André de Jesus Costa

Um ambiente de baixo custo para o processamento de imagens tomográficas 3D

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Orientador: Pedro Abílio Duarte de Medeiros, Professor Associado, NOVA FCT

Júri

Presidente: Carlos Augusto Isaac Piló Viegas Damásio, Professor Associado, NOVA FCT
Arguente: André Teixeira Bento Damas Mora, Professor Auxiliar, NOVA FCT
Vogal: Pedro Abílio Duarte de Medeiros, Professor Associado, NOVA FCT

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2018

Um ambiente de baixo custo para o processamento de imagens tomográficas 3D

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Resumo

Nesta dissertação desenvolveu-se um ambiente de trabalho (PSE Problem Solving Environment) para engenheiros de materiais dedicado à caracterização de materiais compostos a partir de imagens tomográficas. Um composto é constituído por um material base e por reforços. A partir da imagem tomográfica é possível obter informação sobre os reforços, nomeadamente dimensão, localização e orientação. Essa informação permite avaliar a eficácia de um dado processo de construção de um material composto.

O ambiente de execução para este sistema é um computador pessoal equipado com processadores *multicore* e uma placa gráfica dedicada. O software a utilizar é o sistema operativo Windows, o *toolkit* SCIRun para construção de PSE e o CUDA para exploração dos GPUs.

As operações de processamento de imagens tomográficas são exigentes do ponto de vista computacional, mas o facto de poderem ser paralelizadas usando um paradigma SIMD - a mesma operação aplicada a um grande conjunto de dados - permitiu que, usando GPUs, o processamento poderá ser efetuado em tempos compatíveis com um ambiente interativo.

O trabalho a desenvolver baseia-se num esforço anterior - o projeto Tomo-GPU. O desempenho do sistema desenvolvido será comparado com o da versão anterior.

Palavras-chave: Processamento GPU, CUDA, Visualização

Abstract

In this dissertation, a Problem Solving Environment, targeted at Material's Engineering specialists, dedicated to the analysis of tomographic images of samples of composite materials, will be developed.

A composite is made by a base material plus reinforcements. By using tomographic images, it is possible to obtain information about the reinforcements, namely dimension, localization and orientation. That information allows an evaluation of the effectiveness of the process used to build the composite.

The execution environment of this system is a personal computer, equipped with a multicore processor and a dedicated graphics card. The software used is the operating system Windows, the SCIRun toolkit and CUDA for the GPU exploitation.

Regarding computational power, the processing of tomographic images is very demanding, but the fact that the computation can be parallelized using an SIMD paradigm - the same operation applicable to a large group of data - let us foresee that, using GPUs, the processing can be done in times compatible with an interactive environment. The work to be developed is based on a previous effort - the Tomo-GPU project. The performance of both new and the already existent versions Tomo-GPU is compared.

Keywords: GPU Processing, CUDA, Visualization

Índice

Lista de Figuras

Capítulo 1

Introdução

Este documento descreve o desenvolvimento de um sistema informático de baixo custo para permitir a especialistas da área da Ciência de Materiais o estudo de imagens tomográficas de materiais compósitos. Um compósito é composto por um material base e por um conjunto de reforços; para avaliar diferentes processos de fabrico é necessário fazer a chamada *caracterização da população de reforços*, isto é conhecer a forma como os reforços se distribuem pelo compósito, qual a sua dimensão, orientação, etc. A caracterização usa, muitas vezes, imagens a três dimensões obtidas por um tomógrafo.

Neste primeiro capítulo começa-se por descrever a motivação para este trabalho e faz-se uma breve exposição à temática da caracterização de reforços em materiais compósitos; discute-se também a importância de um ambiente gráfico flexível para a análise das imagens tomográficas. De seguida é discutido a forma como o sistema vai ser construído e quais as contribuições esperadas deste trabalho. Por fim, existe uma descrição do conteúdo do documento.

1.1 Motivação para o trabalho

Com o desenvolvimento da tecnologia, a complexidade e a diversidade dos problemas provoca um aumento exponencial da quantidade de dados a ser processada em cada investigação.Todos os dias são desenvolvidas novas infraestruturas e novos algoritmos para análise dos dados produzidos por simulações e experiências científicas.

Assim, a visualização de informação relacionada com volumes de dados de grande dimensão corresponde a um dos novos desafios da era computacional. Todos os processamentos envolvidos requerem software que tem grandes exigências do ponto dos recursos computacionais, quer ao nível do processador quer da placa gráfica.

A caracterização dos reforços existentes nos materiais compósitos pode ser muito

auxiliada se existir um ambiente gráfico dedicado a este fim *(Problem Solving Environment)*.

O objetivo deste trabalho é desenvolver um PSE dedicado à tarefa da análise de imagens tomográficas de materiais compósitos, procurando oferecer um bom desempenho em hardware de baixo custo.

1.1.1 Caracterização de reforços em materiais compósitos

Os compósitos são materiais formados pela junção de componentes distintos, com o objectivo de criar um produto com propriedades de ambas as partes e de qualidade superior. O aço é um dos melhores exemplos disso, combinando as propriedades de dureza do ferro com o de várias ligas metálicas, como o magnésio e o crómio, e também outros materiais como o carbono, para obter uma liga que é mais resistente à oxidação, mais leve, e mais forte do que se fosse somente em ferro.

A aplicabilidade destes compostos na indústria aeronáutica, automóvel e aeroespacial tem crescido bastante nos últimos anos. Como tal é de extrema importância conseguir avaliar os processos de fabrico destes materiais. Essa avaliação é feita caracterizando a população de reforços, nomeadamente quanto ao seu tamanho, integridade, orientação espacial, distribuição , etc.

Anteriormente era necessário recolher amostras do próprio material, danificando o mesmo. Hoje em dia com o uso de tecnologia como a tomografia por raios-X, é possível obter imagens tridimensionais do interior das amostras sem as destruir. Essas técnicas produzem um enorme volume de dados, e o hardware e software disponível produzem informação que inclui erros e artefatos que têm de ser removidos. A separação do material base dos compósitos exige um conjunto de processamentos complexos que têm de ser combinados e afinados de forma a serem conseguidos resultados fiáveis.

1.1.2 Importância de se ter um ambiente gráfico flexível

As aplicações como a da caracterização dos materiais compósitos exigem a combinação de um conjunto de ferramentas, umas já existentes e outras que têm de ser desenvolvidos para fins específicos. A flexibilidade é um aspeto fundamental destes sistemas e uma forma de a conseguir é através da combinação de vários tipos de processamento suportados em módulos independentes.

As ferramentas geralmente disponíveis pelos PSEs incluem componentes para a visualização e modelação de imagens tridimensionais, inteligência artificial, interação com o utilizador, análise numérica, computação distribuída e paralela, e engenharia de software[1].

Para construir um PSE dedicado a um fim específico é necessário dispor além de módulos de uso geral (por exemplo de visualização de dados a três dimensões), componentes que efetuem computações específicas aos problemas em causa. Diversas companhias e grupos de investigação universitários desenvolveram *toolkits* que incluem

- Módulos de uso geral (por exemplo, visualização),

- Bibliotecas que facilitam o desenvolvimento de componentes específico,
- Mecanismos que permitem a transferência de dados entre componentes (por exemplo, através de um padrão *pipeline*.

As referidas ferramentas permitem uma caracterização 3D pormenorizada de compósitos; permitem também uma profunda interatividade com o utilizador, com alterações de parâmetros e configurações que alterem a visualização de conteúdos à medida que sejam ajustados. Tudo isso proporciona uma ajuda computacional aos investigadores na procura de soluções, ou na compreensão de problemas. Alguns esforços anteriores (descritos no capítulo seguinte) foram feitos. No trabalho conducente a esta dissertação, esses esforços foram continuados nas dimensões descritas à frente.

1.2 Abordagem ao problema

Nesta tese vai ser desenvolvido um PSE com as seguintes características:

- Tenha como ambiente de execução alvo um computador pessoal com um processador de vários núcleos e um processador gráfico de uso geral (GPGPU). O sistema operativo deverá ser o Windows, uma vez que é este ambiente que está normalmente disponível diretamente para engenheiros e cientistas.
- Não seja construído de raiz, mas tire partido de um *toolkit* para a construção de PSEs. Existem variadíssimos projetos *open-source*, com documentação, que podem servir de base a este esforço. Para tirar partido de desenvolvimentos anteriores, foi utilizado um PSE chamado SCIRun.
- Suporte a exploração do hardware paralelo existente no computador pessoal, um vez que os algoritmos de processamento de imagens tomográficas de materiais compósitos exigem grandes recursos computacionais e os tempos de processamento têm de ser compatíveis com uma utilização interativa do sistema. Os módulos poderão tomar partido de ambientes como o OpenCL ou o CUDA.
- Permita alterações rápidas aos algoritmos de processamento, diminuindo os tempos necessários à introdução de novas funcionalidades no sistema. O SCIRun, como outros *toolkits* de construção de PSE é um sistema monolítico e a alteração de um módulo exige um procedimento de reconstrução do sistema moroso.

1.3 Contribuições esperadas

Nesta dissertação estão planeadas as seguintes contribuições:

- Um ambiente de caracterização de problemas baseado no SCIRun a executar em sistemas Windows.

- Transporte para este ambiente de algoritmos e componentes, já existentes, para efeitos de processamento de imagens, incluindo módulos da versão anterior do Tomo-GPU.
- Construção de um módulo genérico para execução de aplicações externas.
- Construção de um módulo genérico que posso ser inserido no pipeline de processamento e que invoca um processo independente. Este módulo processa informação numa *thread* que tem um espaço de endereçamento independente do SCIRun, podendo ser recompilado sem obrigar à recompilação do PSE.
- Comparação do desempenho do Tomo-GPU em diferentes versões do SCIRun e do sistema operativo (Windows e Linux).

1.4 Conteúdo do documento

Esta dissertação é constituída por 6 capítulos. No primeiro é apresentado o problema a resolver. Existe uma descrição do mesmo, uma definição da motivação, uma breve explicação da caracterização de imagens tomográficas e uma abordagem aos *Problem Solving Environments*.

O segundo capítulo descreve o estado da arte. Entra-se em pormenor na temática dos PSEs, especificando o caso particular do SCIRun. Esse mesmo PSE foi utilizado como base para o ambiente Tomo-GPU, vocacionado para a caracterização de materiais compósitos. Também são abordadas ambientes de desenvolvimentos de paralelização de programas paralelos, a referir, POSIX Threads, OpenMP, CUDA e OpenCL.

O terceiro capítulo aborda a organização da solução para esta dissertação. Apresenta-se um resumo do funcionamento do SCIRun, dos seus módulos, portas e tipos de dados. Existe também uma pequena descrição do módulo genérico e é detalhado cada módulo principal do Tomo-GPU.

O quarto capítulo explica a implementação da solução. Apresenta um tutorial de como construir um módulo SCIRun e de como converter um módulo antigo da versão 4 do mesmo. É também explicado brevemente o funcionamento de vários módulos Tomo-GPU.

O quinto capítulo fala acerca da metodologia utilizada para a avaliação do sistema, cuja principal componente é a comparação das duas versões SCIRun em Linux com a versão SCIRun 5 em Windows. Esta comparação usa como critério o desempenho.

Por fim o sexto e último capítulo apresenta as conclusões do trabalho, sendo comparados os objetivos enunciados neste capítulo com as contribuições conseguidas.

Capítulo 2

Trabalhos relevantes

Neste capítulo é retratado o estado da arte referente ao tema deste projeto. São referidas em detalhe as propriedades dos PSE, e é discutido também um PSE especializado para o processamento de imagens tomográficas. Também se dedica algum espaço ao trabalho já realizado na integração de computação paralela em PSEs.

2.1 Problem Solving Environments (PSE)

Nesta primeira secção são primeiramente abordadas as características gerais dos PSE, incluindo posteriormente em mais detalhe, o SCIRun.

Problem Solving Environments são aplicações que providenciam facilidades computacionais necessárias para resolver ou estudar um determinado problema. Têm por base o facto de poderem serem usadas por um utilizador que não tenha conhecimento específico sobre o hardware ou software em que se está a trabalhar [7]. O investigador utiliza o ambiente para realizar sequências de passos, sendo exemplos de operações a aplicações de algoritmos a dados e a a renderizações 3D de resultados. Os PSE são hoje uma ferramenta relevante na resolução de problemas en ciência, engenharia e não só.

Estas aplicações estão desenhadas para fornecer um ambiente gráfico *user-friendly*; o utilizador da PSE, pode focar-se mais no problema em si, e não em desenvolver software para realizar todas as operações mencionadas ou a conhecer o hardware específico para correr o programa. O facto de estes sistemas serem modulares, e poderem ser utilizados em vários tipos de investigação, faz com que não seja necessário criar softwares específicos para cada problema diferente a ser tratado. Em alguns casos, os PSE permitem a colaboração numa experiência visual, de investigadores que estão geograficamente separados.

Apesar de serem multidisciplinares, os PSE não são específicos a determinadas disciplinas. Hoje em dia os PSE englobam uma vasta área científica, desde suporte para educação

escolar, bio-medicina, química, geração de software, suporte para empresas, computação em *cloud* entre outros.

A maior parte dos PSE permite que o investigador, interativamente, altere *on the fly* vários parâmetros das operações em curso e também modifique as configurações da visualização do problema, o que permite uma melhor compreensão do fenómeno em estudo.

Alguns PSE são baseadas numa programação modular, em que cada módulo disponível para a aplicação permite realizar uma tarefa diferente no *workflow* da caracterização de um sistema. Cada módulo tem, normalmente, associado inputs e *outputs* específicos. Muitas operações realizadas nos PSE envolvem grandes quantidades de dados e/ou muito tempo de computação; normalmente é possível ver em tempo real, a utilização do CPU ou da memória e também mesmo alterar alguns parâmetros no *workflow* do sistema.

Alguns tipos de módulos disponíveis em PSEs são:

- Visualização: é possível representar dados a duas ou três dimensões, normalmente em formatos normalizados.
- Aplicação de algoritmos: por exemplo, manipulação de matrizes, ordenação de dados.
- Simulação; Podem ser simulações de equações matemáticas ou até mesmo de complexas teorias astrofísicas.
- Afinição e medição de desempenho;

Como alguns dos passos executados tem grandes necessidades computacionais - quer pela grande quantidade de dados envolvida, quer pela complexidade dos algoritmos a executar - é importante que esses módulos possam ser programas que exploram múltiplos processadores.

2.2 Exploração de hardware paralelo pelos módulos de um PSE

Como já foi referido, para garantir a satisfação do especialista da área que está a usar o PSE é vital conseguir que os módulos sejam executados rapidamente, garantindo que o sistema responda também rapidamente a alterações à rede de processamento efetuadas pelo especialista.

Uma das alternativas para conseguir este objectivo, passa pela paralelização dos programas executados pelos módulos de recursos do hardware. Este processo consiste na divisão do processamento em múltiplas tarefas que são distribuídas por distintos elementos processadores. No caso do uso de um computador de secretária, estes elementos processadores são os núcleos (cores) do CPU e os processadores do GPGPU. Nesta dissertação, é explorada uma máquina única.

Para conseguir paralelizar um módulo é preciso dispor de um ambiente de programação que permita:

- gerir a criação e destruição de atividades paralelas independentes (processos ou *threads*);
- sincronizar as ações destas entidades;
- suportar a troca de informação entre elas;

Seguidamente discute-se como é que cada um destes aspetos é suportado num conjunto de plataformas de programação paralela populares, a saber, PThreads, OpenMP, CUDA e OpenCL.

PThreads, mais conhecido por POSIX Threads, é a implementação *standard* para a criação e controlo das *threads*. Este permite a um programa controlar diferentes sequencias de execução em paralelo *(threads)*. É de salienta que as *threads* partilham o mesmo espaço de endereço, enquanto que os processos não. Para se conseguir a paralelização duma aplicação usando esta interface, é necessário ter especial atenção ao sincronizamento das *threads*, bem como ao correto particionamento dos dados entre os mesmos. A biblioteca Pthreads está presente em variados sistemas operativos, incluindo Linux, Android e existe uma adaptação para sistemas Windows. Apesar de existirem mais de 100 funções na Pthreads, elas podem-se distinguir em 2 grupos[2]:

- Gestão das *threads*: criação e destruição;
- Sincronização entre *threads*; mutexes, variáveis de condição e semáforos.

OpenMP, abreviatura de Open Multi-Processing, é também uma interface de multiprocessamento de memória partilhada, mais fácil de usar do que a dos PThreads. É uma plataforma escalável e portátil que proporciona um ambiente flexível para as aplicações. Pode ser usado numa ampla gama de hardware desde super-computadores até computadores pessoais. É baseada num conjunto de anotações suportadas pelos compiladores, variáveis de ambiente e bibliotecas de suporte em tempo de execução que mapeiam os conceitos do OpenMP para *threads* do sistema operativo. Está disponível nos principais sistemas operativos, dos quais fazem parte mais uma vez, Linux e Windows e também MacOS[6].

