Hugo Delgado

Licenciatura Engenharia Informática

Characterization and Surface Reconstruction of Objects in Tomographic Images of Composite Materials

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador : Pedro Medeiros, Professor Associado, Faculdade de Ciências e Tecnologia da UNL - Dep. de Informática

Júri:

Presidente: Ana Maria Dinis Moreira, Prof. Associada, Faculdade de Ciências e Tecnologia da UNL - Dep. de Informática

Arguentes:

Vogais: José Manuel Fonseca, Professor Auxiliar, Faculdade de Ciências e Tecnologia da UNL - Dep. de Eng. Electrótecnica
Pedro Medeiros, Professor Associado, Faculdade de Ciências e Tecnologia da UNL - Dep. de Informática

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Julho, 2013

Characterization and Surface Reconstruction of Objects in Tomographic Images of Composite Materials

A todos os que de uma forma ou de outra contribuíram para que tenha que este trabalho tenha chegado ao fim.

Abstract

In the scope of the project Tomo-GPU supported by FCT / MCTES the aim is to build an interactive graphical environment that allows a Materials specialist to define their own programs for analysis of 3D tomographic images. This project aims to build a tool to characterize and investigate the identified objects, where the user can define search criteria such as size, orientation, bounding boxes, among others. All this processing will be done on a desktop computer equipped with a graphics card with some processing power.

On the proposed solution the modules for characterizing objects, received from the identification phase, will be implemented using some existing software libraries, most notably the CGAL library. The characterization modules with bigger execution times will be implemented using OpenCL and GPUs. With this work the characterization and reconstruction of objects and their research can now be done on conventional machines, using GPUs to accelerate the most time-consuming computations. After the conclusion of this thesis, new tools that will help to improve the current development cycle of new materials will be available for Materials Science specialists.

Keywords: TOMO-GPU, SCIRun, MMDBS, MonetDB, geometric processing, CGAL, GPGPU, openCL

Resumo

No âmbito do projecto Tomo-GPU financiado pela FCT/MCTES o tema central é a construção de um ambiente gráfico interactivo que permita a um especialista de Materiais definir os seus próprios programas para análise de imagens tomográficas 3D. Com este projecto pretende-se construir uma ferramenta que permita caracterizar e pesquisar os objectos identificados, podendo o utilizador definir critérios de pesquisa tais como dimensões, orientação, factores de forma, entre outros. Todo este processamento será feito num computador desktop equipado com uma placa gráfica com algum poder de processamento.

O processo de caracterização e reconstrução dos objectos é um processo computacionalmente dispendioso e que actualmente o seu tempo de processamento deixa os cientistas mais limitados na analise destes objectos. Ter a possibilidade de poder executar toda a computação num computador economicamente acessível, e de uma forma mais rápida do que actualmente se faz, traz bastantes vantagens. Alguns dos temas a abordar têm a ver com a utilização de bases de dados em memória, outro aspecto que poderá vir a ser explorado é a utilização de GPGPUs, para o processo de caracterização dos objectos.

A solução proposta passa por implementar módulos de caracterização de objectos provenientes da fase de identificação com recurso a algumas bibliotecas aplicacionais existentes, mais nomeadamente a biblioteca CGAL. Serão também implementados, com recurso a GPUs e OpenCL, os processos de caracterização e reconstrução que tenham tempos de execução mais longos. Com este trabalho a caracterização de objectos e a sua pesquisa pode passar a ser feita em máquinas convencionais, com recurso a GPUs para acelerar as computações mais morosas. Desta forma é fornecida aos especialistas de materiais mais uma nova ferramenta que vem ajudar a melhorar o actual ciclo de desenvolvimento de novos materiais.

Palavras-chave: TOMO-GPU, SCIRun, MMDBS, MonetDB, processamento geométrico, CGAL, GPGPU, openCL

Contents

1 Background **1**

1.1 Context . . . 1

1.1.1 Tomo-GPU Project . . . 2

1.1.2 SCIRun . . . 4

1.2 Problem . . . 5

1.2.1 Characterizations Data Persistency . . . 6

1.2.2 Object Reconstruction Filter . . . 6

1.3 Approach . . . 6

1.3.1 Characterizations Data Persistency . . . 7

1.3.2 Object Reconstruction Filter . . . 7

1.3.3 Parallelization . . . 8

1.3.4 SCIRun Integration . . . 8

1.4 Thesis Contributions . . . 8

1.5 Thesis Organization . . . 9

2 Object Characterization **11**

2.1 Problem . . . 11

2.2 Relevant Work . . . 12

2.2.1 Computacional Geometry . . . 12

2.2.2 Data Persistency . . . 20

2.2.3 Reducing Execution Time using Available Cores . . . 22

2.3 Solution Organization . . . 24

2.3.1 Organization . . . 24

2.3.2 Storage . . . 26

2.4 Implementation . . . 27

2.4.1 PCA . . . 28

2.4.2 Bounding Boxes . . . 28

2.4.3 Surface Area . . . 29

2.4.4 Tests . . . 30
2.5 SCIRun Integration . . . 30
2.5.1 How to turn the standalone code in a SCIRun module . . . 31
2.5.2 Module position in the TomoGPU software . . . 31
2.6 Parallelization . . . 31
2.6.1 Approach . . . 31
2.6.2 Conclusion . . . 34
2.7 Conclusion . . . 34

3 Object Reconstruction **35**
3.1 Problem . . . 35
3.1.1 Problem Definition . . . 35
3.2 Relavant Work . . . 36
3.2.1 Computation Geometry . . . 36
3.2.2 Space Partitioning . . . 37
3.2.3 Implicit Surface Reconstruction Techniques . . . 38
3.2.4 Image Cleaning . . . 41
3.2.5 Software Libraries . . . 41
3.2.6 GPGPU Architectures . . . 42
3.2.7 Linear Algebra Libraries . . . 48
3.3 Proposed Solution . . . 48
3.3.1 Organization . . . 49
3.4 Implementation . . . 50
3.4.1 Remove Interior Voxels from Object . . . 51
3.4.2 Create Initial Triangulation . . . 51
3.4.3 Compute Surface Normals . . . 51
3.4.4 Poisson Reconstruction . . . 51
3.4.5 Extract Surface Mesh . . . 52
3.4.6 Tests . . . 52
3.5 SCIRun Integration . . . 52
3.6 Optimizing Solution . . . 53
3.6.1 A - Multi-Core Approach . . . 53
3.6.2 B - CPU-GPU Approach . . . 53
3.6.3 C - Meshing Algorithm Replacement . . . 54
3.7 Conclusion . . . 54

4 Conclusions **57**
4.1 Work evaluation . . . 57
4.2 Future work . . . 57

A Mathematical Foundations **65**
A.1 Linear Algebra and Matrices 65
A.1.1 System of Linear Equations and Matrices 65
A.1.2 Eigenvalues and Eigenvectors 66
A.1.3 Solving Systems of Linear Equations 66
A.2 Statistics 68
A.3 Geometric and Analytical Measures 68
A.3.1 Mathematical Foundations 68
A.3.2 Distance Metrics 68

B SciRun Integration **71**

C Reconstruction Examples **73**

1

Background

In this chapter, we start by describing the context of this work, followed by a description of the problem, approach and expected contributions of this dissertation.

1.1 Context

This work has two main focus, the first is to provide an extensible framework for the integration of geometric algorithms on some previously identified particles on datasets, providing data persistency of the computed geometrical measures. The second focus of this work relies with the reconstruction of the surface from incorrectly sampled particles on the *Tomo-GPU* project, offering an alternative solution to the currently available techniques used on the project. The techniques that are currently employed cannot correctly reconstruct the shape on those bad particles and a different approach to the reconstruction should be devised. This tools will be integrated on other project called *Tomo-GPU* project in order to provide the environment where one could better study those particles. The Tomo-GPU Project makes use of the SCIRun software, that is a problem solving environment framework where material specialists have at their disposition a set of modules to choose, connecting them in order to create a workflow for data analysis composed of several independent analysis that are applied to the received dataset. The Tomo-GPU Project is developed as a package of several modules that can be added to a common SCIRun workbench, this work will introduce two additional modules to that project.

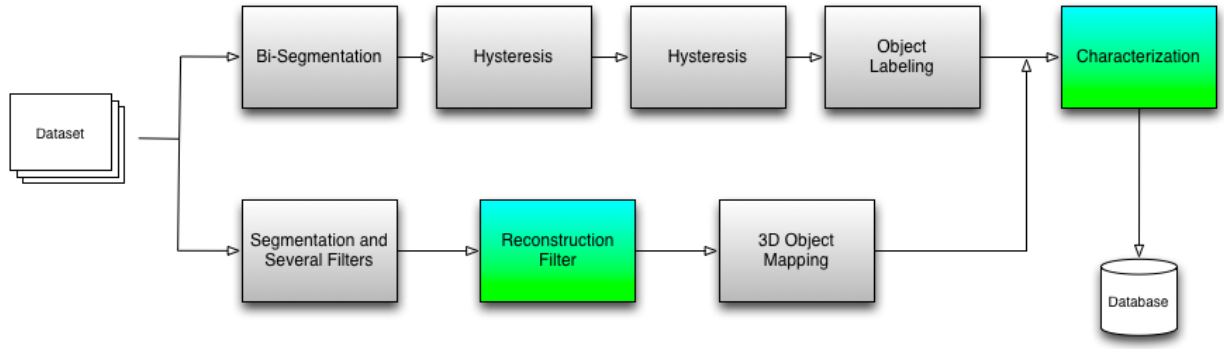


Figure 1.1: Modules Position in *Tomo-GPU* project workflow.

1.1.1 Tomo-GPU Project

This project was founded by FCT/MCTES (PTDC/EIA-EIA/102579/2008 - Ambiente de Resolução de Problemas para Caracterização Estrutural de Materiais por Tomografia) and its main objective is the building of an environment that helps a Materials Science specialist to divide new composite materials, commonly know by composites, that are formed by combining materials together to form an overall structure that is better than the individual components. The idea is to obtain new materials for use in cars, planes, rockets, etc.. For testing new processes of building composite materials, tools for the characterization of the reinforcement population are needed; in the Tomo-GPU project the focus is on the analysis of tomographic images of composite materials.

The *Tomo-GPU* project have some important characteristics that drove the development of this thesis dissertation, and the most important characteristics of the Tomo-GPU environment are presented next:

Ease of use The environment allows the easy specification of a sequence of processing steps of the data.

Flexibility The integration of new capabilities in the system is easy.

Interactivity The processing steps have short execution times that promote an interactive use of the system, where users can change parameters of the processing and visualize.

Affordability The hardware and the software needed have prices that allow many research groups to use it.

Lets now see how this characteristics are obtained under the Tomo-GPU Project:

Ease of use and flexibility are obtained by using a graphical tool to do the data analysis, where one could easily develop different workflows to analyze some input data and better understand the complex data relationships, as so ease of use is achieved in SCIRun, and flexibility cames with the modules framework under the

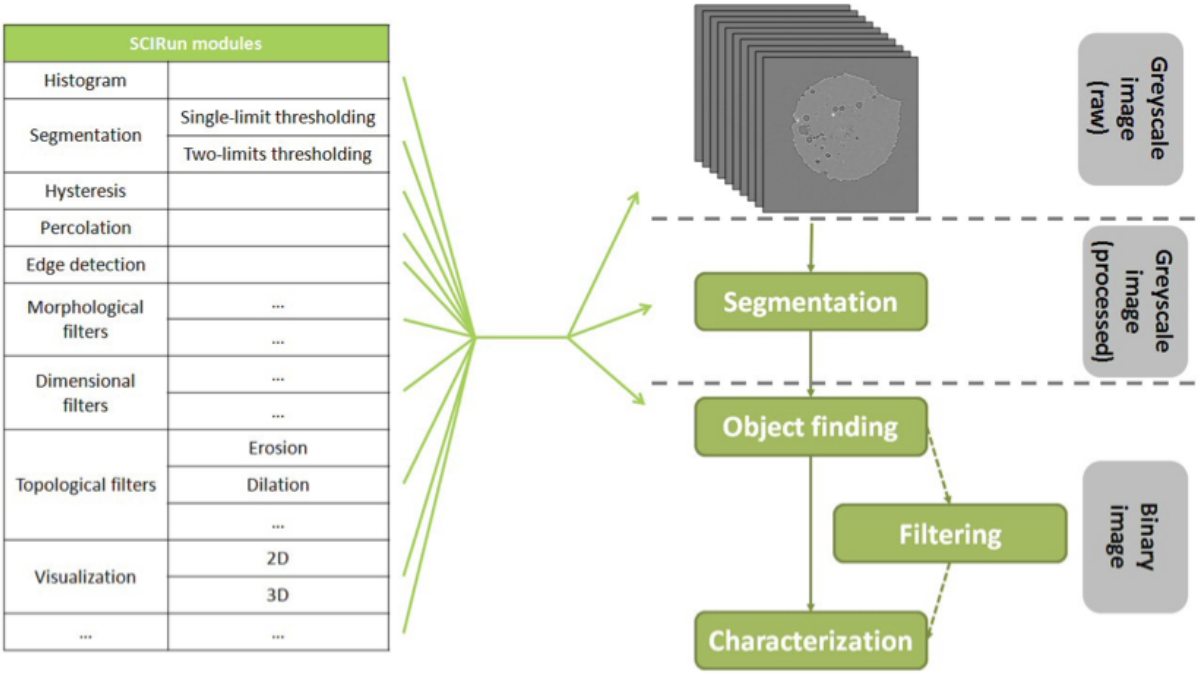


Figure 1.2: Overall view of Tomo-GPU modules
[Ea12]

SCIRun[PJ95], as previously seen this modules could be added or removed from one workflow as needed.

Interactivity and affordability are achieved by targeting an hardware platform based on a desktop personal computer equipped with GPGPUs (General Purpose Graphic Processing Unit) [OLGHKLP07]. The use of the great computing power of this hardware platform allows fast execution times of highly-demanding processing algorithms - for example, 3D image processing - without expensive investments on hardware or access to remote clusters.

Figure 1.2 shows some of the modules that had been developed in the project and how they could be connected to produce an 3D tomographic image analysis pipeline, those modules can be divided in the following categories:

Defect removal 3D Image processing techniques are used to clean the image of artifacts that are originated by the method of image acquisition. The object is to have a monochromatic image where the matrix corresponds to white and objects are black.

Image labeling In the monochromatic image, sets of connected black voxels are identified; each set receives a unique label.

Image filtering operations are applied to individual objects. One example is the removal of objects smaller than a threshold.

Object characterization From each individual object, some characteristics are extracted. In this phase, most of the characteristics are related with geometry - dimensions, bounding box, volume, area, etc..

Object characteristics mining A repository containing object characteristics of different samples is built. Tools for querying that database allow the Materials specialist to evaluate the process used in the creation of the sample.

In this dissertation the focus is at the object characterization and the object filtering categories, but we will also address the last category, object characteristics mining, although not as the primary goal but providing the tools so that functionality could be added when needed.

