Revista Eletrônica Paulista de Matemática, (191):561–582, 2001.
- [9] L. Formaggia, D. Lamponi, and A. Quarteroni. One-dimensional models for blood flow in arteries. Journal of Engineering Mathematics, (47):251–276, 2003.
- [10] K. Group. Opengl, 2019.
- [11] R. Karch, F. Neumann, M. Neumann, and W. Schreiner. A three-dimensional model for arterial tree representation, generated by constrained constructive optimization. Computers in biology and medicine, (29):19–38, 1999.
- [12] G. Karreman. Some contributions to the mathematical biology of blood circulation. reflection of pressure waves in the arterial system. Bull. Math. Biophys., (14):327–350, 1952.
- [13] N. Kouchoukos, L. Sheppard, and D. A. McDonald. Estimation of stroke volume in the dog by a pulse contour method. Circ. Res., (26):611–23, 1970.
- [14] M. Lighthill. Mathematical biofluidmechanics. Philadelphia: Society for Industrial & Applied Mathematics, 1975.
- [15] R. Mates, F. Klocke, and J. Canty. Coronary capacitance. Progress in Cardiovascular Diseases, (31):1–15, 1988.
- [16] D. A. McDonald. Blood flow in arteries. Baltimore: Williams & Wilkins, 1974.
- [17] A. Parker. Algorithms and Data Structures in C++. 1959.

- [18] C. Peskin. Flow patterns around heart valves: a numerical method. J. Comput. Phys., (10):252–271, 1972.
- [19] R. Queiroz. Construção automática de modelos de árvores circulatórias e suas aplicações em hemodinâmica computacional. PhD thesis, Laboratório Nacional de Computação Científica, 2013.
- [20] R. Queiroz, P. Blanco, R. Feijó, and J. Ulysses. Ifmbe proceedings. 49ed.: Springer international publishing. Baltimore: Williams & Wilkins, pages 884–887, 2015.
- [21] N. Stergiopulos, D. Young, and T. R. Rogge. Computer simulation of arterial flow with applications to arterial and aortic stenoses. Journal of Biomechanics, (25):1477–1488, 1992.
- [22] C. Taylor, T. Hughes, and C. Zarins. Finite element modeling of three-dimensional pulsatile flow in abdominal aorta: relevance to atherosclerosis. Annals of Biomedical Engineering, (26):975–987, 1998.
- [23] M. Taylor. The input impedance of an assembly of randomly branching elastic tubes. Biophys. J., (6):29–51, 1966.
- [24] L. Welicki, J. W. Yoder, and R. Wirfs-Brock. The dynamic factory pattern, 2008.
- [25] M. Zamir. ptimality principles in arterial branching. J. Theor. Biol., (62):227–251, 1976.
- [26] M. Zamir and S. Phipps. Network analysis of an arterial tree. American Journal of Physiology, (21):25–34, 1988.