CUDA, mais especificamente Compute Unified Device Architecture, é uma API desenvolvida pela Nvidia com vista ao processamento paralelo em GPGPU. Está disponível nos sistemas operativos mais comuns, (Windows,Linux e MacOS). Esta tecnologia está reservada somente para placas gráficas com *chipset* Nvidia[11].

Um programa em CUDA tem duas partes

- Uma que corre no CPU e que pode ser escrita em múltiplas linguagens de programação nomeadamente C,C++,Python e Fortran;

- Outra que executa no GPU *(kernel)* e que é especificada em C com algumas extensões.

A utilização eficiente do CUDA não é fácil: é necessário definir uma série de factores: A quantidade das *threads*, a sua disposição, acessos de memória global, *cache* e comunicação entre CPU E GPU. A imagem seguinte apresenta uma comparação do desempenho entre CPU e GPU.

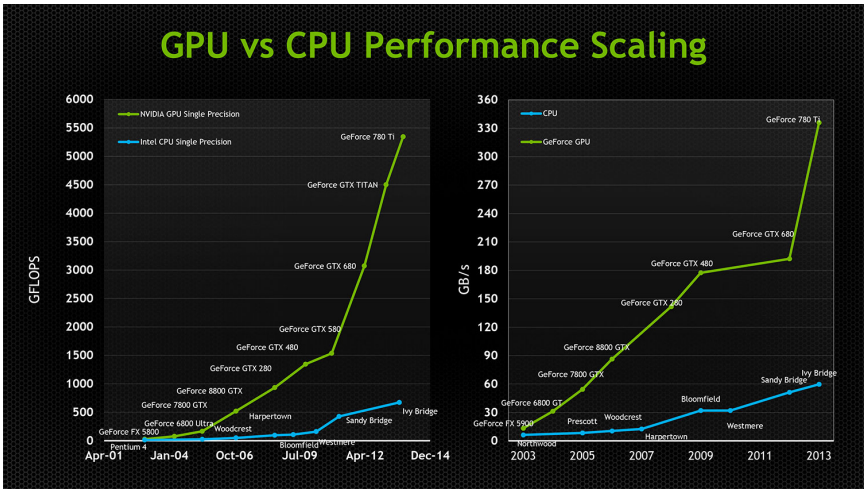

Figura 2.1: Comparação GPU vs CPU
adapted from: https://geotecnologias.files.wordpress.com/2016/01/gpucpu.jpg

Para GPGPUs AMD existe uma outra arquitetura que se pode utilizar chamada **OpenCL**, amplamente usada na indústria, e semelhante em funcionalidade com o CUDA. É uma interface de programação em paralelo de diversos processadores, por exemplo, de CPU e GPUs de um computador, a processadores de dispositivos móveis [6]. É compatível com os principais fornecedores de processadores Intel e AMD, assim como de placas gráficas, a Nvidia e novamente a AMD. É baseada na linguagem C++, estando já incluída como sub-conjunto estático do *standard* C++14. O OpenCl disponibiliza ao utilizador variadas expressões, funções, classes, *templates*, expressões lambda, entre outros. OpenCL é desenvolvida pelo *Khronos Group* e apresenta na altura de escrita desta dissertação, a sua versão mais recente, a 2.2. A figura seguinte apresenta o modelo da plataforma OpenCL. Existe um *host* que possui vários dispositivos OpenCL. Cada um desses dispositivos tem uma ou mais unidades de computação, que por sua vez, tem múltiplos elementos de processamento.

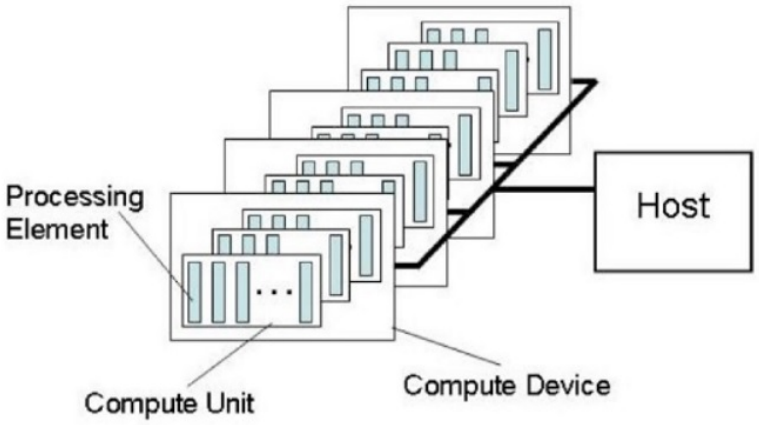

Figura 2.2: Modelo da plataforma OpenCL.

No projeto Tomo-GPU foi esta *framework* que foi usada para explorar os GPUs.

2.3 SCIRun

O *Problem Solving Environment*, que será utilizado nesta dissertação é o SCIRun(Figure 2.4) tem sido desenvolvido pela CIBC (Center for Integrative Biomedical Computing) da Universidade de Utah desde 1999, tendo como objetivo inicial ser "um software extensível, escalável de resolução de problemas relacionados com o campo bio-eléctrico". Desde então, o SCIRun. tem servido de base para o desenvolvimento por muitas outras ferramentas de modelação e visualização como Seg3D e Fluorender(Figure 2.3).

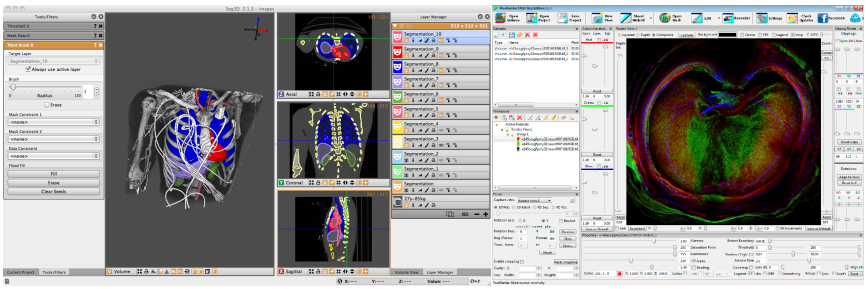

Figura 2.3: Seg3d e FluoRender
adapted from: http://www.sci.utah.edu/cibc-software/seg3d.htmlehttp://www.sci.utah.edu/images/CIBC/TRD/fl_diaphragm.jpg

À semelhança de outros PSE, o SCIRun possibilita a construção gráfica de uma rede de processamento de dados através da interligação de módulos disponíveis num menu. Cada módulo tem uma função específica, seja ela computação de dados, visualização de matrizes, conversão de valores, etc..

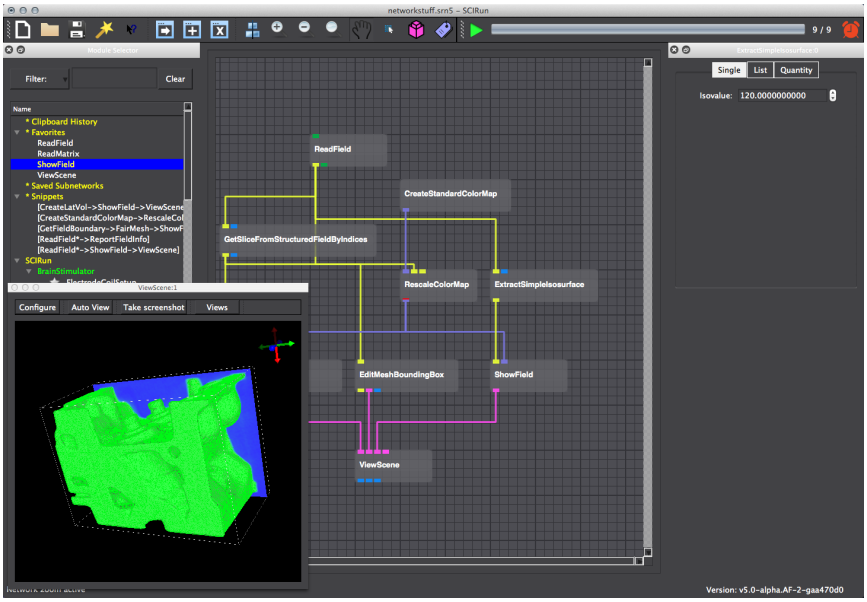

Figura 2.4: SCIRun versão 4
adapted from:
http://sciinstitute.github.io/scirun.pages/BasicTutorial_figures/coloriso.png

O SCIRun está implementado sobretudo em C++ e o seu código fonte está disponível. Neste trabalho será utilizada a versão mais recente (v5) do SCIRun. A principal diferença desta versão em relação às anteriores é a utilização do *toolkit* Qt na interface gráfica. Um programador pode desenvolver novos módulos e integrá-los no menus; há documentação sobre este procedimento e os novos módulos podem ser escritos em diferentes linguagens de programação nomeadamente C++, Python e Matlab.

O código fonte do SCIRun pode ser obtido através do site da instituição.[12] Estão disponíveis versões executáveis para Windows, Linux e MacOS.

2.3.1 Interface para o não especialista

O foco principal da criação do SCIRun foi o de criar um ambiente que permitisse estudar cenários e simulações, aliar ferramentas de visualização com algoritmos existentes e promover o desenvolvimento de novos algoritmos, tudo da maneira mais fácil e eficiente possível para o utilizador. Em especial deu-se ênfase em poder facultar todos estes componentes ao utilizador, sem que o mesmo tivesse de ter conhecimento sobre a forma como um determinado módulo foi construído.

Um módulo do SCIRun, tem associado uma ou várias portas de *input* e *output*. Estas portas são utilizadas para a passagem de informação entre módulos através de conexões; uma porta *input* recebe dados de uma outra porta *output*; as portas de *input* podem estar ligadas apenas a uma porta de *output*, uma porta de output pode estar ligada a várias portas de *input*. O utilizador não especialista pode interligar portas de *input* e *output* de forma gráfica; esta ligação é facilitada pela existência de uma gama de cores associada

com cada tipo de fluxo de informação, por exemplo, uma matriz é representada por uma cor, enquanto uma string é representada por outra.

2.3.2 Execução de um módulo

Em termos da implementação do *dataflow* usado pelo SCIRun é necessário salientar duas alternativas distintas. *demand-driven* e *data-driven*. Segundo [9]:

Basically, in data-driven (e.g., data-flow) computers the availability of operands triggers the execution of the operation to be performed on them, where as in demand-driven (e.g., reduction) computers the requirement for a result triggers the operation that will generate it.

O SCIRun utiliza uma arquitetura baseada em módulos. A execução de um cada módulo segue a abordagem *data-driven*[9], o que significa que a disponibilidade de dados desencadeia a execução a ser realizada sobre eles.

2.3.3 Implementação de módulos para o SCIRun

Como mencionado anteriormente, o SCIRun permite que sejam implementados novos módulos para o sistema. A opção mais frequente para realizar estes módulos, é usar a linguagem C++ e o *toolkit* Qt.

Estes podem ter associados nenhuma, uma ou mais portas de *input/output*. A tarefa de um módulo em si, é escrita na função execute(). Tipicamente este recebe *input*, realiza alguma computação com os dados, e envia o seu *output*.

No capítulo 4 é apresentado um exemplo de como implementar um módulo para o SCIRun.

2.4 PSE para a caracterização de materiais compósitos

Os objetivos de um PSE para a caracterização de materiais compósitos já foram apresentados no capítulo 1; ver também a referência [10]. Em resumo, é preciso processar a imagem 3D da micro/nanotomografia por raio-X para obter informação sobre os reforços; essa informação é sobretudo geométrica (dimensão, orientação especial, distribuição pela amostra). A operação fundamental é conseguir identificar quais são os voxeis que correspondem a reforços; isto corresponde a realizar uma *segmentação da imagem*.

Para se conseguir o objetivo anterior é necessário processar grandes quantidades de dados. Uma imagem tomográfica é representada por uma matriz 3D de números, onde cada número corresponde a um pixel da imagem 3D (daqui em diante designado por voxel). Dimensões típicas desta matriz são 1000x1000x1000 o que se considerarmos que cada voxel corresponde a 1 *byte*, nos leva a ter 1GB de dados para ser processados. O valor associado a cada pixel representa a maior ou menor absorção dos raios X nesse ponto; esta absorção tem a ver com a densidade do material.

Além do desafio associado à dimensão dos dados, a segmentação da imagem também implica processamentos computacionais complexos sobre cada voxel, especialmente se a diferença de densidade entre o material base e os reforços são pequenas; outros aspetos que obrigam a cálculos realizados sobre cada voxel têm a ver com a eliminação de artefatos produzidos pelo processo de obtenção da imagens (por exemplo difração sofrida pelos raios X).

As operações de processamento da imagem 3D têm de poder ser parametrizadas pelo utilizador e é importante que este obtenha uma visualização imediata dos resultados para que, perante a observação que está a fazer, possa corrigir esses parâmetros *(steering)*. Assim, estes processamentos têm de ser realizados o mais rápido possível para permitir uma interface interativa com o utilizador. Uma das maneiras de se conseguir este objetivo, é através do processamento em paralelo. Esta temática foi descrita anteriormente na secção 2.2.

O projeto Tomo-GPU [3] teve como objetivo desenvolver um PSE para a caracterização de compósitos a partir de imagens tomográficas. Embora já existissem esforços anteriores na utilização de processamento paralelo à geração de imagens tomográficas [8] o projeto Tomo-GPU foi inovador na sua aplicação à fase de processamento das imagens 3D [5].

2.4.1 Um exemplo: Tomo-GPU

O ambiente Tomo-GPU é o resultado de um projeto realizado por dois centros de investigação da Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia FCT/MCTES (PTDC/EIA-EIA/102579/2008 - Ambiente de Resolução de Problemas para Caracterização Estrutural de Materiais por Tomografia) cujo objectivo foi apresentar um PSE para auxiliar na correcta caracterização e análise das imagens tomográficas de novos compósitos para o departamento de Ciências dos Materiais[4].

Um dos principais objetivos do Tomo-GPU é proporcionar uma plataforma de desenvolvimento com base num *toolkit* para a construção de *Problem Solving Environments*. O SCIRun foi escolhido como ponto de partida deste projeto por ser *open-source*, contar com uma documentação razoável e estar constantemente a ser atualizado. Permite construir uma aplicação de forma modular. É também possível criar módulos novos e específicos para novos tipos de processamento.

Um outro dos grandes objetivos de a aplicação poder realizar tarefas computacionais exigentes, uma vez que a sua plataforma alvo é um computador pessoal com um ou mais GPGPU *(General Purpose Graphical Processing Unit)* dedicado, seja suficiente para conseguir correr a aplicação.

O Tomo-GPU é portanto construído sobre o SCIRun, através da combinação de módulos já disponíveis (por exemplo para a visualização de conteúdos 3D) com um pacote de módulos desenvolvidos especialmente para o processamento de matrizes 3D que contêm o resultado de tomografias por raios X de amostras de materiais compósitos. Na imagem seguinte é apresentada a estrutura do Tomo-GPU.

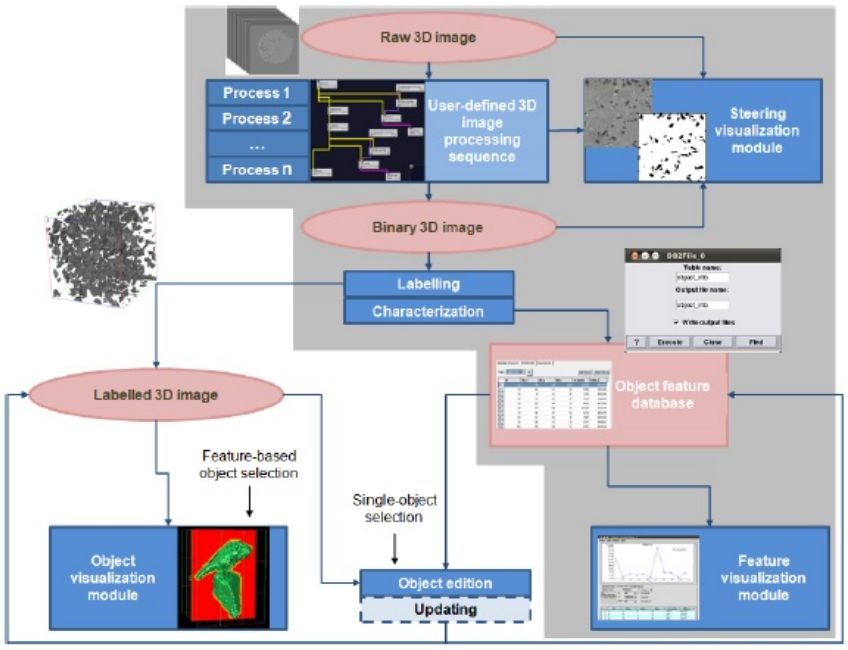

Figura 2.5: Estrutura do Tomo-GPU
adaptado de:[5]

No Tomo-GPU a sequência de processamento de uma imagem tomográfica inclui os seguintes passos pela ordem indicada:

- Obtenção de uma imagem monocromática: reforços (preto), material base (branco);
- Obtenção das características geométricas das partículas;
- Colocação dos valores anteriores numa base de dados;
- Geração de gráficos que permitem visualizar as características dos reforços.