1.1.2 SCIRun

SCIRun is an open source project that was supported by grants from the National Center for Research Resources (5P41RR012553-14) and the National Institute of General Medical Sciences (8 P41 GM103545-14) from the National Institutes of Health[Ins13]. It is a computational workbench developed by the SCI group from from the University of Utah, its an open source licensing software and used worldwide in many universities and research

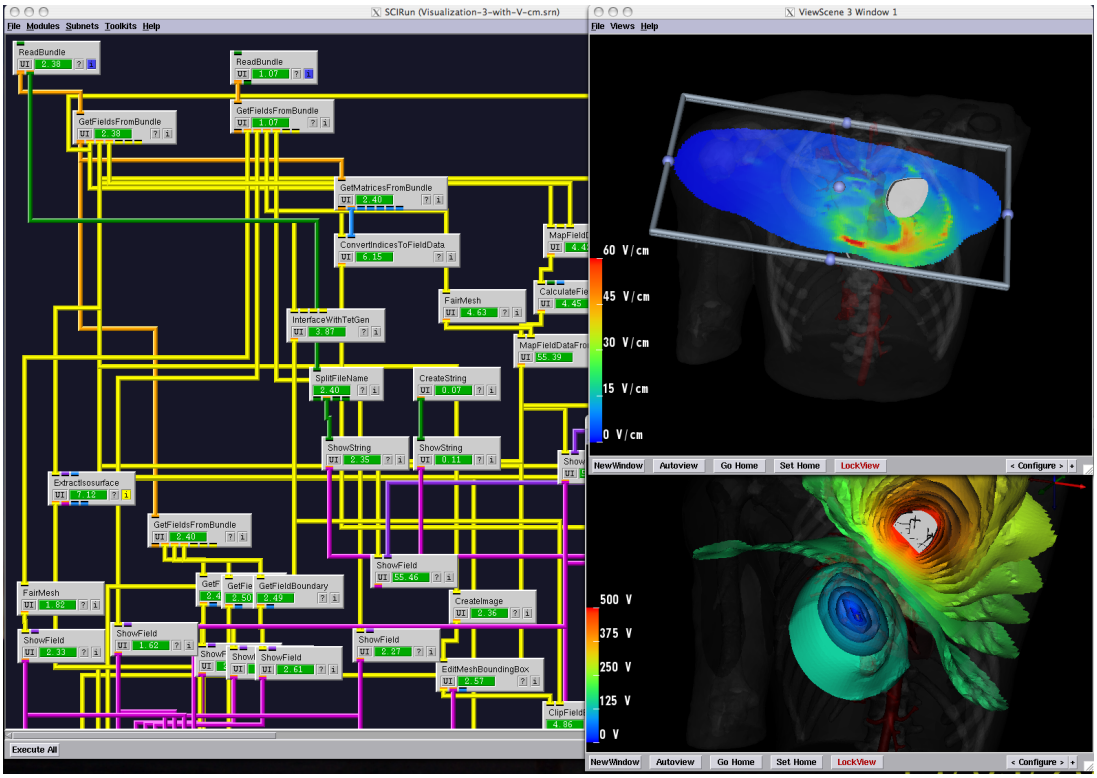


Figure 1.3: Overall view of SCIRun computational workbench [Ins13]

groups, mainly on the field of the biomedical investigation. As one can see on Figure1.3 this software is a tool for building the so-called problem solving environments (PSEs) that allows the construction of programs through a visual programming approach. In the *SCIrun* environment one has a menu with of several modules; each one of the modules has *input ports* and *output ports*. The general idea is that each module produces, in one of its output ports, a modified version of the data that is received through one or more of its input ports.

For the graphical front end *SCIRun* uses the *Tcl/TK*, and have an *TCLInterface* to provide an abstraction layer that make the task of moving data between the *Tcl* and the C++ portions of *SCIRun* transparent to the user. On the *Tcl* side, the code access variables such as regular *Tcl* variables and on the C++ side the code needs to declare those variables under a module class and access them.

Being an open source software and its used worldwide brings some advantages such as having an active community where problems could be exposed. The available developer manual isn't very well documented and that introduce a learning curve before a developer could fully use the available features on the framework.

1.2 Problem

There are several problems addressed on our work, but mainly we will deal with the geometric characterization and reconstruction of previously identified particles, that from now on we denote by objects. For now there is the need to have a tool for characterizing geometrically, with some defined characterizations, the objects and store the resulting values for latter access. But the number of characterizations that need to be stored could increase according to the needs of the *Tomo-GPU* project.

Under a different scope, in some specific datasets the resulting output from previous processing stages on Tomo-GPU Project presented to be processed contains errors and the objects received may not be complete, so this tool should be able to deal with noise introduced either by the nature of the composite materials or accumulated over the previous processing stages.

The need to have an interactive system for the characterization and reconstruction of the objects, the size of the data and the available hardware for the scientists suggests the use of parallelization techniques, either by using the *CPU* or *GPU*. It will be accessed, preferably on all stages of the processing, how these improvements could be used and what are the benefits or not of its use. The reconstruction phase is the major focus for such access because of the processing times of the necessary operations to perform a successful reconstruction.

1.2.1 Characterizations Data Persistency

There is the need to store the data from the characterizations for latter access, to fully understand the objects a sample should be processed several times producing several datasets to be analyzed, although they are analyzed on different workflows on the *SCIRun*. If the data between different ct-scans remains available, other tools may be built to query that data. With that capability one could easily search for specific features on several objects among different samples. In order to do that there should be added support to store the results for later searching and should be easy to extend this data repository, in the case of different extraction methods are created and integrated on the SCIRun. Although not being the major goal from this work, this data should be structured in some way that could easily allow any scientist to latter access it, and fully infer about it.

1.2.2 Object Reconstruction Filter

The *Tomo-GPU* project already contain some modules to extract the shape from an object. On some of the sampled materials the currently employed techniques didn't behave correctly and the extracted shape from the identified objects could contain holes or present some outliers, difficulting their accurate representation. None of the previous filter modules on the *Tomo-GPU* project could perform a good reconstruction mainly because the sparsity of the sampled points on the object surface. Under this scenario a different approach to the reconstruction should be investigated so that the real shape from that object could be fully recovered, covering possible holes and removing the outliers. This implementation should be integrated onto the *Tomo-GPU* project, to latter be used on real samples.

1.3 Approach

In order to comply with the requisites for the project this work have been divided in two modules, one for the object characterization and other for the object reconstruction.

Although the main focus of this work is under the image analysis, this work started to study the availability to use in-memory databases and how they could be used in a *GPGPU* environment, for storage and fast query of the characterization data returned from the characterization module, soon we saw that the amount of data and the required features for this project at this stage doesn't require such techniques, and we choosed to introduce this data abstraction so latter if is found that such techniques could be used this modules could still be used for the object characterization.

Next we go into further detail about how this work is organized, starting by describing the object characterization module and how the data persistency and the integration with the *Tomo-GPU* project has been achieved under this module. Lastly we present the object reconstruction module.

1.3.1 Characterizations Data Persistency

The geometrical characterizations and the data persistency are ensured by a framework that can perform the desired goals. It is extensible in the sense that abstract the used database and easing the implementation of future characterizations by allowing to extend the current set of characterizations with new characterizations.

1.3.1.1 Characterizations Framework

Beside the capability of this module to perform the required computations it has been developed as an module where some components may be plugged providing a framework for the characterization of the objects. These components are abstracted by a characterization class and are made available to the user by a *GUI* component where one could choose which of the characterizations should be performed, this way it is easy to extend the system with new characterizations as they are needed.

This module have an graphical interface, allowing one to choose which of the characterizations should be computed, and at launch it will receive the objects and for each received object will compute the choosed geometric characterizations.

1.3.1.2 Data Layer Abstraction

In order to allow the characterized objects to be latter queried, these results will be stored in a repository, this repository should contain the different values from each characterization from all objects, and although the small expected size for the characteristics repository in terms of expected columns, this project as a part of a bigger project may grew in the number of characteristics to process, so it is to expect that the system should easily accommodate such changes. This is accomplished using an abstraction layer that will serve to interface different dbms systems through *SQL*, allowing for further changes on the data layout easily. By storing the data using a *dbms* we gain expressiveness given by *SQL* allowing a rich set of language to query the data, portability and persistency on the data are accomplished by the choosed *dbms*.

To provide the data persistency under the characterization module, we have introduced a data abstraction layer to abstract various *dbms's* using *odbc*, that way any *dbms* that have a connectivity driver to *odbc* could be used to store the data for latter access, and that way portability is introduced to the data framework.

1.3.2 Object Reconstruction Filter

Here the problem focus is to fully reconstruct the surface from an object in order to latter be processed by other modules of *Tomo-GPU* project. It will be developed an module for the *SCIRun* that is capable of perform the reconstruction of objects from samples containing noise. There are several algorithms capable of doing that and mainly what they do

is to reconstruct the surface based on inferring some properties at each point of the surface based on some local or global measures. We have chosen to use the poisson surface reconstruction algorithm, that is an algorithm that tries to reconstruct the object surface according to global measures of the object voxels, it is highly resilient to data noise and is capable of perform an surface reconstruction with good detail. This process is suitable to errors since the original data also have errors and it isn't complete, but it can infer the surface from those objects in a very accurate way, eliminating ghost voxels, and filling the holes of the objects on received input and with that approximate the real surface from that object.

This reconstruction phase won't be used on all datasets, but only on those that one had previously identified as datasets containing noise. The data may come on in two different ways depending in the previous used filters on other stages of the *Tomo-GPU* project, but mainly the data is composed of all the voxels from the object or only voxels that the previous algorithms have identified as surface voxels. To provide an easy to perform the reconstruction the *GUI* of the module allow to select if the interior voxels should be removed from the sampled object or not.

1.3.3 Parallelization

As interactivity is one of the keywords from this project, all the computations should be fast, and to accomplish this an assessment of the use of parallelization on the available hardware has been employed. On the characterization module, we have chosen to use multicore-cpu shared memory parallelization techniques such as *OpenMP* and *PThreads*, on the reconstruction module we have introduced parallelization using the gpu using *OpenCL*.

1.3.4 SCIRun Integration

1.4 Thesis Contributions

The expected contributions of the thesis are:

A tool for object characterization A set of modules for extracting geometric characteristics of objects identified in tomographic images will be designed, implemented, assessed and integrated in the SCIRun environment. The execution times will be evaluated and versions for execution in GPGPUs will be developed. These versions will be compared with former ones regarding efficiency.

A tool for object reconstruction A set of modules that will allow to fully reconstruct one object even on the presence of holes or gaps of the original object. Such tool will be used on bad datasets that are exposed noise introduced by the tomograph and the earlier processing by the image cleaning algorithms.

A repository of object characteristics Introducing a dbms as a repository system, and abstracting it from the development, will be the basis of what a consultant of the project called a "google of the particles".

The components above will enhance the functionality of the problem solving environment, making it more attractive to Materials Science specialists. Besides this contributions to the Tomo-GPU project, know-how about tools for geometric computing, in-memory databases and application parallelization will be obtained.

1.5 Thesis Organization

On the second chapter we will show all the work behind the geometric characterization from a point set. On the third chapter we will show how can surface reconstruction from an noise and unoriented point set may be achieved. Lastly on fourth chapter we will state our conclusions.

2

Object Characterization

In this chapter we start by introducing the problem, all the know-how and tools to perform the required characterizations. After that is presented the proposed solution and implementation details, showing how the implemented framework work with the underlying databases and how it could be used on a multi-core shared memory architecture leveraging the previous implementation. We finish by introducing the integration onto the *SCI-Run* framework and showing the results from the implemented functionality.

2.1 Problem

Besides each received sample contains several objects, we start to look at the problem as the geometric analysis of one single object. Under this scope we have one object represented as a set $\mathcal{X}$ of n three-dimensional unoriented points $\mathcal{X} = \{x_1, x_2, ..., x_n\}$, from all

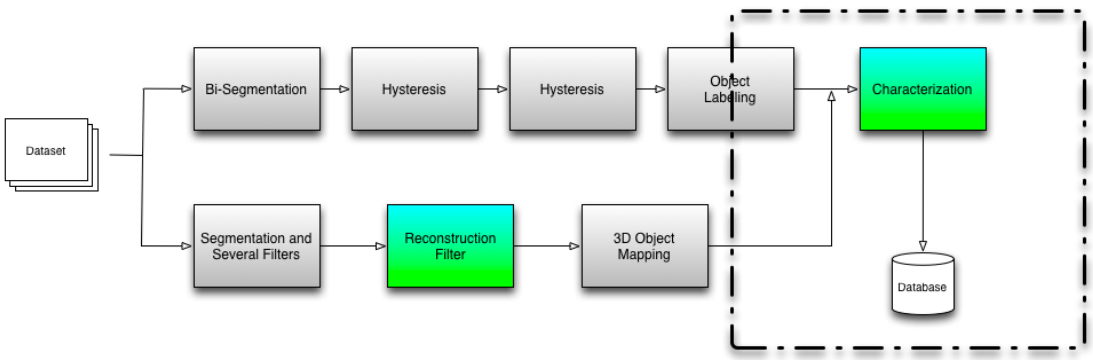


Figure 2.1: Modules Position in *Tomo-GPU* project workflow.

the voxels of the object including its interior and boundary. As seen, for this point set it should be computed its centroid, principal directions, object volume, object surface, axis aligned bounding box and the object oriented bounding box.

Each received object represent a previously identified particle on the ct-scan composed of several voxels. The ct-scan has been decomposed on a regular grid in a three-dimensional space, and we receive all the voxels that belong to each identified object. We assume that the objects are compact, don't have holes inside, all the voxels from the object are well sampled and that each voxel has another voxel connected to it. By connected voxel we have defined that each voxel has at least another voxel on one of its twenty six neighbor voxels.

The implemented framework besides computing the respective values for each characterization it should also store the resulting values on an underlying database for latter access. This framework ideally should be extensible by allowing an easy integration of other possible characterizations.

2.2 Relevant Work

Most of the presented algorithms rely upon mathematical or statistical models, where the analysis over the data is computed. We have compiled some mathematical concepts that are required to perform the characterizations and it can be found on Appendix A.

2.2.1 Computacional Geometry

For the given point set P, representing all points from an object in 3d-space where each point has the same density value, we start by introducing some mathematical concepts required to perform the required characterizations.

2.2.1.1 Geometric Characterization

Centroid One could think in the centroid in terms of physics, as the center of mass or barycenter, from an object. In geometry it is computed by the average of its geometric positions, and in some cases the centroid may not belong to the object.

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n}$$

Bounding Boxes There are mainly two kinds of bounding boxes that should be computed for an object, a axis-aligned and a minimal-area bounding box, or object bounding box. The axis-aligned bounding box is commonly used for computing intersections on objects in complex scenes, since it is easy perform analytical tests for intersection, the object oriented box is the smallest box containing the object, this box give a better approximation of the object shape but that may have impact on performance, for example on a intersection test additional computations must be performed.

Axis-Aligned Boundig-Box An axis-aligned bounding box or *AABB* is simply a rectangular parallelepiped whose faces are each perpendicular to the origin, here we want to figure the minimum axis-aligned parallelepiped that fully encloses all of the objet voxels, this is done by computing the minimum and maximum values on all three axis, x, y and z, that are given by two points $P_{min} = (X_{min}, Y_{min}, Z_{min})$ and $P_{max} = (X_{max}, Y_{max}, Z_{min})$.

Oriented Bounding Box This bounding box of the object can also be seen as minimum bounding rectangle that fully encloses an object. On this work we are only interested on the object-oriented bounding box as a simple bounding parallelepiped whose faces are parallel to the basis vectors representing the principal components obtained from PCA, as explained on 2.2.1.1.

Volume There are analytical procedures to compute the volume from a solid, although in our case we are only interested in computing an approximate solution for the volume, that is simply given by the sum of the volume of each voxel from the object. Since our object is sampled on a regular spaced grid and each voxel have the same volume, the total volume is implicit defined by the number of voxels from the object. This is possible because each sample contains also some meta-information about each voxel spacing in regard to the real size of the sample.

Surface Area Since the sampled object represent a discretization of the real object under a regular grid there are different techniques that could be used to compute an approximate value for the object surface area. One way is to reconstruct the surface from the object by a triangular mesh and compute the total area from the surface as the sum of all the areas of the triangles that compose the surface. Other simple process that may be applied on this specific case is to compute all the surface voxels from the input object and for each one of them compute the total area as the sum of the areas from all of the exposed faces from each voxel the surface, for that we define $\Omega = \{\Omega_1, \Omega_2, ..., \Omega_{n'}\}$ as a subset of $\mathcal{X}$, representing all the voxels from the object that belongs to its surface. This definition is possible since the object is represented on a regular grid in a *three-dimensional* space and all the voxels have the same size and are equally spaced.

Principal Component Analysis Principal component analysis is a basic component of many geometric computing and processing algorithms.[GAP08] Mainly it helps to identify patterns in data, it is a powerful statistical tool to analyze data and is mainly used on point sets, having many practical uses such as pattern finding or image compression, where PCA its used to express the original data in terms of the eigenvectors and eigenvalues. In theory it allows to transform an number of variables correlated or not in a slower number of variables called principal components.With that the dimension of the data is reduced and a new meaning for the data may be found.

