HeMoLab: Laboratório de Modelagem em Hemodinâmica

Ignacio Larrabide e Raúl A. Feijóo

O aumento na complexidade das intervenções cirúrgicas exige cada dia um maior uso de técnicas precisas que permitam predizer o resultado dos procedimentos médicos. O atual grau de desenvolvimento alcançado pelas técnicas de modelagem computacional unida ao crescimento da performance dos computadores permite o estudo, desenvolvimento e solução de modelos computacionais capazes de estudar a resposta do corpo humano a estes procedimentos. A modelagem e simulação computacional, computação gráfica e realidade virtual, fornecem as ferramentas necessárias para representar, com alto grau de detalhe, os fenômenos que acontecem dentro do organismo de um paciente de maneira não invasiva. Com esta visão nasceu o conceito do HeMoLab - Laboratório de Modelagem em Hemodinâmica, uma ferramenta que reúne as mais novas técnicas de modelagem do Sistema Cardiovascular Humano com o objetivo de se tornar útil na difícil tarefa de modelar o corpo humano. Desenvolvimento de modelos personalizados do sistema arterial completo, processamento de imagens médicas para obtenção de geometrias dos pacientes, geração de modelos tridimensionais de distritos arteriais específicos, simulação integral do sistema arterial ajustado ao comportamento de um paciente, integração destes dados com as geometrias extraídas das imagens do paciente são algumas das funcionalidades mais salientáveis desta ferramenta.

Categories and Subject Descriptors: [ ]:

General Terms:

Additional Key Words and Phrases:

1. INTRODUÇÃO

Nestes últimos anos pesquisadores das áreas de engenharia biologia e medicina começaram, a introduzir ferramentas computacionais de predição nas práticas médicas. O atual grau de desenvolvimento alcançado pelas técnicas de modelagem computacional, conjuntamente com o rápido crescimento da performance de cálculos dos computadores, têm permitido o estudo, desenvolvimento e solução de modelos[1] computacionais altamente sofisticados capazes de antecipar com aceitável grau de precisão, os resultados de importantes procedimentos médicos. A modelagem e simulação computacional aliada à computação gráfica e realidade virtual, permitem representar com alto grau de detalhe e de maneira não invasiva, os fenômenos que acontecem dentro do organismo de um paciente.

Uma das potenciais aplicações desta tecnologia é no planejamento terapêutico e cirúrgico das mais variadas doenças cardiovasculares. A este respeito, é fundamental o uso de modelos matemáticos que representem a dinâmica do Sistema Cardiovascular Humano (SCVH) assim como a sua simulação computacional. Diferentes características desta dinâmica são de interesse, e.g., campos de velocidade do sangue dentro das artérias, propagação das ondas de pressão na árvore arterial, estado de tensões nas paredes arteriais, tensões oscilatórias, etc. Estes modelos devem permitir ainda a incorporação de informações específicas dos pacientes. Estas informações podem chegar de diferentes maneiras, formatos, e até incompletas em alguns casos (imagens médicas, propriedades mecânicas dos tecidos, leituras de pressão ou fluxo em determinados pontos da anatomia, etc.).

Algumas das aplicações mais destacáveis destes modelos para assistir na medicina são:

—*Assistir no treinamento e aprimoramento de técnicas cirurgias:* permitindo a estudantes aprender os procedimentos ou a cirurgiões experimentes aperfeiçoar as suas técnicas e habilidades.

—*Estudar o resultado de diferentes intervenções cirúrgicas:* permitindo determinar o melhor procedimento a ser utilizado em um determinado paciente, a melhor maneira de realizar este procedimento e até avaliar os potenciais resultados.

—*Representar fenômenos como absorção, difusão e transporte de substâncias bioquímicas nos tecidos da parede arterial:* reproduzindo por exemplo, o que acontece no caso de utilização de stens revestidos com fármacos.

—*Indicar risco de ruptura de aneurismas e desenvolvimento de hiperplasia:* diferentes estudos indicam que fatores hemodinâmicos (tensões oscilatórias e a conseqüente fatiga dos tecidos, altos gradientes de tensão nas paredes arteriais, regiões de re-circulação, elevados tempos de residência das partículas) estão intimamente relacionados ao risco de ruptura de aneurismas assim como ao desenvolvimento de hiperplasia. Os modelos permitem estudar este tipo de fatores.

—*Caracterização de materiais/tecidos biológicos de maneira não invasiva:* os tecidos que apresentam certos tipos de doenças (e.g., hiperplasia, calcificação, etc.) mostram as suas propriedades mecânicas alteradas. A determinação deste tipo de

[1]Ver glossário na Seção A.

propriedades tem um grande impacto no diagnóstico destas doenças. Problemas inversos é uma área com grande potencial neste tipo de aplicações.

Existem atualmente modelos que representam, com aceitável grau de detalhe, as condições do fluxo sanguíneo dentro das artérias assim com a propagação de ondas de pressão nas artérias [Formaggia et al. 1999; Hughes and Lubliner 1973; Formaglia et al. 2003; Causin et al. 2004; Deparis et al. 2005; Urquiza et al. 2006; Cebral and Lohner 1999]. Representar o SCV completo com modelos tridimensionais é, no momento, impensável por causa do altíssimo custo computacional. Para conseguir modelar o SCV como um todo, na atualidade são utilizados modelos multidimensionais. Estes modelos utilizam uma formulação diferente nas distintas partes do sistema dependendo do nível de detalhe que se deseje:

—Modelos 3D: Estes modelos são utilizados somente para reproduzir pequenos distritos do SCV. Nas regiões representadas com este tipo de modelos dispõe-se de grande detalhe do fluxo sanguíneo.

—Modelos 1D: São utilizados para representar as maiores artérias do corpo humano. Estes modelos são uma simplificação da equações de Navier-Stokes 3D. Os mesmos são obtidos ao assumir uma série de hipóteses no fluxo: 1) os vasos são retos, 2) o perfil de velocidade em qualquer ponto é conhecido e 3) uma relação constitutiva entre a pressão e a área do vaso.

—Modelo 0D: Utilizados na representação do comportamento do fluxo sanguíneo a partir de um ponto (e.g., para representar a resistência oferecida pelas arteríolas e capilares), estes modelos "condensam" em um ponto a resistência oferecida por uma parte do sistema não modelada em detalhe.

—Acoplamento 1D-3D: Os modelos de diferente dimensionalidade são acoplados de maneira que se comportem como um todo. Em particular, os modelos 1D são utilizados para fornecer condições de contorno ao modelos 3D que sejam consistentes com o comportamento sistêmico. Desta maneira, os diferentes tipos de modelos interagem obtendo-se um comportamento global complexo.

As soluções destes modelos são achadas utilizando técnicas numéricas como o Método dos Elementos Finitos [Hughes 2000]. Pelo fato das formulações de diferente dimensionalidade terem propriedades diferentes, informações heterogêneas consistentes com cada modelo devem ser associadas em um modelo sistêmico integrado. Não existe atualmente uma ferramenta que permita integrar estes modelos em um mesmo ambiente de trabalho. Para suprir esta falta pensou-se em uma nova ferramenta que integre e permita manipular as diferentes informações em um único ambiente de trabalho possibilitando a criação deste tipo de modelos, o HeMoLab.

A abrangência e complexidade dos modelos mencionados faz com que o desenvolvimento desta ferramenta seja uma tarefa complexa a qual requer da interação de especialistas de diferentes áreas (engenharia, medicina, computação, entre outras). São analisados, na seguinte seção, alguns requerimentos básicos desta ferramenta.

2. ANÁLISE DE REQUERIMENTOS

A criação de modelos do SCV requer da integração de diferentes funcionalidades e dados. Somente com o objetivo de ilustrar esta idéia, são citados alguns exemplos. No caso, pode ser de interesse dos usuários a reconstrução de geometrias a

partir de imagens médicas do paciente. Uma outra alternativa seria parametrizar a árvore arterial para aproximar o seu comportamento o máximo possível do observado em um indivíduo determinado, para o qual deve-se permitir modificar as propriedades mecânicas da árvore assim como adicionar ou eliminar segmentos arteriais nas regiões que tenham maior ou menor relevância. Também, pode ser de interesse analisar em detalhe a hemodinâmica (fluxo, pressão, estado de tensões, etc.) em um certo distrito do sistema de um determinado paciente com o fim de obter aquelas geometrias. Aparecem então diferentes requerimentos a serem considerados no desenvolvimento:

—*Processamento de imagens:* para poder extrair a geometria das artérias de um determinado paciente é necessário fornecer ferramentas para visualizar e processar estas imagens. Uma vez processadas estas imagens, a geometria das artérias ou estruturas de interesse nelas presentes devem ser reconstruídas tridimensionalmente (e.g., triangulação das superfícies).

—*Visualização/Edição da árvore 1D:* devem-se fornecer ferramentas que permitam visualizar e editar os parâmetros da árvore arterial 1D de maneira a poder reproduzir as características e comportamentos específicos de diferentes pacientes.

—*Visualização/Edição da geometrias 3D:* uma vez obtidas as geometrias (superfícies) a partir das imagens dos pacientes, as mesmas devem ser processadas/refinadas para, a partir delas, gerar as malhas de elementos finitos volumétricas que serão utilizadas na discretização de elementos finitos para calcular as aproximações numéricas das equações de fluidos/sólidos correspondentes.

—*Acoplamentos entre os diferentes modelos:* para conseguir que os modelos funcionem de maneira integrada, é necessário acoplar os modelos 1D e 3D.

—*Solução numérica de equações diferenciais:* os modelos mencionados anteriormente fornecem uma série de equações cujas soluções correspondem aos campos de interesse (velocidade do fluxo, pressão, etc.). As soluções destas equações podem ser aproximadas utilizando diferentes técnicas numéricas.

—*Visualização dos resultados:* os resultados do cálculo numérico são gigantescos volumes de dados. Para poder interpretá-los, estes dados devem ser presentados de uma maneira amigável (técnicas de visualização científica [referencias visualizacion científica]).

Estes requerimentos foram separados em módulos com diferentes funcionalidades. Na seqüência são analisados os requerimentos específicos de cada um deles.

2.1 Módulo 1: Processamento de Imagens

O processamento de imagens tem sido largamente utilizado na medicina [Hohne et al. 1990; Suri et al. 2001]. Talvez esta seja a maneira mais efetiva de se utilizar dados reais para simulações computacionais. O objetivo deste módulo é permitir ao usuário ler e processar imagens médicas. Geralmente as imagens médicas são recebidas no formato padrão DICOM [Bidgood 1998], este formato é amplamente utilizado pelos aparelhos de aquisição. No entanto, algumas vezes estes dados podem ser fornecidos em outros formatos, por exemplo como seqüencias de imagens BMP, JPG, PNG ou em uma única imagem que contém os diferentes planos. Por este motivo, este módulo deve permitir a leitura de diferentes formatos de imagens.

Existem diversos filtros usualmente utilizados para processar uma imagem [Aubert and Vese 1997; M.J.Black et al. 1998; Frangakis and Hegerl 2001; Gonzalez and Woods 2001; Larrabide et al. 2005]. No caso de imagens médicas, estas costumam apresentar ruído o que dificulta o processo de identificação das diferentes estruturas. Filtros de suavização Gaussiano, suavização anisotrópico, filtros morfológicos, dentre outros são usualmente utilizados para remover ruído e melhorar a qualidade de uma imagem degradada.

Uma vez que a imagem foi filtrada, é necessário identificar as diferentes regiões que nela se encontram. Este processo é chamado de segmentação. Neste caso particular este processo será utilizado para identificar artérias ou outras estruturas relacionadas ao sistema cardiovascular presentes nas imagens (paredes arteriais, aneurismas, etc.). Existem na bibliografia uma variedade de métodos para segmentar imagens, variando segundo a aplicação específica [Boscolo et al. 2002; Li et al. 1995; Zhang et al. 2001; Larrabide et al. 2006; Malladi and Sethian 1997; Larrabide et al. 2003].

Tanto os filtros de imagens quanto os métodos de segmentação, são configurados com diferentes parâmetros, precisando cada um deles um tratamento especial. Processamento de imagens médicas é na atualidade uma das áreas mais ativas de produção científica. Por este motivo devem-se considerar mecanismos que permitam incorporar métodos já existentes assim como novos métodos que possam ser desenvolvidos no futuro.

Finalmente, uma vez selecionada a região de interesse dentro da imagem, a geometria (triangulação) que a representa deve ser gerada [Lorensen and Cline 1987]. A partir desta primeira versão da geometria do vaso (ou estrutura segmentada), será gerada a malha de elementos finitos correspondente (Seção 2.3).