Regra geral , numa sequência de processamento Tomo-GPU, o sistema tem um ou mais módulos de visualização em tempo real de uma imagem 3D e um ou mais módulos responsáveis por alterar vários parâmetros de processamento.

2.5 Conclusões

A análise feita neste capítulo permitiu definir com clareza o ambiente e software a utilizar para a construção da nova versão do ambiente Tomo-GPU que é objeto desta dissertação:

Um computador desktop/portátil a executar o sistema operativo Windows para permitir que o sistema seja executado no mesmo hardware que o especialista de materiais usa no seu dia a dia;

A utilização do software SCIRun para a contrução do PSE uma vez que se trata de um ambiente conhecido e que garante as características de modularidade e interatividade conhecidas;

A exploração de computação paralela para diminuir o tempo de execução dos módulos que efetuam processamentos demorados;

A exploração dos múltiplos cores do CPU e do GPU que será efetuada nos ambientes mais adequados, em princípio OpenMP para os CPUs e CUDA ou OpenCL para os GPUs.

A grande vantagem da exploração desta arquitetura heterogénea é que os cores dos CPUs asseguram as operações de entrada/saída e podem executar de forma muito eficiente um pequeno conjunto de *threads*. Uma placa gráfica possui uma arquitetura optimizada para executar a mesma operação sobre dados distintos, suportando eficientemente milhares de *threads* em simultâneo. Este último aspeto é particularmente importante para o processamento das matrizes 3D em que estão guardados os dados correspondentes à imagem tomográfica inicial e às que são obtidas por transformações sucessivas desta.

Capítulo 3

Organização da solução

Neste capítulo é apresentada a organização da solução. É explicado o funcionamento do SCIRun e dos seus componentes. De seguida é apresentado como foi realizado o transporte de módulos já existentes para a nova versão do SCIrun. Finalmente, é também mencionado o processo de criação e funcionamento do módulo genérico e dos módulos mais importantes do Tomo-GPU.

3.1 Funcionamento do SCIRun

O principal objetivo do SCIRun, é o de proporcionar um ambiente eficiente na qual cientistas, de diversas áreas, podem criar e visualizar simulações e cenários, para o desenvolvimento de algoritmos.

Cada aplicação do SCIRun é construída através da junção de vários componentes (módulos) de maneira a formar uma rede entre eles. Depois cada módulo é responsável por realizar uma tarefa computacional diferente. Estas tarefas variam desde a visualização de conteúdos (matrizes, texto, histogramas) a computação de algoritmos. Cada módulo pode também ter elementos no interface gráfica (UI), que sirvam para alterar parâmetros no módulo. Nesta secção ir-se-á descrever em detalhe, o funcionamento de cada componente do SCIRun.

3.1.1 Módulos

Um módulo representa um algoritmo ou operação, a ser executada numa aplicação SCIRun. Cada módulo é desenhado como uma caixa num fluxograma SCIRun. Em cada módulo, podem existir ou não, portas de *input* ou de *output*. Um módulo tem portas de *input* na sua parte superior, e as portas de *output* na sua parte inferior. Essas portas servem para receber e enviar dados entre módulos. Como descrito no capítulo anterior, o SCIRun utiliza

uma abordagem *data-driven* na execução dos módulos, isto é, a execução dos mesmos é desencadeada pela disponibilidade de dados. O fluxo da execução de uma rede com vários módulos é demonstrado na imagem seguinte[9].

```
execute_list = modules that requested execution;
resend_list = empty;
foreach module in the execute_list {
   foreach module connected to an output {
     if the connected module is not in the execute list,
       add it
   }
   foreach module connected to an input {
     if the connected output port has a cached dataset,
       add it to the resend list
     else add it to the execute_list
   }
}
cull all ports from the resend_list whose modules appear
    in the execute list
send resend messages to the ports in the resend list
send execute messages to the modules in the execute list
```

Figura 3.1: Pseudo-código da execução de uma rede de módulos.

Cada módulo tem associado um fluxo de execução independente (daqui para a frente *thread*). Um módulo é executado de novo se existe novos dados a serem recebidos na porta de *input*, se algum dos parâmetros do módulo são alterados, ou quando os dados são requeridos por uma porta *output*. A execução dos módulos é controlada por num escalonador (*scheduler*).

Quando algo se altera na rede, por exemplo através de uma ação do utilizador, o escalonador é notificado e analisa o grafo da rede de módulos para decidir quais são os módulos que precisam de uma nova execução. O processamento do grafo determina quais os módulos que necessitam de ser executados, que incluem o módulo que o utilizador ativou e recursivamente, todos os que têm portas de *input* ligadas a módulos que foram selecionados para execução.

O escalonador ativa um módulo enviando uma mensagem para uma caixa de correio com disciplina FIFO (*First-in/First-out* na qual o *thread* associado ao módulo está bloqueado. Este *thread* executa o código associado ao módulo. Esta execução implica geralmente, uma leitura de dados da porta de *input*, uma operação com esses dados, e enviar os resultados através de um ou mais portas de *output*.

A imagem seguinte exemplifica uma aplicação SCIRun. Esta gera um cubo, que é depois segmentado por vários planos ao longo dos eixos dos ZZ.

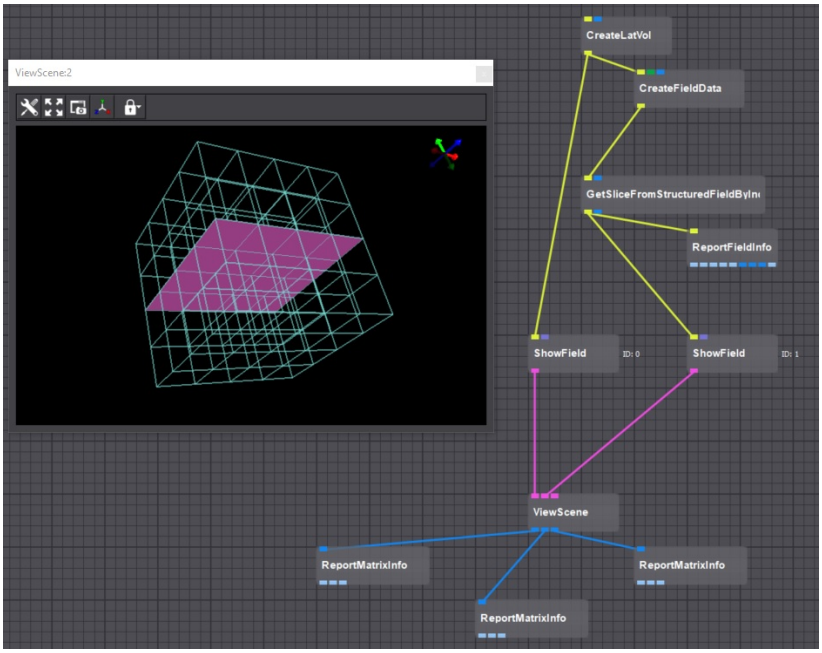

Figura 3.2: Exemplo de uma rede de módulos SCIRun.

Como já referido, esta arquitetura de rede de módulos do SCIRun permite que seja possível construir e interagir com simulações científicas complexas, sem submergir o utilizador não-especialista em informática com detalhes irrelevantes para o tratamento de dados que este está a efetuar.

3.1.2 Portas

As portas providenciam a comunicação de dados entre os vários módulos da aplicação. Cada porta tem associado somente um tipo de dados. Por exemplo um porta de *output* que produza uma matriz, só se pode conetar com uma porta de *input* que esteja associada a matrizes. Para ajudar o utilizador, cada tipo de dados tem uma cor de porta e conexão distinta. Um texto tem associada a cor verde e uma matriz tem associada a cor azul. Apesar de por exemplo, uma porta esteja associada a uma matriz, ele pode receber vários tipos de matrizes. Tanto uma *DenseMatrix* como uma *SparseMatrix* utilizam a mesma tipo de porta *MatrixPortTag*.

O SCIRun disponibiliza os seguintes tipos de portas:

- *MatrixPortTag*
- *ScalarPortTag*
- *StringPortTag*
- *FieldPortTag*
- *GeometryPortTag*

- *ColorMapPortTag*
- *BundlePortTag*
- *NrrdPortTag*
- *DatatypePortTag*

3.1.3 Tipos de dados

Em conjunto com os mecanismos de fluxo de informação já descrito, o SCIRun apresenta também uma grande flexibilidade de tipo de dados. Isso permite que existam diferentes tipos de aplicações, simulações e computações no SCIRun. De salientar que, dada a modulação e a abstração do SCIRun, é sempre possível implementar mais tipos de dados para a aplicação.

3.1.3.1 Matrizes

O SCIRun providencia neste momento quatro tipos de implementações de matrizes:

- *DenseMatrix*: Implementação de uma matriz densa, que guarda um único bloco com todas as linhas e colunas da matriz.
- *TriDiagonalMatrix*:Implementação que guarda três elementos por linha.
- Matrizes esparsas: Sob a forma de *SparseRowMatrix* e *SymSparseRowMatrix*. A diferença entre elas é que a *SymSparseRowMatrix* é a versão simétrica de *SparseRowMatrix*. Ambas guardam a matriz completa mas a implementação simétrica permite utilizar o mesmo algoritmo para a multiplicação, e para o cálculo da matriz transposta. Permitindo assim, eliminar o custo da multiplicação para o cálculo da matriz transposta, que é bastante alto.

Cada uma dessas implementações partilham várias funções e estruturas de dados. Como por exemplo, *nrows()* que retorna o numero de linhas de uma matriz, e a *maxValue()* que devolve o maior valor encontrado na matriz.

3.1.3.2 *Mesh*

Meshes em SCIRun são utilizadas para representar grelhas tetraédricas não estruturadas. Uma *Mesh* consiste num conjunto de nós e de um conjunto de elementos. Cada nó contém a sua localização no espaço 3D, enquanto que cada elemento contém um apontador para quatro nós diferentes. Um nó da *Mesh* possui também uma lista de elementos à qual está conetado. Além disso, cada elemento contém uma lista com os seus elementos vizinhos.

3.1.3.3 *Fields*

Um *Field*, ou campo em português, define uma função escalar ou vetorial numa dada região do espaço. Em computação científica representa regra geral, uma quantidade física ou um número, Por exemplo voltagem, temperatura, pressão, entre outros. No SCIRun esta estrutura é implementada de duas maneiras. *ScalarField* e *VectorField*, correspondendo à função escalar ou vetorial respetivamente. *ScalarField* possui várias implementações, incluindo: *ScalarFieldUG* que contém um *Mesh* e valores para cada nó ou elemento, e *ScalarFieldRG* que define um campo escalar utilizando uma grelha com amostras regulares.

A imagem seguinte mostra a estrutura de classes de *VectorField*, sendo as de *ScalarField* muito semelhantes[9].

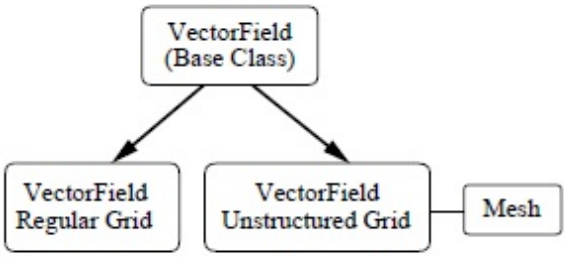

Figura 3.3: estrutura de classes de *VectorField*.

3.1.3.4 Outros tipos de dados

Existem muitos mais tipos de dados implementados no SCIRun; alguns são semelhantes ao tipos de dados normalmente usados em programação. *Strings*, como conjunto de caracteres e valores booleanos de verdadeiro/falso. Alguns dos tipos de dados do SCIRun ainda não mencionados são os seguintes:

- *ColorMap*, utilizado por bastantes módulos de visualização, proporciona um mapeamento de dados para cores.
- *Geometry*, representa objetos renderizáveis.
- *MultiMesh*, conjunto de *meshes* de várias resoluções.
- *Image*, uma imagem 2D.

3.2 Transporte de módulos para o SCIRun 5

A parte mais importante da solução de esta dissertação, passa por realizar o transporte dos módulos Tomo-GPU desenvolvidos no SCIRun4. Embora toda a lógica do código do

SCIRun4 e do SCIRun5 seja a mesma, existem várias diferenças nas suas implementações. Esta secção apresenta como será tratada a conversão dos módulos para o novo SCIRun.

3.2.1 Alteração do código fonte

A versão anterior do SCIRun utiliza Tcl/Tk para sua interface gráfica enquanto que a nova versão do SCIrun utiliza Qt. Não é possível a conversão automática de Tcl/Tk para Qt, pelo que todos os módulos, têm de ser criados com uma interface gráfica baseada em Qt construída de raíz.

Vários métodos e estruturas de dados do SCIRun4 já não existem, estes tem de ser substituídos pele seu equivalente do SCIRun5. Por exemplo a definição das portas de um módulo já não é feita através de uma simples inclusão de um ficheiro *header*. No SCIRun5 as portas de cada módulo são considerados estruturas de dados, e cada porta do módulo é definido no respetivo ficheiro *header* do módulo.

Uma outra alteração entre as versões do SCIRun 4 e o SCIRun 5 é a implementação dos mutexes. A versão anterior utilizava a interface *pthreads*, em *Windows*. Apesar de existir uma implementação *pthreads* para *Windows* ela está em 32 bits, não sendo compatível com esta aplicação, que está em 64 bits. Utilizou-se então a própria implementação do *Windows* dos mutexes, os *threadHandles*. As chamadas para criar, destruir, bloquear e desbloquear os mutexes são muito semelhantes, pelo que a sua alteração foi trivial.

3.2.2 Módulo genérico

Um dos objetivos planeados para esta dissertação, era a implementação de um módulo SCIRun que pudesse executar programas externos ao SCIRun. A grande vantagem de este módulo é que ele permite integrar num *workflow* SCIRun módulos externos ao SCIRun, e assim, evitar os gastos de tempo da (re)compilação do SCIRun. Se esta facilidade não existisse, cada alteração ao programa em questão envolveria uma reconstrução de todo o SCIRun. Assim pode-se criar um programa, com a linguagem de programação que o utilizador preferir, compilá-lo e de seguida executá-lo através deste módulo genérico.

Para facilitar a transmissão de *input* e *output* entre o SCIRun e o módulo genérico, adaptou-se a ideia inicial de uma comunicação direta entre módulos pelas ligações entre portas. Em vez disso para o *input* e *output* utilizam-se leitura e escritas de ficheiros. O módulo genérico recebe os dados a computar normalmente através da sua porta de *input*e escreve-os num documento. Quando o módulo inicializar o programa externo, este irá usar os dados do ficheiro. Quando a sua computação estiver completa, o programa em questão escreverá os seus resultados num outro ficheiro, que será lido pelo módulo genérico e traduzido para o tipo de dados SCIRun pretendido.

A figura 3.4 representa o funcionamento do módulo genérico criado.

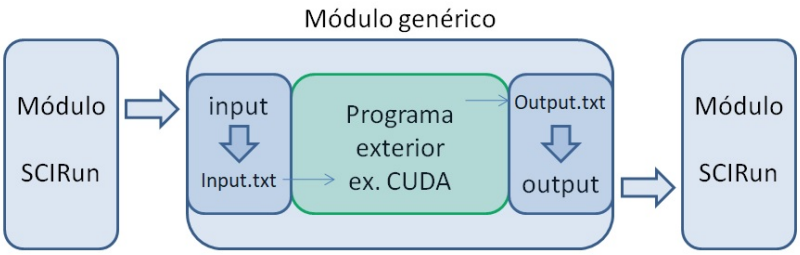

Figura 3.4: Esquema de funcionamento do módulo genérico.

3.3 Módulos Tomo-GPU

O Tomo-GPU é composto por um conjunto de módulos SCIRun, especializados para a análise, computação e visualização de imagens de tomografia computadorizada. Inicialmente este projeto tinha sido desenvolvido para a versão anterior do SCIRun, mas no âmbito de esta dissertação, procedeu-se à implementação do Tomo-GPU para a nova versão do SCIRun, a versão 5.

O Tomo-GPU possui módulos de vários tipos. Eles podem ser divididos em 4 categorias:

- Caracterização
- Processamento de imagem
- Ferramentas
- Visualização

Dentro do Tomo-GPU existem alguns módulos que são essenciais para o objetivo para o qual ele foi criado, a caracterização de imagens tomográficas de compostos. Os seguintes módulos são considerados os mais importantes e realizam as computações mais específicas para esta temática. Todos eles fazem uso de OpenCL para o processamento em paralelo no GPU.