To better understand this analysis one could look at it like a transformation on the objects to a new coordinate system such that the greatest variance comes to lie on its first principal component, the second greatest variance on the second principal component and so on.[APG12] So by this kind of analysis one will constrain each one of the variable in terms of the other two, and latter obtaining the values that minimize such new representation.

Principal Component Metodology In order to extract the principal component the next steps should be done:

1. Determine the centroid for X
2. Subtract the centroid to X, in order to center the data around its mean value
3. Compute the 3×3 covariance matrix M from X
4. Compute the eigenvalues and the eigenvectors from M
5. Order the eigenvalues and eigenvectors starting with the one with maximal eigenvalue

2.2.1.2 Polygonization

Delaunay based Here at first it is constructed the Delaunay triangulation, or its dual[1] voronoi diagram, partitioning the sample points into a finite set of tetrahedra, as presented on **??** its main advantages relies with its uniqueness. After the construction of the triangulation it is needed to figure out wich of the simplices belongs to the surface. There are a variety of proposed algorithms for constructing a Delaunay triangulation.

Region Growing It is a technique for solving geometric problems where an algorithm starts with a initial seed or complex of the final mesh and the solution is incremented by glueing other pieces to the initial seed and so on until the final mesh is constructed. Typically this kind of algorithms have complex strategies to deal with the intersections where the pieces of the mesh being added connect to the already constructed mesh to avoid the duplication of the final mesh.

Surface Splatting As presented on [ZPBG01], is a reconstruction technique that aims the direct rendering from point-based objects. They use a point-based algorithm, and provides a splat primitive that could be applied for large resolution laser scan ranges. Using a weighted sum of radially symmetric basis functions. It is also introduced an extension to Heckbert's resampling theory to process point-based objects.

[1] the duality for any finite set S of points in the plane between the Delaunay triangulation of S and the Voronoi diagram of S

Optimized Sub-Sampling Using an surface splatting technique the processing costs are still proportional to the number of primitives[ZPBG01] used to represent an object. This technique addresses specifically this problem, presenting a sub-sampling technique for dense point clouds that are adjusted to the particular geometric properties of circular or elliptical surface splats.

Progressive Splat Generation This technique has been proposed for interactive ray-tracing of point-based models, that uses an full splat geometry estimating the error. They compute all the splats and after that they are ordered and an iterative algorithm that will progressively reach the number of desired splats and minimizing the global error of the global reconstruction.

2.2.1.3 Convex Hull

It is by definition the smallest convex set that contains a finite point set $\mathcal{P}$, it is also known by polytope. There is also a geometrical notation from the convex hull of $k+1$ points that are affinely independents called *k-simplex*, a line segment for example is a 1-simplex, a triangle a 2-simplex and a tetrahedron a 3-simplex. In a d-dimensional space a facet from the convex hull are a (d-1)-simplice. There are known algorithms for incrementally compute the convex hull and their complexity in a 3-dimensional space is O(n log n), but can be improved by inserting the points in random order. [Cha93]

2.2.1.4 Voronoi Diagram

Their usage can be tracked back to Descartes in 1664 and its formal study and definition on a 2- and 3-dimensional space at 1850 by Dirichlet, latter the n-dimensional general space appeared at 1908 by a Ukrainian mathematician Georgy Fedosievych Voronyi. A Voronoi diagram or Dirichet tessellation is the division of a space M in a set S of seeds or sites s in M, in which exists a concept of influence that the region of s exerts on a point x of M, where the region of s consists of all points x for which the influence of s is the strongest, over all $s \in S$. There are several variants of this diagram depending upon different objects classes, distance functions and embedding space.

2.2.1.5 3D Triangulation

Usually on the literature when one talk about triangulations a notation appears with the name of simplicial complex, that is the formal definition for a topological space that is constructed by gluing together several geometrical simplices. On this work we denote a triangulation $T(\mathcal{P})$ of a finite set of points $\mathcal{P} \in \mathbb{R}^3$ a decomposition of the domain of the convex hull $Conv(\mathcal{P})$ of the point set composed by tetrahedrons whose vertices are the points of $\mathcal{P}$. Since no more restrictions apply to this formulation there exist many triangulations for a given set of points $\mathcal{P}$, but the shape from the triangles are important and

a common measure the ensure the quality of the triangulation is the size of the internal angles from each tetrahedron that compose the $T(\mathcal{P})$.

Delaunay Triangulation The Delaunay triangulation $\mathcal{DT}$ from a set of points $\mathcal{S}$, from now on denoted by $\mathcal{DT}(\mathcal{S})$, is a $\mathcal{T}(\mathcal{S})$ that guarantees that no point, $\mathcal{P} \in \mathcal{S}$, is inside the circumscribed sphere of any tetrahedra $\mathcal{T} \in \mathcal{S}$. There are several algorithms proposed in the literature to compute the Delaunay triangulation such as incremental construction, divide and conquer and sweeping.

Other Triangulations Besides the previously presented triangulations others exist such as a regular triangulation, a constrained triangulation and their Delaunay counterparts. A regular triangulation is a subset of a Delaunay triangulation where each point in the triangulation is associated with a weight factor. If all points in the triangulation have the same weight the regular triangulation is equal to a Delaunay triangulation but in most cases this don't happened. A constrained triangulation is a triangulation that has some enforced segments in the triangulation. Usually a regular triangulation can also be a Delaunay triangulation but the same doesn't happened with the constrained triangulation since the requirements from the Delaunay triangulation cannot be imposed. Some papers state a solution for that problem adding specific vertices to the triangulation in order to maintain the Delaunay requirements but this come with several complexity.[Pau]

2.2.1.6 Alpha-Shapes

An *alpha-shape*, or α*-shape*, is a reconstruction technique that is capable of reconstruct the shape of a given point set S in the plane, that can be further extended two higher dimensions such as 3-dimensional space. For an α*-shape* of a point set $\mathcal{P}$ we denote the graph with the points of $\mathcal{P}$

On 3D the α-shape algorithm uses the delaunay triangulation and the convex hull of the point set. At first creates a delaunay triangulation that is restricted to the convex hull of the point set, then to each vertex of the triangulation is assigned a weight that is represented as a radius related to a given alpha value for that shape. Bigger alpha values will increase the radius of each point and the algorithm extracts all the edges that are on the boundary of the circle or sphere with that radius. So it's easy to see that for small alpha values the resulting shape is simply the point set and with the increase of the alpha value the shape that is extracted will increase. By using this technique the extracted shape could have holes that are iteratively covered by increasing the alpha-value and testing the resulting mesh.

2.2.1.7 Software Libraries

CGAL Typically geometrical libraries gave us the data models and algorithms to perform geometrical computations on those data models. These tools are widely used in

some areas such as computer graphics, medical imaging and scientific visualization. This geometrical algorithms could be triangulations, Voronoi diagrams, mesh generation, geometry processing among others. The choice of the library will be limited to open source libraries, and preferentially portable among different platforms. The two libraries that will be considered on this work are CGAL and Wild Magic, and we will look at each one of them in greater detail in order to figure how they could be better used to perform the necessary characterizations.

Introduction Computational Geometry Algorithms Library, also known as CGAL, is a computer graphics software library that is used mainly for geometric and mathematical computations, it is a well documented library that is widely used in many areas and have support for multiple data structures and geometry algorithms. Developed in C++, has bindings for other languages such as Java and Perl, but they are not complete and only contains a subset of the modules from the library. It makes use of generic programming using templates and is targeted at being a generic and modular library. Having an clear focus on geometry is implemented with de facto standards *STL*, *Boost* or *BLAS*.

Main goals Its main purpose was to gather the existing geometric algorithms and make them available for industrial application. At it algebraic foundations CGAL aimed at exact computation on top of objects that are defined by algebraic curves and surfaces, and its modular implementation allows to detach the algorithms from one fixed number implementation allowing the algorithms on the library to be used among different number representations. There are several kernels available, ranging from exact predicates with inexact constructions, exact predicates with exact constructions and exact predicates with

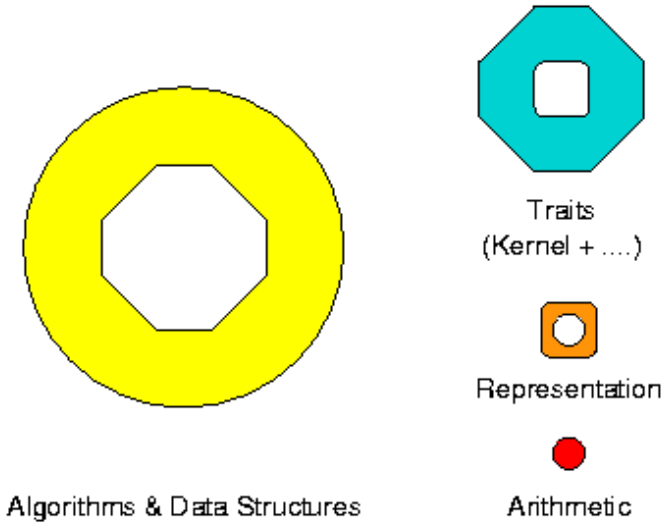


Figure 2.2: Generic Design of CGAL.
[FGKSS98]

exact constructions with sort, allowing fast computations.

Design It is a modular library composed by several layers, and can be seen as structured in three layers together with a support library for visualization purpose. The core library have the basic non-geometric functionality. The kernel library have the basic geometric objects, like points and lines and basic operations to work with them such as computing intersection and distance among objects, it is also split in three parts to deal with two-dimensional, three-dimensional and general-dimensional objects, having for all dimensions Cartesian and homogeneous representations. Finally the basic library has more complex geometric objects and data structures such as polygons and algorithms to work with there data structures such as convex hull or the union of two polygons.

Packages CGAL is structured under packages according to their purpose we will make a presentation of the relevant packages to our work and how can they be used on this work. Looking at CGAL packages we find the Principal Component Analysis package that is composed of functions to analyze sets of 2-dimensional and 3-dimensional point sets, such as the computations of axis-aligned bounding boxes, centers of mass and principal component analysis. All of them will be needed on this project, also this principal component analysis allow us to perform the computation of moment of inertia for surface triangle meshes[GAP08], needing only to convert the representation of an object to a set of tetrahedra. The Surface Reconstruction from Point Sets package has a set of methods that could allow to extract a mesh from a point set, extracting an isosurface[2] from the dataset and reconstructing the surface as a set of tetrahedral, this brings the ability to after that make the principal component analysis and also the other characterizations needed.

As seen this library have the fundamentals to figure geometric indicators from objects that is needed in our work and it is our main choice for the development of the project.

Triangulations The classes under CGAL triangulation package have two template parameters providing the geometric traits and the data structure to use on the underlying triangulation. It have two- and three-dimensional triangulations from a set of points and they are represented as a simplicial complex whose domain is not restricted to interior of the points to be triangulated but covers their convex hull. [BDTY00] One can found under the two-dimensional triangulations on a plane several triangulations such as a Delaunay, regular, constrained and constrained Delaunay triangulations, in three-dimensions CGAL doesn't provide constrained and constrained Delaunay triangulations.

Here we will focus under the 3D Delaunay triangulation package presented by CGAL and mainly they provide the partition of the space $\mathbb{R}^d$ onto cells of $d+1$ vertices, some of those vertices are defined as infinite and are linked to each face of the convex hull from

[2] A surface that represents points of a constant value within a volume of space

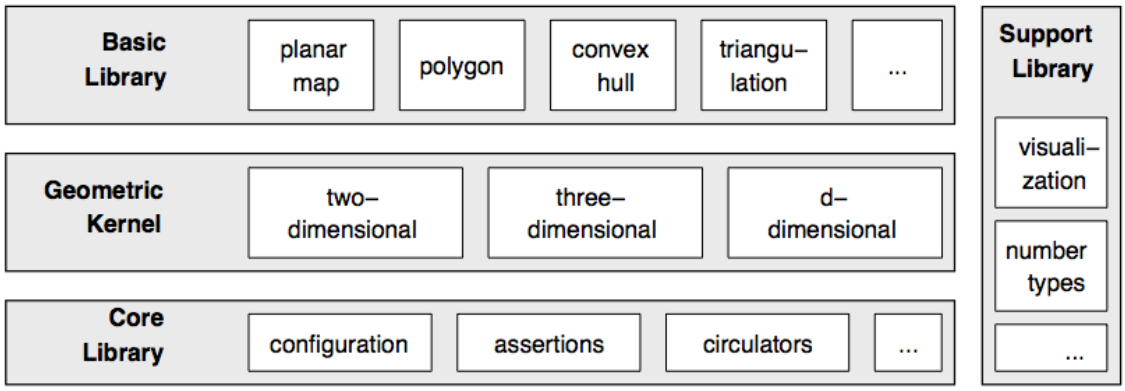


Figure 2.3: Structure of CGAL.
[FGKSS98]

the point set, that way is possible to enclose the full space and allow to deal with degeneracies[3]. Those infinite vertices don't have no geometrical meaning but will simplify the computation by easily identifying the facets that belong to the surface from the object.

Triangulation Implementation Design The triangulations on *CGAL* are provided by a model that separates the combinatorial structure from the geometric information, it is built upon two layers. The combinatorial structure, belong to the top layer, and is provided by the triangulation data structure that is parameterized with the geometric kernel. This layer is presented as a model for example in the case of the regular and delaunay triangulations providing implementations for them. Then the vertex and cell base classes, belong to the bottom layer, they store the elementary incidence and adjacency and other geometric information and are parameterized by the triangulation data structure. This is a very modular architecture allowing to extend the basic functionality over the cells and vertexes easily extending those classes, or replacing for example the underlying triangulation data structure. This structure is used through the CGAL library where triangulations are needed, for example the alpha-shapes package is parameterized with a triangulation data structure and implement specific cell and vertex base classes with information for the alpha value for that cell.

Surface Extraction CGAL provides several algorithms to extract a surface from an object, most of the algorithms presented are very time demanding but the *alpha-shapes* package provides a rapid extraction of an approximate mesh from the point set. Since it could be directly mapped to our data and that the processing times are reduced we have used the *alpha-shapes* package to extract the mesh from an object. Under this package besides the common fixed *alpha-shape* that is parameterized with an *alpha-value*, *CGAL* also provides methods for finding the minimal *alpha-value* that fully covers the surface.

[3] is a limiting case in which a class of object changes its nature so as to belong to another, usually simpler, class.

2.2.2 Data Persistency

In order to achieve data persistency one could use a *dbms*. Although providing a framework to easily store and retrieve some data, it also provides the portability and flexibility to easily extend the system that we are targeting. Next we introduce the general design ideas from some existent dbms's and we will look at their characteristics targeting the goals on this project.

DBMS Is a software program that enable users to create and maintain databases. Over the years it have become a very complex system where the data can be stored and retrieved in a very efficient way. The most common form of DBMS is a relational database, also known as *RDBMS*, where the data is stored into tables that have relationships between them, that is also what we will use.

Design Commonly *DBMS's* stores the data values in a row-store fashion way, where the tuples are stored in sequential blocks on memory or disk[Pla11]. This form of architecture fits well for query insertions, where the inserted rows could be inserted without overhead, and simple queries where some row is retrieved, but not so well for column scans where the retrieved data isn't contiguous. This row oriented architecture was mainly oriented to maximize the I/O traffic on queries and thus minimizing the number of block read/writes[Bon02]. This has append mainly because hard disks are used to store the data, so DBMSs are optimized to it. But with that a problem arises, because of this data organization even if one wanted to optimize the database performance using lower latencies memories, such as main memory, the data isn't optimized for those devices, so several penalties arise of using this design on different hardware with different capabilities. Although the increasing frequencies of memory is stalled their sizes continuos to increase and todays computers have good amounts of it for low costs, this with the reduced latencies compared to hard disks, seems to be a good reason to justify their usage on current DBMS.

ACID Also it is typical for DBMS's to comply with atomicity, consistency, isolation and durability properties known as ACID. These are important rules when designing a successful commercial DBMS cause they bring guarantees to the stored information and the transactions that are issued will remain consistent even under system failures. They are capable of multiple connections allowing several clients to perform simultaneous queries to the database. Unfortunately this brings more complexity and overhead in transactions and DBMS core system making it a very reliable software but with some loss in performance.

Portability By choosing to use an RDBMS with SQL capabilities we are ensuring that easily new queries that could fully exploit complex relationships among objects could

be more easily created, and we also guarantee interoperability among different DBMS frameworks.