2.2 Módulo 2: Modelo 1D

A maior parte do sistema arterial é representada utilizando uma simplificação unidimensional das equações de Navier-Stokes, este modelo é chamado de Modelo 1D [Hughes and Lubliner 1973; Formaglia et al. 2003]. Existe no momento uma geometria "padrão" baseada na árvore arterial proposta por Avolio [Avolio 1980], a mesma esta composta de 128 segmentos arteriais que representam as maiores artérias do corpo humano. O primeiro requerimento deste módulo será então poder ler e escrever este tipo de árvores. Estas árvores encontram-se armazenadas em arquivos de texto formatados (o formato destes arquivos é o definido pelo Solver GP, Seção 4.6).

Existem neste tipo de árvores dois tipos de elementos: segmentos e terminais Windkessel. Cada segmento representa uma artéria, fornecendo informações de fluxo, pressão e área em todos os pontos da árvore arterial. Por sua vez, os terminais condensam em um ponto (modelo 0D, baseados no modelo proposto em [Frank 1899]) características de uma região completa. Isto permite simplificar a estrutura da árvore de maneira que não é necessário fornecer propriedades dos distritos condensados, porém os mesmos não provêem detalhes de fluxo e pressão nas regiões condensadas.

Dois tipos de informações caracterizam este modelo:

—*geométricas* e

—*paramétricas*.

Os dados *geométricos* irão definir a topologia da árvore representada. Ao incorporar maior quantidade de elementos nesta árvore, obtém-se uma melhor aproximação (mais precisa) nos resultados numéricos associados. Para permitir modificar a geometria da árvore será necessário fornecer ferramentas de edição que permitam agregar e eliminar elementos desta árvore assim como modificar os já existentes.

Os dados paramétricos são aqueles que definem as características mecânicas das artérias, quais sejam: diâmetro da artéria[2]; espessura da parede arterial; percentual de elastina, colágeno, e músculo liso na parede, pressão externa no vaso, permeabilidade da parede, curva de ejeção do coração, dentre outras. Estes parâmetros devem assumir valores dentro de determinados intervalos, caso contrário o modelo não teria sentido físico. A verificação deste valores evitará então inconsistências nos modelos.

2.3 Módulo 3: Modelo 3D

Outra funcionalidade importante é permitir a simulação do fluxo sanguíneo em determinados trechos da árvore arterial de maneira a poder analisar em detalhe as características do fluxo naquela região.

As superfícies geradas a partir das imagens médicas costumam ser de má qualidade (estas costumam ter triângulos tipo "agulhas" onde a área se aproxima de zero mas o comprimento das arestas e maior a zero, ou muito pequenos onde todas as arestas são da mesma ordem mas a área do elemento é próxima que zero). Por este motivo são necessárias ferramentas (filtros que operam sobre a malha modificando a posição dos pontos e a conectividade dos triângulos) que permitam melhorar as superfícies eliminando este tipo de características. Uma malha de boa qualidade para a simulação computacional se caracteriza por estar composta de triângulos praticamente equiláteros (todos os seus lados são quase iguais). A precisão dos resultados (associados aos modelos) dependerá diretamente da qualidade destas malhas e do tamanho dos elementos desta malha.

Também existe a necessidade de incorporar informações específicas dos pacientes nestas geometrias. Informações como espessura das paredes arteriais, percentual de colágeno, elastina e músculo liso irão influir na resposta do modelo do sistema arterial. Caso estas informações possam ser extraídas das imagens dos pacientes ou de outros estudos associados a ele, elas irão enriquecer os modelos. Pode ser de interesse também, modelar o comportamento do fluido utilizando diferentes relações constitutivas (fluido Newtoniano ou não-Newtoniano).

Todas estas informações deverão ser salvas de maneira persistente (em arquivos). Em particular, no formato definido pelo SolverGP, o qual será utilizado para calcular as soluções dos modelos.

2.4 Módulo 4: Acoplamento

Uma vez gerada a geometria da árvore 1D, a(s) geometria(s) 3D e determinados os correspondentes parâmetros, estas informações devem ser reunidas gerando um

[2]O diâmetro e uma característica geométrica da artéria, mesmo assim este entra como um parâmetro no modelo matemático simplificado, por este motivo é considerada um dado paramétrico.

único modelo. Os passos seguidos são:

—identificar/selecionar os segmentos arteriais do modelo 1D que serão substituídos por uma geometria 3D,

—remover estes segmentos da árvore 1D,

—associar os pontos de entrada/saída da geometria 3D, identificados por grupos de elementos, com os segmentos correspondentes na árvore 1D.

Para isto será necessário crias grupos de elementos no modelo 3D que caracterizem os pontos de conexão entre ambos modelos. As informações correspondentes aos modelos 1D, 3D e ao acoplamento devem ser salvas em um formato compatível com o do SolverGP.

2.5 Módulo 5: Simulação Numérica

O solver numérico SolverGP é um framework pensado para atender diferentes necessidades de acordo com o problema sendo modelado. O problema modelado é descrito utilizando uma série de arquivos formatados (os quais contém informação da geometria, acoplamentos, parâmetros e condições iniciais e de contorno associadas ao problema) em um formato definido especificamente para este framework. Nestes arquivos encontra-se a informação necessária para montar o sistema de equações correspondente ao problema modelado.

Os parâmetros utilizados na resolução dos sistemas de equações (resolvedor direto ou iterativo, esquema de integração temporal, etc.) de cada problema são definidos nestes arquivos. Logo, o problema sendo considerado, encontra-se completamente definido em termos dos arquivos de entrada do SolverGP. Por este motivo, a leitura/escrita destes arquivos é um ponto fundamental no aproveitamento desta ferramenta.

2.6 Módulo 6: Visualização e Análise

Uma vez realizada a simulação numérica, a solução do problema modelado (e.g., campos escalares no caso da pressão, campos vetoriais no caso da velocidade e deslocamento, etc.) é representada pelo valor (numérico) destes campos em cada ponto do domínio. Estas informações representam uma quantidade enorme de dados os quais não podem ser analisados diretamente sem tratamento. Para interpretar estes dados, são utilizadas técnicas de visualização científica [Haber and McNabb 1990; Gillilan and Wood 1995; Scateni 1995; Schroeder et al. 2000] que permitem representar a informação de maneira compreensível. Por exemplo, o campo de velocidades pode ser representado utilizando "glyphs" que indiquem a direção e intensidade do campo nos diferentes pontos do domínio. O deslocamento das paredes arteriais provocado pela variação da pressão no interior dos vasos pode ser visualizado aplicando uma deformação na geometria consistente com os deslocamentos em cada ponto. Existem inumeráveis maneiras de representar estes dados dependendo da informação sob análise. Para isto, são necessárias estruturas de dados (para armazena-los), filtros (para processar esta informação) e bibliotecas gráficas especialmente desenhadas para visualização de grandes volumes de dados mantendo boa performance (tirando proveito do hardware disponível, e.g., OpenGL [OpenGL]).

3. O FRAMEWORK UTILIZADO

Nesta seção é analisado o ambiente de trabalho e ferramentas utilizadas no desenvolvimento. Em seguida é feita uma revisão da arquitetura do software ParaView, o qual foi utilizado como base para o desenvolvimento.

3.1 Ambiente de trabalho

O desenvolvimento do HeMoLab está sendo feito no sistema operacional Linux, na distribuição Ubuntu 6.06.1 LTS. A linguagem de desenvolvimento é ANSI-C++, e os compiladores utilizados são c++ e gcc (correspondentes ao GCC versão 4.0.3).

O IDE (Integrated Development Environment) escolhido é o Eclipse SDK (versão 3.1.2), com os módulos (plug-ins) Eclipse C/C++ Development Tooling - CDT (versão 3.0.2) para edição de arquivos C/C++ e Subclipse (versão 0.9.105) para interação com o repositório. O controle de versões é feito utilizando um repositório Subversion (SVN - 1.3.1). Na configuração de projeto é utilizado o CMake (versão 2.2 - patch 3).

Para gerenciar o projeto é utilizado o dotProject (versão 2.0.1), e para gerenciar as tarefas está sendo utilizado o Bugzilla (versão 2.16.7). A equipe de desenvolvimento é constituída por:

—1 gerente de projeto,

—2 programadores senior,

—5 programadores junior,

—1 graphic designer.

3.2 ParaView

Para suprir os requerimentos mencionados na seção anterior, está sendo desenvolvido um software que reúne as ferramentas mencionadas dentro de um mesmo ambiente de trabalho. O ambiente escolhido para desenvolver as ferramentas foi o ParaView [Inc. a], o qual será o marco destas funcionalidades.

O ParaView é uma aplicação originalmente desenvolvida para visualização de dados científicos, com interface gráfica simples e que pode ser parametrizada pelos próprios usuários. Este software também incorpora métodos de processamento de imagens, além de proporcionar suporte para computação distribuída e portabilidade para vários sistemas operacionais. Também possui a capacidade de se comunicar através de diferentes formatos de arquivos. Dependendo da arquitetura em que as aplicações funcionam, o tamanho dos conjuntos de dados que o ParaView suporta pode variar.

Embora o ParaView seja em boa parte, um programa interpretado, este apresenta o rendimento de um programa compilado, com as facilidades e extensibilidade que uma linguagem interpretada oferece. O mesmo pode ser estendido por meio do uso de arquivos XML descrevendo novos módulos, aproveitando rotinas escritas na linguagem TCL que por sua vez utiliza bibliotecas compiladas feitas em C++. A arquitetura do ParaView foi criada para ser modular, afinal o processamento de dados, a renderização e o controle de interface do usuário podem funcionar em processos separados, em diferentes computadores, podendo estes inclusive estarem funcionando em diferentes plataformas. O ParaView utiliza como base uma das mais

utilizadas bibliotecas de visualização e renderização 3D, o Visualization ToolKit (VTK [Inc. b]). O VTK foi escrito em C++ e usa primitivas OpenGL para a renderização. Já a interface de usuário (User Interface - UI) do ParaView foi escrita na linguagem TCL.

O ParaView foi criado pela Kitware Inc. juntamente com os Laboratórios de Los Alamos, Sandia e Lawrence Livermore. O aspecto mais significativo do ParaView é a separação da interface gráfica do usuário (Graphical User Interface - GUI) dos servidores de processamento.

Este requerimento foi atendido criando um Gerenciador de Servidores (chamado *Server Manager*, responsável pela comunicação entre os diferentes processos) e de uma linguagem para comunicar objetos em diferentes processos (a Linguagem Cliente-Servidor, é uma linguagem especialmente criada para poder chamar métodos em objetos VTK de maneira remota).

As características reunidas pelo ParaView que o fizeram a melhor alternativa para ser o ambiente de desenvolvimento das funcionalidades acima descritas. Abaixo, as mesmas são resumidas:

—*Aplicação Open-Source:* o ParaView é uma aplicação Open Source sob a licença GNU, implicando que qualquer pessoa ou grupo tem o direito de usar e/ou distribuir o software do modo que este é, ou introduzindo as modificações que bem entendam necessárias sem pagar por direitos autorais ou patentes.

—*Desenvolvido na linguagem ANSI-C++:* o que o faz eficiente e robusto, além de permitir que o mesmo seja recompilado em diferentes plataformas.

—*Extensibilidade:* a arquitetura do ParaView permite estender facilmente as suas funcionalidades, permitindo agregar novas rotinas de maneira simples.

—*Paralelismo:* esta ferramenta foi pensada e desenvolvida de maneira a poder trabalhar em vários processos e inclusive em vários computadores ao mesmo tempo. Esta característica torna-se muito atraente quando o tamanho dos dados é muito grande.

—*Versatilidade:* permite tratar diferentes formatos de dados (e incluir novos) de maneira independente sem ser esta uma limitação na sua performance.

—*Baseado em VTK:* esta biblioteca é amplamente utilizada e conhecida, o que facilita a sua manipulação e o desenvolvimento de novos módulos.

3.3 Arquitetura do ParaView

O ParaView baseia-se em uma arquitetura Cliente-Servidor. Sua arquitetura foi pensada para ser modular, permitindo a separação do processamento de dados, a visualização e a interface de usuário em diferentes processos/computadores. Desta maneira não resulta difícil montar um cenário no qual o resultado de uma simulação (geometria + dados) processada em um supercomputador, é visualizada em um computador com menor quantidade de processadores mas com um hardware de renderização 3D de alta performance enquanto a interface de usuário é apresentada em um computador de escritório simples.