- *Segmentation*
- *Hysteresis*
- *Image Labeling*
- *Image Cleaning*
- *VisAttributes*

3.3.1 *Segmentation e Bi-segmentation*

O módulo *Segmentation* possui 2 implementações distintas, que podem ser selecionadas no seu UI: Segmentação e Bi-Segmentação.

O propósito de ambas é o de dividir uma imagem em 3 cores. Branco, preto e cinzento. Em termos computacionais, aqui a cor branca é definida com o valor 255, e o preto com o valor 0. O cinzento depois serão todos os outros valores entre 0 e 255. É definido no UI do módulo os valores mínimos e máximos na qual os cinzentos são transformados em branco, ou preto. Se o valor do *voxel* for inferior ao valor mínimo estipulado, o *voxel* será preto. Se o valor for maior que o máximo estipulado, este será branco. Todos os cinzentos são posteriormente convertidos para um único valor de cinzento 127.

A figura seguinte demonstra o algoritmo para este módulo. De salientar que é possível criar um *thread* com este excerto de código para cada *voxel*, permitindo um fácil e eficiente mapeamento para um GPU.

```
for cada voxel da imagem do
    if cor ≤ minimo then
        cor ← preto
    else
        if cor ≥ máximo then
            cor ← branco
        else
            cinzento ← cinzento(127)
```

Figura 3.5: Pseudo-código do algoritmo da Segmentação.

3.3.2 *Hysteresis*

A histerese consiste essencialmente na conversão de *voxels* cinzentos para branco e preto, dentro das regiões cinzentas, que estão geralmente entre regiões brancas e pretas. Este algoritmo analisa a vizinhança de cada *voxel* e, por exemplo, se existirem *voxels* vizinhos brancos, e nenhum preto o *voxel* cinzento ficará branco. Se na vizinhança existirem *voxels* de ambas as cores, a cor branca ou preta é atribuída aleatoriamente. A histerese recebe como *input* uma matriz 3D com três valores, branco, preto e cinzento, e devolve como *output* a mesma matriz, mas somente com valores brancos e pretos. Esta operação pode ser definida pelo seguinte pseudo-código:

```
for cada voxel cinzento do
    examina a cor da fronteira
    if cor predominate é branco then
        interior ← branco
    else
```

if cor predominate é preto **then**
    $interior \leftarrow preto$
else
    $interior \leftarrow Random(branco|preto)$

Figura 3.6: Pseudo-código do algoritmo da Histerese.

3.3.3 *Image Labeling*

Esta fase conclui o processo de segmentação da imagem atribuindo a cada reforço uma eiqueta única. Essa análise é feita através de um algoritmo chamado *ObjectIdentifier* [10].

Image Labeling é utilizado no Tomo-GPU quando uma imagem já passou pela segmentação e histerese. Este módulo recebe uma matriz 3D, em que cada *voxel* já está caracterizado como somente branco ou preto.

Image Labeling produzirá dois *outputs*:

- Uma matriz 3D em que cada região constituída por uma série contínua de *voxels* pretos é denominada como sendo um objeto. De salientar que cada um desses objetos,etiquetada com um rótulo distinto.
- O segundo *output* trata-se de uma série de inteiros com várias informações acerca da imagem, tais como:
  - Número de objetos distintos,
  - Para cada objeto, o respetivo rótulo, número de *voxels* e as coordenadas de cada *voxel*.

Estes dois *outputs* são produzidos para auxiliar na optimização dos processos seguintes, nomeadamente a caracterização individual de cada objeto. Utilizando o output gerado, é realizada uma análise a cada objeto identificado. Em pormenor, será feita para cada objeto, uma caraterização do ponto de vista geométrico. Será calculado volume, momento de inércia, área, entre outros.

3.3.4 *Image Cleaning*

Após a execução da Bi-segmentação, histerese e *Image Labelling* é obtida uma lista de reforços, em que para cada reforço existe o seu identificador, a sua dimensão em voxeis e a localização de cada voxel. A operação de *Image Cleaning* remove os reforços com dimensão inferior a um mínimo especificado. O funcionamento deste módulo é brevemente descrito no seguinte pseudo-código.

for cada partícula da imagem **do**
    if dimensão da partícula $<$ limite mínimo **then**
        remove partícula

Figura 3.7: Pseudo-código do algoritmo do Image Cleaning.

3.3.5 *VisAttributes*

VisAttributes é um módulo de visualização que mostra as características das partículas. As partículas que foram identificadas no *Image Labelling*. A imagem seguinte exemplifica a utilização do módulo *VisAttributes* para mostrar a distribuição da dimensão das partículas[5].

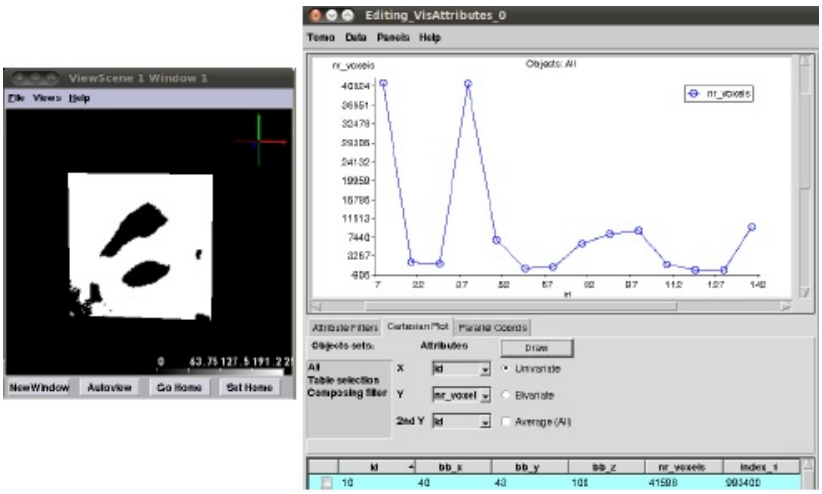

Figura 3.8: Módulo de visualização *VisAttr* do Tomo-GPU.

3.4 Conclusões

Este capítulo preparou o caminho para o capítulo seguinte onde se descreve o transporte do Tomo-GPU para o SCIRun 5. Com este objectivo foram estudados diversos aspetos, tais como:

- A forma de funcionamento das redes e dos módulos do SCIRun,
- As diferenças entre o SCIRun 4 e o SCIRun 5.
- A funcionalidade dos principais módulos do Tomo-GPU.

Este transporte só é possível porque o SCIRun é um sistema completamente aberto, disponibilizando todo o seu código fonte e tendo também acessível documentação sobre vários aspetos do sistema, desde a forma de o utilizar até à descrição da forma de desenvolver novos módulos [13].

Capítulo 4

Implementação da solução

Neste capítulo descreve-se o transporte do Tomo-GPU para o SCIRun5.

4.1 Módulos SCIRun

Nesta primeira secção é explicado como é criado um novo módulo no SCIRun.

Para criar um módulo SCIRun apenas são necessários três ficheiros. Um ficheiro de configuração *modulo.module*, e os seus ficheiros de código fonte e *header (modulo.cc e modulo.h)*. O ficheiro de configuração inclui uma descrição do módulo, a que tipologia pertence, bem como os nomes de todos os outros ficheiros necessários para a execução do mesmo.

No entanto para módulos mais avançados, são necessários mais componentes. Existe a possibilidade de construir módulos com uma interface gráfica específica ao mesmo. Para tal são necessários mais outros três ficheiros: além do ficheiro com o UI, criado através do editor de UIs do QT *(QT Creator)* são também necessário seu *source code* e respectivo ficheiro de *header*.

Para aqueles módulos que requerem uma lógica ou um código mais extensivo, existe a possibilidade de incluir ficheiros com os algoritmos desse módulo. Isto contribui para uma fatorização e optimização do SCIRun, e do próprio código a ser desenvolvido, pois é sempre possível re-utilizar algoritmos de outros módulos. Nos módulos em causa, apenas é necessário acrescentar mais dois ficheiros, nomeadamente o *source code* com o algoritmo e o seu respectivo *header*.

4.1.1 Ficheiro de configuração

Como referido anteriormente, um módulo SCIRun necessita de principalmente do ficheiro *.module* e dos ficheiros com de *source code* e *header*. Este ficheiro de configuração contém

todas as informações necessárias para que o *module factory* do SCIRun possa fazer todas as ligações necessárias para a inclusão do módulo. É um ficheiro de texto com essencialmente 3 campos: *"module", "algorithm"* e *"UI"*. Essencialmente em cada campo existe a definição da existência, ou não, de ficheiros de algoritmo ou de UI para o módulo. Se existirem, é então referido o nome e a localização desses ficheiros. Este ficheiro deve ser colocado em *src/Modules/Factory/Config*. Nessa pasta existe inclusivé um ficheiro *CMakeLists.txt* que tem de ser sempre actualizado, sempre que se acrescenta um novo ficheiro *.module*.

De seguida são apresentados dois exemplos de um ficheiro de configuração, um deles com algoritmo e UI, e outro com somente a definição de uma UI.

```
{
  "module": {
    "name": "@ModuleName@",
    "namespace": "Fields",
    "status": "description of status",
    "description": "description of module",
    "header": "Modules/Template/ModuleTemplate.h"
  },
  "algorithm": {
    "name": "@AlgorithmName@Algo",
    "namespace": "Fields",
    "header": "Core/Algorithms/Template/AlgorithmTemplate.h"
  },
  "UI": {
    "name": "@ModuleName@Dialog",
    "header": "Interface/Modules/Template/ModuleDialog.h"
  }
}
```

Figura 4.1: Exemplo de ficheiro com algoritmo e UI.

```
{
    "module": {
        "name": "BandPass",
        "namespace": "Tomo",
        "status": "new module",
        "description": "TomoGPU module.",
        "header": "Modules/Tomo/BandPass.h"
    },
    "algorithm": {
        "name": "N/A",
        "namespace": "N/A",
        "header": "N/A"
    },
    "UI": {
        "name": "BandPassDialog",
        "header": "Interface/Modules/Tomo/BandPassDialog.h"
    }
}
```

Figura 4.2: Exemplo de ficheiro sem algoritmo, e com UI.

4.1.2 Ficheiro *.header*

O ficheiro *header* funciona como um típico C++ *header*. Este contém especificações acerca da estrutura do *source code*. No caso do SCIRun este contém também informações acerca do portas a serem utilizadas pelos módulos. Nomeadamente quantas existem e o tipo. É também no ficheiro de *header* onde são instanciadas as variáveis do UI a serem utilizadas pelo *source code*. O *header* deve ser colocado em *src/Modules/...*. Semelhantemente ao ficheiro de configuração, existe também um ficheiro *CMakeLists.txt* que deve ser actualizado não só com o *header* mas com o código fonte. O exemplo seguinte representa o ficheiro *header* de um dos módulos do Tomo-GPU, o BandPass.

```
#ifndef MODULES_TOMO_BandPass_H
#define MODULES_TOMO_BandPass_H

#include <Dataflow/Network/Module.h>
#include <Modules/Tomo/share.h>

namespace SCIRun {
namespace Modules {
namespace Tomo {

class SCISHARE BandPass : public SCIRun::Dataflow::Networks::Module,
public Has1InputPort<FieldPortTag>,
public Has1OutputPort<FieldPortTag>
{
public:
  BandPass();
  virtual void execute() override;
  virtual void setStateDefaults() override;

  INPUT_PORT(0, InputField, Field);
  OUTPUT_PORT(0, OutputField, Field);

  static const Core::Algorithms::AlgorithmParameterName LowerGrey;
  static const Core::Algorithms::AlgorithmParameterName LowerSpinBox;
  static const Core::Algorithms::AlgorithmParameterName HigherGrey;
  static const Core::Algorithms::AlgorithmParameterName HigherSpinBox;
  static const Core::Algorithms::AlgorithmParameterName OtherSpinBox;
  static const Core::Algorithms::AlgorithmParameterName OtherSlider;
  static const Core::Algorithms::AlgorithmParameterName WriteChecked;
  static const Core::Algorithms::AlgorithmParameterName SetConstantValues;


  MODULE_TRAITS_AND_INFO(ModuleHasUI)
};
}}}


#endif
```

Figura 4.3: *header* do módulo BandPass do TomoGPU.

Por ordem de ocorrência deve-se salientar os seguintes pormenores:

```
#ifndef MODULES_TOMO_BandPass_H
#define MODULES_TOMO_BandPass_H

#include <Dataflow/Network/Module.h>
#include <Modules/Tomo/share.h>
```

A definição e o *include* devem estar presentes em todos os módulos. Especificamente o *share.h* deve ser o último *include*, senão não é possível fazer *build* em *Windows*.

```
namespace SCIRun {
namespace Modules {
namespace Tomo {
```

O último *namespace* deve corresponder ao especificado no ficheiro de configuração.

```
public HasIInputPort<FieldPortTag>,
public HasIOutputPort<FieldPortTag>
```

Neste campo é definido o número e o tipo de portas do módulo. É possível existir até o máximo de 7 portas de *input* e 9 portas de *output*. Existe também a possibilidade de não haver portas de *input* e de *output*. De mencionar também a existência de portas dinâmicas. Estas actuam como um vector de portas. Dentro do tipo de portas de salientar os mais comuns, *Field,Matrix, Nrrd* e *String*.

```
public:
  BandPass();
  virtual void execute() override;
  virtual void setStateDefaults() override;
```

Estas funções são obrigatórias para todos os módulos.

```
INPUT_PORT(0, InputField, Field);
OUTPUT_PORT(0, OutputField, Field);
```

Na definição das portas, cada uma delas deve ter um nome único entre elas. Cada tipo de porta deve corresponder ao definido anteriormente.

```
static const Core::Algorithms::AlgorithmParameterName LowerGrey;
```

Estas variáveis são utilizadas pelo *source code* para aceder a parâmetros existentes no UI. Cada um deles está definido no ficheiro .ui em *Interface/Modules*.

```
MODULE_TRAITS_AND_INFO(ModuleHasUI)
```

Esta linha é também obrigatória para qualquer *header*. Deve corresponder ao definido anteriormente no ficheiro de configuração.

4.1.3 Ficheiro *source code*

Como num típico ficheiro *source* de C++, aqui existe toda a programação e lógica do módulo. Semelhante ao *header*, a sua localização deve ser em *src/Modules/....* Também nesta directoria existirá um ficheiro *CMakeLists.txt* que deverá ser actualizado.

Para além de ser necessário instanciar as portas e os valores *default* dos parâmetros, existe também a obrigação de em *MODULE_INFO_DEF(BandPass, Tomo, SCIRun)* estar especificado no segundo parâmetro a diretoria à qual o módulo pertence. Tal como está descrito no *header* e no ficheiro de configuração. No construtor do módulo, deve conter Module(staticInfo) para efeitos de construção da aplicação SCIRun. Por fim é na função *execute()* que os dados de *input* são tratados e onde acontece a sua computação. É também aí onde o *output* é criado e enviado para a porta de saída do módulo caso ela exista.

```
#include <Modules/Tomo/BandPass.h>
#include <Dataflow/Network/ModuleStateInterface.h>

using namespace SCIRun;
using namespace SCIRun::Modules::Tomo;
using namespace SCIRun::Core::Datatypes;
using namespace SCIRun::Dataflow::Networks;
using namespace SCIRun::Core::Algorithms;


const AlgorithmParameterName BandPass::LowerGrey("LowerGrey");
const AlgorithmParameterName BandPass::LowerSpinBox("LowerSpinBox");
const AlgorithmParameterName BandPass::HigherGrey("HigherGrey");
const AlgorithmParameterName BandPass::HigherSpinBox("HigherSpinBox");
const AlgorithmParameterName BandPass::OtherSpinBox("OtherSpinBox");
const AlgorithmParameterName BandPass::OtherSlider("OtherSlider");
const AlgorithmParameterName BandPass::WriteChecked("WriteChecked");
const AlgorithmParameterName BandPass::SetConstantValues("SetConstantValues");

MODULE_INFO_DEF(BandPass, Tomo, SCIRun)

BandPass::BandPass() : Module(staticInfo_)
{
    INITIALIZE_PORT(InputField);
    INITIALIZE_PORT(OutputField);
}

void BandPass::setStateDefaults()
{
  auto state = get_state();
  state->setValue(LowerGrey, 80);
  state->setValue(LowerSpinBox, 80);
  state->setValue(HigherGrey, 100);
  state->setValue(HigherSpinBox, 100);
  state->setValue(OtherSlider, 100);
  state->setValue(OtherSpinBox, 100);
  state->setValue(SetConstantValues, 0);
  state->setValue(WriteChecked, 0);
}


void
BandPass::execute()
{
```

Figura 4.4: *Source* simplificado do módulo BandPass do Tomo-GPU.