There are several DBMS frameworks available, and currently the target machine will run under Microsoft Windows, we will flavor one that is capable of running on that operating system. We have studied two DBMSs, the MonetDB and CSQL, both are RDBMSs fully supporting the SQL has query language, have a main-memory design and are column-oriented. CSQL although of being an open source solution the server-side only have binaries under Linux, so we have preferred the Monet.

2.2.2.1 MonetDB

Monet is a in-memory open source relational DBMS that has a complete vertical fragmentation of data, is column oriented, optimized for query intensive applications and it was designed with focus on bulk processing[Bon02]. Monet stores each column in a binary association table(BAT) table, so a column composed by a set of records is stored as a set of (key,value) tuples, being the key the identifier for the current record line on that column and the value the actual value for that record on the column. BAT's are mapped to memory using memory mapped files as two memory arrays. A relational algebra have been implemented in order to work with this BAT files, and every results for the queries are also a collection of BATs.

Column-oriented RDBMS Monet was designed so it can be a backend RDBMS platform capable of interpreting several language and coupling with other DBMS systems that could rely on Monet to store the data in a column-oriented way and highly optimized for queries. It it focused on query intensive applications and the framework is designed to explore the usage of the underlying hardware and optimizing their data structures to main a memory execution, reducing at the maximum the CPU stalls, so optimizing its performance.

MAL In order to perform all the logic of a relational model, Monet have a *BAT* algebra that accomplishes that. The BATs and algebra stay inside the MAL[4] framework. This is a assembly-like language that the Monet core is capable of executing and have all the functionalities to perform the operations of a typical DBMS on this column-fashion architecture. This intermediate language provides the abstractions needed to allow interoperability with other DBMSs and allow for good performance optimizations.

Architecture At it architectural design Monet is composed of three layers, on the top level are the query compilers that translate the queries from SQL, or other languages, to algebraic query plans. Bellow the query layer is an optimization layer that optimizes the generated MAL algebraic plans to a more compiler friendly MAL execution plan,

[4]Monet Assembly Language

exposing several tight loops that help compilers to achieve better instruction parallelism and thus optimizing the performance. The bottom layer is the execution layer that have the relational algebra used to process the MAL execution plans together with the BAT files containing the data.

Conclusion Although this column-oriented dmbs give us very good performance for querying the database the insertions of records are slower than in a typical row-oriented DBMS, this is because of the overhead needed to insert a row in several columns, in this program there are few insertion queries so the drawbacks are few.

2.2.2.2 UnixODBC

It is an open source specification for providing developers with a predictable API with which to access data sources. By choosing to use it one gains portability, since it works under all windows and linux platforms. This framework can be seen as a driver manager allowing the easily configuration of the underlying data sources, by allowing the user or the system administrator to provide files for mapping a connection to a data source, these file is called DSN, or Data Source Name, that is a file containing all the information to access the underlying *dbms*.

2.2.2.3 TiODBC

Although the use of an *odbc* successfully provides the needed portability it is quite complex to write a simple program to access a database since the *API* is from a very low level. For that a small library called *TiODBC* is used that wrap the *UnixODBC* library making it much easier to work on with the dbms to insert and retrieve values.

2.2.3 Reducing Execution Time using Available Cores

The CPU has more than 30 years of development and since the 60's their frequencies become much more faster than memory frequencies and interconnection buses that supplies them with data, since then the CPU starved for data and several techniques have been used to alleviate this problem. Modern desktop CPUs architecture are a mix of a RISC[5] and CISC[6] architectures.

2.2.3.1 Architecture

A RISC processor is simpler to implement in the form that instructions are atomic achieving more efficiency and more processing power. X86 processor architecture, the one used on main desktop computer processors, is CISC and those instructions are decoded into simpler RISC instructions, making this architecture a mix of RISC and CISC. They also

[5]Reduced Instruction Set Computer
[6]Complex Instruction Set Computer

make use of a pipelined instruction set, so are superscalar processors,[Wik12] allowing to perform multiple instructions.

SIMD Since the release of MMX from Intel and 3DNow from AMD several registers have been added to processors making them capable of SIMD[7] instructions that can be used to graphics and other uses, but must be addressed with very low programming languages such as assembly also some compilers make use of this registers to achieve better performance, but in order to achieve it processors need to extract parallelism from the instructions.

Speculative optimizations Such as multiple execution units, out-of-order processing and brach prediction have been developed to increase the overall performance. Most of this developments tried to exploit the principles of temporal and spatial locality in code. CPUs are optimized to deal with few threads that have high data locality and a high percentage of conditional branches.[Gla09]

Cache Several levels of cache have been added to processors to reduce the gap between memory and processor frequencies, and avoid processor stalls. With the increased number of cores inside the same die the level grow to three levels, usually distributed by one first level inside each core and a second level shared among a group of two cores, with the appearance of more cores a third level have been introduced that is shared by all cores inside the same die.

MIMD Since current processors are multi-core they belong to a MIMD[8] computer architecture class where each core is seen as an independent computing unit with its local memory, and could access a shared memory between all cores. Each of those cores are capable of executing independent tasks that could share data between them or not.

2.2.3.2 Programming Models

The parallel programming models that exist may be divided into classes based on their assumptions they make about the underlying memory architecture[ref. wiki parallel computing]. On the hardware approached the communication occurs through a shared address space, so we will look at some of the APIs that exist to program this hardware. Here we will look at pthreads and OpenMP two widely used APIs to achieve parallelization in shared memory architectures.

PThreads are also known as POSIX Threads is a standard to work with threads. Several implementations are available on Unix POSIX-conformant systems and also Windows systems. Threads are independent flow of task that exists inside a process,

[7] Single-intruction Multiple-data
[8] Multiple-instruction Multiple-data

they share the same process resources, but they are lighter than a process, and so can achieve better performance than a process fork. On the Pthreads API we can found several functions grouped within four groups:

- Thread management
- Mutexes
- Condition variables
- Synchronization

In order to use it programmers should break the computational work within different threads and deal with data partitioning and synchronization among those threads. The threads are explicitly created and the paradigm to use when programming must be different from single threaded.

OpenMP is more recent than pthreads and tries to be a portable and scalable model for parallel programming. It accomplishes that with pragmas added to code sections making them parallel, those pragmas are prepared by the compiler on a preprocessing stage, preparing that peace of code to run in parallel. The data decomposition provided by *OpenMP* is done automatically by the framework and in general the original (serial) code don't need to be dramatically changed in order to run the new directives in parallel. *OpenMP* uses directives to control the number of created threads, synchronization and flow control of the work among threads.

2.3 Solution Organization

Our proposed solution is composed of one module, integrated onto the SCIRun environment, that will do the characterizations using the *CGAL* library and store the results on an underlying *dbms* through *odbc*.

On the next section we start by describe a sequential version of the characteristics extraction.

2.3.1 Organization

As seen on section 2.2 the *centroid* and an *AABB* are used to compute other geometric characteristics such as *PCA*, as so, it has been decided to create a class that represents an identified object, where some common measures of an object have been added, e.g. centroid or axis-aligned bounding box. Those are values that could be computed incrementally at the time of the insertion of points on each object, that way some more complex characterizations algorithms could be built, e.g. *PCA* or *OOBB*.

The characterizations that are added to the system extend an abstract class named *Characterization*, and each object have a list of the characterizations to be performed. The sample that is loaded, containing all the objects, is implemented in a class named *Sample*

whith all the functionality to load the sample from the *SCIRun* environment. At the end of the characterization stage the resulting data from all the characterizations is stored on the specified database. In order to integrate all this work on the *TomoGPU* project, a *SCIRun* module have been produced containing a *GUI* to choose the required characterizations and set the connection details from the database connection where to store the results.

Next we look into further detail each of the early presented classes, that are also on Figure 2.4.

Characterization Module It contains the *GUI* implemented using a *tcl/tk* interface onto *SCIRun*, that allow to choose the required characterizations and connection details. The available characterizations are *PCA*, *OOBB* and *area*, the user could also choose the name for the database connection details already configured on the *odbc* layer.

Sample This is the class that have most of the logic for the module, it starts to load and initialize the objects from the input into each object class storing them in a array. The number of objects on each sample and the meta information for the sample is available at the initialization of the sample, as so, it is initialized on the database that information for the sample and received an id for the sample that is stored also this class. After successfully load all object it stores each object *AABB* and *Centroid* values on the database and launches the computations of the characterizations on each object. At the end it fetches all the results from each object building and inserting a *SQL* string on the underlying connection for each selected characteristic. It contains the methods for creating the *SQL* strings that are inserted on the database, assembling a insert string for each characterization data of that object containing on each string all of the values of that characterization.

Geometric Object Class Each loaded object is represented by a class named *GeometricObject* that contains some common functionality over an thee-dimensional point set. It extends the std::deque class and provides iterators over the object points. Each object besides the *id* for that object have the *AABB* and *Centroid* for that object, giving also accessors methods for that values.

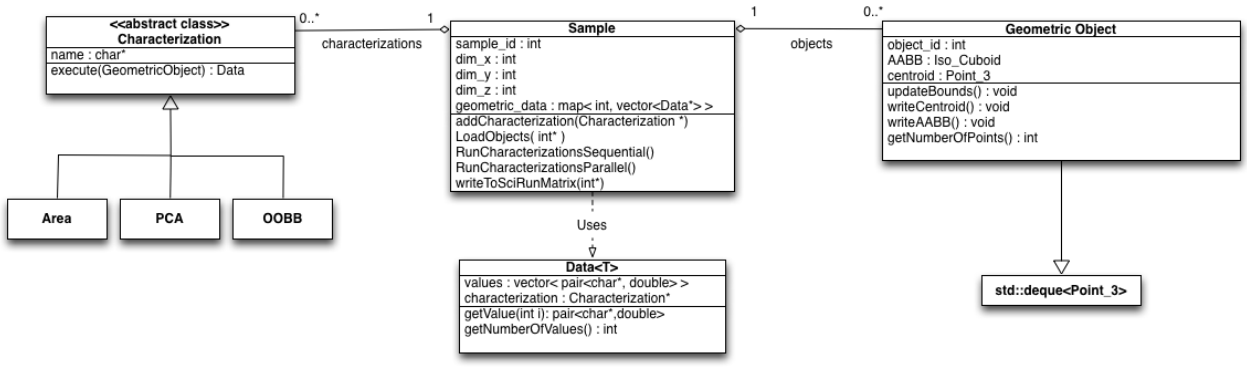


Figure 2.4: Class Diagram for the Characterization Module.

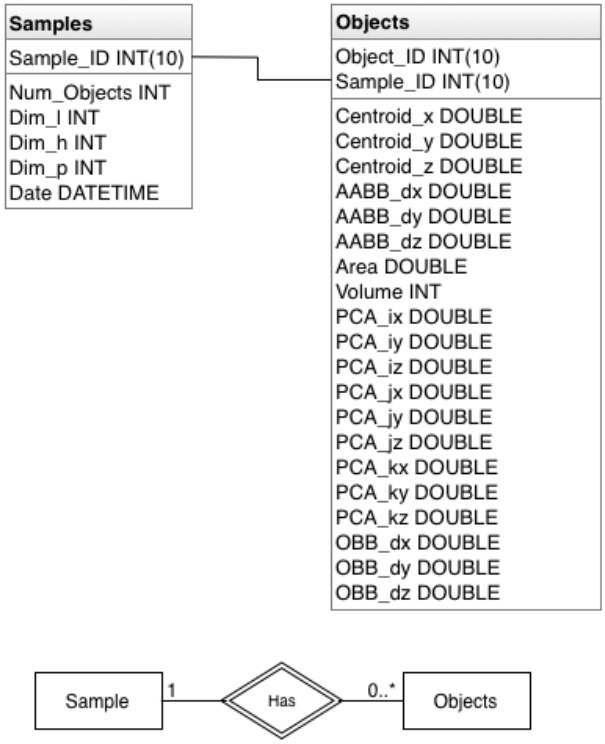


Figure 2.5: Database Layout and Entity-Relationship diagram of the database.

Characteristic Abstraction This is an abstract class that will serve to interface all the characterization sub classes. It contains the name of the characterization, a execute function to launch the characterization on an object. The resulting data is created with the values from one characterization, as so each characteristic that extends this class must return the correct keys and values to be latter stored on the database.

Data This is the data that is created by each characterization process and stored on the sample class. It have a vector data structure with all the sub-values from that characterization stored in a pair<key,value>, being the key the name for the characteristic sub-value and it's respective value represented as a double value.

2.3.2 Storage

To provide the connectivity to the database layer we have used the *odbc*, as explained earlier. This abstraction layer is very useful to abstract the concrete *dbms* used, although to be able to use it on a project usually implies to the programmer create another layer on top of *odbc* to provide a simple way to interact with the database. There are already some thin libraries that could be used to do exactly that, avoiding to write such layer and more easily providing the required functionality.

Since we can't assure what is the used *dmbs* we choosed to isolate the code that does the communication with the databases in order to assure that only one access at a time is made to the database. The system stores info for the sample to be loaded and at each

sample insertion the database assigns an id for the sample, that is returned in order to at the time of the insertion of the characterizations data this id for the sample could be used. Each object on a sample is already identified with an id, so we have used

2.4 Implementation

As seen previously some of the characteristics are implicit defined on the data such as the volume, others will be performed as explained on 2.2.1. We next provide the implementation details and discuss the results of the performed characterizations, but before that we show how it is structured the internal workflow for the framework. Besides providing some characterizations the framework also have to perform the storage to the underlying database.

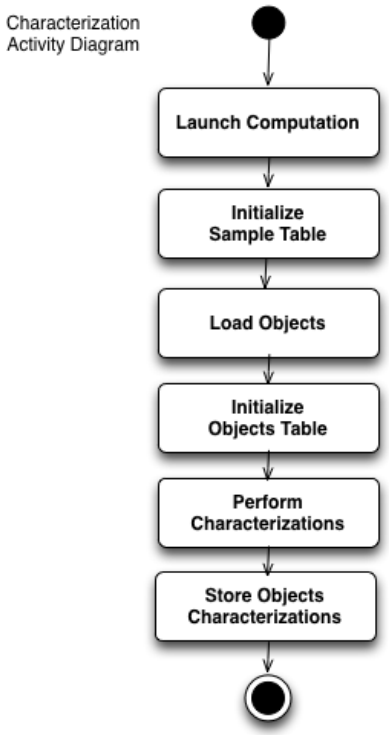


Figure 2.6: Activity diagram for the characterization module.

As can be seen on 2.6 at the initialization phase it is inserted on the database the data for the sample and the database is queried for the id of the inserted sample, with the id for the sample and the id for the object it is also inserted on the database a row for the current object, also after each object initialization the initial values for the object basic measures, such as the centroid are also stored on the database. The total number of database insertions at the initial phase is related to the number of objects on the sample since we update the info on the database for each object, this step is computed sequentially. Later

with the data returned from the characterizations, that is stored on the sample class under a *map<K,V>* where the *K* is the id for the object and the *V* a list of *data*, is assembled a *SQL* string for each data and inserted on the database. This access to the database is performed at the end of the computations and is also performed sequentially since we can't guarantee that the access to the underlying database is thread safe, because that is dependent of the choosed *dbms*. Although the system is capable of correctly insert the data on the underlying *dbms*, the modules are expecting that the respective tables have been previously created on the *dbms* and made available through an *odbc* data source file, also the name for each implemented characterization column to be stored on the characterizations table is given by the key value on the data that is returned by each characterization process. The data to be inserted on the database is built as a string using the *INSERT (K1,K2,K3) ON DUPLICATE KEY UPDATE VALUES (V1,V2,V3)*, from all the values from an object characterization data, meaning that at this finalization stage we will update n rows on the characterization table, being n the number of objects on the sample.

2.4.1 PCA

PCA is done through CGAL, here the process described on 2.4.1, will first extract the 3×3 covariance matrix from the object points, and latter the eingenvalues and eigenvectors are extracted through linear least squares fitting over a plane. This will give the first two directions and the third one is extracted by computing the normal vector of such plane. The resulting orthogonal axis from the principal directions is centered on the computed centroid of the object.