Neste contexto existem quatro componentes que integram o ParaView:

—Client (CL);

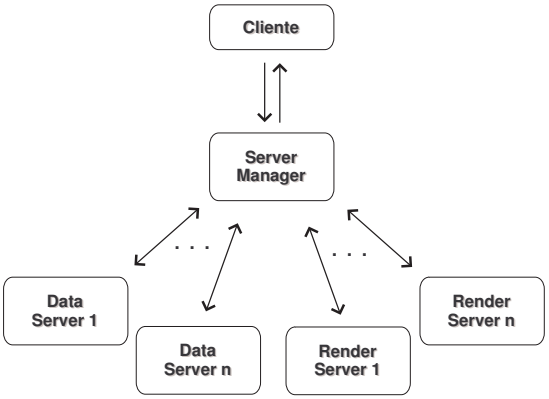

Fig. 1. Arquitetura Cliente-Servidor do ParaView.

—Server Manager (SM);

—Render Server (RS) e

—Data Server (DS).

Cada um destes componentes tem funções específicas, as mesmas são descritas a continuação.

3.4 Client - CL

O Cliente é responsável por passar os valores/dados, fornecidos pelo usuário, para o Server Manager. Estes valores são lidos utilizando controles de interface (abas) comuns, e.g., caixas de texto, botões, etc. Também é responsabilidade do cliente manter consistentes as informações mostradas na interface com os dados que se encontram no Server Manager.

A GUI dos módulos de leitura/escrita/filtrado são especificados utilizando arquivos XML. Existem dois tipos de arquivos XML: a) XML Cliente: especificam as abas necessárias na GUI para controlar os leitores/escritores/filtros que serão utilizados pelo usuário para introduzir dados (e.g., parâmetros dos filtros); b) XML Servidor: relacionam as classes no Server Manager (*proxies*) com os correspondentes leitores/escritores/filtros no servidor (objetos VTK). Utilizando estes proxies as classes de interface do cliente se comunicam com as classes nos diferentes servidores.

Para cada aba[3] que se deseje agregar na interface de um determinado módulo, existe um tag correspondente no XML. Estes controles podem ser abas simples (e.g., um botão, caixa de texto, etc.) ou componentes complexos (compostos por varias abas, por exemplo, um painel que contem vários campos, barras de rolagem, e botões). Desta maneira, ao criar novos filtros, podem ser criados controles que permitam ao usuário introduzir as informações específicas daquele filtro.

[3]Ver glossário em Seção A

3.5 Server Manager - SM

O Server Manager (ou Gerenciador de Servidores) é o módulo responsável pela comunicação entre o Render Server, o Data Server e o Cliente. Também é responsável por manter uma cópia do estado dos servidores por meio do uso de propriedades. Nele existem referências a proxies (objetos locais) que se comunicam com os objetos VTK (objetos criados remotamente) nos servidores, fornecendo também uma interface para criação e gerenciamento de objetos naqueles processos possivelmente em paralelo. O server manager possui os seguintes componentes:

—Proxies - `vtkSMProxy` e suas subclasses, fornecem uma interface local para se comunicar com os objetos VTK, que residem no(s) servidor(es). As suas responsabilidades são as seguintes: Criar os objetos VTK no servidor; Manter referências aos objetos VTK no servidor usando os *Ids Cliente Servidor(CSID)*[4]; Manter uma cópia do estado dos objetos que se encontram no lado servidor;

—Propriedades - `vtkSMProperty` e suas subclasses, fornecem o acesso do cliente à interface (API - Application Programming Interface) dos objetos no lado servidor representado pelos proxies;

—Domínios - `vtkSMDomain` e suas subclasses, representam uma coleção/intervalo de possíveis valores que as propriedades podem assumir.

Os diferentes proxies são gerenciados pelo Proxy Manager (`vtkSMProxyManager`). O Proxy Manager é uma classe Singleton [Gamma et al. 1995] (existe uma única instância dela em memória) responsável por manter referência aos proxies e suas correspondentes propriedades de acordo ao especificado nos arquivos XML do servidor (os quais serão descritos em breve). Cada proxy esta relacionado a um objeto no servidor por meio do CSID. O CSID e o objeto no servidor são criados pelo Process Module (`vtkProcessModule`, também singleton), responsável por atribuir um CSID único no sistema a cada objeto criado através dele e de obter a referência desse objeto dado o CSID.

A configuração do SM é armazenada em arquivos XML. Estes arquivos irão especificar com que objetos no servidor cada proxy se comunica e quais são as propriedades a ele associadas. Desta maneira, o usuário ou o desenvolvedor tem um mecanismo dinâmico de associar proxies (no SM) com objetos VTK (nos servidores). Enquanto os arquivos XML do cliente especificam que controles devem aparecer na interface para permitir ao usuário alterar os dados e parâmetros dos diferentes módulos e filtros, o XML do servidor determina qual será o proxy responsável por tratar/enviar estes dados ao servidor e quais serão as propriedades associadas e esse proxy.

Dentro dos arquivos XML do servidor existe um único marcador (tag) XML chamado (ServerManagerConfiguration), por sua vez este contem marcadores de um único tipo (`ProxyGroup`). Este marcador é utilizado para agrupar proxies de acordo com a sua funcionalidade (e.g., leitores/escritores, filtros, widgets 3D, etc.), também define no seu atributo "name" o grupo ao qual proxies definidos dentro dele pertencem, alguns exemplos podem ser "sources", "filters" e "3d_widgets". Um breve exemplo é apresentado na Fig. 2.

[4]O id cliente servidor é um valor único a nível de sistema que permite identificar objetos no(s)

```
<ServerManagerConfiguration>
   <ProxyGroup name="sources">
        <SourceProxy name="CubeSource" class="vtkCubeSource">
           [...]
        </SourceProxy>
   </ProxyGroup>
   <ProxyGroup name="filters">
        <SourceProxy name="SurfaceSmooth" class="vtkSmoothPolyDataFilter">
           [...]
        </SourceProxy>
   </ProxyGroup>
</ServerManagerConfiguration>
```

Fig. 2. Exemplo código XML com proxies em diferentes grupos.

Um exemplo simples da chamada de um proxy via XML é o caso dos leitores, fontes e filtros (todos estes tipos de filtros são considerados fontes de dados ou "sources"). Cada source é representado (no XML) pelo elemento `SourceProxy` (um sub-marcador do `ProxyGroup`). O marcador `SourceProxy` tem os seguintes atributos: *name,* que é um identificador **único** desse source proxy, o mesmo não pode estar repetido em outro source proxy; *class,* que é o nome da classe source que será criada no servidor ao se criar esse proxy. Dentro deste marcador, encontram-se definidas as propriedades associadas a este proxy (como mostrado na Fig. 3). Desta maneira é feita uma associação única entre um source proxy e uma classe VTK (source) no servidor. O proxy cumpre a função de estabelecer a comunicação entre o processo no qual o proxy se encontra e a classe correspondente em um servidor. Este exemplo considera o caso no qual o source proxy supre todas as funcionalidades necessárias para dialogar com um objeto do tipo source no servidor. De uma maneira semelhante são associados outros tipos de proxies com outras classes VTK (e.g., filtros mais complexos, widgets 3D, etc.).

As propriedades são utilizadas para representar a chamada a um método (e qualquer argumento a ele associado) de um objeto VTK armazenado no CL, SM, DS ou RS. As propriedades têm um estado interno (atributos que representam os parâmetros de entrada do método) o qual pode ser enviado para o objeto VTK a ela associado. Geralmente, o desenvolvedor não cria diretamente instâncias das propriedades, elas são criadas e atribuídas quando o arquivo e/ou strings de configuração XML são processados pelo ParaView. Na Fig. 3 é apresentado um exemplo simples de configuração de um source proxy com propriedades associadas a ele. Neste exemplo também é utilizado o domínio, o qual especifica a faixa de valores permitidos para esta propriedade.

3.6 Render Server - RS

Este módulo representa todo o processamento requerido para mostrar e renderizar um ou vários conjuntos de dados. Toda vez que um conjunto de dados é carregado ou gerado pelo ParaView, ele é primeiramente filtrado para gerar a uma

---

servidor(es).

```
<SourceProxy name="CubeSource" class="vtkCubeSource">
   <DoubleVectorProperty
       name="XLength"
       command="SetXLength"
       number_of_elements="1"
       default_values="1.0">
       <DoubleRangeDomain name="range" min="0"/>
   </DoubleVectorProperty>
   <DoubleVectorProperty
       name="YLength"
       command="SetYLength"
       number_of_elements="1"
       default_values="1.0">
       <DoubleRangeDomain name="range" min="0"/>
   </DoubleVectorProperty>
</SourceProxy>
```

Fig. 3. Exemplo código XML com propriedades e domínio.

representação poligonal. Posteriormente, é criada uma série de mappers, atores, propriedades e tabela de cores (LookUpTables - LUTs) que permitirão visualizar a representação poligonal destes dados.

Existe no ParaView uma API abstrata para simplificar o processo de agregar novos algoritmos de renderização. O Render Module (representado pela classe `vtkRenderModule` e a sua interface de usuário) é o objeto de renderização de mais alto nível no ParaView. O Render Module é responsável pela criação e gerenciamento de todas as janelas de renderização, manejo dos eventos de renderização e algoritmos de renderização em paralelo. No nosso caso, os algoritmos de renderização disponibilizados pelo ParaView serão suficientes.

3.7 Data Server - DS

O Data Server é o módulo do ParaView responsável por carregar ou criar conjuntos de dados e aplicar filtros nestes dados. Os conjuntos de dados podem ser partidos para ser manipulados em diferentes processos do DS. Se o RS não existe, cada nó do data server é capaz de renderizar os resultados da sua porção do conjunto de dados para que uma imagem composta de todas as partes possa ser mostrada no CL (processo no qual reside a interface de usuário).

Este módulo contém leitores, fontes e filtros VTK. Nele são lidos e processados os dados para finalmente criar as representações geométricas necessários para a renderização. Cada DS tem um pipeline VTK idêntico, e cada processo é indicado a ler a parte dos dados que a ele corresponde manipular.

3.8 Linguagem Cliente-Servidor

A Linguagem Cliente-Servidor do VTK, é uma linguagem independente da plataforma capaz de criar, utilizar (executar métodos) e destruir objetos VTK. Mensagens criadas pelo **cliente** são enviadas ao **servidor** onde elas são interpretadas e

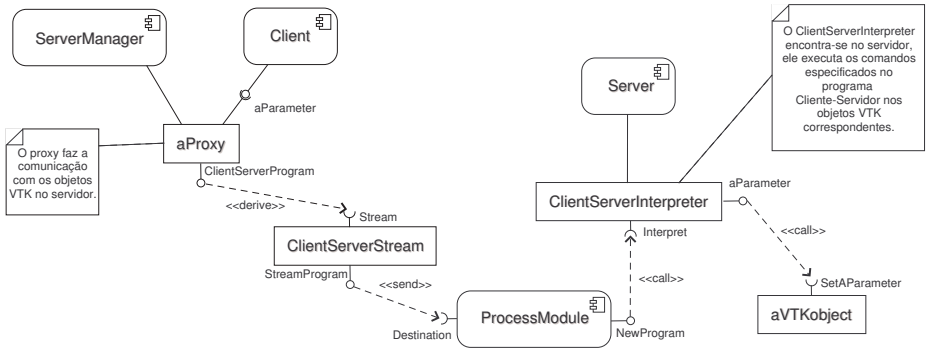

Fig. 4. Comunicação entre Cliente e Servidor.

executadas[5]. Desta maneira, o cliente e o servidor podem-se encontrar no mesmo espaço de memória, em diferentes processos, em diferentes computadores ou até diferentes plataformas. Dentro do ParaView, o SM faz o papel do cliente e os RS, DS e CL cumprem o de servidor.

Os programas Cliente-Servidor VTK são escritos nos *Streams Cliente-Servidor*, os quais são posteriormente interpretador pelo *Intérprete Cliente-Servidor*[6]. Existe uma instância deste interprete em cada processo do ParaView, incluído o SM. O Process Module (`vtkProcessModule`) no SM tem uma instância do stream no qual o programa Cliente-Servidor será escrito. Os objetos no SM (geralmente os proxies), irão escrever estes programas utilizando a linguagem, e posteriormente pedirão ao Process Module para enviar o programa escrito ao servidor correspondente. Estes programas serão recebidos pelos processos correspondentes, onde cada interprete os executa localmente (Fig. 4).

Instâncias do Process Module são criadas no CL e nos Servidores (SM, RS, DS) com diversos propósitos. Elas mantém referências locais aos interpretes de streams e provêem uma interface para chamar métodos em objetos remotos usando a linguagem Cliente-Servidor. Podem também reunir informação dos nós servidores usando objetos de tipo *information* (classe `vtkPVInformation`). Também provêem métodos para obter o número de "partições" e o id de cada uma delas, assim como métodos que geram e retornam CSIDs únicos.