4.1.4 Ficheiros de UI

Para formar uma UI para um módulo são necessários três ficheiros, um ficheiro *src* .cc, um *header* e um ficheiro de design .ui criado em Qt. Este ficheiro .ui é criado no Qt *designer*. Estes ficheiros devem estar localizados na diretoria *src/Interface/Modules/...* . Como nos casos anteriores existe nesta diretoria um ficheiro *CMakeLists.txt* que precisa de ser atualizado com estes três ficheiros para a correta integração do módulo ao SCIRun.

4.1.4.1 Ficheiro de design

É neste ficheiro que é especificado o tipo, o posicionamento, as propriedades de todos os *widgets* da interface gráfica do módulo. A janela *Widget Box* permite ao utilizador escolher e colocar novos objetos no UI. O editor de propriedades permite, como o nome indica, modificar a propriedades dos *widgets*, incluindo nome, tipo de *input*, tamanho,

valores iniciais, entre outros. Um dos pormenores interessantes neste ficheiro é que o comportamento *default* do ficheiro, atribuí um tipo de valor "Fixo"em ***sizePolicy***. Isto faz com que o tamanho real do módulo seja muito menor do que o tamanho verdadeiro do mesmo. (Propriedade ***geometry***). Deve-se atribuir a ***size policy*** o valor *"Prefered"* e fazer corresponder o valor de ***geometry*** ao atributo ***minimumSize***. A imagem seguinte serve de exemplo de um típico ficheiro de UI de um módulo do SCIRun. Neste caso trata-se da UI do BandPass, presente em Tomo-GPU.

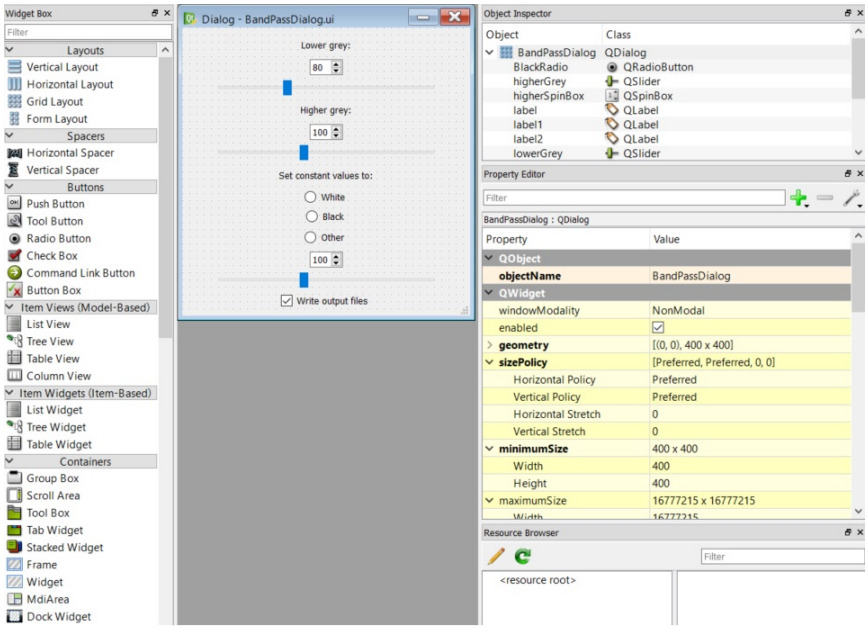

Figura 4.5: Ficheiro UI do módulo BandPass do Tomo-GPU.

4.1.4.2 Ficheiro *header* do UI

Este ficheiro correponde ao tradicional C++ *header* de um ficheiro *source code*, neste caso da parte do UI. Este ficheiro comporativamente aos anteriores, possui regra geral nenhumas ou poucas alterações para o ficheiro *template* do SCIRun. Somente em casos de especificamento de funções presentes no código fonte é que o *header* do UI é alterado. De seguida é apresentado um exemplo de um *header* de um UI de um dos módulos do Tomo-GPU, o BandPass.

```
#ifndef INTERFACE_MODULES_TOMO_BandPassDialog_H
#define INTERFACE_MODULES_TOMO_BandPassDialog_H

#include <Interface/Modules/Tomo/ui_BandPassDialog.h>
#include <Interface/Modules/Base/ModuleDialogGeneric.h>
#include <Interface/Modules/Tomo/share.h>

namespace SCIRun {
namespace Gui {

class SCISHARE BandPassDialog : public ModuleDialogGeneric,
  public Ui::BandPassDialog
{
    Q_OBJECT

public:
  BandPassDialog(const std::string& name,
    SCIRun::Dataflow::Networks::ModuleStateHandle state,
    QWidget* parent = 0);

    private Q_SLOTS:
                void setLowerSlider();
                void setLowerSpinBox();
                void setHigherSlider();
                void setHigherSpinBox();
                void setOtherSlider();
                void setOtherSpinBox();
};

}
}

#endif
```

Figura 4.6: Ficheiro *header* do UI de BandPass, presente no Tomo-GPU.

4.1.4.3 Ficheiro *source code* do UI

É aqui no código fonte do UI que se estabelece a ligação entre o UI e *source code* do módulo em si. A principal parte de este código é a correspondência das variáveis do *source code* com os elementos do UI. Cada tipo de *widget* diferente corresponde a uma função *"Manager"* diferente. De salientar que nenhum valor dos *widget* do tipo *slider* são utilizados pelo código fonte. Somente as *spinboxs* correpondendo aos valores do *slider* é que são usadas. Estas são interligadas através de uma função *connect()* que guarda os valores e suas mudanças, numa variável a ser posteriormente utilizada no código fonte.

```
#include <Interface/Modules/Tomo/BandPassDialog.h>
#include <Modules/Tomo/BandPass.h>
#include <Dataflow/Network/ModuleStateInterface.h>

using namespace SCIRun::Gui;
using namespace SCIRun::Dataflow::Networks;
using namespace SCIRun::Modules::Tomo;

BandPassDialog::BandPassDialog(const std::string& name, ModuleStateHandle state,
  QWidget* parent /* = 0 */)
  : ModuleDialogGeneric(state, parent)
{
  setupUi(this);
  setWindowTitle(QString::fromStdString(name));
  fixSize();

  addSpinBoxManager(lowerSpinBox, BandPass::LowerSpinBox);
  connect(lowerGrey, SIGNAL(valueChanged(int)), this, SLOT(setLowerSpinBox()));
  connect(lowerSpinBox, SIGNAL(valueChanged(int)), this, SLOT(setLowerSlider()));

  addSpinBoxManager(higherSpinBox, BandPass::HigherSpinBox);
  connect(higherGrey, SIGNAL(valueChanged(int)), this, SLOT(setHigherSpinBox()));
  connect(higherSpinBox, SIGNAL(valueChanged(int)), this, SLOT(setHigherSlider()));

  addSpinBoxManager(otherSpinBox, BandPass::OtherSpinBox);
  connect(otherSlide, SIGNAL(valueChanged(int)), this, SLOT(setOtherSpinBox()));
  connect(otherSpinBox, SIGNAL(valueChanged(int)), this, SLOT(setOtherSlider()));

  addCheckBoxManager(writeChecked, BandPass::WriteChecked);
  addRadioButtonGroupManager({whiteRadio,BlackRadio,otherRadio},BandPass::SetConstantValues);
}

void BandPassDialog::setLowerSlider() {
    lowerGrey->setValue(lowerSpinBox->value());
}

void BandPassDialog::setLowerSpinBox() {
    // Store the decimal portion of the spin box so that it's not lost when the slider value changes.
    int temporary = lowerSpinBox->value() - (int)lowerSpinBox->value();
    lowerSpinBox->setValue(lowerGrey->value() + temporary);
}
```

Figura 4.7: Excerto do código fonte do UI de BandPass, presente no Tomo-GPU.

4.1.5 Ficheiros de Algoritmo

Os módulos que tenham um ou vários algoritmos associados, vêem-se acrescido de pelo menos mais dois ficheiros. Nomeadamente um *header* e o respetivo *source code* Os algoritmos devem ficar localizados na directoria *src/Core/Algorithm/....* Mais uma vez existe um *CMakeLists.txt* que tem conter os ficheiros a serem adicionados. Ambos os ficheiros .h e .cc seguem as regras tradicionais de um típico projeto C++. Nos dois exemplos abaixo é demonstrado um template para os ficheiros de algoritmo do SCIRun. Estes exemplos podem ser encontrados na diretoria *srs/Core/Algorithm/Template* [13].

```
#ifndef CORE_ALGORITHMS_FIELDS_@AlgorithmName@_H
#define CORE_ALGORITHMS_FIELDS_@AlgorithmName@_H

#include <Core/Algorithms/Base/AlgorithmBase.h>
#include <Core/Algorithms/Field/share.h>

namespace SCIRun {
  namespace Core {
    namespace Algorithms {
      namespace Fields {
// declare parametes and options in header when not part of standard names.
        ALGORITHM_PARAMETER_DECL(Knob1);
        ALGORITHM_PARAMETER_DECL(Knob2);


         class SCISHARE @AlgorithmName@Algo : public AlgorithmBase
         {
         public:
           @AlgorithmName@Algo();
           virtual AlgorithmOutput run_generic(const AlgorithmInput& input) const;
         };
       }}}}
 #endif
```

Figura 4.8: *Header* do exemplo disponível em *srs/Core/Algorithm/Template*

```
#include <Core/Algorithms/Field/@AlgorithmName@Algo.h>
#include <Core/Algorithms/Base/AlgorithmVariableNames.h>
#include <Core/Algorithms/Base/AlgorithmPreconditions.h>
#include <Core/Datatypes/Legacy/Field/FieldInformation.h>
#include <Core/Datatypes/Legacy/Field/VField.h>
#include <Core/Datatypes/Legacy/Field/VMesh.h>
#include <Core/Logging/Log.h>

using namespace SCIRun;
using namespace SCIRun::Core::Datatypes;
using namespace SCIRun::Core::Algorithms;
using namespace SCIRun::Core::Algorithms::Fields;

//  this function is for setting defaults for state variables.
//  Mostly for UI variables.
@AlgorithmName@Algo::@AlgorithmName@Algo()
{
  using namespace Parameters;
  addParameter(Knob1, false);
  addParameter(Knob2, 1.0);
}

//main algorithm function
AlgorithmOutput @AlgorithmName@Algo::run_generic(const
     AlgorithmInput& input) const
{
  auto inputField = input.get<Field>(Variables::InputField);

  FieldHandle outputField(inputField->deep_clone());
  double knob2 = get(Parameters::Knob2).toDouble();
  if (get(Parameters::Knob1).getBool())
  {
    // do something
  }
  else
  {
    // do something else
  }

  AlgorithmOutput output;
  output[Variables::OutputField] = outputField;
  return output;
}
```

Figura 4.9: Código fonte do exemplo disponível em *srs/Core/Algorithm/Template*

4.2 Conversão de módulos SCIRun4

Nesta secção é explicado o procedimento para a conversão de módulos do SCIRun4 para o mais recente SCIRun5. Este processo pode ser complicado porque existem muitas mudanças a nível de funções, infraestrutura, e interface gráfica. É recomendado utilizar um módulo já existente como ponto de partida e alterar a partir daí, especialmente na parte da interface gráfica que foi completamente remodelada. A principal razão para estas necessidades de

alteração tem a ver com o abandono do Tcl/Tk que foi totalmente retirado do SCIRun5 a favor do Qt.

Essencialmente a conversão dos módulos do SCIRun4 segue as mesmas regras da criação de um novo módulo. O código fonte do módulo pode quase totalmente ser reutilizado, existindo apenas alguns pormenores a serem alterados como por exemplo:

- O tipo de portas já não é um ficheiro *header*. Todos os includes mencionando */FieldPort.h, MatrixPort.h* podem ser retirados.
- *input_handle(...)* já não existe, entrando em vigor getRequiredInput("nome da porta").
- Ficheiros de UI tem de ser completamente contruídos de raíz. Incluindo toda a lógica entre componentes do UI, bem como a própria UI.
- Maior parte dos ficheiros de *include* são diferentes, têm outro nome e estão em diretorias diferentes.

4.3 Implementação do módulo genérico

A implementação do módulo genérico segue as bases de construção de um módulo novo. Este tem uma pequena interface gráfica para o utilizador, e o respectivo código fonte. A interface gráfica lança através de um botão, um explorador de ficheiros *Windows* em ambiente *Windows* ou *Linux* em sistemas *Unix*. O módulo possui também uma linha de texto, que apresenta a localização do executável selecionado.

No que diz respeito à parte lógica do módulo, utiliza-se a chamada da função *system()* que dada uma localização de um ficheiro .exe, .bat ou .com este executa-o como se estivesse numa linha de comandos. Existem várias maneiras de lançar um executável num programa C, mas optou-se por esta implementação, pois permite que o código do módulo SCIRun entre o sistema *Windows* em *Linux* não necessita de alterações entre os 2 sistemas operativos.

É apresentado de seguida um lançamento de um programa CUDA, através deste mesmo módulo genérico. Este módulo traz uma grande vantagem ao sistema SCIRun. Ele permite correr programas externos à plataforma SCIRun. Ou seja, para realizar uma dada computação específica que não esteja disponível no SCIRun, não é necessário escrever um módulo novo e recompilar e reconstruir a aplicação. Pode-se escrever um programa externo, na linguagem que desejar, Java, C, ou C++, desde que possa compilar como um executável.

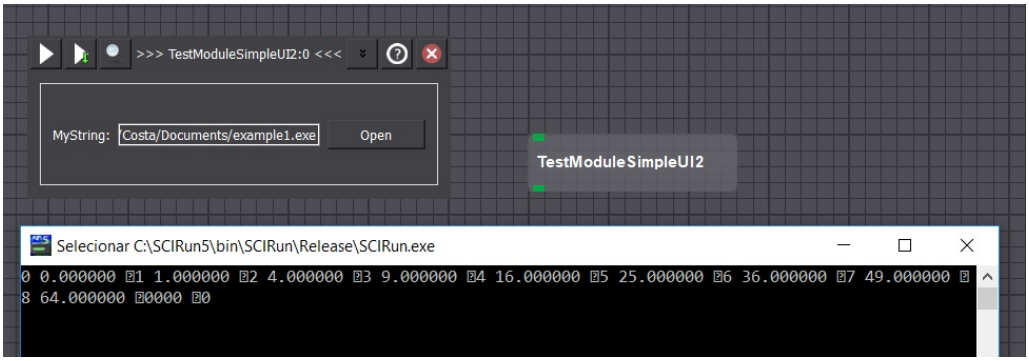

Figura 4.10: Utilização do módulo genérico dentro da aplicação SCIRun.

4.4 Implementação dos módulos Tomo-GPU

O Tomo-GPU apesar de ser constituído essencialmente pelos 5 módulos principais,(*Segmentation, Hysteresis, Image Labelling, Image Cleaning e VisAttributes*) é na verdade um pacote de mais de 30 módulos especializados na caraterização de imagens tomográficas. De seguida são apresentadas vários exemplos de implementações dos diversos módulos Tomo-GPU. A maior parte dos módulos têm interfaces gráficas semelhantes, pelo que apenas serão aqui demonstrados alguns exemplos.

4.4.1 Segmentação

Dentro dos módulos convertidos do Tomo-GPU, encontra-se um dos principais para o processo de caracterização de compósitos, a segmentação. Este módulo apresenta uma interface gráfica que requer alguns cuidados. No SCIRun 5, os *sliders* não mostram *feedback* ao utilizador. É necessário ligar o valor de um *slider* com uma *spinbox* de valores. De salientar também que cada objeto da interface gráfica do SCIRun tem associado a si mesmo um estado. Essa componente é nova no SCIRun5, no SCIRun 4 cada componente era associado a nome, aqui existe a noção de estado. Cada objeto tem associado um estado, exceto componentes que agrupem valores ou escolhas. Por exemplo os botões de rádio que diferenciam o método a ser utilizado neste módulo, se a segmentação, se a bi-segmentação é somente um estado.