Lets see how it can be computed under an 3-dimensional space:

Center the Data There is the need to subtract the centroid from each data, thus centering the dataset around the origin

Compute the Covariance Matrix Compute the 3×3 covariance matrix for the data set such as stated on A.2

Extract Features Vector This is the vector with all the three principal components, eigenvectors and eigenvalues are extracted using linear least squares fitting

Here the data is fitted onto a plane to extract the first three principal components using least squares fitting.

2.4.2 Bounding Boxes

Other common way of representing a parallelepiped on 3d space is using an center point C, representing its centroid, an orthogonal set of vectors $\{\vec{u}, \vec{v}, \vec{w}\}$, and three scalars representing the half-width, half-height, and half-depth.

AABB This is a trivial computation, it should be evaluated the minimal and maximal values under all axis, and extracted the values from the AABB.

OBB To extract the *OBB* the initial data set is centered around origin, the are applied affine transformations to rotate the dataset to fit the principal component vectors extracted previously by *PCA*, to each one of the planes xy, xz and yz, on each fitting one bounding box is extracted, and is retrieved the one with less area. The resulting *OBB* is the represented by the three dimension values of the length, width and height of the bounding box and the three orthogonal vectors representing the principal components and the object centroid.

2.4.3 Surface Area

In order to compute the surface area it is first needed to perform an approximation of the shape from the object or to extract it's surface mesh. With that it is possible to compute the surface area as the sum of all the areas from the exposed complexes that compose the obtained mesh or shape. We have several choices to perform this step. At a initial phase we have used the alpha-shapes package from the *CGAL* to extract the mesh of the object. This algorithm as presented could introduce holes on the resulting mesh and with that errors on the final measurement. To avoid that and as explained the algorithm must be used with several alpha-values and for each test if the mesh is closed or not. The *CGAL* already contained all the software to provide that but on bigger samples the computing times could break the required interactivity.

A second alternative was then implemented and since the dataset received is closed and don't have holes a simple algorithm could be implemented to extract the voxels of the surface on the initial object. As so, to compute the area first it is extracted the axis aligned bounding box from the object, and then a grid is constructed with all the voxels from the object. This is a boolean grid that has a true value if the voxel at (x,y,z) belongs to the voxels from the object and not only the surface. After that it is searched for each voxel if it belongs to the surface or not, figuring for each of the voxel six faces if there exist another voxel connected to that face, the faces that don't have any voxel connected to it are the faces from the surface and the voxel of that face can be marked as an surface voxel.

This is a simple algorithm that have a complexity of O(2*N), being N the number of voxels of the object, and fits the desired needs of the project approximating on the discrete grid the total area of the surface.

2.4.3.1 Surface Extraction

2.4.4 Tests

The framework have been tested for each one of the required computations, and the results are presented here. All the tests that are presented on this section where produced on a desktop computer with a quad-core Intel Xeon E5506 @2.13 GHz cpu, 12 GB RAM DDR-3 800 MHz memory, the operation system(OS) used is Linux Ubuntu 10.04.4 LTS (kernel 2.6.32-45). All the computations perform sequentially and we obtained the following measures. The tested samples are derived from a 400x400x400 voxel 3d matrix, that when received has 272 objects and a total number of voxels from all objects of 2691749.

Submodule	Time (s)	% of the total time
Initialization	0.410	65.71
PCA	0.071	11.38
OBB	0.071	11.38
Area	0.072	11.54
Finalization	0.214	34.29
Total Time	0.624	100

Initialization include the time to read all the data from the input and the initialization of all the *Geometric Objects* with those data, including the initial computing of the object centroid and axis-aligned bounding box. After that each module is started and the obtained times are extracted since the module starts to work until it returns the data. As one could see all the characterizations are fast to execute, and the system achieves the needed interactivity.

2.5 SCIRun Integration

In this section we start by giving a brief description of how the code developed can be integrated in the *TomoGPU* system as a *SCIRun* module. In a second part we show how the characterization module is positioned in the global system.

In order to integrate the system on the *Tomo-GPU* project there was the need to develop some modules for the *SCIRun* environment, as so, a graphical *tcl/tk* file was developed that should give to the user the ability to enable or disable each characterization submodule, and parametrize each characterization when needed. This file GUI module have several checkboxes, one for each characteristic, that will be mapped to boolean variables on the scirun extended module class.

2.5.1 How to turn the standalone code in a SCIRun module

SCIRun is organized as pipeline of modules, where the execution of a module is fired by the arrival of data at an input port. The data sent by the previous module corresponds to a SCIRun mesh object, where the 3D matrix data is accessible.

On appendix B one could see how the interaction with *SCIRun* is achieved.

2.5.2 Module position in the TomoGPU software

The characterization module receives a sequence of integers with the following organization

- an integer with the number of objects
- for each object
  - the ID object identification
  - the number N of voxels of the object
  - N integers, one for each voxel; each integer C represents the voxel position in the sample. Being L, H and P the dimensions of the sample, respectively in x, y and z, This integer is coded as

$$C = L \times H \times z + L \times y + x$$

This format is produced by the Object Labeling module represented in figure 2.7; the output of this module can be processed by the object cleaning module. This module can eliminate objects according to a given criteria - for example, deleting objects that are too small.

This module does not have an output port, as it is the last step in the pipeline. Its output is the already mentioned database of objects with its relevant characteristics.

2.6 Parallelization

2.6.1 Approach

There are several ways to provide parallelization and the system architecture provided the independence among different characterizations, as so, we have choosed to parallelize all the objects and their characterizations, that way when a new characterization is added to the system it will be launched in parallel with all the others. Instead of parallelizing the computations among several objects it could be also parallelized each characterization algorithm. Since the algorithms are relatively simple and with linear complexity, they don't justify their parallelization, in either cases any characterization further added to the framework could also be parallelized.

The pseudo code on the sample file for launching the computations is:

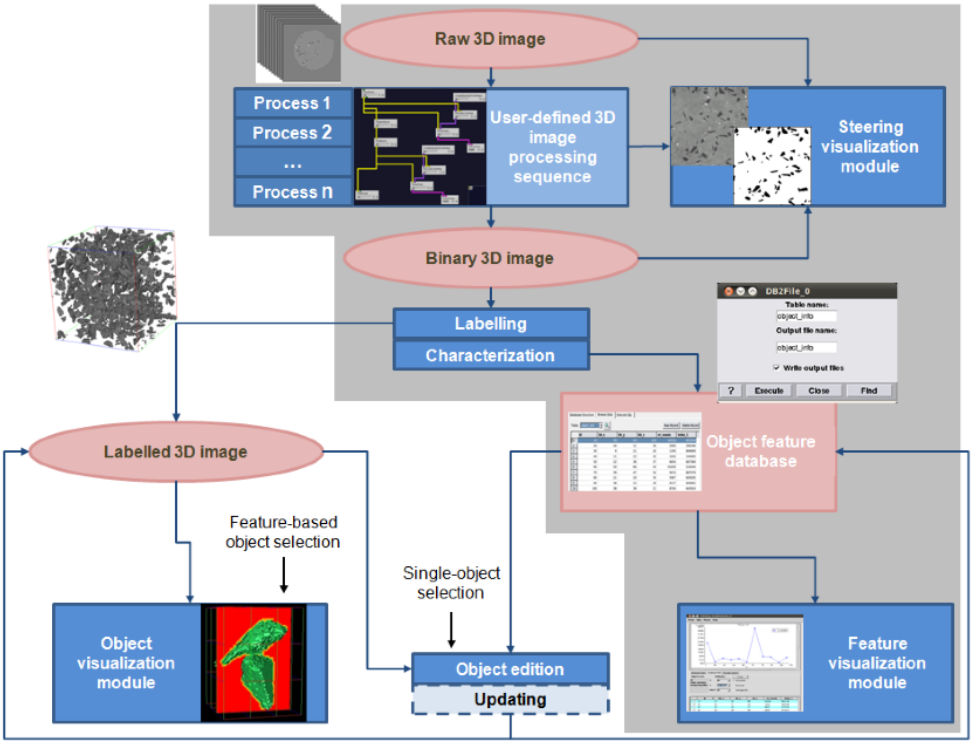


Figure 2.7: General organization of the TomoGPU system.

```
initialization phase
for all objects on sample
  for all characterizations
    characterization->execute(object)
finalization phase
```

To provide the parallelization we used *OpenMP*, and there are two phases that run in sequential, as so, couldn't be parallelized. A first initialization phase where all the data is loaded and the objects initialized, and a finalization phase where the data is stored on the database.

2.6.1.1 Characterizations Parallelization

Each individual characterization is independent from all the other characterizations and it was possible to launch every characterizations on each object in parallel. The sample file was changed in order to launch all the characterizations on parallel, adding a *omp parallel for* pragma on the first for loop, executing the loop over the determined number of threads. On this work only the three characterizations *OOBB*, *Area* and *PCA* are launched in parallel, but that number may increase with the introducing of new characterizations. Next we show the results for this parallel implementation comparing them with the first sequential implementation.

```
initialization phase
```

```
  for all objects on sample
    #pragma omp parallel for
    for all characterizations
      characterization->execute(object)
  finalization phase
```

Sample	2 Threads	4 Threads	8 Threads
$100 \times 100 \times 100$	–	–	–
$200 \times 200 \times 200$	–	–	–
$400 \times 400 \times 400$	0.553	0.563	0.575

2.6.1.2 Objects Parallelization

The different objects on each sample are also independent and after successfully loaded all the objects, all the objects could be analyzed in parallel through the usage of an OpenMP pragma parallel for.

```
  initialization phase
  #pragma omp parallel for
  for all objects on sample
    for all characterizations
      characterization->execute(object)
  finalization phase
```

Sample	2 Threads	4 Threads	8 Threads
$100 \times 100 \times 100$	–	–	–
$200 \times 200 \times 200$	–	–	–
$400 \times 400 \times 400$	0.534	0.526	0.565

2.6.1.3 Collapsing Work

Although the previous approaches ensure the parallelization of the work on some cases they won't maximize the usage of the cpu power, and since the smallest unit for this parallelization is each characteristic extraction such extraction may be complex and it is difficult to assure that the same amount of work is distributed among all of the threads. OpenMP provides a collapse keyword that helps achieving that by flattening the nested loop for, allowing to represent all the work on one dimension and then dividing the work among the available threads.

Sample	2 Threads	4 Threads	8 Threads
$100 \times 100 \times 100$	–	–	–
$200 \times 200 \times 200$	–	–	–
$400 \times 400 \times 400$	0.549	0.507	0.540

2.6.2 Conclusion

The parallelization techniques employed on the characteristics extraction allow to fully use the processing power of the *CPU* allowing more characterizations to be launched in parallel and providing an interactive workbench.

2.7 Conclusion

The number of cores that can be added to *SMP* architectures is limited, typically *SMP* architectures typically scale up to eight processors, for more scalability other architectures are needed because of the single shared bus used and caches complexity. So its fundamental to have good frameworks that could truly be scalable through several architectures.

3

Object Reconstruction

In this chapter we deal with the reconstruction of the surface from a point set sampled on an object surface. We start to introduce the problem that must be solved and take a survey over some mathematical concepts required to perform such reconstruction. After this introductory stage we dive onto the most common used techniques to perform such reconstruction introducing the available algorithms and what are the available libraries to perform it. At the end of this chapter we show how the implement features are organized and discuss about what are the improvements that can be applied and how some of them are achieved in order to reduce the overall reconstruction time.

3.1 Problem

The presented problem can be generically seen as the extraction of a triangular mesh of an object surface from a unoriented and unorganized point set that contains a several amount of noise.

3.1.1 Problem Definition

In similar way to what where received previously on the characterization stage, presented on Section 2.1, the received dataset have the same data layout to represent the objects on a sample. Here the input is also the set $\mathcal{P}$ of n unoriented points from voxels belonging to the object that could be sampled either on the object surface or in all the voxels from the object, depending on what modules are used on previous modules of the *Tomo-GPU* workflow. On those bad datasets it is expectable that near the surface object several artifacts may appear, or some parts of the surface are missing due to the

characteristics of the reinforcements matrix on the presented sample of the composite material being analyzed. Having said that one should reconstruct the shape, representing a closed object, that best approximates the given set of points $\mathcal{S}$ extracting its mesh for visualization and reintroducing again on the *Tomo-GPU* workflow all the voxels of the reconstructed object, including interior voxels.

3.2 Relavant Work

Reconstructing an object surface from an unoriented and unorganized point set is a complex and compute intensive task, there are several papers, books and dissertations to approximate the surface and this is a high topic of academical and industrial research, ranging from the computational methods to the data structures needed to achieve a truthful reconstruction from the sample. The reconstruction pipeline must fully interpret the discrete data that is received dealing with possible noise and filling holes when needed, reconstructing an object shape on sparse and with possible wrongly sampled data of the object.

3.2.1 Computation Geometry

There exist vast documentation around techniques for reconstructing a surface from a point set, and there are several ways of representing an surface Ω, but mainly they can be classified as explicit or implicit representations.[GVJWG09] Most of the different techniques provide some assurances of the reconstruction and share various issues and problems. We will describe the those representations and see some of the reconstructions available under each different representation.

3.2.1.1 Explicit Representation

Besides the two representations presented earlier at Section 2.2.1.2 for, the *Delaunay-based* and *region-growing* techniques, there exist also a parametric surface representation among others. We do not intend to give a complete enumeration of all the different techniques since the literature around such techniques is vast.

A parametric surface representation can be expressed as the graph of a function $f : \Omega \to \mathbb{R}$ defined on some region $\Omega \subseteq \mathbb{R}^d$, and d in general is given by $d = 2$.[Wen10] Using this terminology one could model a terrain, where the patches $X \subseteq \Omega$ decipt certain points on a map and a data value $f_i = f(x_i)$ is the height at point i. Although being very flexible approach for representing a surface they imply the use of global consistency, of the mappings, has to be guaranteed, as so, some operations on parametric surfaces are rather inefficient. Examples of parametric surfaces are subdivision surfaces or triangle meshes, and although is very easy to enumerate points on the surface, by evaluating the value of f at different parameters in the domain Ω

3.2.1.2 Implicit Representation

An implicit surface, or a compact, orientable manifold, is more demanding than an parametric surface, they can be constructed from an point set $X = \{x_1, .., x_n \subseteq S\}$, composed of millions of point in $\mathbb{R}^3$, representing the points on a object surface. Under the implicit surfaces there are mainly two different approaches to build accurate models to represent it. One that tries to find local parameterizations of the object, however for some complex models that approach is limited, and other approach tries to describe S as the zero-level set of a function F, i.e. $S = \{x \in \Omega : F(x) = 0\}$.[Wen10]

The major advantage of implicit surfaces over the parametric surfaces relies on the classification of surface points, on implicit surfaces one should look at the value of the function F at the given point.

3.2.1.3 Conversion between representations

There are several works that propose the conversion from the volume representation to the polynomial mesh representation of its surface **??**, extracting the surface using Marching Cubes algorithm. Most of the this proposed methods use the Marching Cubes algorithm present artifacts that come from the fact the algorithm process the data from a discrete volume and sampling the implicit surface $f(x, y, z) = 0$ is performed on the basis of a uniform spatial grid.[KBSS01] They present a representation of the discrete field that relies on the directed distances in x, y and z directions, instead of using the scalar distance, allowing for finding more accurate samples without increasing the overall complexity. It is also presented an adapted Marching Cubes algorithm in order to detect sharp features based on the local distance field information and its gradient. Additional control points are inserted on the mesh, giving it an significantly reduced alias and the guarantee that the surface normals of the approximation quickly converge to the original surface's normals.[KBSS01]

3.2.2 Space Partitioning

3.2.2.1 Kd-Tree

3.2.2.2 Octree

3.2.2.3 Marching Cubes

It is an algorithm for extracting a triangular mesh that extracts an isosurface from a *three-dimensional* scalar field, usually represented as voxels in a three-dimensional grid where each voxel contains an value for a density of the sampled material. The algorithm works seeded with an isovalue and the voxels with that isovalue are extracted and it is computed an cut that is applied based on the connectivity of neighbors density values. Since our data is in a binary format, that is, we only know if a specific voxel belongs to the surface or not, its appliance to a successful mesh extraction imply that holes may arise on

the extracted mesh, and in order to present the surface as a closed mesh some algorithms should be used to fill the resulting holes.