3.8.1 *Streams Cliente-Servidor.* Os Streams Cliente-Servidor (classe `vtkClientServerStream`) são utilizados para comunicar os diferentes processos do ParaView, por meio da linguagem Cliente-Servidor do VTK. Cada instrução nesta linguagem é chamada de mensagem, onde cada uma destas mensagens tem um comando, zero ou mais argumentos e um token `End` (Tab. I). Estas mensagens são armazenados em um formato binário independente de plataforma.

[5]Os termos cliente e servidor neste casso são utilizados de uma maneira sutilmente diferente. Por **cliente** entende-se o processo, módulo ou classe que utiliza esta linguagem para chamar um método em um objeto VTK, o qual é chamado de **servidor**.
[6]Na classes vtkClientServerStream e vtkClientServerInterpreter respectivamente

	comando	argumentos	fim
mensagem 0	`New`	string(vtkObject),id(5)	`End`
mensagem 1	`Invoke`	id(7),string(SetXResolution)	`End`
mensagem 2	`Delete`	id(5)	`End`

Table I. Exemplo de comandos na linguagem Cliente-Servidor do ParaView

4. HEMOLAB - LABORATÓRIO DE MODELAGEM EM HEMODINÂMICA

O nome HeMoLab, significa Laboratório de Modelagem em Hemodinâmica. Este software agrupa dentro de um mesmo ambiente diferentes ferramentas que permitem criar modelos do SCVH. Seguindo os requerimentos mencionados na Seção 2, estas funcionalidades foram implementadas como módulos do software ParaView. Na seção seguinte é descrita a organização dos pacotes e módulos desenvolvidos.

4.1 Pacotes e Módulos

Devido a complexidade dos requerimentos do HeMoLab foi necessário separá-los de acordo com as funcionalidades necessárias em cada um deles. Assim, as funcionalidades foram separadas de duas maneiras diferentes: por pacote e por módulo. Os pacotes seguem o mesmo conceito utilizado pelo VTK para separar classes de acordo ao tipo de funcionalidade que eles provêem, sejam:

—**Common:** agrupa funcionalidades comuns, são classes de infraestrutura que controlam as regras do negócio.

—**Filtering:** reúne filtros implementados no HeMoLab, estes podem ser aplicados em diferentes tipos de dados (superfície, volume, etc.).

—**Graphics:** classes utilizadas para representar geometrias complexas (árvore 1D, malhas de superfície/volume 3D, etc.).

—**GUIClient:** define componentes utilizados na interface de usuário.

—**Imaging:** agrupa os diferentes filtros de imagens.

—**IO:** agrupa as classes de entrada/saída (leitura/escrita).

—**Server Manager:** reúne as classes responsáveis por comunicar os diferentes processos (proxies).

—**Widgets:** classes utilizadas na interação do usuário com as estruturas de dados através de eventos de mouse e de teclado.

Estes pacotes oferecem classes agrupadas de acordo ao tipo de funcionalidade. Assim, estas classes são utilizadas pelos distintos módulos para fornecer funcionalidades mais complexas. Os seguintes módulos foram criados dentro do HeMoLab:

—hmImageProcessing,

—hm1DModule,

—hm3DModule,

—hmCoupling e

—SolverGP.

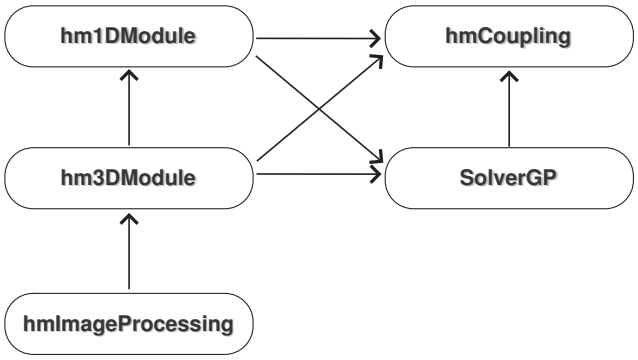

Fig. 5. Módulos do HeMoLab e sua interação.

Estes módulos podem trabalhar de maneira independente ou compartilhar informações segundo seja necessário. Nas seções seguintes estes módulos são estudados em detalhe.

4.2 Módulo de Processamento de Imagens - hmImageProcessing

No processo de modelagem do SCV, é de interesse reproduzir o comportamento deste sistema sob diferentes condições. Para poder gerar informação que seja útil para os médicos, os modelos devem ser ajustados para reproduzir o comportamento específico de um determinado paciente. Neste processo será necessária a incorporação de informação personalizada. Uma importante fonte destas informações são as imagens médicas, as quais podem ser adquiridas de diferentes fontes (MRI, CT, Ultrasom, dentre outras). Serão necessárias, então, ferramentas que permitam trabalhar com imagens médicas de diferentes tipos e em diferentes formatos.

O módulo hmImageProcessing tem por objetivo fornecer estas ferramentas a partir das quais, por exemplo, as geometrias das artérias serão obtidas. Neste caso, uma imagem médica deve ser processada [Bonnet 1996; M.J.Black et al. 1998; Canny 1983; Frangakis and Hegerl 2001; Gonzalez and Woods 2001; Jain 1989] utilizando diferentes filtros de melhoramento de imagens para posteriormente identificar as diferentes regiões nela presentes (processo chamado de segmentação [Larrabide et al. 2003; Held et al. 1997]). As ferramentas providas permitem:

—*Leitura/escrita:* leitores de imagens em diferentes formatos são incorporados permitindo carregar formatos padrão imagens médicas (e.g., DICOM). Para agregar novos formatos, basta simplesmente implementar o leitor/escritor como uma classe VTK e agregar a chamada ao mesmo no arquivo XML correspondente a este módulo. Os escritores (*writers*) podem ser utilizados para salvar imagens processadas (e.g., o usuário deseja salvar uma imagem depois de ter aplicado uma série de filtros). O proprio VTK fornece *writers* de diferentes formatos, os mesmos são agregados usando os arquivos XML.

—*Filtros:* as imagens podem ser processadas utilizando diferentes filtros, por exemplo:

—Suavizado Gaussiano: convolução de uma imagem com um kernel Gaussiano.

Componente (`Classe`)	Responsabilidade
Image Workspace (`vtkPVHMImageWorkspaceWidget`)	Interface de usuário do visualizador de imagens.
Image Workspace Proxy (`vtkSMHMImageWorkspaceWidgetProxy`)	Comunica o CL com um widget 3D no RS.
Image Workspace Widget (`vtkHMImageWorkspaceWidget`)	Permite visualizar os planos da imagem nas direções x,y e z.

Table II. Classes do Módulo de Processamento de Imagens

Este tipo de filtro "suaviza" a imagem e é normalmente utilizado para remoção de ruido [Alvarez et al. 1992].

—Suavizado Anisotrópico: semelhante ao caso anterior, com a particularidade de preservar as bordas na imagem. Este tipo de filtro é utilizado quando se deseja eliminar ruido da imagem mas preservando as características [M.J.Black et al. 1998; Perona and Malik 1990].

—Filtro de Denoising: utilizado para eliminar diferentes tipos de ruido de uma imagem. Dentre eles se considera o filtro baseado na derivada topológica [Larrabide et al. 2005].

—Filtro da Media e da Mediana: são normalmente utilizados para remover outros tipos de ruido, como ruido de impulso, com o qual os filtros mencionados têm dificuldades [Guichard and Morel 2001].

—Filtro Limiar ou "Threshold": este filtro permite selecionar os pixels/voxels da imagem que satisfazem uma determinada condição (i.e., que se encontram dentro de uma faixa de valores) [Jain 1989]. Desta maneira, um valor v_1 é atribuído aos pixels que satisfazem esta condição e um outro valor v_2 ao resto.

—*Segmentação:* é de interesse também, selecionar dentro de uma imagem diferentes objetos ou regiões (processo chamado de segmentação). No caso estes objetos serão as artérias, as suas paredes, um aneurisma, etc. Existem diferentes métodos que permitem realizar este tipo de operação, alguns dos quais são geralmente aplicados às imagens médicas [Banga and Ghorbel 1993; Bezdek 1981; Boscolo et al. 2002]. Os mais utilizados foram contemplados no módulo de processamento de imagens. Também foi incorporado um novo método de segmentação baseado na derivada topológica [Larrabide et al. 2005].

Diferentes pacotes foram utilizados para proporcionar estas funcionalidades, a interação das classes de interface, proxies, filtros e classes VTK, são especificadas utilizando arquivos XML (`hmImagingClient.xml` e `hmImagingServer.xml`) seguindo o mecanismo proposto no ParaView.

O módulo hmImageProcessing esta composto por classes dos pacotes GUIClient, Imaging, Server Manager e Widgets utilizados da seguinte maneira:

—**GUIClient:** Este pacote provê as classes de interface que permitem aos usuários modificar parâmetros e interagir com os filtros no servidor. Para visualizar os planos da imagem, é utilizado *image workspace*, o qual mantém uma referência a um *image workspace proxy* que realiza a comunicação com widget 3D (*image workspace widget*) responsável de mostrar o slice selecionado. O widget 3D pode-se encontrar em um outro processo. O *image workspace* permite ao usuário sele-

cionar/visualizar um slice nas direções principais (i.e., x, y e z). Este componente da GUI é adicionado quando se deseja visualizar o resultado de um determinado filtro. Além do *image workspace*, cada filtro irá ter associado a ele uma ou mais abas que permitam controlar os seus parâmetros.

—**Imaging:** Os filtros específicos para processamento de imagens desenvolvidos especificamente para este projeto vem deste pacote, dentre eles: D_T Restoration (remoção de ruido por meio da derivada topológica), Voxel Grow (Segmentação por crescimento de regiões), D_T Segmentation (segmentação utilizando a derivada topológica). Eles são implementados como filtros VTK e posteriormente adicionados por meio dos arquivos XML.

—**Server Manager:** O *image workspace* precisa de um *image workspace proxy* para se comunicar com a classe responsável pela visualização da imagem em 3D. Assim, este proxy realiza a conexão entre o CL e o RS (no qual se encontra o widget 3D).

—**Widgets:** Para visualizar a imagem, foi desenvolvido um widget 3D específico. O *image workspace widget* utiliza 3 diferentes instâncias da classe `vtkImagePlaneWidget` para mostrar cortes da imagem nas direções principais (x, y e z). O único objetivo deste widget é a visualização da imagem (por planos) e ele não produz nenhum tipo de processamento/filtrado na imagem. A associação deste widget a uma imagem qualquer, e.g., a saída de um filtro aplicado em uma outra imagem, é feita no arquivo `hmImagingClient.xml` ao se associar a um filtro um *image workspace*. Assim, automaticamente a saída do filtro será visualizada usando este widget 3D. Caso o mesmo não seja utilizado, somente a bounding box da imagem é visualizada.

Pretende-se incorporar novos filtros de processamento de imagens neste módulo. Existe também a possibilidade de se agregar filtros de outras bibliotecas (e.g., ITK [Ibanez et al. 2005]). Nesta biblioteca existem implementados alguns dos filtros de imagens e algoritmos de segmentação mais utilizados.

4.3 Módulo 1D - hm1DModule

Os modelos 1D [Hughes and Lubliner 1973; Olufsen 1998; Formaglia et al. 2003; Urquiza et al. 2006] permitem representar de maneira simplificada o fluxo sanguíneo nas maiores artérias do corpo humano. Desta maneira, pode ser de interesse para os usuários gerar modelos personalizados 1D e estudar a resposta no comportamento do SCVH em diferentes condições (e.g., ao se modificar propriedades mecânicas do modelo, ao se modificar a geometria da árvore, etc.). Para suprir estes requerimentos, deve-se permitir ao usuário gerar modelos 1D personalizados e modificar os já existentes. Estes modelos personalizados serão a entrada do SolverGP (resolvedor numérico, descrito em maior detalhe na Seção 4.6), o qual realizará a correspondente simulação (independentemente da existência de uma geometria 3D associada). Isto evidencia a existência de uma relação direta entre o hm1DModule e o SolverGP (leitura/escrita do formato dos arquivos do deste solver).

O módulo hm1DModule permite ler, editar propriedades, modificar a geometria e salvar modelos da árvore 1D no formato do SolverGP. Esta característica permite a geração de novos modelos personalizados assim como a edição dos já existentes.