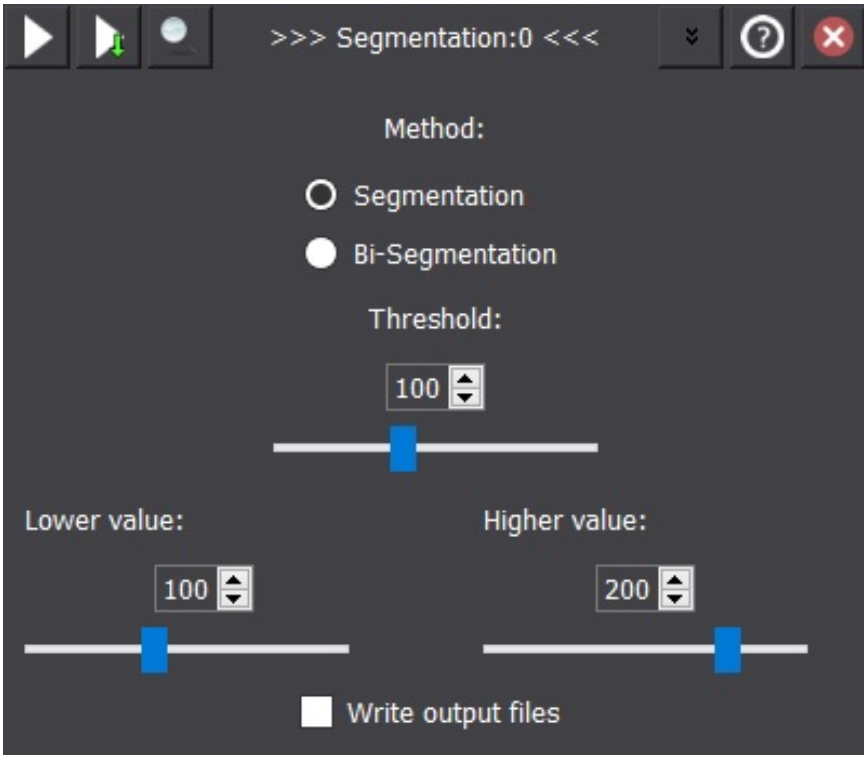

Figura 4.11: Interface gráfica do módulo *Segmentation* dentro da aplicação SCIRun em ambiente Windows.

Apresenta-se de seguida a comparação entre a interface gráfica em Windows 4.11 e a do Linux 4.12.

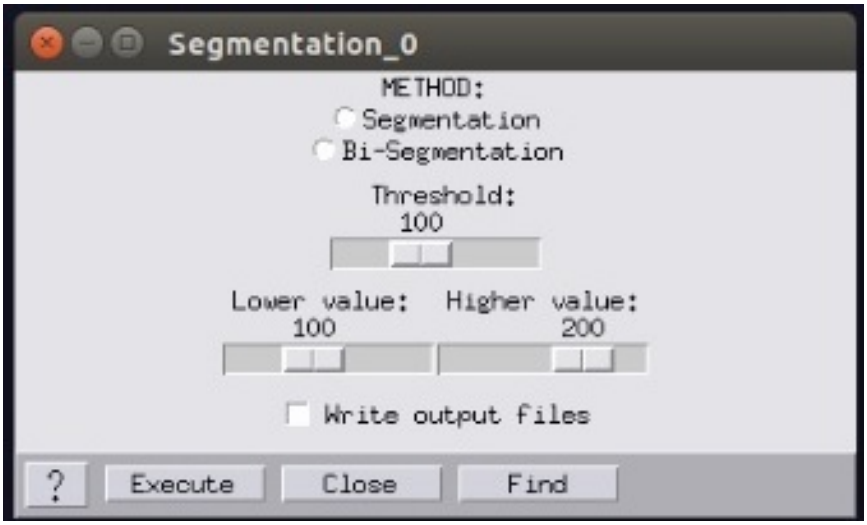

Figura 4.12: Interface gráfica do módulo *Segmentation* dentro da aplicação SCIRun em ambiente Linux.

Este módulo utiliza também OpenCL, como tal este módulo tem associado um ficheiro *kernel* .cl. Esse ficheiro *kernel* tem funções que serão executadas em dispositivos OpenCL, ou seja, maioritariamente placas gráficas ou processadores. Esses ficheiros são associados ao código fonte dos módulos através da sua definição no CMakeList.txt inserido na diretoria onde estão todos os outros módulos Tomo-GPU.

Para o correto funcionamento do sistema, os ficheiros de *kernel* tem de estar na mesma diretoria que o executável do SCIRun, numa pasta OpenCL. É de notar que como referido anteriormente, para todos os módulos Tomo-GPU criados, o método de obtenção de *input* e envio de *output* difere do SCIRun 4 pois a função para obter a informação a partir das portas do módulo mudou no SCIRun 5.

4.4.2 Histerese

Para converter a histerese para a versão 5 do SCIRun, como todos os módulos Tomo-GPU, foi necessário re-implementar toda a sua interface gráfica. Tal e como a segmentação, a histerese também utiliza OpenCL nas suas computações, tendo também um ficheiro *kernel* associado. Apresenta-se de seguida a sua interface gráfica no editor Qt.

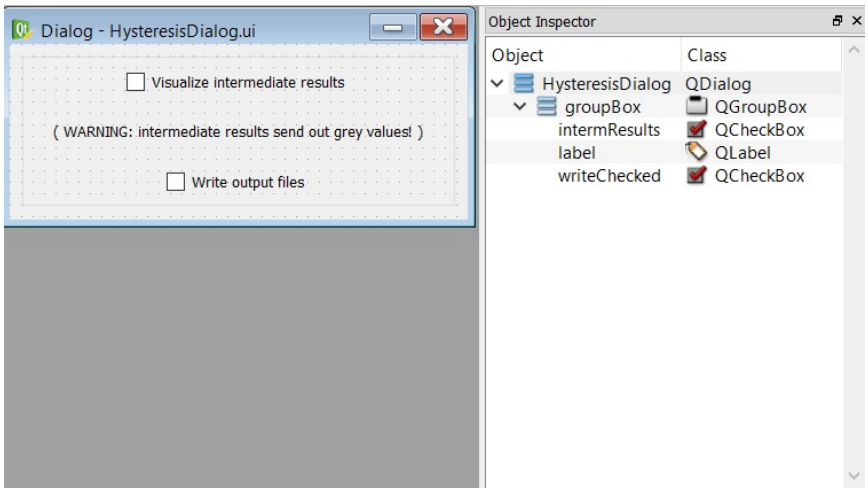

Figura 4.13: Interface gráfica da histerese dentro do editor Qt.

A histerese tem uma interface gráfica bem menos complexa que a segmentação, utilizando somente 2 objetos na sua interface gráfica. Aqui associa-se uma variável booleana para cada *checkbox* da interface gráfica.

4.4.3 *Image Labelling*

Talvez o módulo mais importante de todo o Tomo-GPU, este módulo é responsável pela identificação dos diferentes objetos presentes nas imagens tomográficas. Este módulo não apresenta qualquer tipo de interface gráfica, mas apresenta vários aspetos interessantes do

ponto de vista da programação. O *Image Labelling* possui diversas dependências com outras classes que por sua vez utilizam mutexes. Como referido no capítulo 3, a interface *pthread* foi substituida pelos *thread handles* disponíveis em *Windows*. Apresenta-se de seguida um pequeno exemplo da sua adaptação, com a sua contra parte em Linux comentada.

```
graph(new std::unordered_map<int, std::unordered_set<int> >(DEFAULT_HASHTABLE_SIZE))
{
    //pthread_mutex_init(&exmut, NULL);
     exmut = CreateMutex(NULL, FALSE, NULL);
}

bs::Graph::~Graph()
{
    delete graph;

    //pthread_mutex_destroy( &exmut );
    CloseHandle(exmut);
}


// Get block.
int bs::Graph::getBlock()
{
    int result = -1;

    //pthread_mutex_lock( &exmut );
    WaitForSingleObject(exmut, INFINITE);

    if(counter < blocks)
        result = counter++;

    //pthread_mutex_unlock( &exmut );
    ReleaseMutex(exmut);

    return result;
}
```

Figura 4.14: Exemplo da utilização da interface *thread handles* disponíveis em Windows.

Uma outra curiosa adaptação foi a completa substituição do método para contar ciclos de relógio do processador. De Linux para Windows, os vários métodos necessários para essa computação transformam-se apenas numa linha de código. A parte comentada é utilizada na versão Linux.

```
//void access_counter(unsigned *hi, unsigned *lo)
//{
    //asm("rdtsc; movl %%edx,%0; movl %%eax,%1"
    //  :"=r" (*hi), "=r" (*lo)
        //:
//      : "%edx", "%eax"
    //  );
//}

//void start_counter()
//{
    //access_counter(&cyc_hi, &cyc_lo);
//}

double get_counter()
{
    //unsigned ncyc_hi, ncyc_lo;
    //unsigned hi, lo, borrow;
    //double result;

    //access_counter(&ncyc_hi, &ncyc_lo);
    //lo = ncyc_lo - cyc_lo;
    //borrow = lo > ncyc_lo;
    //hi = ncyc_hi - cyc_hi - borrow;
    //result = (double)hi * (1 << 30) * 4 + lo;

    //if (result < 0)
        //fprintf(stderr, "Error: counter returns neg value: %.of\n", result);

    return (double)__rdtsc();
}
```

Figura 4.15: Exemplo da utilização da função rdtsc() em Windows.

4.4.4 ViewDataSlices

ViewDataSlices é o módulo Tomo-GPU com a interface gráfica mais complexa. Ela apresenta mais de 60 objetos que representam as diversas variáveis a serem utilizadas na execução e interação do módulo. Este módulo é responsável apenas pela visualização das diferentes *slices* realizadas nas amostras.

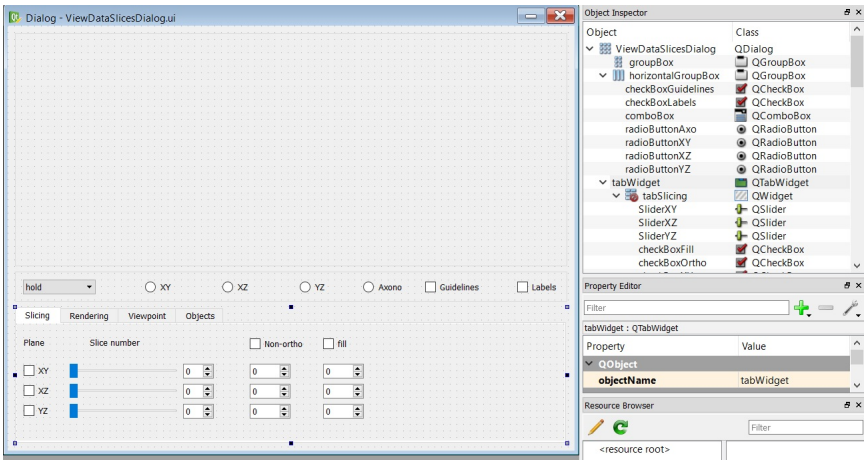

Figura 4.16: Interface gráfica de ViewDataSlices dentro do editor Qt.

Toda a sua interface gráfica foi transportada para o SCIRun 5 com sucesso, apenas a componente funcional, não foi incluída, devido a algumas interfaces do SCIRun 4 terem sido excluídas completamente no SCIRun 5, neste caso o Tcl/Tk.

4.5 Conclusões

Este capítulo detalhou vários aspetos da implementação realizada. O ênfase principal é nas duas possibilidades para transportar os módulos do SCIRun 4 para o SCIRun 5, a saber

- Através da modificação do seu código fonte, que indice especialmente na parte da interface gráfica que usa o Qt.
- Transformando o código de um módulo de forma a torná-lo um programa autónomo e usando o módulo genérico para o lançar em execução.

O capítulo seguinte avalia o transporte dos módulos para o SCIRun 5 principalmente. verificando a sua funcionalidade e comparando o desempenho das versões SCIRun4 e SCIRun 5; também é feita um comparação do desempenho da versão SCIRun5 nos sistemas Linux Ubuntu 16.04 e Windows 10.

Capítulo 5

Avaliação

Neste capítulo avalia-se a funcionalidade e o desempenho dos módulos transportados para o SCIRun

5.1 Metodologia

Para efeitos de avaliação do sistema, ir-se-á comparar o desempenho de várias redes de processamento do Tomo-GPU no SCIRun. Serão utilizadas várias amostras de dimensões e tipos de material diferentes e será medido o tempo de execução. Existirão 3 versões a ser testadas, SCIRun 4 em Linux, SCIRun 5 em Linux, e SCIRun 5 em Windows. Esta avaliação irá permitir estabelecer a comparação de desempenho entre as duas versões.

Para efeitos de medição do desempenho, a versão 4 do SCIRun, já disponibiliza uma medição da duração de cada módulo. Essa duração está representada em cada módulo do SCIRun e é vísivel em tempo de execução, na própria aplicação. O tempo demonstrado em cada módulo representa o instante em que o módulo concluiu a sua execução, desde que toda a rede foi iniciada. Como tal, a duração de cada módulo corresponde à diferença de tempo, entre o tempo demonstrado nesse módulo, com o módulo anterior.

Para que a medição do desempenho seja o mais fidedigna possível entre as diferentes versões do SCIRun, implementou-se um sistema de medição da duração de cada módulo a ser avaliado. A medição da duração dos módulos é somente inicializada após a obtenção do *input* e medida até ao instante anterior em que o módulo envia o seu *output*. Após a execução do módulo é enviado para a consola, uma linha de texto contendo a duração do mesmo.

5.1.1 Carateristicas da máquina de testes

Todos os testes serão realizados num portátil MSI-GE62 2QC, com as seguintes caraterísticas:

Processador: Intel i5-4210H com 2.9GHz,

Placa gráfica: Nvidia GeForce GTX960M com 2GB GDDR5,

Memória RAM: 8GB DDR3 de 1600MHz,

Disco Rígido 1TB a 7200rpm.

O sistema *Windows* utilizado, é o *Windows 10 Pro* de 64 bits com um disco rígido de sensivelmente 800GB com 350GB disponíveis. A versão da *driver* da placa gráfica utilizada é a 391.01. O sistema Linux é uma distribuição Ubuntu 16.04 de 64 bits, com um disco de 50GB. Ambas as versões do SCIRun estão instaladas na mesma distribuição. A versão da *driver* da placa gráfica utilizada é a 384.130. Em ambos os casos é utilizada a plataforma CUDA 9.0. Para efeitos de compilação do SCIRun, a versão 4 é compilada utilizando o gcc-4.6. A versão 5 do SCIRun necessita do gcc-4.8.

5.1.2 Medição do desempenho

Para efeitos de medição de desempenho, as várias versões do SCIRun foram sujeitas a diferentes tipos de amostras. Essas amostras são ficheiros .nrrd (*nearly raw raster data*), característicos por representarem imagens *raster* ou *bitmap* n-dimensionais. Esse tipo de ficheiros são muito utilizados em visualização científica e em aplicações de processamento de imagens.

A primeira amostra é uma amostra .nrrd de dimensões 96x128x96 disponibilizada pelo próprio SCIRun 5, que foi testada em todas as versões do SCIRun. De seguida, pela incompatibilidade do SCIRun 5 em Linux, os ficheiros de amostras com imagens tomográficas não foram possíveis de testar. Essas amostras foram testadas no SCIRun 4 em Linux, e no SCIRun 5 em Windows. Nas versões do SCIRun 4 em Linux e SCIRun 5 em Windows foram testadas as imagens tomográficas com as seguintes dimensões:

- 100x100x100
- 200x200x200
- 400x400x400
- 1024x1024x1024

Em todos os casos, foi medido o desempenho dos 2 primeiros módulos utilizados no processo de caracterização de imagens tomográficas de compostos do Tomo-GPU. A segmentação e a histerese.

5.2 Desempenho do Tomo-GPU original em Linux, versão SCIRun4

Nesta primeira secção, é avaliado o estado atual do Tomo-GPU na versão 4 do SCIRun, a correr em ambiente Linux. A rede de módulos utilizada para a primeira iteração de resultados é demonstrada na imagem seguinte. Esta figura coincide também com o resultado do primeiro teste da amostra 96x128x96. A rede de módulos é constituída por um módulo *ReadField* para importar e carregar o ficheiro de *input .nrrd*, e os 2 módulos Tomo-GPU *Segmentation* e *Hysteresis* ficam responsáveis por desempenhar respetivamente, a segmentação e a histerese da amostra.

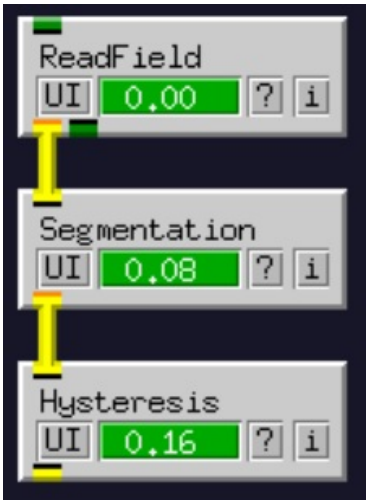

Figura 5.1: Resultado da primeira iteração da amostra 96x128x96.