3.2.2.4 Marching Tetrahedra

3.2.3 Implicit Surface Reconstruction Techniques

We will look into further detail how can one build an implicit surface reconstruction, introducing some different techniques, some use a global function approximation such as radial basis functions and others use a local approximation to build the global approximation function.

Least Squares Surfaces The least squares have been presented early at **??** as an technique for solving overdetermined systems of equations, where instead of solving the equations exactly, it finds a minimization of the sum of the squares of the residuals. By performing this approximation for surface fitting in $\mathbb{R}^3$, one must solve the minimization problem:

$$\min_{f \in \prod_m^d} \sum_{i=1}^{n} \|f(p_i) - f_i\|^2,$$

that fitted using quadratics on three dimensions, $f(x)$ can be written as:

$$f(x) = b(x)^T c,$$

where $b(x) = [b_1(), ..b_k(x)]^T$ is the polynomial basis vector and $c = [c_,.., c_k]^T$ the vector of the unknown coefficients that should be minimized.

This approximation allows to rewrite the system using matrices and algebra gives us the tools to solve such approximation, also this approach allows to express the function f that best fits the model, and each point on the dataset have an constant weight factor, influencing in the same way the resulting function f. Sometimes one would like that f have a local approximation of the data, for that a *weighted least squares* approximation exists in the literature where the weight of each point depend on the distance to the centroid.

Moving Least Squares It is a general method proposed by David Levin in 98, for near-best fit approximations on $\mathbb{R}^d$. It is a local method that tries to locally approximate the surface using polynomials. Every point have it's own support plane, avoiding common problems of piecewise parameterizations for shapes, e.g., parametrization dependence, distortions in the parameterizations and continuity issues along the boundaries of pieces.[ABCOFLS03]

Radial Basis Functions Where first introduced to computer graphics by Hardy [Har71]

and since then, RBFs have gained popularity in several disciplines, but the first surface reconstruction algorithm that uses RBFs appeared by Savchenko et al. [SPOK95] that proposed to use RBFs to interpolate implicit surfaces, Carr et al. [CFB97] used a thin-plate spline RBF to interpolate surfaces on 3D medical images and Turk and O'Brien [TO99] built a RBF implicit surface built up from the interpolation of a point set, in a similar process to the thin-plate interpolation, providing a surface with minimal curvature passing through the point set. The RBF interpolation is the problem of interpolating a multivariate function $f : \Omega \in \mathbb{R}^3 \rightarrow \mathbb{R}$, from a set of sample values $\{f(x_i)\}_{i=1}^{N}$ on a point data set $\{x_i\}_{i=1}^{N}$, as so, it is only needed to approximate f locally by a real-valued function ϕ at each point x_i, this function, is the radial basis function and is dependent upon the Euclidean distance from each point x_i to the mean value so that $\phi(||x - x_i||) = \phi_i(x)$. In order to solve this system Turk and O'Brien [TO99] used a symmetric LU decomposition. The mainly disadvantage of this method is that the interpolation process is very time consuming, if the data points goes above a few thousand.

Multi-level Partition of Unity Y. Ohtake, A. Belyaev, and H.-P. Seidel. A multi-scale approach to 3d scat- tered data interpolation with compactly supported basis functions. In SMI '03: Proceedings of the Shape Modeling International 2003, page 292, Wash- ington, DC, USA, 2003. IEEE Computer Society.

and

Nikita Kojekine, Ichiro Hagiwara, and Vladimir V. Savchenko. Software tools using csrbfs for processing scattered data. Computers Graphics, 27(2):311– 319, 2003.

Poisson Reconstruction This technique as presented on [KBH06] relies on the Poisson equation to reconstruct a smooth surface from an oriented point set and it's main challenge is to accurately compute an indicator function $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ for the object representation valued with positive values for points inside the object and with negative values for points outside, with that it is possible to extract the surface from the object sampling points with $f(x, y, z) = 0$. Mathematically it can be proved that there is an integral relationship between the oriented sampled points on a surface and the gradient of the indicator function f, reducing the problem to the problem of finding the scalar function f whose gradient operator best approximates the vector field $\vec{V}$ defined by the model surface, or $\min_f \|\vec{\nabla} f - \vec{V}\|$. By applying the divergence operator this variational problem could be stated as a poisson problem and formulated as the scalar function f whose Laplacian equals the divergence of the vector field $\vec{V}$,

$$\Delta_f \equiv div(\vec{\nabla} f) = div(\vec{V}) \equiv \nabla \cdot \vec{\nabla} f = \nabla \cdot \vec{V}.$$

The indicator function f is a piecewise linear constant function of several patches f_o, sampled on the input points p_i and to allow it's gradient explicit computation

one have to convolve the indicator function with a smoothing filter. The gradient from the surface model is given by the inward vector of the normal vector at each surface point. Having a continuous vector field $\vec{V}$, the implicit function can be extracted by integrating the vector field $\vec{V}$. This is a problem that may not have solution but it is still possible to minimize the error difference between the estimated surface and the model through the usage of least squares fitting, but first one have to define a space of functions in which to discretize the domain accurately near the reconstructed surface so that the the resulting *divergence* and *laplacian* operator are sparse and the evaluation of a function is expressed as the linear sum of F_o at some point q requiring only to evaluate some neighbor subset to q. With the vector field $\vec{V}$ defined it is then necessary to solve for the function F such that the the gradient of F is closest to $\vec{V}$, but with that other problem arises since the X and the coordinate functions of $\vec{V}$ are in the space of X, but the

To do it one need first to define the gradient field identifying the relationship between gradient of the indicator function and an integral of the surface normal field. This surface integral is approximated over the dataset and later the indicator function for this gradient field is reconstructed as a Poisson problem. In fact, this method is identical to a method presented on [Kaz05] that uses Stokes' Theorem to define the Fourier coefficients of the indicator function.

In order to extract the surface it is used a variant of the marching cubes algorithm that is adapted to work on a octree structure, this algorithm is similar to what we have presented early the only difference is the usage of the octree as a space partitioning structure.

3.2.3.1 Surface Normal Estimation

In order to correctly estimate the surface normals at some point defined on the surface, there should be taken into account the geometrical neighbors from that point on the surface. Sometimes a neighbor defined on the surface isn't exactly the same as the neighbors from that point geometrically, for example two cities that are separated by a mountain can be close on a straight line, but the path from one city to the other is bigger. The same happens with the points defined on a surface and if we choose wrongly the neighbors for the computation of the normal at that point that surface normal will be wrong, most of the times this is achieved by using a different function to estimate the distance from one point to a surface as shown previously. Other problem may arise from the noise of the image because additional points may appear and the normal that we compute will be also wrong. There are several techniques to improve the quality of the computed surface normals, a simple approach is to introduce some filter that will smooth the data before the normals are extracted, a second approach is to increase the number of neighbor points that will be used to compute the normals at that point. This second approach can have problems since as said previously the points choosed to compute the surface

normal should be the points near that point that are defined on the surface. With that is easy to figure that is very complicated to correctly extract the normal the surface normals and another techniques are used align and keep consistent with the surface all the extracted normals, with that we ensure that all the surface normal are constantly oriented approximating a smooth surface and improving the extracted surface.

3.2.4 Image Cleaning

The algorithms under this section will change the data and make it behave in a expectable manner removing some parts of it that doesn't behave as expected, removing points that are away from the from the others. This is very helpful especially when data have noise and we need for example to extract the surface normal vectors at some input points.

Those techniques can and should be used in a pre-stage before the real processing starts when the received data contain noise. Algorithms that fall under this domain are common filters common known as convolutions, as for example the gaussian filter that for each point computes an weighted average of it's neighbors and replaces it's value with the average value. There are other class of algorithms under this domain that instead of generate another data set use that average spacing between the points and with that remove the points that have a bigger distance from it's neighbors that the average and usually it's given a maximal number of points that will be removed from the dataset. Other the points will the With that as is expectable the data presented to

3.2.5 Software Libraries

There are several software libraries that contains some of the algorithms that we have presented on this work, one thing that we noticed is that we couldn't find a library that presents a reconstruction pipeline that could be easily adapted to our requirements. The algorithms that work with unorganized point sets are complex and must be adapted to work with our data. Under the different possible reconstructions we targeted the *poisson* reconstruction. It can be found mainly with on two different implementations. The first one is the one presented on the *Hugues Hoppe* web page [**Hoppea**] and the other is presented on the *CGAL* under the *Surface Reconstruction from Point Sets* package. Since we have already used the *CGAL* to compute the geometric characterizations under the first chapter and that reconstruction although have a different implementation than the one presented on *Hoppe* page, the two achieve the same final result and could perform the required reconstruction.

3.2.5.1 CGAL

Point Set Processing This package of CGAL provides the required framework to reconstruct an surface from a given point set, it have algorithms for the normal estimation and orientation, smoothing and simplification of point sets.

Under this domain *CGAL* offers algorithms for average spacing, outlier removal, simplification, smoothing, normal estimation an orientation. Average spacing computes the average spacing of all the points on the point set over it's nearest neighbor points. Outlier removal sorts the points based on the distance to their neighbors and removes the points that are away from their neighbors, the number of points to be removed is passed to the algorithm.

Surface Reconstruction from Point Set This poisson surface reconstruction have some requirements such as it needs the received voxels to be oriented, that means, that all voxels should have also a normal vector representing the voxel orientation, since the received dataset doesn't contain those normal vectors we will have to infer them first to fully reconstruct the surface. This package implements a variant of the earlier presented reconstruction poisson algorithm solving the poisson equation onto the vertices of a 3D Delaunay triangulation instead of the octree structure as introduced earlier.

Surface Neighbors If we have point set $\mathcal{P}$ sampled from a closed surface $S \in \mathbb{R}^3$, the tangent plane T_x to the surface at a point x, it is proven at [BF02] that the intersection of T_x with $Vor(x)$, is inside this cell, a reasonable approximation of the surface S. With this perspective in mind it could be proven that the intersection of a three-dimensional Voronoi diagram with a plane is a two-dimensional power diagram, that is built by projecting the points onto the plane, having for each point as weight its negative squared distance to the plane, and as seen, CGAL provides package that are capable of computing its dual, that is the regular triangulation in 2-dimensions. With that it is possible to define the surface neighbors from a point that is the dual from explained intersection between the plane and the voronoi diagram. The CGAL classes provided for that are the Regular_triangulation_2, that is parameterized by a geometric kernel, a triangulation data structure and Voronoi_intersection_2_traits_3 class that is parameterized by a geometric kernel.

Mesh Generation *CGAL* presents a meshing framework that is completely parameterizable even in the size or in the shape of the resulting elements that compose the It uses the concept of restricted triangulation, in order to restrict the sampled points on a triangulation to the input domain. It is capable of handling sharp or not sharp domains and it uses an concept of oracle that is capable of answer to some questions about the shape that is extracted.

3.2.6 GPGPU Architectures

3.2.6.1 Hardware

NVIDIA Fermi The hardware used on this study was NVIDIA's Fermi architecture, that represent the current trend on GPUs.

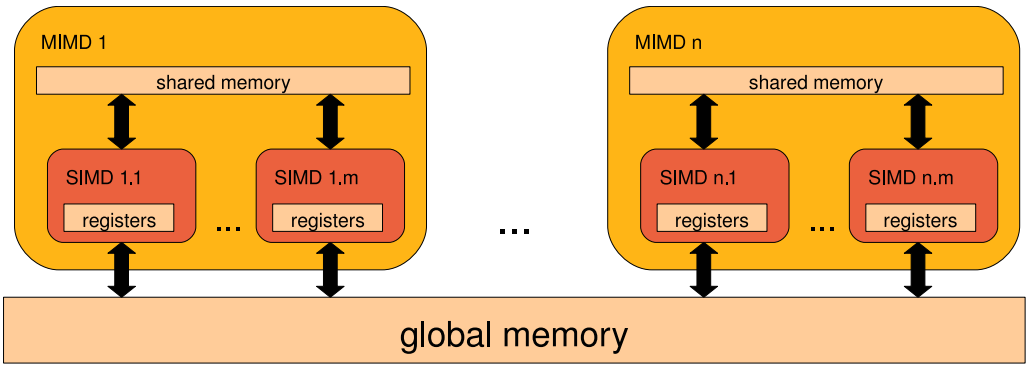


Figure 3.1: GPU with n MIMD multiprocessors and n x m SIMD processors.

NVIDIA's Fermi cards are highly parallel arithmetic processing units that are capable of achieving high bandwidth, around 172.8GB/s the theoretical maximum for GDDR5, and where mainly used for computer graphics processing. Today a level of abstraction have been added creating a virtual machine that programers could use as an additional co-processor capable of achieving good speedups on some heavy computational work. As they represent a different architecture from current CPU architectures we will look at their conceptual design and how they are composed in order to better understand how they can be used to perform computations.

Architecture NVIDIA's Fermi cards architecture, as seen in figure 3.1, is composed of multiple cores known as SMs[1], each SM could be seen as an MIMD. Each card generally has 14 or 16 of them and this architecture could scale up to 18 SM's, the hardware used has 14 SM's. Inside each SM are four main execution units composed by two groups of 16 processing units, one group of 16 load/store units and other group of 4 SFU's[2]. Along with that execution units is a 32KB register file, 64KB of configurable RAM[3], and thread logic hardware. The memory operations that feed the processing units with data are handled by the load/store units. The SFU's are used to handle special mathematical operations such as sin, cos and exp and others. Within thread logic the Fermi has two groups of scheduler / dispatch blocks, each one of them send a warp to any of the execution units, making it a true dual-issue design. The double-precision floating-point operations completes in two cycles, half of the performance than single-point operations, similar to what appends on current CPU's. One could look at each SM as a simper in-order, dual-core processor with 32 thread processors thus 16 inside each core. That give us 448 multiple processing units, composed by 32 threads $\times 14$ SM . The level of multithreading depth in each SM is of 48 concurrent threads, thus 672 concurrent threads composed by 48 threads $\times 14$ SM .

[1]Streaming Multiprocessors
[2]special function units
[3]Random-access memory

ISA PTX[4] defines a virtual machine and ISA[5] for general purpose parallel thread execution and provides a stable programming model and instruction set for general purpose parallel programming.[NVI10] It allowed for the hardware implemented shaders and graphical primitives to be implemented outside the graphic card and afterwards compiled to run there, with minimal lost on their performance and allowing this architecture to become general-purpose computing on graphics processing units(GPGPU). With it programs written on multiple languages such as C, C++, Fortran could be compiled to a intermediate machine-independent ISA to run through multiple GPU generations.

Memory Hierarchy This architecture provides for 64K of L1[6] local memory in each SM, this memory can be split as cache and shared memory, giving to programer the change to choose the best case to use. An L2[7] memory is also found, having 768KB in size and is used by the 512 cores. This L2 memory is capable of performing a set of memory read-modify-write operations that are atomic.[Gla09]. The GPU also have another local memory hierarchy that is the DRAM, it have six 64-bit DRAM channels capable of up to 6GB of GDDR5 DRAM. To this last memory hierarchy we call global device memory.

Conclusion As seen a GPU is capable of executing a large number of threads in parallel and its programming model supposes that operates as a coprocessor to the main CPU, running compute intensive portions of applications spread around multiple threads. It follows a Single Instruction Multiple Thread(SIMT) approach with shared memory, that enables programmers to write thread-level parallel code for independent, scalar threads as well as data-parallel code for coordinated threads.[NVI10] A warp is a group of threads that run the same instruction through different data following the SIMD model. At each cycle each SM could issue two warps of 32 threads, organized on two groups of 16 threads groups each.

Heterogeneous Hardware Architectures When we talk about heterogeneous hardware we are referring to the use of a CPU and GPU that could collaborate to perform computations improving the computational power. These two components could be integrated inside the same die or connected through a high bandwidth bus like PCIe.