Componente (`Classe`)	Responsabilidade
1D Straight Model (`vtkHM1DStraightModel`)	Armazena a estrutura de árvore e gerencia a maneira na qual ela é criada.
Segment (`vtkHM1DSegment`)	Armazena informação correspondente ao segmento (incluindo seus elementos e nós).
Terminal (`vtkHM1DTerminal`)	Armazena informação correspondente ao terminal.
1D Straight Model Grid (`vtkHM1DStraightModelGrid`)	Armazena a representação geométrica da árvore 1D.
1D Straight Model Source (`vtkHM1DStraightModelSource`)	Gera, a partir do 1D Straight Model, o grid que o representa.
1D Straight Model Reader (`vtkHM1DStraightModelReader`)	Gerencia a leitura dos arquivos de dados do solver.
1D Straight Model Writer (`vtkHM1DStraightModelWriter`)	Gerencia a escrita dos arquivos de dados do solver.
Segment Proxy (`vtkSMHM1DSegmentProxy`)	Comunica o CL com um segmento no DS.
Terminal Proxy (`vtkSMHM1DTerminalProxy`)	Comunica o CL com um terminal no DS.
Straight Model Widget Proxy (`vtkSMHMStraightModelWidgetProxy`)	Comunica o CL com um widget 3D no RS.
Segment Widget (`vtkPVHMSegmentWidget`)	Oferece a interface correspondente aos parâmetros dos segmentos.
Terminal Widget (`vtkPVHMTerminalWidget`)	Oferece a interface correspondente aos parâmetros dos terminais.
Element Widget (`vtkPVHMElementWidget`)	Oferece a interface correspondente aos parâmetros dos elementos.
Heart Widget (`vtkPVHMHeartWidget`)	Oferece a interface correspondente aos parâmetros do coração.
Straight Model Source Widget (`vtkPVStraightModelSourceWidget`)	Oferece a interface correspondente aos parâmetros da árvore 1D.
Straight Model Source 3D Widget (`vtkHM1DStraightModelWidget`)	Apresenta a árvore 1D ao usuário.
XY Plot Widget (`vtkKWHMXYPlotwidget`)	Visualiza as curvas dos resultados no gráfico.

Table III. Classes do Módulo 1D

Os pacotes e correspondentes componentes utilizados por este módulo são determinados nos arquivos `hm1DModuleClient.xml` e `hm1DModuleServer.xml`. No arquivo `hm1DModuleClient.xml` é chamado o componente da interface de usuário utilizado para controlar a interação com as classes no servidor, e caso for necessário as propriedades que ele utiliza. O arquivo `hm1DModuleServer.xml` especifica as relações entre proxies e objetos VTK no servidor assim como as propriedades associadas a estes proxies.

Os diferentes componentes deste módulo encontram-se distribuídas, de acordo com a sua funcionalidade nos diferentes pacotes. Os pacotes utilizados por este módulo são Common, Graphics, GUIClient, IO, Server Manager e Widgets.

—**Common:** Para representar a estrutura de dados de uma árvore 1D diferentes componentes foram desenvolvidos. O *1D straight model* é responsável por gerenciar a maneira na qual uma árvores 1D é edição ou criada, ele é criado no

DS. Nesta classe existem implementadas as regras que controlam a criação e inserção de novos *segmentos arteriais* e *terminais* na árvore. Seguindo a natureza da árvore arterial, a mesma foi desenhada utilizando o modelo "Composite" [Gamma et al. 1995], este permite tratar os elementos da árvore independentemente do seu tipo (*terminais* ou *segmentos*). Os *segmentos* e *terminais* armazenam as informações que correspondem ao modelo, i.e., propriedades mecânicas e geométricas da árvore e dos seus elementos e informações próprias da malha de elementos finitos utilizada para achar as soluções das equações 1D correspondentes.
Existem também diferentes restrições na maneira na qual novos *segmentos* e *terminais* podem ser adicionados na árvore. No caso de agregar um novo *segmento* na árvore, este deve ter no seu extremo (aquele que não se encontra conectado ao resto da árvore) um *terminal*[7]. Este *terminal* é incorporado automaticamente pelo *1D straight model*. Quando neste modelo são conectados dois *segmentos* com diferentes diâmetros, é colocado entre eles um *terminal*. Este é utilizado para modelar a queda de pressão que ocorre entre os dois *segmentos*. No caso dos dois *segmentos* ter o mesmo diâmetro, o *terminal* não é necessário. Estas restrições são levadas em consideração no momento de adicionar *segmentos* e *terminais* pelo *1D straight model*.

—**Graphics:** Para poder visualizar a árvore, foi necessário criar uma representação geométrica dele. Esta representação é montada a partir das informações existentes no *1D straight model*, o qual armazena comprimento e posição dos *segmentos* e *terminais* da árvore. Desta maneira a árvore arterial é percorrida para gerar as representações dos seus *segmentos* (estes compostos de elementos[8] e nós) e *terminais*. Esta representação é armazenada pelo *1D straight model grid* (extensão do poly data de VTK), a qual representa os diferentes componentes da árvore (*segmentos*, *terminais*, nós e elementos) como pontos e linhas dentro de um cell array. A estas linhas e pontos é atribuída uma propriedade (um escalar) indicando o seu tipo. O *1D straight model source* é o responsável por gerar o *1D straight model grid* a partir do *1D straight model*. Desta maneira, a dependência entre a os dados e a sua representação fica centralizada nesta classe.

—**GUIClient:** As classes neste pacote são responsáveis por mostrar propriedades dos componentes da árvore assim como permitir ao usuário modifica-las. Neste caso as abas utilizadas são *element widget*, *segment widget*, *heart widget*, *terminal widget* e *straight model source widget*. Quando uma árvore é aberta (ou criada), um *straight model source widget* é criado pelo ParaView (esta associação é especificada no arquivo XML do cliente, descritos na Seção 3.5). Esta irá mostrar as propriedades (usando instâncias das abas mencionadas, i.e., *element widget*, *segment widget*, *heart widget* e *terminal widget*) para mostrar as propriedades dependendo da interação do usuário, do modo de operação no qual ele se encontre (edição ou visualização) e do componente que ele tenha selecionado. Estas classes são painéis compostos com "labels", "text boxes" e "botões" dentre outras abas

[7]Este terminal representa a resistência oferecida pelo resto do sistema depois daquele ponto.
[8]Por elemento entende-se elemento finito, onde as propriedades destes podem ser alteradas. Os nós armazenam informações geométricas complementares como diâmetro do segmento naquele ponto.

simples que permitem editar e visualizar os valores correspondentes aos diferentes componentes. Uma vez que os valores são modificados, estes são armazenados em variables internas.

O *heart widget* permite visualizar/editar as propriedades do coração. Neste tipo de modelo o coração é representado com um *terminal* "especial" que injeta no sistema uma quantidade de sangue determinada por uma curva. Esta curva de ejeção muda de um indivíduo para outro. Assim, é provida uma ferramenta que permite ajustar a curva de acordo com o observado no paciente.

Quando deseja-se enviar estas modificações ao servidor, a informação é passada utilizando proxies ou propriedades. Um proxy é utilizado quando a operação é de maior complexidade (e.g., se requer efetuar uma seqüencia de chamadas a métodos). As propriedades são utilizadas quando a interação é simples.

—**IO:** Para poder armazenar de maneira persistente árvores criados ou poder ler árvores já existentes, estas informações devem poder ser salvas de maneira persistente (i.e., em arquivos). Diferentes formatos podem ser adotados para salvar estas informações (XML, CSV, etc.), no entanto em uma primeira etapa, o principal objetivo é compartilhar estas informações com o SolverGP (Sec. 4.6). Este software, pensado para solução de problemas discretos, permite um uso simples de diferentes tipos de elementos finitos (i.e., diferentes modelos). Neste contexto, a definição das malhas de elementos finitos, das suas propriedades, condições de contorno e condições de iniciais são definidas através de arquivos de texto formatados. Para ler/escrever estes arquivos foram desenvolvidas diferentes classes no pacote IO. O componente principal para a leitura é o *1D straight model reader* (extensão do data reader do VTK), responsável por coordenar a leitura dos diferentes arquivos (feita por classes especificas) e gerar o correspondente *1D straight model*. De forma equivalente existe um componente responsável pela escrita (*1D straight model writer*, o qual gera os arquivos do modelo com geometria, propriedades mecânicas, condições iniciais e configuração do algoritmo de resolução). Desta maneira é feita a interação entre o módulo 1D e o SolverGP.

—**Server Manager:** A comunicação entre o cliente e os diferentes servidores (RS e DS) é feita pelo SM. Como mencionado anteriormente, no SM coexistem as classes responsáveis por esta comunicação, os *proxies*. A comunicação entre o CL e o DS, é feita pelos proxies correspondentes aos *segmentos* e *terminais* (*segment proxy* e *terminal proxy* respectivamente) os quais permitem trocar informações entre estes processos. Assim, estas classes são utilizadas para se comunicar com os componentes da árvore que se encontram no *1D straight model* no servidor. As informações são enviadas pelos *proxies* utilizado *streams*. Para identificar o objeto no qual um método deve ser chamado, o *CSID* daquele objeto é utilizado. Desta maneira, no cliente somente são criados um *segment proxy* e um *terminal proxy*, os quais são "apontados" para diferentes *segmentos* ou *terminais* usando o *CSID* correspondente.

Por outro lado, existe a necessidade de comunicar o CL com o RS (para identificar diferentes interações do usuário com o widget 3D da árvore e os seus componentes). Esta comunicação é responsabilidade do *straight model widget proxy*, o qual envia informações (principalmente de configuração da visualização) da árvore ao correspondente widget no RS. Também, esta classe armazena re-

ferências aos *proxies de segmentos e terminais*, utilizados para interagir com elementos da árvore no DS.

—**Widgets:** A representação visual da árvore 1D é feita utilizando um widget 3D especial. Este widget, *straight model source 3D widget*, é responsável por controlar a interação com a árvore 1D, seleção de elementos da árvore (segmentos, terminais, elementos e nós), assim como a configuração e gerenciamento dos gráfico para visualização de resultados (gráficos 2D correspondentes a área, pressão, fluxo, etc.). Utilizou-se o esquema de *callbacks* (implementando o pattern "Observer" [Gamma et al. 1995]) do VTK, e implementado um mecanismo que permite ao usuário selecionar diferentes componentes da árvore. Cada componente da árvore é representado por um ator diferente, ao qual encontra-se associado o *CSID* do componente correspondente na árvore 1D no DS. Quando um determinado componente (representado por um ator) é selecionado no widget 3D, o seu CSID é identificado. Desta maneira é possível "dialogar" com todos os elementos da árvore, podendo-se obter e/ou modificar as suas informações. O widget 3D permite a seleção de *segmentos* (assim como dos seus nós e elementos) e de *terminais*. Ao selecionar um *segmento*, por exemplo, as suas informações são mostradas na interface permitindo visualizar e/ou modificar o seu valor.
Este widget 3D gerencia também a visualização dos resultados do processamento da árvore. O *xy plot widget* é utilizado pelo *straight model source 3D widget* para mostrar as curvas correspondentes a pressão, fluxo, área, etc., permitindo selecionar trechos do gráfico para estudar detalhes destas curvas.

Desta maneira, o módulo hm1DModule é responsável pela criação/ edição/ visualização de resultados de modelos 1D. Estes modelos são lidos/armazenados em um formato específico. Além da leitura de arquivos de dados, é feita a leitura dos resultados da simulação, permitindo a visualização e análise destes resultados.

4.4 Módulo 3D - hm3DModule

O objetivo do Módulo 3D é prover ferramentas que permitam, a partir de uma geometria primeira aproximação da artéria ou região de interesse, gerar uma malha de elementos finitos de boa qualidade, adequadamente refinada e ainda permitam gerar os arquivos correspondentes a este problema prontos para realizar a simulação computacional (utilizando o SolverGP, Seção 4.6).