Neste teste e nos seguintes a rede de módulos foi corrida 5 vezes. Os tempos de execução para a primeira amostra são os seguintes:

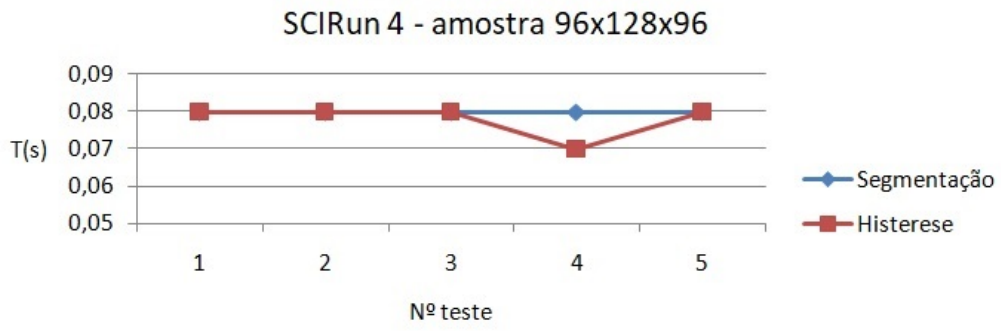

Figura 5.2: Duração dos módulos Tomo-GPU da amostra 96x128x96 no SCIRun4.

	Segmentação	Histerese
Valor mínimo	0,08	0,07
Valor máximo	0,08	0,08
Média	0,08	0,078

Figura 5.3: Análise das durações da amostra 96x128x96 no SCIRun4.

De seguida foram testadas no SCIRun 4 as imagens tomográficas anteriormente referidas. É expetável que, com o aumento do tamanho da amostra, o tempo de duração dos módulos aumente, especialmente na histerese, pois consoante o tamanho do *input* esta tem de realizar mais ciclos. A primeira dessas amostras corresponde a um cubo de dimensões 100x100x100 que gerou os seguintes resultados:

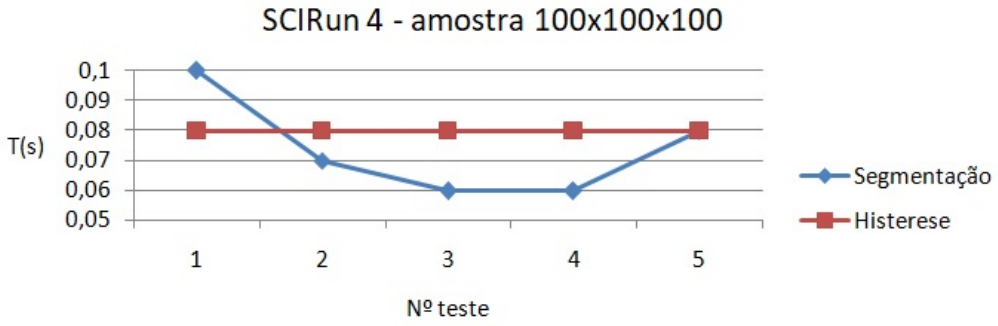

Figura 5.4: Duração dos módulos Tomo-GPU da amostra 100x100x100 no SCIRun4.

	Segmentação	Histerese
Valor mínimo	0,06	0,08
Valor máximo	0,1	0,08
Média	0,074	0,08

Figura 5.5: Análise das durações da amostra 100x100x100 no SCIRun4.

Em comparação com a primeira amostra, apesar do tamanho ter aumentado consideravelmente, existiu um aumento muito pouco significativo na duração dos módulos. Esse aumento, em média, não excedeu os 0,02 segundos para ambos os módulos. Apresenta-se de seguida os resultados para a amostra de 200x200x200.

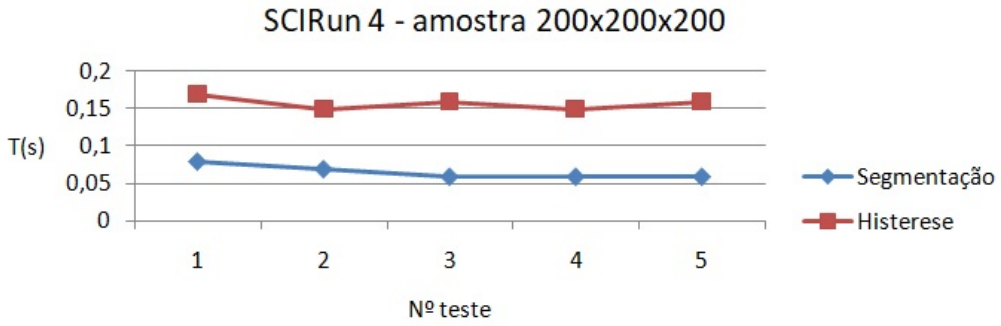

Figura 5.6: Duração dos módulos Tomo-GPU da amostra 200x200x200 no SCIRun4.

	Segmentação	Histerese
Valor mínimo	0,06	0,15
Valor máximo	0,08	0,17
Média	0,066	0,158

Figura 5.7: Análise das durações da amostra 200x200x200 no SCIRun4.

Este teste demonstrou um aumento considerável na duração da computação da histerese. Com uma amostra com o dobro das dimensões da anterior, a histerese demorou, em média, quase o dobro do tempo. A segmentação em média demorou exatamente 1 centésima de segundo a menos que a amostra anterior, aumento esse que pode ser considerado negligenciável, pois essa diferença de tempos pode ser explicada pela diferente carga do processador na altura dos testes. A seguir duplicou-se novamente o tamanho da amostra, agora com dimensões 400x400x400. Seguem-se então os seguintes resultados.

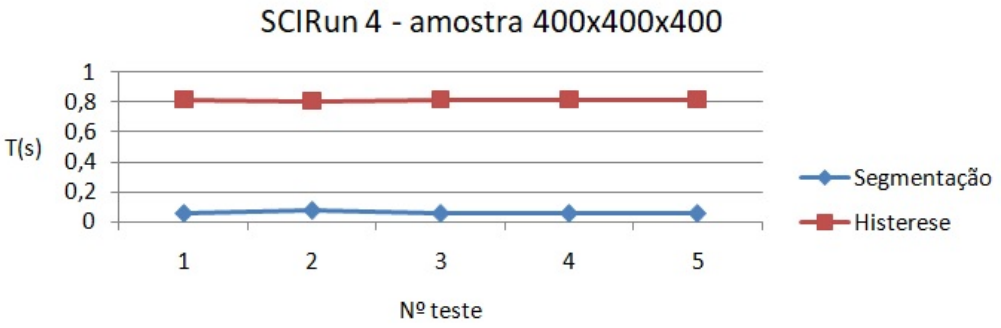

Figura 5.8: Duração dos módulos Tomo-GPU da amostra 400x400x400 no SCIRun4.

	Segmentação	Histerese
Valor mínimo	0,06	0,81
Valor máximo	0,08	0,82
Média	0,064	0,818

Figura 5.9: Análise das durações da amostra 400x400x400 no SCIRun4.

Novamente a duração da segmentação não é afetada pelo aumento da amostra. Por sua vez a histerese apresenta um grande aumento na sua computação. Esta demora 5 vezes mais que o teste anterior, mesmo aumentando 8 vezes o tamanho da amostra. Apresenta-se de seguida os resultados sobre a amostra de maior dimensão, um cubo de 1024x1024x1024.

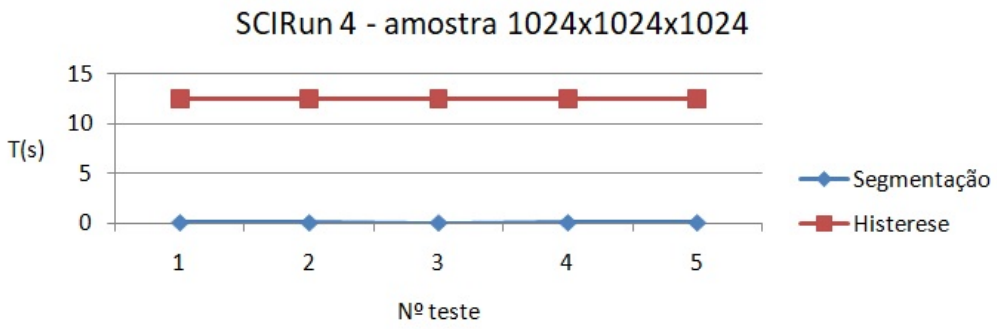

Figura 5.10: Duração dos módulos Tomo-GPU da amostra 1024x1024x1024 no SCI-Run4.

	Segmentação	Histerese
Valor mínimo	0,06	12,5
Valor máximo	0,08	12,53
Média	0,072	12,512

Figura 5.11: Análise das durações da amostra 1024x1024x1024 no SCI-Run4.

Apesar do aumento bastante considerável da amostra, pode-se concluir que não existe praticamente diferença entre uma imagem de dimensão 3x3x3 e 1024x1024x1024 para o

cálculo da segmentação. Já no caso da histerese, comparando as 2 últimas amostras, um aumento de quase dobro e meio na dimensão da amostra resultou que a duração fosse 15 vezes superior à anterior.

5.3 Desempenho do Tomo-GPU em Linux, versão SCIRun5)

Nesta secção foi testada a nova versão do SCIRun mas em ambiente Linux. O teste utiliza uma rede de módulos igual à da versão 4 do SCIRun. Como o SCIRun 5 não possui um medidor de duração incorporado, criou-se um que, como mencionado anteriormente, mede o tempo de execução entre o instante após a recepção do *input* e antes do envio do *output*. De notar que na versão do SCIRun 5, optou-se por uma medição sensível ás milésimas e não às centésimas como no SCIRun 4. Apresentam-se de seguida os tempos de execução do único teste que foi possível realizar. Esta limitação dos testes deve-se a uma limitação da versão SCIRun5 em Linux no processamento de ficheiros no formato *.nrrd*.

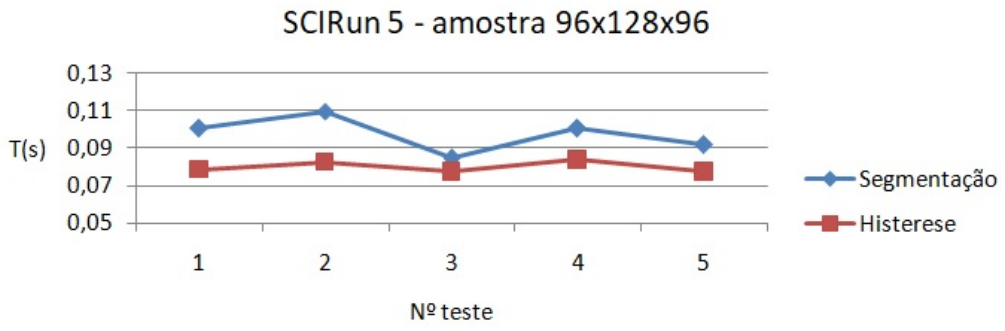

Figura 5.12: Duração dos módulos Tomo-GPU da amostra 96x128x96 no SCIRun5 em Linux.

	Segmentação	Histerese
Valor mínimo	0,085	0,078
Valor máximo	0,11	0,084
Média	0,0978	0,0804

Figura 5.13: Análise das durações da amostra 96x128x96 no SCIRun5 em Linux.

Uma primeira comparação que se pode fazer acerca do desempenho entre o SCIRun 4 e o SCIRun 5 em Linux, é que o desempenho da histerese foi melhor que o da segmentação.

Mais conclusões serão apresentadas de seguida, na comparação entre as versões SCIRun 4 em Linux e o SCIRun 5 em Windows.

5.4 Desempenho do Tomo-GPU em Windows, versão SCIRun5

Nesta secção é avaliado a versão 5 do SCIRun, em ambiente Windows. Todos os testes efetuados anteriormente pelo SCIRun 4 serão também reproduzidos. Utilizou-se exatamente a mesma rede de módulos que os testes anteriores.

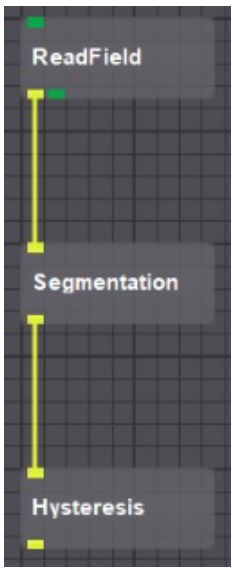

Figura 5.14: Rede de módulos utilizada nos testes do SCIRun5 em Windows.

De seguida é apresentado os resultados obtidos pelo primeiro teste, utilizando a amostra de 96x128x96.

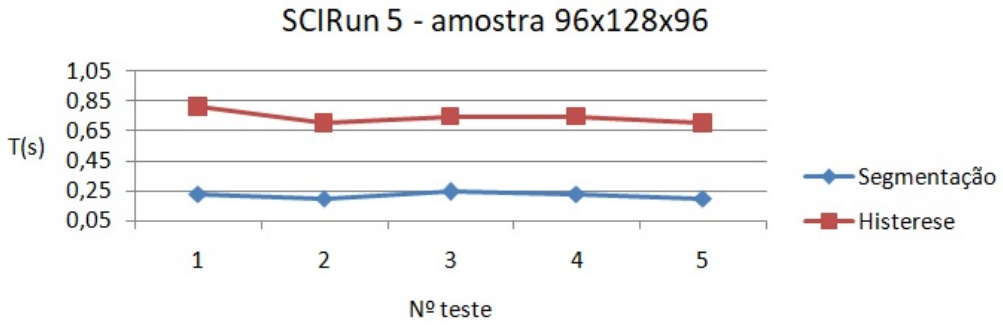

Figura 5.15: Duração dos módulos Tomo-GPU da amostra 96x128x96 no SCIRun5 em Windows.

	Segmentação	Histerese
Valor mínimo	0,203	0,703
Valor máximo	0,249	0,812
Média	0,2246	0,7432

Figura 5.16: Análise das durações da amostra 96x128x96 no SCIRun5 em Windows.

Já na amostra inicial pode-se confirmar um aumento considerável da duração dos módulos, em comparação com os SCIRun em ambiente Linux. Até mesmo o processo de segmentação, que nos testes anteriores registou quase nenhuma alteração, aqui demorou pouco mais que o dobro do tempo. A histerese por sua vez registou um duração quase 9 vezes superior.

De seguida são apresentados os testes às amostras tomográficas. Será feita para cada um dos testes, uma resumida comparação entre as amostras anteriores. A primeira amostra diz respeito a um cubo de dimensão 100x100x100.

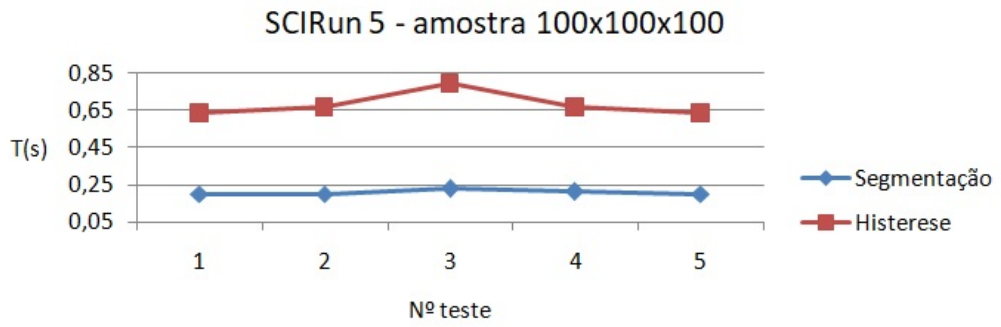

Figura 5.17: Duração dos módulos Tomo-GPU da amostra 100x100x100 no SCIRun5 em Windows.

	Segmentação	Histerese
Valor mínimo	0,203	0,64
Valor máximo	0,234	0,796
Média	0,2122	0,6838

Figura 5.18: Análise das durações da amostra 100x100x100 no SCIRun5 em Windows.

Entre os 2 primeiros testes em SCIRun 5 em Windows, não se apresentam grandes diferenças na duração da computação das amostras. Existe um aumento de sensivelmente 2 centésimas entre os 2 módulos Tomo-GPU. A seguir são apresentados os resultados obtidos pela amostra de 200x200x200.

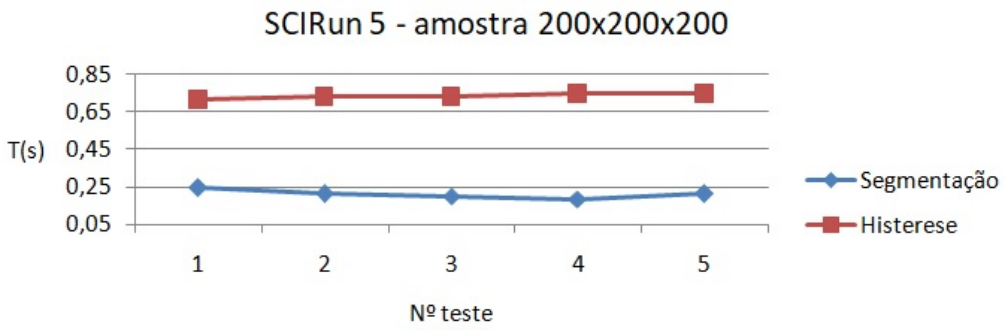

Figura 5.19: Duração dos módulos Tomo-GPU da amostra 200x200x200 no SCIRun5 em Windows.