Currently a wide branch of new chips were introduced by, SoC[8] on Intel side and APU[9] on ATI. These chips generally follow the RISC architecture and include also a GPU core on the same die. Its a cross-functional chip with enough power to run a operating system and achieve good graphic performance with lower power consumption and reduced price. They mark a new direction on the market that now tries to create a better

[4]Parallel Thread Execution
[5]Instruction set architecture
[6]first-level cache
[7]second-level
[8]System-on-a-Chip
[9]Accelerated Processing Unit

ratio between power and performance and have hardware with higher degrees of parallelization, fitting like a glove to the current desktop market needs. With a similar chip that has programable hardware for graphic processing delays are reduced, and tend to be very power-efficient by the use of simpler architectures reducing processors design complexity. They follow a in-order design in regard to threads execution, and the CPU and the GPU share the same interconnection mechanism to memories, so they have lower bandwidth than a dedicated GPU and this interconnection represents a possible bottleneck for this architecture.

Those chips although very efficient to the applications needed by a desktop user they cannot compete with a dedicated GPU, that doesn't suffer from this memory bottleneck and are composed of many-cores, but they don't are really designed to be highly parallel processors. They are not so good to do high data parallel and high computational workloads, because of their lower degrees of scalability and sharing the same memory than CPUs creating memory access bottlenecks.

3.2.6.2 Programming Models

CUDA It is a general purpose parallel computing architecture, it allows developers to use C programming language and is composed of three key abstractions. A hierarchy of thread groups, a single unified shared memory, and barrier synchronizations. With this the developer is exposed with fine-grained data parallelism and thread parallelism nested within coarse-grained data parallelism and task parallelism[NVI11]. It emphasize the divide to conquer paradigm where an problem is breaked into smaller subsets that are processed in parallel. The greatest difficulty to achieve a good performance relies with a good approach from the programmer to better divide the data between those threads minimizing data decencies and the minimizing the necessity for synchronism. When doing that one should be aware of the underlying hardware and execution models to better partition the the data among threads under execution warps.

It has an execution model composed by an host, the CPU and a device, the GPU. The host sends the data and issue commands to the device. The programmer has to write the code for the two devices. It have to program the CPU to send the requests to the GPU and also GPU using kernels, that will process the data in parallel among warps.

OpenCL It is a open standard for programming a heterogeneous collection of computing devices into a single language. It includes a framework for parallel programming composed of a programing language, API[10], libraries and a runtime system to support software development. With it a programmer can write a general purpose program that could execute on GPUs. It was developed to be portable, but true portability in terms of performance is yet to be achieved. Because it is a low level language, subtle differences on hardware architectures could mean great performance loss, and programs need to be

[10] Application Programming Interface.

fine-tuned to other architectures.

The OpenCL specification used on this work was 1.2. On Figure 3.2(a) is represented its architectural overview. To better understand it and learn how to use in our benefit we will start by introducing the next core models[Ea11]:

Platform Model On Figure 3.2(b) its an overview of OpenCL platform model. This model consists of a host connected to one or more OpenCL devices, and the computations on a device occur within the processing elements. The existing processing elements within a compute unit execute a single stream of instructions as SIMD units or as SPMD units.

Memory Model The OpenCL memory model is divided onto four distinct areas: *Global Memory*, *Constant Memory*, *Local Memory* and *Private Memory*. The Host and OpenCL device memory are independent of each other, and can interact by explicitly copying data or by mapping an unmapping regions of a memory object.

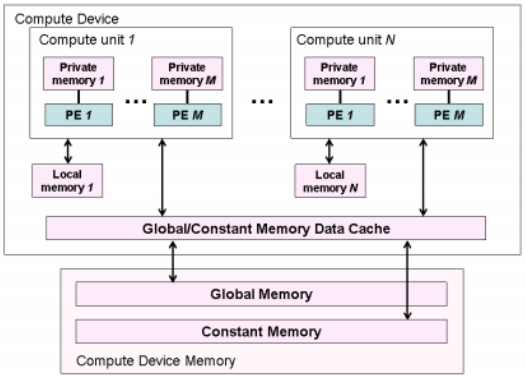
Execution Model The execution of an OpenCl program occurs at two distinct parts: on one side exists the host program that executes on the host and on the other side the kernels that execute on OpenCL devices. OpenCL explicitly support two programming models; the data parallel programming model and the task parallel programming model.

Programming Model Besides the early referred supported programming models OpenCL also supports hybrid models of them. Data parallelism can be achieved on a explicit or implicit way. On explicit the programmer defines the number of work-items and also how they are divided on work-groups, on implicit the work-items division into work-groups is managed by OpenCL. To achieve Task Parallelism a single instance of a kernel is executed independently. This allows programmers to express parallelism using vector data types and enqueuing multiple tasks.

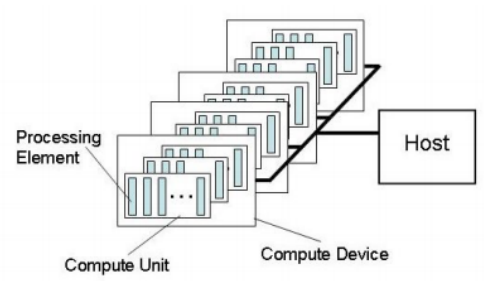
3.2.6.3 Conclusion

Whether we are targeting SoC's or a common desktop with CPU + GPU, having a programming model that could easily abstract all of the this hardware heterogeneity and allow programmers to better use the hardware at his disposal is crucial. As we have seen CUDA tries to address this problems on the NDIVIA hardware giving to programmers the tools needed to effectively achieve it, but it lacks in portability through diverse hardware. OpenCL[11] framework was released in 2008, it is an effort from several companies to create a open, royalty-free standard for cross-platform, parallel programming[Gro11b] model capable of running through heterogenous hardware composed of CPU's and GPU's and currently is being widely supported by hardware manufacturers.

[11] Open Computing Language.



(a) OpenCL Device Architecture



(b) OpenCL Platform Model

Figure 3.2: OpenCL Architectural Overview [Gro11a]

OpenCL is a heterogeneous framework capable of fully exploiting the power of the parallel architectures available on current hardware. It could be used to target any CPU, GPU, or other hardware that are compatible with OpenCL, so the programmer must be aware of the targeted hardware to better use it. SoC's or CPU+GPU desktops could both be used to obtain performance gains on compute intensive tasks, although they have different parallel capabilities and latencies between the device memory and the main memory.

Bom relativamente aos gpu's a questão é mais complicada. Moldar um pipeline de reconstrução à sua utilização numa placa gráfica é bastante complexo, no nosso caso não trouxe grandes vantagens. As placas graficas apresentam grandes vantagens mas também podem trazer dores de cabeça caso não sejam usadas correctamente. Em primeiro lugar temos a questão da espacialidade dos dados, os algoritmos que temos que podem correr na placa grafica se não beneficarem de estruturas de dados que lhes tragam essa localidade podem tornar-se mais lentos do que a sua execução no cpu. Outra questão importante é a questão da divergencia, sendo a placa gráfica composta por grupos de trabalhadores, esses mesmos trabalhadores devem todos executar o mesmo trabalho. Isso implica estrategias diferentes para a execução dos algoritmos vejamos o caso simples do calculo da bounding box de um objecto: No caso do cpu o algoritmo apenas mantém

um valor para o maximo e outro para o minimo e vai iterar sobre todos os valores presentes actualizando esse valor caso o valor actual seja maior ou menor que o maximo ou o minimo consoante se actualizar um valor ou outro. Para mapear o mesmo algoritmo para o gpu temos o problema como vamos mapear os dados? que threads recebem que valores? No caso mais simples em que cada um dos valores é mapeado a 1 thread como vai essa thread conseguir saber os valores dos vizinhos? O simples caso da comparacao do valor com um valor que venha de tras vai introduzir divergencia na execucao dentro das threads e atrasos no tempo de execução também. Apesar de os gpu's poderem ser usados para realizar computação GPGPU o mesmo não quer dizer que vamos apenas simplesmente colocar lá a informação e pedir para que seja obtida. Os algoritmos têm de ser adaptados, repensados, reformulados para que possam tirar todo o proveito de toda esta maquinaria e arquitectura subjacente. Se bem aproveitada todo este hardware pode tornar-se uma ferramenta bastante util para a resolução de problemas que de outra maneira seriam impossiveis de resolver ou não consigam ser resolvidas em tempo util.

3.2.7 Linear Algebra Libraries

The mathematical problems that are presented on this work are formulated as a linear problem and we use a solver[12] library to compute the solution and there are several available libraries. This topic is a topic of high research and exist several different algorithms to solve different problems. The matrices from the system can be dense of sparse and for different matrices different approaches may be applied. Under the open source libraries one can found the *ViennaCL*, *clMAGMA* and under commercial libraries one can found *Culatools*.

3.3 Proposed Solution

The proposed solution is composed of a *SCIRun* module that will receive the datasets at it's input, perform a reconstruction on it and retrieve the correct shape from the sampled objects, inserting onto the *Tomo-GPU* workflow a triangular mesh representing the shape from each of the reconstructed objects. This module will perform as a filter over the input data reconstructing the shape of the object to a shape implicitly defined by the dataset and retrieving a closed surface mesh for each object. With that it is possible to preview the surface or send it for other modules to extract the voxels from the object for example for latter classification.

We have choosed to use the *poisson reconstruction* technique in order to reconstruct the surface from each object. We had to choose a reconstruction among the ones studied, flavoring the reconstructions that are available as an open source software libraries so they could be used and changed if needed. The *poisson reconstruction* is a technique that fits the desired goals of this project but in order to be used with the received dataset there

[12] A solver is a generic term indicating a piece of mathematical software, that 'solves' a mathematical problem.

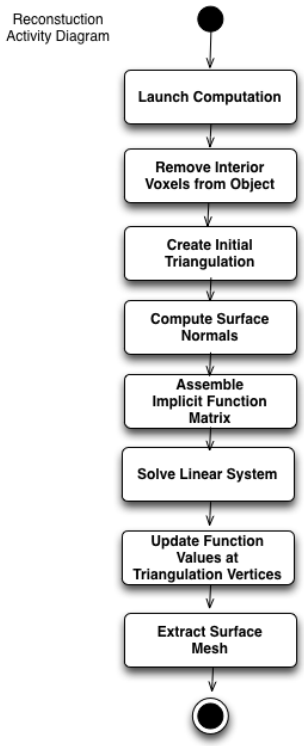


Figure 3.3: Activity diagram for the surface reconstruction pipeline.

are some requisites that must be computed such as the extraction of the oriented normal vector at each of the input points. We have used the reconstruction available on the *CGAL* and implemented the required algorithms so that the poisson reconstruction could be applied. Identifying the points on the surface of the object when needed and computing the normal vector from each of the points on the surface. The oriented points are then used to feed the reconstruction algorithms to extract the desired surface. This reconstruction pipeline have a high processing time and because of that we improve the achieved reconstruction by replacing some subsetps of the computation with others always with the objective of reduce the total reconstruction time.

3.3.1 Organization

All the logic to perform the reconstruction is under a reconstruction module, the structure for this module is similar to the lastly presented on chapter 2.3 and contain a reconstruction main file, a sample file, the reconstruction class and the reconstruction data. This module differs from the one for the characterization since it doesn't store the computed values to a database and, instead of that, for each object it sends the list of the triangles that compose the resulting mesh. The *GUI* have a boolean checkbox to allow the user to perform the extraction of the interior of the object prior the reconstruction.

Sample Class The sample class read all the data from the input containing all the objects and for each one of the them performs the reconstruction.

Reconstruction This class is implemented as a stateless box, it receives all the points and return the list of triangles that compose the object surface. It's composed of several steps and at its internal workflow emulates a pipeline where the data is passed from one operation to other until the resulting data is returned.

Remove Interior Voxels from Surface At this step depending upon the selection flag presented on the *GUI* it is computed which voxels belong to the surface or not. At the next steps of the reconstruction it is required to receive only the points from the surface as so the user have an option to enable or disable such computation.

Triangulation Initial triangulation of the extracted point set sampled on the object surface.

Normal estimation For each point of the surface, is computed a normal vector perpendicular to the tangent plane from the surface at that point based upon a number of neighbor points.

Linear System Assembly The matrices for the system are initialized from the points on the surface.

Linear System Solver The poisson function is solved for the surface points

Update Function Value at Triangulation Vertices Here for all the voxels from the triangulation it is computed the value of the function by interpolation over the obtained values at the surface points.

Isosurface Extraction An isosurface is extracted from the points of the surface using the poisson function

Geometric Object This GeometricObject is equal to the one used on characterization, it is used by the sample file and the reconstruction file to read the received point set for each object.

Data Reconstruction Class This file will hold the values from the reconstructions, each geometric object at the sample file have such object. Internally it stores the resulting mesh from the reconstruction, and a getter function to return the triangular mesh.

3.4 Implementation

All of the algorithms used on this module are derived from cgal libraries, and there interest on this project to use the poisson reconstruction because of all of its beneficts in approximating the plausible surface, as so, we have choosed to use this reconstruction technique

3.4.1 Remove Interior Voxels from Object

In a similar way to the presented early for the Objects it is constructed and AABB data structure to store all the points from the object. Here a tree data structure is build to extract to allow the efficient data query. In order to find the nearest neighbors

3.4.2 Create Initial Triangulation

3.4.3 Compute Surface Normals

We estimate the normal direction of each point from an object surface upon their neighbors, since the data that is present to reconstruction contains errors, it is important to have a good number of neighbors for the normal estimation on each point. We used thirty two neighbor points to estimate the normal at a given point on the surface points. Internally *CGAL* presents two different algorithms on the Point Set Processing package to extract the normal vectors from points either through *PCA* or through *Jet Fitting*. As seen previously *PCA* computes the least linear fitting of the data to a plane and the orthogonal vector of the plane on the specified point is retrieved, on *Jet-Fitting* instead of computing the plane that best approximates the neighborwod of p it tries fit an jet surface over the neighbors. The retrieved neighbors from each point are retrieved by the k-nearest neighbor search on the kd-tree that will retrieve all the neighbors on the 3-dimensional euclidean space, and mainly for non-convex surfaces, that doesn't mean that the points lies on the same patch or neighbor patches of the point patch on the surface. The second package to estimate normals from point sets presented on *CGAL* is the *Jet-Fitting* that tries to fit an

Even using a large number of points to extract the normals at the surface points those normals where incorrectly classified either using the geometrical neighbors or their natural neighbors and some normals where pointing to the interior of the object. Since the normals from each point play a central role on the used reconstruction, we introduce a simple pos-processing step that after the normals are estimated each one of them are tested against the interior of the object to see if they point to inside or not, if so we use the symmetric normal from the one computed previously. With this simple step it was possible to correctly orient the computed normals and achieve a better reconstruction.

Neighbor Selection We have choosed to use the natural neighbors provided by the delaunay triangulation on the *CGAL* package

Normal Orientation In a similar way to what is proposed on

3.4.4 Poisson Reconstruction

After the initial triangulation and with the oriented points on the surface

3.4.4.1 Assemble Implicit Function Matrix Coeficients

3.4.4.2 Solve Linear System

The *CGAL* could use the *Eigen* or the *Taucs* library

3.4.4.3 Update Function Values at Triangulation Vertices

3.4.5 Extract Surface Mesh

At this stage we have at each vertex of the triangulation the values for F and we wish to extract the a closed triangular mesh for a specific isovalue for f.

3.4.6 Tests

The framework have been tested for each one of the required computations, and the results are presented here. All the tests that are presented on this section where produced on a desktop computer with the characteristics presented next:

CPU Quad-core Intel Xeon E5506 @2.13 GHz

Memory 12 GB RAM DDR-3

SO Linux Ubuntu 10.04.4 LTS (kernel 2.6.32-41)

Next we present the reconstruction times for one object, and for one complete sample set. We measured two object having 5k points one and 30k points the other. We also show the measures for a complete sample, composed of six objects.

Submodule	Time (s) 5K	Time (s) 30K	Time (s) sample
Surface Points Extraction	0.22	0.36	1.72
Points Normal Extraction	0.34	0.86	3.56
Poisson Solver	1.68	3.2	14.50
Surface Meshing	0.76	0.79	4.75
Total Time	3	5.21	24.39

As one could easily see the most of the algorithms are time consuming being the most expensive step at solving the poisson system. The number of objects on a sample varies from sample to sample and the one presented here have six particles.