Neste módulo são reunidos diferentes filtros para processamento de superfície os quais permitem processar a triangulação de uma geometria para melhorar a sua qualidade. Assim, uma vez que a região de interesse foi isolada em uma imagem médica, é reconstruída a superfície (triangulação) que caracteriza aquela região. Esta primeira triangulação da geometria costuma ser de má qualidade e precisa de um processamento adequado. Para realizar o processamento das malhas de elementos finitos, foram incluídos diferentes métodos clássicos de geração e otimização de malhas não estruturadas (triangulares e tetraédricas). Estes algoritmos encontram-se implementados em um software chamado Gemesis3D [Vénere 1996]. As funcionalidades oferecidas por este software foram adicionadas e disponibilizadas dentro do módulo 3D. Depois que a superfície for melhorada, é gerada uma malha de volume (utilizando o algoritmo de Delaunay [Baker 1989] para geração de malhas de volume), posteriormente a malha de volume (composta de tetraedros) é

Componente (`Classe`)	Responsabilidade
Full Model (`vtkHMFullModel`)	Armazenar informações correspondentes a um modelo completo.
Mesh (`vtkHMMesh`)	Armazenar a geometria da malha (superfície ou volume).
Surface Reader (`vtkHMSurfaceReader`)	Leitor de arquivos ".sur" (pontos e conectividade).
Surface Writer (`vtkHMSurfaceWriter`)	Escritor de arquivos ".sur" (pontos e conectividade).
Volume Reader (`vtkHMVolumeReader`)	Leitor de arquivos ".vwm" (pontos e conectividade).
Volume Writer (`vtkHMVolumeWriter`)	Escritor de arquivos ".vwm" (pontos e conectividade).
Trisurf Widget (`vtkPVHM3DTrisurfFilter`)	Oferece a interface correspondente dos filtros/parâmetros do Trisurf.
Dela 3D Widget (`vtkPVHM3DDelaFilter`)	Oferece a interface correspondente aos parâmetros do Dela3D.
Qopt Widget (`vtkPVHM3DQoptFilter`)	Oferece a interface correspondente aos parâmetros do Qopt.

Table IV. Classes do Módulo 3D

otimizada [Vénere 1997].

Obtidas as malhas de elementos finitos, pode ser de interesse para o usuário realizar a simulação computacional somente do modelo 3D (impondo condições de contorno, é possível realizar a simulação do modelo 3D isolado). Isto pode ser útil em diferentes circunstâncias, e.g., geração das condições iniciais do modelo acoplado. Neste caso devem ser fornecidas as condições no contorno nas entradas/ saídas da geometria. Assim, o módulo hm3DModule pode gerar os arquivos do modelo 3D independente da árvore 1D (arquivos de entrada do SolverGP).

As classes e pacotes utilizadas para fornecer estas funcionalidades são especificadas nos arquivos `hm3DModuleClient.xml` e `hm3DModuleServer.xml`. No arquivo do cliente são definidas as classes de interface utilizadas para criar filtros VTK e controlar seus parâmetros. O arquivo XML do servidor irá relacionar os diferentes proxies com os filtros e classes VTK no servidor.

As funcionalidades deste módulo são implementadas utilizando classes dos pacotes Common, Filtering, Graphics, GUIClient, IO e Server Manager.

—**Common:** Neste pacote encontra-se o componente responsável por manter e armazenar informações correspondentes a um modelo 3D (completo), chamado de *full model*. Por modelo completo entende-se aquele no qual as equações correspondentes (fluido o sólido) são modeladas de maneira completa e sem hipóteses simplificativas que reduzem a dimensionalidade do problema, como no caso do modelo 1D. Um exemplo de isto são as equações de Navier-Stokes utilizadas para modelar o comportamento de um fluido em um distrito arbitrário. No caso, *full model* armazena a informação da malha, grupos de elementos e parâmetros associados ao modelo. Logo, cada grupo tem informação de quais elementos correspondem ao grupo assim como dos campos escalares/ vetoriais/ tensoriais para descrever as propriedades a ele associadas. Pode-se observar que este modelo se

mantém genérico e flexível, permitindo tanto representar a malha de elementos finitos da parede arterial (neste caso as mesmas poderiam utilizar um modelo de membrana, de cascara ou de solido conforme o caso) como a correspondente ao fluido (no caso para as equações de Navier-Stokes).

—**Filtering:** Como o funcionamento do Gemesis3D é baseado em um esquema de dados e filtros aplicados nestes dados, foi relativamente fácil incorporar as suas funcionalidades para trabalhar em um ambiente baseado em VTK. Os filtros deste módulo podem ser diferenciados de acordo com o tipo de malha processada, as que podem ser caracterizadas como sendo: de superfície (malhas triangulares) ou de volume (malhas tetraédricas).
Os filtros de superfície incluem funcionalidades que permitem otimizar a qualidade da malha de acordo com diferentes critérios (estes filtros encontram-se no modulo Trisurf do Gemesis 3D). Este filtros introduzem modificações na forma e topologia da malha. Dentre eles: filtro de suavizado, divisão de triângulos obtusos, inserção de nós (refinamento), colapsar triângulos pequenos (satisfazendo um certo critério de tamanho), dentre outros. A aplicação sucessiva (com bom critério por parte do usuário) destes filtros na malha irá melhorar a sua qualidade, até se obter uma malha de superfície com qualidade suficiente.
Uma vez que a malha de superfície for melhorada, uma malha de volume (tetraedros) é gerada a partir dela. Para gerar esta malha volumétrica, são utilizados dos algoritmos diferentes: método Delaunay [Delaunay 1934; Vénere and Dari 1990], método Frontal [Peraire et al. 1988; Peraire et al. 1991] e método de Octrees [Frey et al. 1994; Shephard and Georges 1991; Yerry and Shephard 1984] (módulo Dela3D do Gemesis 3D). Estes métodos são amplamente utilizados para geração de malhas não estruturadas de elementos finitos e se mostram eficientes na geração deste tipo de malhas. No entanto os resultados destes métodos não necessariamente resultam em malhas de volume de boa qualidade. Aparece então a necessidade de otimizar estas malhas. Para isto é utilizado um otimizador de malhas de volume [Vénere 1997] (módulo Qopt do Gemesis 3D). Estas funcionalidades são implementadas no software Gemesis 3D e foram adaptadas utilizando o esquema "Adapter" [Gamma et al. 1995] de maneira a prover uma interface transparente para poder acessar estas funcionalidades.

—**Graphics:** As classes de representação de geometria são incluídas neste pacote. Em particular o componente *mesh* (é uma especialização do unstructured grid do VTK), irá armazenar a malha formada por coordenadas e conectividade. Ao aplicar um filtro nesta malha, esta será modificada gerando-se uma nova malha (processada) na saída. As classes neste pacote são responsáveis por armazenar a representação da malha (seja de superfície ou de volume).

—**GUIClient:** Para permitir ao usuário escolher a operação/filtro a se realizar na geometria foram desenvolvidos 3 componentes: *trisurf widget*, *dela 3D widget* e *qopt widget*. No caso do *trisurf widget* oferece a interface para escolher uma das diferentes operações possíveis em uma superfície (e.g., suavizado de superfície, divisão de triângulos obtusos, inserir nós para refinar a malha, dentre outros). Cada uma destas funções têm diferentes parâmetros, os quais são requeridos ao usuário pelo componente. O *dela 3D widget* permite definir tamanho característico dos triângulos de acordo ao grupo. Com estas informações a malha de volume é ge-

rada. O *qopt widget* permite ver e analisar de maneira quantitativa a qualidade da malha em valores médios (elemento médio) e absolutos (pior elemento). O filtro Qopt irá analisar a malha para intentar melhora-la operando por "clusters" de elementos, o número máximo de elementos neste cluster é o parâmetro que controla o algoritmo. Este parâmetro é controlado usando a interface.

—**IO:** O software Gemesis3D tem definido um formato de arquivos para representar superfícies. Estes arquivos baseiam-se simplesmente em descrever pontos e conectividade da malha. Além disto, estes arquivos permitem definir grupos de elementos, por exemplo, de acordo com o seu tipo. Para poder utilizar geometrias já existentes e exportar geometrias para serem processadas com outras versões do Gemesis 3D, foram implementados módulos de leitura/escrita deste formato. Estes são o *surface reader/writer* para leitura e escrita de superficies (arquivos ".sur") e volume reader/writer para leitura e escrita de volumes (arquivos ".vwm"). Além de permitir ler/escrever arquivos com as geometrias, foi implementada a funcionalidade de ler/escrever os modelos completos (incluindo condições de contorno, propriedades mecânicas, condições iniciais e configuração do algoritmo de resolução) no formato do SolverGP para que estes possam ser analisados (i.e., simulação).

—**Server Manager:** Neste caso, o *source proxy* já implementado no ParaView foi suficiente.

Como no caso do módulo 1D, o módulo 3D pode trabalhar de maneira independente dos outros módulos, o que da maior versatilidade à ferramenta. Para a visualização de resultados destas simulações são atualmente utilizados os mecanismos providos pelo ParaView (e.g., visualização de campos escalares por meio de tabelas de cores, representação de campos vetoriais com glyphs ou stream tubes, representação de deslocamentos por meio de deformações na malha, dentre outros). Mesmo assim novos mecanismos podem ser facilmente adicionados assim que sejam necessários. O calculo de tensões nas paredes arteriais (Wall Shear Stress - WSS) e tensões oscilatórias (Oscillating Shear Stress Index - OSI) deverá ser incorporado posteriormente.

4.5 Módulo de Acoplamento - hmCoupling

O fluxo do sangue no corpo humano é reproduzido (de maneira aproximada) utilizando dois modelos com características diferentes: o modelo 1D e o modelo 3D. Cada um deles tem características e informações próprias que, ao ser unidos, representam de maneira integral o comportamento do sistema cardiovascular.

Para a geração de modelos personalizados acoplados 1D-3D, foi criado o módulo hmCoupling. Este tipo de modelos acoplados, fornece informações locais da hemodinâmica (e.g., detalhe do fluxo sanguíneo na bifurcação da carótida, estados de tensões nas paredes das artérias, etc.) consistentes com as condições (fluxo, pressão, etc.) providas pelo resto do sistema arterial. Os modelos acoplados (ou multidimensionais [Formaggia et al. 1999; Formaggia et al. 2002; Lamponi 2004; Urquiza et al. 2006]) requerem associar terminais do modelo 1D com grupos de elementos no modelo 3D. Para realizar esta associação, o módulo hm3DModule permite a criação de grupos de elementos, os quais serão associados com terminais no modelo 1D. Armazenar esta informação, assim como salvá-la em forma de um

Componente (`Classe`)	Responsabilidade
Full Model to Straight Model Coupling (`vtkHMFullModelTo1DStraightModelCoupling`)	Armazenar informações correspondentes a um modelo completo.
Full Model to Straight Model Coupling Widget (`vtkPVHMFullToStraightModelCoupling`)	Oferece a interface para associar grupos no modleo 3D e terminais no modelo 1D.
Full Model to Straight Model Coupling Reader (`vtkHMFullModelTo1DStraightModelCouplingReader`)	Leitor de modelos acoplados.
Full Model to Straight Model Coupling Writer (`vtkHMFullModelTo1DStraightModelCouplingWriter`)	Escritor de modelos acoplados.
Full Model to Straight Model Coupling Proxy (`vtkSMHMCouplingProxy`)	Comunica o CL e o DS.

Table V. Classes do Módulo 3D

modelo personalizado integrado, è responsabilidade do módulo hmCoupling. O formato utilizado para salvar o modelo é o do SolverPG, o qual contempla esta situação (modelos acoplados 1D-3D).

Para poder fazer com que estes dois modelos interajam, é necessário especificar quais grupos de elementos do modelo 3D irão ser conectados com quais terminais do modelo 1D. Todas estas funcionalidades são implementadas utilizando os pacotes Common, GUIClient, IO e Server Manager.

—**Common:** A informação propria do acoplamento é armazenada utilizando o *full model to straight model coupling*. Para cada acoplamento é armazenada a informação correspondente (no caso, direção da conexão, referência ao *full model* e o numero de grupo, referência ao *1D straight model* e o id do terminal no modelo 1D). Esta é toda a informação necessária para poder gerar a especificação do modelo acoplado.

—**GUIClient:** Para associar grupos no *full model* e terminais no *1D straight model* é utilizado o *full model to straight model coupling widget*. Assim, são escolhidos o *1D straight model* e o *full model*. Logo, por cada acoplamento deve ser relacionado um terminal no *1D straight model* e um grupo no *full model*. Esta informação é enviada ao servidor utilizando o proxy correspondente (*full model to straight model coupling proxy*).

—**IO:** Uma vez especificado do modelo acoplado o mesmo deve ser salvo no formato do SolverGP. Para isto foram criados dois componentes, um de leitura e um de escrita para este salvar as informações do modelo.

—**Server Manager:** As informações fornecidas pelo usuário são enviadas às classes correspondentes no servidor (*full model to straight model coupling*, responsável por armazenar as informações dos acoplamentos). De forma semelhante ao feito nos outros módulos, esta responsabilidade corresponde aos proxies. O componente *full model to straight model coupling proxy* irá comunicar estas informações às classes no servidor. Uma vez que as informações estejam completas o sistema permite salvar a especificação do modelo acoplado.

Os modelos multidimensionais cobram uma grande importância devido à sua versatilidade. Com este tipo de formulação é possível obter informação detalhada

do fluxo sanguíneo em qualquer região do corpo sendo a mesma consistente com e respondendo à interação com o resto do sistema.