	Segmentação	Histerese
Valor mínimo	0,187	0,718
Valor máximo	0,249	0,749
Média	0,215	0,7368

Figura 5.20: Análise das durações da amostra 200x200x200 no SCIRun5 em Windows.

Neste teste, os resultados entre as duas últimas amostras são novamente muito semelhantes. De registar um subida na duração da histerese, em média de 5 centésimas de segundo. A duração da segmentação praticamente não sofreu alterações. Seguidamente apresentam-se os resultados obtidos utilizando a amostra de 400x400x400.

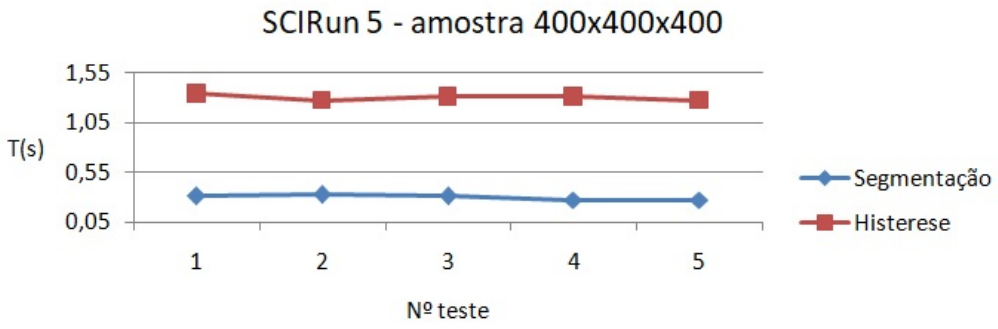

Figura 5.21: Duração dos módulos Tomo-GPU da amostra 400x400x400 no SCIRun5 em Windows.

	Segmentação	Histerese
Valor mínimo	0,281	1,281
Valor máximo	0,335	1,343
Média	0,3042	1,3058

Figura 5.22: Análise das durações da amostra 400x400x400 no SCIRun5 em Windows.

Comparativamente ao teste anterior, todos os módulos demoraram mais tempo na sua computação. A segmentação, em média demorou sensivelmente mais uma décima de segundo, enquanto que a histerese sofreu um aumento na sua duração de aproximadamente 77,2%. Finalmente, é apresentado o último teste, que diz respeito à amostra de maior dimensão, 1024x1024x1024.

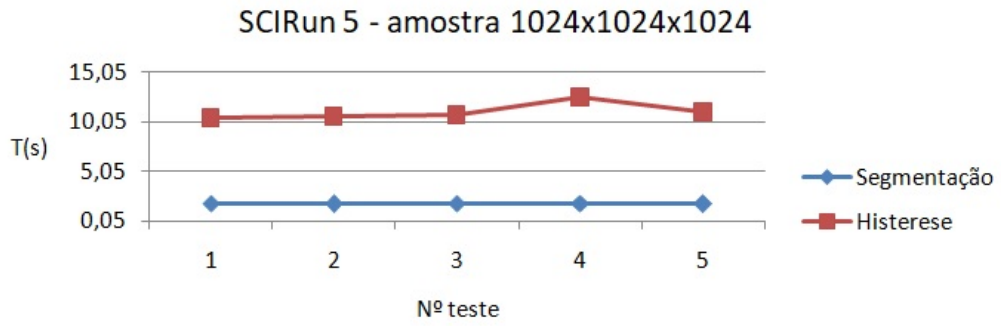

Figura 5.23: Duração dos módulos Tomo-GPU da amostra 1024x1024x1024 no SCI-Run5 em Windows.

	Segmentação	Histerese
Valor mínimo	1,843	10,474
Valor máximo	1,906	12,544
Média	1,8698	11,0788

Figura 5.24: Análise das durações da amostra 1024x1024x1024 no SCI-Run5 em Windows.

Relativamente à amostra de 400x400x400, este teste revelou novamente aumentos na duração da computação. Enquanto que a segmentação demorou 6 vezes mais que o teste anterior, por sua vez a histerese demorou pouco mais que 8 vezes. Na secção seguinte são realizadas várias comparações entre os diferentes SCIRun testados.

5.5 Comparações entre versões

Nesta presente secção, são expostas as conclusões obtidas, com base nos testes efetuados. Utilizando a primeira amostra, comum a todas as versões do SCIRun testadas, concluiu-se que o SCIRun 5 em Windows apresenta o pior desempenho. Já as versões em Linux, apresentam uma performance muito similar.

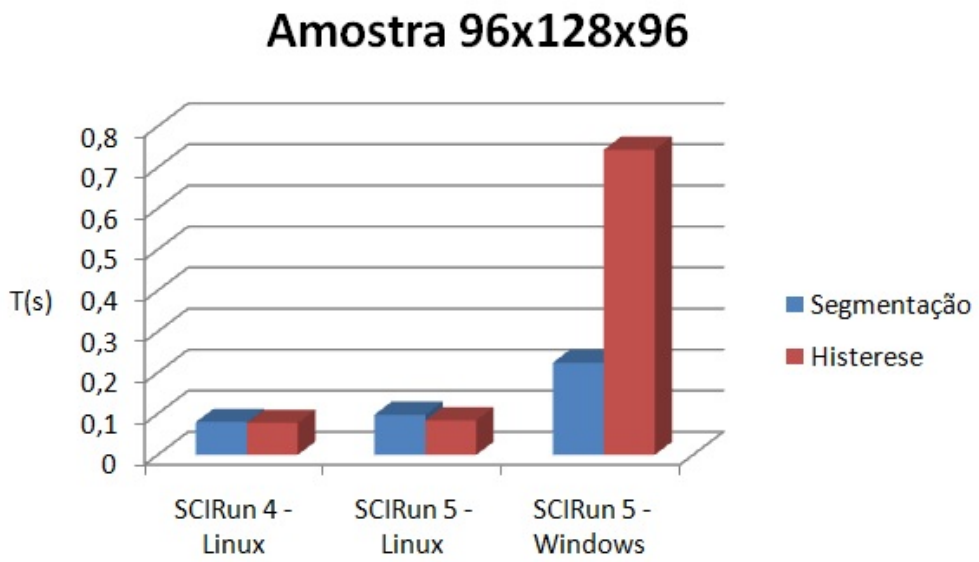

Figura 5.25: Duração dos módulos, utilizando a amostra 3x3x3 entre os vários SCIRun.

Seguidamente com base nas variadas amostras tomográficas testadas, conclui-se que na maior parte das vezes, o SCIRun 4 em Linux, apresenta uma perfomance superior, excepto na histerese da amostra de 1024x1024x1024 em que o SCIRun 5 em Windows é mais que 1 segundo mais rápido. Em contra partida, nessa mesma amostra, a segmentação dura em média 1,8698 segundos *versus* os 0,072 do SCIRun 4 em Linux, praticamente 2 segundos mais rápido. São apresentados de seguida um gráfico que apresenta uma comparação entre as várias medições observadas e uma tabela com a médias dos valores representados no gráfico.

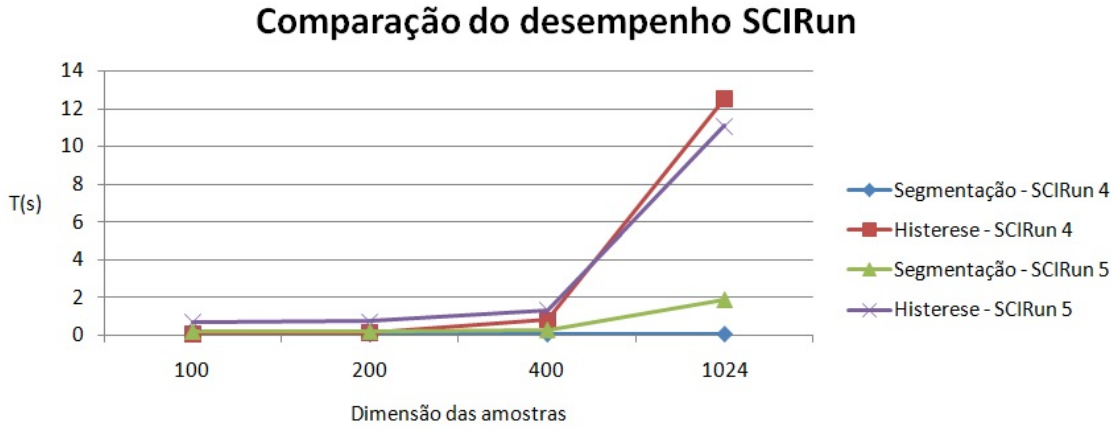

Figura 5.26: Duração dos módulos, de todas as amostras entre os vários SCIRun.

SCIRun 4 - Linux			SCIRun 5 - Windows	
Segmentação	Histerese	Dimensão da amostra	Segmentação	Histerese
0,074	0,08	100x100x100	0,2122	0,6838
0,066	0,158	200x200x200	0,215	0,7368
0,064	0,818	400x400x400	0,3042	1,3058
0,072	12,512	1024x1024x1024	1,8698	11,0788

Figura 5.27: Média de todos as durações da segmentação e da histerese, entre as versões SCIRun 4 em Linux e SCIRun 5 em Windows.

5.6 Conclusões

Os testes efetuados, embora com algumas limitações, permitiram, por um lado verificar a funcionalidade dos módulos transportados, e por outro efetuar comparações de desempenho entre as várias versões.

As conclusões a retirar é que o transporte dos módulos para o Windows 10, foi um sucesso quase total, tendo sido feito o transporte de todos os módulos com exceção de dois (*ViewDataSlices* e *VisAttributes*). Foi também demonstrada a viabilidade da alternativa módulo genérico, que foi demonstrada pela incorporação de um programa que utiliza CUDA.

Quanto à comparação dos desempenhos teria sido necessário mais tempo para fazer comparações mais sistemáticas e, sobretudo para procurar perceber as razões para as diferenças. Nesta área, a conclusão mais importante é que, no mesmo hardware, o desempenho da versão Windows 10 não é muito inferior à do Linux Ubuntu 16.04. Não houve disponibilidade de tempo para procurar uma justificação para as diferenças encontradas.

Capítulo 6

Conclusões

É feita uma comparação entre os objetivos iniciais da tese e aquilo que foi conseguido; o trabalho futuro tem sobretudo a ver com a parte do trabalho incialmente previsto e que não foi possível realizar.

6.1 Conclusões

Dentro do âmbito de esta dissertação conseguiu-se obter uma versão do tomo-GPU que pode ser usado numa máquina pessoal, executando um sistema operativo Windows 10; o transporte para este sistema operativo é importante para garantir uma maior disseminação destes ambientes dedidados à resolução de problemas.

As experiências efetuadas permitiram demonstrar que num computador portátil equipado com um processador com dois núcleos de processamento e um GPU os tempos de execução dos módulos mais exigentes em termos de processamento são compatíveis com uma utilização interativa. Está assim disponível um sistema de caracterização da população de reforços em materiais compósitos, que pode ser usado por um especialista da área de Materiais no seu computador portátil.

O trabalho permitiu transportar o Tomo-GPU para uma versão mais recente do SCIRun; esta conversão assegura uma maior longevidade do ambiente uma que a anterior versão do SCIRun é suportado em software (Tcl/Tk) que começa a mostrar a sua idade. A conversão para o SCIRun exigiu um esforço razoável uma vez que a interface gráfica teve alterações profundas devido à mudança para a biblioteca Qt.

Um desenvolvimento promissor foi a demonstração de incorporar no SCIRun programas autónomos que são lançados a partir de módulos genéricos. Esta abordagem permite uma maior rapidez no desenvolvimento de um módulo, uma vez que não obriga a uma (demorada) recompilação do sistema sempre que se fazem alterações no processamento.

Esta abordagem do módulo genérico também permite o transporte para o SCIRun 5 de módulos do SCIRun baseados em bibliotecas que deixaram de ser suportadas (por exemplo, o Tcl/Tk).

6.2 Trabalho futuro

O trabalho futuro pode incidir na conclusão do transporte do Tomo-GPU para o SCIRun 5 na continuação do desenvolvimento do Tomo-GPU acrescentando ao ambiente novas funcionalidades.

Em relação à primeira dimensão, o trabalho deverá sobretudo tirar partido do desenvolvimento efetuado no sentido de se ter um módulo genérico. Esse módulo genérico deverá ser melhorado na sua integração com o SCIRun, permitindo que além de ser feito o seu lançamento no SCIRun, também possa receber e enviar informação através das portas suportadas; neste momento, a forma de módulos SCIRun enviarem e receberem informação do módulo genérico é através de ficheiros. O módulo genérico também permitirá transportar para o SCIRun 5 o único módulo do Tomo-GPU que ainda não foi portado para este sistema. Este módulo usa o Tcl/Tk e tem uma interface gráfica tão complexa que não foi possível, em tempo útil, transformá-lo no sentido de passar a ser suportado pelo Qt.

Quanto à evolução do Tomo-GPU, podem ser adicionados ao sistema novas funcionalidades que sejam relacionado com o processamento de imagens tomográficas dos mesmos materiais, por exemplo procurando tratar compósitos em que as densidades do material base e dos reforços é próxima.

Bibliografia

[1] M. Abrams, D. Allison, D. Kafura, C. Ribbens, M. Rosson, C. Shaffer e L. Watson. *PSE Research at Virginia Tech: An Overview*. Department of Computer Science, Virginia Tech, 1998.

[2] A. Brodtkorb, C. Dyken, T. Hagen, J. Hjelmervik e O. Storaasli. "State-of-the-art in heterogeneous computing". Em: *Scientific Programming* 18.1 (2010), pp. 1–33. ISSN: 1058-9244.

[3] T. Cadavez, S. C. Ferreira, P. Medeiros, P. J. Quaresma, L. A. Rocha, A. Velhinho e G. Vignoles. "A Graphical Tool for the Tomographic Characterization of Microstructural Features on Metal Matrix Composites". Em: *International Journal of Tomography & Statistics* 14.S10 (2010), pp. 3–15.

[4] H. Delgado. "Characterization and Surface Reconstruction of Objects in Tomographic Images of Composite Materials". Tese de mestrado. Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2013.

[5] F. Fernando Birra, M. Encarnação, A. Lopes, P. Medeiros, N. Oliveira, B. Preto, P. Quaresma e A. Velhinho. "Tomographic image analysis of reinforcement distribution in composites using a flexible and material's specialist-friendly computational environment". Em: *Journal of Synchrotron Radiation* (2013).

[6] B Gerassimos. *Multicore and GPU Programming An Integrated Approach*. Morgan Kaufmann Publishers, 2015. ISBN: 9780124171374.

[7] S. Kawata, H. Usami, Y. Hayase, Y. Miyahara, M. Yamada, M. Fujisaki, Y. Numata, S. Nakamura, N. Ohi, M. Matsumoto, T. Teramoto, M. Inaba, R. Kitamuki, H. Fuju, Y. Senda e Y. Tag. "A Problem Solving Environment: PSE for Distributed Computing". Em: *Int. J. High Perform. Comput. Netw.* 1.4 (dez. de 2004), pp. 223–230. ISSN: 1740-0562. DOI: 10.1504/IJHPCN.2004.008351. URL: http://dx.doi.org/10.1504/IJHPCN.2004.008351.

[8] G. Laszewski, J. Insley, I. Foster, J. Bresnahan, C. Kesselman, M. Su, M. Thiebaux, M. Rivers, S. Wang, B. Tieman e I. McNulty. "Real-Time Analysis, Visualization, and Steering of Microtomography Experiments at Photon Sources". Em: *Actas da 9th SIAM Conference on Parallel Processing for Scientific Computing* (2000).

[9] S. G. Parker e C. R. Johnson. "SCIRun: a scientific programming environment for computational steering". Em: *Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*. Supercomputing '95. San Diego, California, United States: ACM, 1995. ISBN: 0-89791-816-9. DOI: http://doi.acm.org/10.1145/224170.224354. URL: http://doi.acm.org/10.1145/224170.224354.

[10] B. Preto, F. Birra, A. Lopes e P. Medeiros. "Object Identification in Binary Tomographic Images Using GPGPUs". Em: *International Journal of Creative Interfaces and Computer Graphics* 2 (2013), pp. 40–56.

[11] J Sanders e E Kandrot. *CUDA By Example An Introduction to General-Purpose GPU Programming*. Addison-Wesley, 2010. ISBN: 9780131387683.

[12] SCIInstitute. *The NIH/NIGMS Center for Integrative Biomedical Computing*. URL: http://www.sci.utah,edu/cibc-software/scirun.html.

[13] J. Tate. *SCIRun5ModuleGeneration*. Center for Integrative Biomedical Computing Scientific Computing Imaging Institute, University of Utah, 2018.