4.6 Solver numérico - SolverGP

A simulação do fluxo sanguíneo é feita por meio do uso de Dinâmica dos Fluidos Computacional (Computational Fluid Dynamics - CFD [Donea and Huerta 2003; Girault and Raviart 1986; Temam 1979]) e o Método dos Elementos Finitos (Finite Element Method - FEM). Para achar uma aproximação numérica dos modelos do SCV, é utilizado um software de calculo numérico chamado Solver GP [Urquiza and Vénere 2002]. Este software pode ser estendido para considerar problemas não lineares, interação fluido estrutura, acoplamento entre modelos 1D e 3D, dentre outras.

O SolverGP foi pensado como uma arquitetura genérica, a qual permite implementar facilmente solvers numéricos para métodos como FEM, FDM (Finite Difference Method), FVM (Finite Volume Method), assim como solvers para outros métodos discretos como redes elétricas e redes neurais, etc., tanto para problemas estacionários como transientes. Além disso, é possível implementar qualquer método que resulte em um sistema de equações algébricas que possa ser construído a partir de contribuições sucessivas de operadores locais (matrices elementares, i.e., em uma matriz global são somados subconjuntos de coeficientes correspondentes a contribuição dos diferentes elementos). A versatilidade deste aplicativo baseia-se na capacidade de um ensamblador simbólico e numérico que opera com matrizes elementares de qualquer tipo com um número arbitrário de graus de liberdade por nó. Este framework alcança um alto grau de reutilização. A única tarefa delegada ao usuário é a implementação das matrices elementares e a sua estrutura de acoplamento. Neste caso os elementos utilizados já existem implementados no SolverGP.

Uma vez definidas as equações utilizadas por um determinado modelo, as mesmas são utilizadas nos elementos correspondentes da malha. Todas estas configurações são feitas através de arquivos de texto. Os arquivos de entrada deste software estão divididos pelos dados que eles contém, sendo estes:

—Parâmetros básicos - (*Basparam.txt*): Contém parâmetros básicos do solver, os mesmos encontram-se definidos por grupo de elementos. Neste arquivo existe o chamado aos elementos correspondentes a cada grupo cada um dos quais estão especificados na biblioteca de elementos do solver [Urquiza and Vénere 2002]. Também contém as configurações do resolvedor numérico.

—Geometria - (*Mesh.txt*): Este arquivo define a geometria do problema. Contém a malha de elementos finitos.

—Parâmetros da malha - (*Param.txt*): Armazena os parâmetros utilizados por cada elemento. Estes parâmetros mudam de acordo com o modelo associado ao elemento.

—Condições iniciais - (*Inifile.txt*): Indica para cada nó da malha a condição inicial associada a ele. Utilizado para problemas transientes.

—Resultados - (*Dataout.txt*): Arquivo de saída. Contem os resultados da simulação gerados pelo Solver GP.

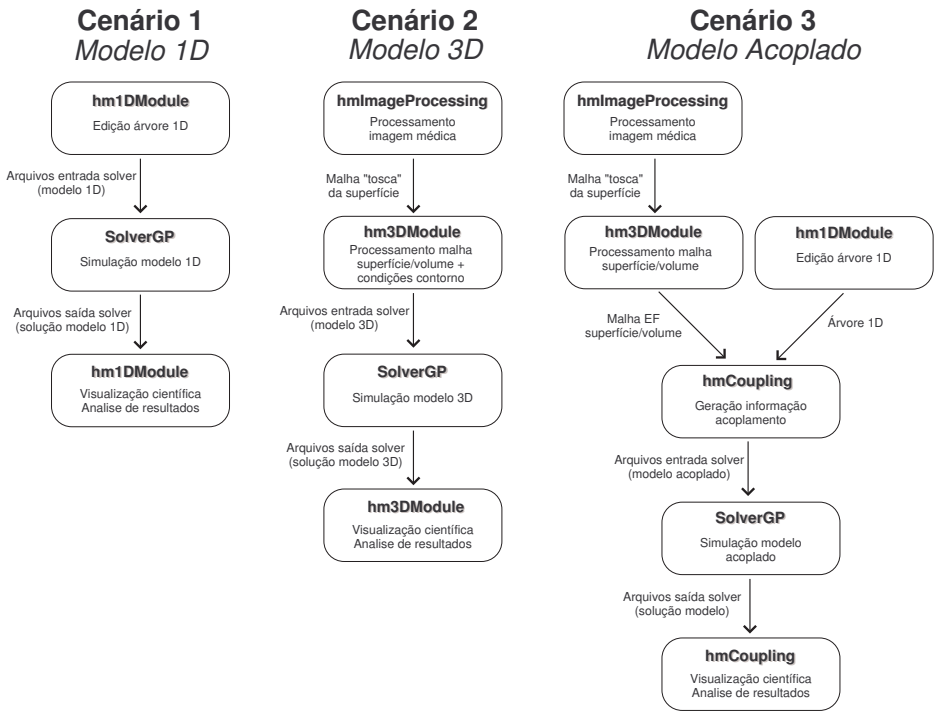

Fig. 6. Três exemplos de cenários.

4.7 Cenários

O sistema HeMoLab foi pensado de maneira modular. Assim, dependendo do objetivo do usuário, este poderá utilizar um ou mais módulos do sistema, sem estar restrito à utilização de um único tipo de modelo. Na operação do sistema estes módulos podem ser utilizados em 3 diferentes cenários (Fig. 6):

4.7.1 *Cenário 1 - Modelo 1D.* Caso o usuário deseje simplesmente analisar a resposta do sistema cardiovascular a uma perturbação determinada (na geometria, nas propriedades mecânicas, etc.), este pode realizar a modificação desejada e analisar o modelo. Por exemplo, o médico deseja ajustar o modelo a um determinado paciente. Neste caso, diversas medidas (pulso, pressão) são feitas em diferentes pontos do paciente. Ajustando os parâmetros e propriedades mecânicas/geométricas do modelo, o especialista procura a combinação de valores que irá produz o melhor ajuste ao observado no paciente.

Neste caso, o hm1DModule é utilizado para edição da árvore 1D. Uma vez feitas as modificações são gerados os arquivos de entrada do SolverGP (contendo o modelo). Assim, o SolverGP computa a solução do modelo gerando os resultados. Estes resultados são finalmente analisados pelo hm1DModule. Uma vez que o modelo 1D esta ajustado e aproxima o observado no paciente, o especialista pode partir para a geração do modelo 3D.

4.7.2 *Cenário 2 - Modelo 3D.* Se o único interesse é estudar condições do fluxo em uma determinada geometria, a mesma pode ser analisada de maneira independente. A geometria pode ser gerada a partir de uma imagem médica do paciente ou lida de um arquivo (e.g., ".sur"). Isto é de interesse, por exemplo, para gerar condições iniciais para um modelo acoplado. A geometria é processada (melhoramento de superfície, geração da malha de volume, melhoramento da malha de volume), posteriormente são determinadas as suas propriedades mecânicas, e gerados os arquivos correspondentes ao modelo 3D (entrada do SolverGP).

O hm3DModule é utilizado para gerar/ otimizar a malha de elementos finitos e gerar a especificação do modelo pronto para o SolverGP calcular a sua solução. Dependendo do caso, os resultados podem ser diretamente analisados ou utilizados como condição inicial no caso acoplado.

4.7.3 *Cenário 3 - Modelo Acoplado.* Talvez o cenário mais interessante seja o que permite estudar de maneira integrada o sistema arterial. Esta opção permite estudar o comportamento detalhado das condições do fluxo (e.g., regiões de recirculação, tensões de corte e tensões oscilatórias, etc.) em um determinado distrito do SCVH consistente com as condições fornecidas pelo resto do sistema. Neste caso, devem ser definidos os pontos nos quais os modelos de diferente dimensionalidade estão conectados. Obtida (do usuário) a informação do acoplamento, os arquivos podem ser gerados para simular o modelo.

Este modelo pode ser utilizado com diversas finalidades, por exemplo para estudar o risco de ruptura de aneurisma. Neste caso, as imagens do paciente são analisadas para extrair a geometria das artérias e do aneurisma. Esta geometria e processada utilizando o hm3DModule para gerar uma malha de elementos finitos adequada e condições iniciais para o campo de pressão e velocidade dentro daquele distrito. Paralelamente, utilizando o hm1DModule a árvore 1D é ajustada para reproduzir as condições observadas no paciente. Estas duas informações são agregadas utilizando o módulo hmCoupling e a simulação é feita. Estes resultados irão complementar a informação da qual o especialista dispõe para tomar suas decisões.

5. TRABALHO FUTURO

No HeMoLab, foram considerados modelos 1D de tubos retos, modelos 3D de geometrias arbitrarias e modelos acoplados multidimensionais. Na medida que novos modelos sejam desenvolvidos, o interesse é adiciona-los como novos plugins. Dependendo da representação destes modelos, serão necessárias novas ferramentas para edição de geometria e parâmetros.

Para a geração e refinamento de malhas elementos finitos são utilizadas ferramentas relativamente simples. Em alguns casos, é de interesse gerar malhas mais refinadas en determinadas regiões. O desenvolvimento de melhores ferramentas para gerar estas malhas assim como a utilização de refinamento adaptativo é uma necessidade importante a ser considerada em próximas versões. Uma outra ferramenta útil a ser considerada e a possibilidade de editar nós e elementos de maneira manual.

Em particular, umas das mais diretas aplicações e utilidades do software correspondem a: 1) predição de ruptura de aneurismas, 2) colocação de stents. No caso de aneurismas, são utilizados diferentes indices para estudar a probabilidade de rup-

tura (analise de tensões), os quais ainda devem ser incorporados no software. Para estudar o impacto/resultado de uma intervenção com stent, é necessária a criação de uma biblioteca de geometrias de stents, permitindo ao especialista experimentar com varias delas na geometria do paciente, para achar a mais adequada, estudar a pressão necessaria para colocá-lo no sitio (para que não fique solto, nem provoque a ruptura do tecido), as perturbações que produzirá no fluxo, etc.

Outra alternativa é a geração de modelos 1D a partir de geometrias da árvore arterial completa. Utilizar o eixo de uma geometria de tubos 3D para gerar um grupo de artérias, as artérias de um determinado órgão, ou do corpo todo, permitirá incorporar de maneira rápida geometrias mas completas da árvore arterial dadas as geometrias em forma de triangulação das artérias. Atualmente está em desenvolvimento um algoritmo para obter as "center lines" de uma geometria tubular para formar a árvore 1D correspondente.

O ParaView oferece uma série de ferramentas para visualização de dados. Estas ferramentas são genéricas e permitem visualizar qualquer tipo de informação (seja tridimensional, variante no tempo, etc.). Os modelos utilizados e as informações correspondentes são bem específicos, por este motivo a alternativa de desenvolver módulos de visualização especialidades aparece como uma boa alternativa (especificamente para visualizar o deslocamento das ondas de pressão em tubos, mudança de área, dentre outros).

6. CONCLUSÕES

A cada dia é mais importante o uso de ferramentas de predição em áreas como biologia e medicina. A modelagem e simulação computacional junto às técnicas de visualização gráfica vem sendo crescentemente utilizadas para estes fins, já que permitem representar com alto grau de detalhe os fenômenos químicos e mecânicos envolvidos. Assistência no treinamento de médicos, predição do resultado de diferentes intervenções cirúrgicas, simulação do transporte de fármacos e substâncias pelo fluxo sanguíneo, predição do risco de ruptura de aneurismas, caracterização de materiais são somente algumas das potenciais aplicações destas técnicas em medicina.

Visando estas possibilidades foi desenvolvido o HeMoLab. Esta ferramenta foi desenvolvida sobre o ParaView, um software de visualização científica que utiliza computação paralela para processar grandes volumes de informação. Completamente desenvolvido em ANSI-C++, este software pode ser executado em diferentes plataformas trabalhando em paralelo.

Para modelar os fenômenos mencionados anteriormente, são utilizados modelos 1D, modelos 3D e modelos acoplados. Foram desenvolvidos módulos diferentes que permitem criar modelos personalizados de cada tipo, sejam: hm1DModule, hm3DModule, hmCoupling. Para poder criar modelos personalizados, é necessário incorporar informações específicas dos pacientes. Este tipo de informações geralmente vem no formato de imagens. Para isto foi criado também um módulo de processamento de imagens hmImageProcessing. Desta maneira as funcionalidades ficam claramente separadas em módulos independentes. Esta característica permite trabalhar separadamente com cada módulo ou de maneira integral com vários módulos (cenários apresentados na Fig. 6). A utilização deste software no am-

biente médico irá fornecer informações mais precisas e muito valiosas na hora do diagnostico.

A. TERMINOLOGIA

Nesta seção são introduzidas as definições de alguns termos utilizados ao longo do texto com o simples objetivo de assistir ao leitor.

Aba: componente ou controle da interface de usuário, e.g., botão, *label* de texto, marco (*frame*), barra de rolagem (*scroll bar*), menu, etc.

Elemento: Na discretização de um domínio contínuo, a mínima porção continua (e.g., em uma triangulação um triângulo, em uma tetraedrização um tetraedro, em una malha regular de quadriláteros um quadrado, etc.)

Id Cliente Servidor (CSID): identificador único a nível de sistema que permite identificar objetos no servidor.

Malha: conjunto de figuras geométricas (triângulos, quadriláteros, tetraedros, prismas, etc.) que se compõem para gerar uma geometria mais complexa. Geralmente são especificadas como um conjunto de pontos e a conectividade entre eles.

Método dos Elementos finitos (FEM): método usado para encontrar a solução aproximada de equações em derivada parciais.

Modelo: conjunto de hipóteses sobre a estrutura ou o comportamento de um sistema físico pelo qual se procuram explicar ou prever, dentro de uma teoria científica, as propriedades do sistema.

Modelo personalizado: modelo correspondente a um individuo particular, o qual considera as suas características mecânicas e geométricas.

Nó: ponto nodal. Locais no quais os valores das incógnitas (pressão, velocidade, deslocamento, etc.) serão aproximados.

Segmento (arterial): trecho da arvore 1D que representa uma arteria. As maiores artérias do corpo humano são representadas usando segmentos.

REFERENCES

Alvarez, L., Guichard, F., Lions, P., and Morel, J. 1992. Axiomatisaton et nouveaux operatéurs de la morphologie mathematique. *Comptes Rendus de l'Académie des Sciences t. 315, Série I*, 265–268.

Aubert, G. and Vese, L. 1997. A variational method in image recovery. *SIAM Journal of Numerical Analisys 34,* 5, 1948–1979.

Avolio, A. P. 1980. Multi-branched model of the human arterial system. *Med. Biol. Eng. Comp. 18*, 709–718.

Baker, T. 1989. Automatic mesh generation for complex three dimensional regions using a constrained delaunay triangulation. *Engineering Computations 5*, 161–175.

Banga, C. and Ghorbel, F. 1993. Optimal bootstrap sampling for fast image segmentation: application to retina images. In *IEEE ICASP 93*. Vol. 5. Minneapolis USA, 638–641.

Bezdek, J. 1981. *Pattern Recognition with fuzzy Objective Function Algorithms*. Plenum Press, New York.

Bidgood, W. 1998. Clinical importance of the dicom structured reporting standard. *Int J Card Imaging 14,* 5 (October), 307–315.

Bonnet, A. 1996. On the regularity of the edge set of mumford-shah minimizers. *Progress in Nonlinear Differential Equations 25*, 93–103.

Boscolo, R., Brown, M., and McNitt-Gray, M. 2002. Medical image segmentation with knowledge-guided robust active contours. *Radiographics 22,* 2, 437–448.

Canny, J. 1983. Finding edges and lines in images. Tech. Rep. AI-TR-720, Massachusetts Instutite of Technology, Artificial Intelligence Laboratory.

Causin, P., Gerbeau, J. F., and Nobile, F. 2004. Added-mass effect in the design of partitioned algorithms for fluid-structure problems. Tech. Rep. 5084, Institut National de Recherche en Informatique et en Automatique - INRIA.

Cebral, J. R. and Lohner, R. October 1999. From medical images to cfd meshes. In *Proceedings, 8th International Meshing Roundtable*. South Lake Tahoe, CA, U.S.A., 321–331.

Delaunay, B. 1934. Sur la sphere vide. *Bull. Acad. Sci. USSR VII: Class. Scil, Mat. Nat.*, 793–800.

Deparis, S., Fernandez, M. A., and Formaggia, L. 2005. Acceleration of a fixed point algorithm for fluid-structure interaction using transpiration conditions. *Submited to M2AN Math. Model. Numer. Anal*.

Donea, J. and Huerta, A. 2003. *Finite Element Methods for Flow Problems*. John Wiley & Sons, Chichester.

Formaggia, L., Gerbeau, J. F., Nobile, F., and Quarteroni, A. 2002. Numerical treatment of defective boundary conditions for the navier-stokes equations. *SIAM J. Numer. Anal. 40,* 1, 376–401.

Formaggia, L., Nobile, F., Quarteroni, A., and Veneziani, A. 1999. Multiscale modelling of the circulatory system: a preliminary analysis. *Comput. Visual Sci. 2*, 75–83.

Formaglia, L., Lamponi, D., and Quarteroni, A. 2003. One-dimensional models for blood flow in arteries. *Journal of Engineering Mathematics 47*, 251–276.

Frangakis, A. S. and Hegerl, R. 2001. Noise reduction in electron tomographic reconstructions using nonlinear anisotropic diffusion. *J. Struct. Biol. 1,* 135, 239–250.

Frank, O. 1899. Die grundform des arteriellen pulses. *Z. Biol. 37*, 483–526.

Frey, P., Sarter, B., and Gautherie, M. 1994. Fully automatic mesh generation for 3d domains based upon voxel sets. *International Journal for Numerical Methods in Engineering. 37*, 2735–2753.

Gamma, E., Helm, R., and Johnson, R. 1995. *Design Patterns. Elements Of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley. Gam E 95:1 1.Ex.

Gillilan, R. E. and Wood, F. 1995. Visualization, virtual reality, and animation with the data flow model of computing. *Computer Graphics 29,* 2 (May), 55–58.

Girault, V. and Raviart, P.-A. 1986. *Finite Element Methods for Navier-Stokes Equations. Theory and Algorithms*. Springer-Verlag, Berlin.

Gonzalez, C. R. and Woods, R. E. 2001. *Digital Image Processing - Second Edition*. Prentice Hall.

Guichard, F. and Morel, J.-M. 2001. Image analysis and p.d.e.s.

Haber, R. B. and McNabb, D. A. 1990. Visualization idioms: A conceptual model for scientific visualization systems. *In Visualization in Scientific Computing - IEEE Computer Society Press*, 74–93.

Held, K., Kops, E. R., Krause, B. J., 3rd, W. M. W., Kikinis, R., and Muller-Gartner, H. W. 1997. Markov random field segmentation of brain mr images. *IEEE Trans. Med. Imaging 16,* 6, 878–886.

Hohne, K. H., Fuchs, H., and Pizer, S. M. 1990. *3D imaging in medicine: algorithms, systems, applications*. Springer-Verlag, New York.

Hughes, T. J. R. 2000. *The Finite Element Method - Linear Static and Dynamic Finite Element Analysis*. Prentice-Hall.

Hughes, T. J. R. and Lubliner, J. 1973. On the one-dimensional theory of blood flow in larger vessels. *American Elseiver Publishing Company - Mathematical Biosciences 18*, 161–170.

Ibanez, L., Schroeder, W., Ng, L., and Cates, J. 2005. *The ITK Software Guide*, Second ed. Kitware, Inc. ISBN 1-930934-15-7, http://www.itk.org/ItkSoftwareGuide.pdf.

Inc., K. ParaView. http://www.paraview.org.

Inc., K. Vtk: The visualization toolkit. http://www.vtk.org.

Jain, A. K. 1989. *Fundamentals of Digital Image Processing*. Prentice Hall.

Lamponi, D. N. 2004. One dimensional and multiscale models for blood flow circulation. Ph.D. thesis, École Polytechnique Fédérale de Lausanne.

Larrabide, I., Feijóo, R. A., Novotny, A. A., Taroco, E., and Masmoudi, M. 2005. An image segmentation method based on a discrete version of the topological derivative. In *World Congress Structural and Multidisciplinary Optimization 6, Rio de Janeiro*. International Society for Structural and Multidisciplinary Optimization.

Larrabide, I., Fiorentini, S., and Vénere, M. J. 2003. Voxel grow: A region growing segmentation technique. In *International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications (CSITeA03)*.

Larrabide, I., Novotny, A. A., Feijóo, R. A., and Taroco, E. 2005. A medical image enhancement algorithm based on topological derivative and anisotropic diffusion. In *Proceedings of the XXVI Iberian Latin-American Congress on Computational Methods in Engineering - CILAMCE 2005 - Guarapari, Espírito Santo, Brazil*.

Larrabide, I., Novotny, A. A., Feijóo, R. A., and Taroco, E. 2006. Topological derivative as a tool for image processing: Image segmentation. Tech. rep., LNCC - Laboratório Nacional de Computação Científica.

Li, H., Deklerck, R., DeCuyper, B., Hermanusa, A., Nyssen, E., and Cornelis, J. 1995. Object recognition in brain ct-scans: knowledge-based fusion of data from multiple feature extractors. *IEEE Trans. Med. Imaging 14*, 212–229.

Lorensen, W. and Cline, H. 1987. Marching cubes: A high resolution 3d surface construction algorithm. In *In Proc. of ACM SIGGRAPH*. 163–169.

Malladi, R. and Sethian, J. A. 1997. Level set methods for curvature flow, image enhancement, and shape recovery in medical images. In *Proc. of Conf. on Visualization and Mathematics*. Springer-Verlag, Germany, 329–345.

M.J.Black, Sapiro, G., Marimont, D., and Heeger, D. 1998. Robust anisotropic diffusion. *IEEE Transactions on Image Processing 7,* 3, 421–423.

Olufsen, M. S. 1998. Modeling the arterial system with reference to an anestesia simulator. Ph.D. thesis, Roskilde Universitetscenter-IMFUSA.

OpenGL. OpenGL. http://opengl.org.

Peraire, J., Peiró, J., Formaggia, L., Morgan, K., and Zienkiewicz, O. 1988. Finite element euler computations in three dimensions. *International Journal for Numerical Methods in Engineering 26*, 2135–2159.

Peraire, J., Peiró, J., and Morgan, K. 1991. Adaptive remeshing for three-dimensional compressible flow computations. *Journal of Comp. Physics 103*, 449–466.

Perona, P. and Malik, J. 1990. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Machine Intell. 12,* 7, 629–639.

Scateni, R., Ed. 1995. *Scientific Visualization '95: proceedings of the International Symposium, Cagliari, Italy, 27–29 September, 1995*. World Scientific Publishing Co., Singapore; Philadelphia, PA, USA; River Edge, NJ, USA.

Schroeder, W. J., Avila, L. S., and Hoffman, W. 2000. Visualizing with vtk: A tutorial. *IEEE Comput. Graph. Appl. 20,* 5, 20–27.

Shephard, M. and Georges, M. 1991. Automatic three-dimensional mesh generation by the finite octree technique. *International Journal for Numerical Methods in Engineering 32*, 709–749.

Suri, J., Singh, S., and Setarehdan, K., Eds. 2001. *Advanced algorithmic approaches to medical image segmentation: state-of-the-art applications in cardiology, neurology, mammography and pathology*. Springer-Verlag.

Temam, R. 1979. *Navier-Stokes Equations. Theory and Numerical Analysis*. North Holland, New York.

Urquiza, S. A., Blanco, P. J., Vénere, M. J., and Feijóo, R. A. 2006. Multidimensional modelling for carotid artery blood flow. *Computer Methods in Applied Mechanics and Engineering 195,* 33-36 (July), 4002–4017.

Urquiza, S. A. and Vénere, M. J. 2002. An application framework architecture for fem and other related solvers. In *Mecánica Computacional*, S.R.Idelsohn, V.E.Sonzogni, and A.Cardona, Eds. Vol. 22. Santa Fé - Paraná , Argentina, 3099–3109.

Vénere, M. J. 1997. Optimización de la calidad de mallas de elementos finitos mediante cambios localizados de topologia. *Rev. Int. de Mét. Num. para Cálculo y Diseño en Ing. 13*, 3–13.

Vénere, M. J. Noviembre 1996. Tecnicas adaptativas para el método de elementos finitos en dos y tres dimensiones. Ph.D. thesis, Instituto Balseiro, Universidad Nacional de Cuyo.

Vénere, M. J. and Dari, E. A. 1990. Análisis comparativo de algoritmos para obtener triangulaciones delaunay. *Mecánica Computacional 10*, 491–506.

Yerry, M. and Shephard, M. 1984. Automatic three dimensional mesh generation by the modified octree technique. *International Journal for Numerical Methods in Engineering 20*, 1965–1990.

Zhang, Y., Brady, M., and Smith, S. 2001. Segmentation of brain mr images through a hidden markov random field model and the expectation-maximization algorithm. *IEEE Transactions on Medical Imaging 20,* 1, 45–57.