3.5 SCIRun Integration

The module is implemented following the notation on section 2.5, the structure for the module is the same than the previous one, the main difference is that this module have

two output ports, one returning the complete point set for this object and other to return the triangulated mesh from the reconstruction.

3.6 Optimizing Solution

As seen previously the achieved solution as presented couldn't achieve interactivity and for fully reconstruct all objects on a sample it could take several seconds. One way to improve the computation times from the used algorithms is to launch several computations at the same time. In the same way as presented on the first chapter we work with a *multi-core* processor and there is space to achieve higher utilization ratios using parallelization.

We will start to provide a parallelization on the surface extraction for all the objects, and lastly we will aboard the *gpgpu* parallelization.

3.6.1 A - Multi-Core Approach

The simplest form of parallelization that have been employed is the parallelization of all the objects on the sample. Since all the reconstructions that are endorsed over the received points are independent in the data and in the functionality, it was possible to attempt the parallelization of each object. By using *OpenMP* one could easily launch all the reconstructions in parallel taking advantage from the multi-core cpu available on the underlying hardware. Each one of the subsets on the reconstruction is data dependent over the next one, so it isn't possible to parallelize the three operations.

In the same way that we have proceeded on the second chapter of this work, the first efforts for using parallelization was made using *OpenMP* and since all the framework where already implemented to launch the reconstructions we have launched all the reconstructions in parallel using *OpenMP* as in Chapter 2. There we have seen that a good number to be used on the tested machine was four or eight threads simultaneously.

3.6.1.1 Tests

3.6.1.2 Conclusion

This tests show that the applied parallelization have improved the solution, reducing the total time from all the reconstructions to about one fourth of the previous total time. It is a good improvement but to provide an interactive response time for the reconstructions a different approach must be used. By looking at the individual times of each object reconstruction we can easily see that it's time doesn't fit for the desired goals.

3.6.2 B - CPU-GPU Approach

3.6.2.1 Tests

..

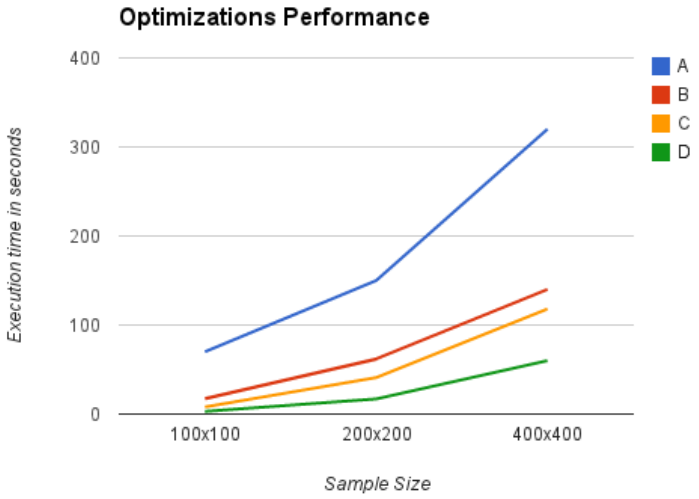


Figure 3.4: GPU with n MIMD multiprocessors and n x m SIMD processors.

3.6.2.2 Conclusion

This tests show that the applied parallelization have improved the solution, reducing the total time from all the reconstructions to about one fourth of the previous total time. It is a good improvement but to provide an interactive response time for the reconstructions a different approach must be used. By looking at the individual times of each object reconstruction we can easily see that it's time doesn't fit for the desired goals.

3.6.3 C - Meshing Algorithm Replacement

3.6.3.1 Tests

3.6.3.2 Conclusion

3.7 Conclusion

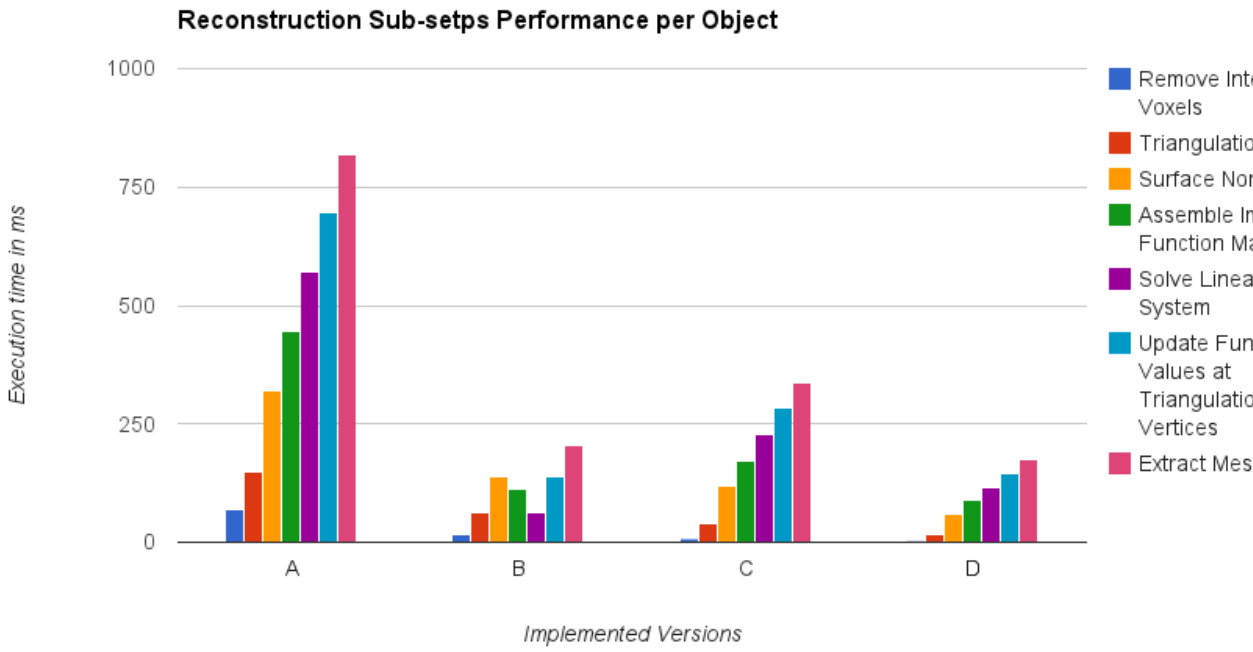


Figure 3.5: GPU with n MIMD multiprocessors and n x m SIMD processors.

4

Conclusions

4.1 Work evaluation

Regarding the objectives stated in the 1st chapter all of them where achieved. It was possible to produce two *SCIRun* modules that are integrated in the *TomoGPU* project providing the required framework.

- It have been an characterization module that is modular in the sense that could be expanded with more characterizations easily. This module is integrated on the *Tomo-GPU* project and is capable of perform the characterizations and store their values on the underlying database layer.
- Under the reconstruction module we have implemented a reconstruction pipeline that is capable of reconstruct and extract a triangular mesh from the object surface sending it to the *Tomo-GPU* workflow for posterior analysis or visualization.

4.2 Future work

There is space for enhancements in both modules:

- Regarding the characterization module
  - more characteristics could be included, according to the wishes of Materials specialists
  - for some more complex characterizations that could be performed it might be interesting to define some kind of dependency graph in order to introduce the definition of dependent characterizations, that way the start of a

characterization might be dependent of previous characterizations. Providing that way more complex characterizations that are composed of several sub-characterizations and that way avoid to replicate the computations of the previous characterizations. For a simple example we can think on the definition of the *OBB* where the computation is dependent of the computation of the *PCA* to extract the principal vectors from the object. By assembling such dependency graph we could launch the two computations independently in parallel and the *OBB* computation will be launched after the *PCA* computation returned avoiding that way to recompute the *PCA* to perform the *OBB* computation. On our implementation the times for all the characterizations are reduced and we didn't find the need to improve the solution with that, anyway this could be an interesting feature to improve the overall framework.

- Regarding the surface reconstruction module
  - more phases of the surface reconstruction process could be offloaded to the *GPU* to improve the overall performance. Under this topic there are several ways to improve the solution. Either by the data structures for the spatial decomposition of the dataset, and for the extraction of the final mesh.
  - alternative methods for surface reconstructions could be implemented; the analysis made in section 3.2 concluded that further research is needed to evaluate if the Poisson method is the most adequate for different types of samples with different number of objects and where the number of voxels composing the skin can also change.

Bibliography

[ABCOFLS03] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. Silva. "Computing and rendering point set surfaces". English. In: *IEEE Transactions on Visualization and Computer Graphics* 9.1 (Jan. 2003), pp. 3–15. ISSN: 1077-2626. DOI: 10.1109/TVCG.2003.1175093. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1175093'escapeXml='false'/>.

[APG12] P. Alliez, S. Pion, and A. Gupta. "Principal Component Analysis". In: *CGAL User and Reference Manual*. 4.1. CGAL Editorial Board, 2012.

[BDTY00] J.-D. Boissonnat, O. Devillers, M. Teillaud, and M. Yvinec. "Triangulations in CGAL (extended abstract)". In: *Proceedings of the sixteenth annual symposium on Computational geometry - SCG '00*. New York, New York, USA: ACM Press, May 2000, pp. 11–18. ISBN: 1581132247. DOI: 10.1145/336154.336165. URL: http://dl.acm.org/citation.cfm?id=336154.336165.

[BF02] J.-D. Boissonnat and J. Flototto. "A local coordinate system on a surface". In: *Proceedings of the seventh ACM symposium on Solid modeling and applications - SMA '02*. New York, New York, USA: ACM Press, June 2002, p. 116. ISBN: 1581135068. DOI: 10.1145/566282.566302. URL: http://dl.acm.org/citation.cfm?id=566282.566302.

[Bon02] P. A. Boncz. "Monet: A {Next-Generation} Database Kernel For {Query-Intensive} Applications". PhD thesis. Universiteit van Amsterdam, 2002. URL: http://oai.cwi.nl/oai/asset/14832/14832A.pdf.

[CFB97] J. C. Carr, W. R. Fright, and R. K. Beatson. "Surface interpolation with radial basis functions for medical imaging." In: *IEEE transactions on medical imaging* 16.1 (Mar. 1997), pp. 96–107. ISSN: 0278-0062. DOI: 10.1109/42.552059. URL: http://www.ncbi.nlm.nih.gov/pubmed/9050412.

[Cha93] B. Chazelle. "An optimal convex hull algorithm in any fixed dimension". In: *Discrete & Computational Geometry* 10.1 (Dec. 1993), pp. 377–409. ISSN: 0179-5376. DOI: 10.1007/BF02573985. URL: http://link.springer.com/10.1007/BF02573985.

[Ea11] A. M. Et al. *{OpenCL} programming guide*. Addison-Wesley, 2011.

[Ea12] A. V. Et al. "A {Problem-Solving} Environment for reinforcement distribution characterization in composites using tomographic images". In: *1st Meeting of Synchrotron Radiation Users from Portugal*. 2012.

[FGKSS98] A. Fabri, G.-j. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. "On the design of {CGAL} a computational geometry algorithms library". In: *Softw. – Pract. Exp* 30 (1998), p. 2000.

[Fra] T. U. G. Franz Aurenhammer. "Voronoi Diagrams". In: (). URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.61.7055.

[Gla09] P. Glaskowsky. *White Paper {NVIDIA's} Fermi*. White Paper. NVIDIA, 2009. URL: http://www.nvidia.com/content/PDF/fermi_white_papers/P.Glaskowsky_NVIDIA's_Fermi-The_First_Complete_GPU_Architecture.pdf.

[GVJWG09] A. Gomes, I. Voiculescu, J. Jorge, B. Wyvill, and C. Galbraith. *Implicit Curves and Surfaces: Mathematics, Data Structures and Algorithms*. Springer, 2009, p. 366. ISBN: 184882405X. URL: http://www.amazon.com/Implicit-Curves-Surfaces-Mathematics-Structures/dp/184882405X.

[Gro11a] K. Group. *{OpenCL} 1.2 Specification*. Specification. Khronos Group, 2011.

[Gro11b] K. Group. *{OpenCL} - The open standard for parallel programming of heterogeneous systems*. http://www.khronos.org/opencl/. 2011. URL: http://www.khronos.org/opencl/.

[GAP08] A. Gupta, P. Alliez, and S. Pion. *Principal Component Analysis in {CGAL}*. Rapport de recherche RR-6642. INRIA, 2008, p. 13. URL: http://hal.inria.fr/inria-00327027.

[Har71] R. L. Hardy. "Multiquadric equations of topography and other irregular surfaces". In: *Journal of Geophysical Research* 76.8 (Mar. 1971), pp. 1905–1915. ISSN: 01480227. DOI: 10.1029/JB076i008p01905. URL: http://doi.wiley.com/10.1029/JB076i008p01905.

[Ins13] S. C. I. Institute. *SCIRun Web Page*. 2013. URL: http://www.sci.utah.edu/cibc-software/scirun.html.

[Kaz05] M. Kazhdan. "Reconstruction of solid models from oriented point sets". In: (July 2005), p. 73. URL: http://dl.acm.org/citation.cfm?id=1281920.1281931.

[KBH06] M. Kazhdan, M. Bolitho, and H. Hoppe. "Poisson surface reconstruction". In: (June 2006), pp. 61–70. URL: http://dl.acm.org/citation.cfm?id=1281957.1281965.

[KBSS01] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. "Feature sensitive surface extraction from volume data". In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH '01*. New York, New York, USA: ACM Press, Aug. 2001, pp. 57–66. ISBN: 158113374X. DOI: 10.1145/383259.383265. URL: http://dl.acm.org/citation.cfm?id=383259.383265.

[LCYL13] S. Lee, H. Cho, K.-J. Yoon, and J. Lee, eds. *Intelligent Autonomous Systems 12*. Vol. 194. Advances in Intelligent Systems and Computing. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. ISBN: 978-3-642-33931-8. DOI: 10.1007/978-3-642-33932-5. URL: http://www.springerlink.com/index/10.1007/978-3-642-33932-5.

[MAT] K. R. MATTHEWS. *ELEMENTARY LINEAR ALGEBRA*. Tech. rep. ELEMENTARY LINEAR ALGEBRA K. R. MATTHEWS DEPARTMENT OF MATHEMATICS UNIVERSITY OF QUEENSLAND.

[NVI10] NVIDIA. *{PTX:} Parallel Thread Execution {ISA} Version 2.0*. Tech. rep. NVIDIA, 2010.

[NVI11] NVIDIA. *{CUDA} C Programming Guide*. Tech. rep. NVIDIA, 2011.

[OO85] Org.cambridge.ebooks.online.book.Author@7203cf70 and Org.cambridge.ebooks.onlin *Algebra through practice*. Ed. by T. S. Blyth and E. F. Robertson. Vol. 4. Cambridge: Cambridge University Press, 1985. ISBN: 9780511600616. DOI: 10.1017/CBO9780511600616. URL: /ebook.jsf?bid=CBO9780511600616.

[OLGHKLP07] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. Lefohn, and T. J. Purcell. "A Survey of {General-Purpose} Computation on Graphics Hardware". In: *Computer Graphics Forum* 26.1 (2007), pp. 80–113. URL: http://www.blackwell-synergy.com/doi/pdf/10.1111/j.1467-8659.2007.01012.x.

[PJ95] S. G. Parker and C. R. Johnson. "{SCIRun:} a scientific programming environment for computational steering". In: *Proceedings of the 1995 {ACM/IEEE} conference on Supercomputing {(CDROM)}*. Supercomputing '95. New York, {NY}, {USA}: 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.

[Pau] U. T. M. Paulo Roma Cavalcanti. "Three-Dimensional Constrained Delaunay Triangulation: A Minimalist Approach". In: (). URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.6295.

[Pla11] H. Plattner. "{SanssouciDB:} An {In-Memory} Database for Processing Enterprise Workloads". In: *BTW*. 2011, pp. 2–21.

[SPOK95] V. V. Savchenko, A. A. Pasko, O. G. Okunev, and T. L. Kunii. "Function Representation of Solids Reconstructed from Scattered Surface Points and Contours". In: *Computer Graphics Forum* 14.4 (Oct. 1995), pp. 181–188. ISSN: 0167-7055. DOI: 10.1111/1467-8659.1440181. URL: http://doi.wiley.com/10.1111/1467-8659.1440181.

[SE03] P. J. Schneider and D. H. Eberly. *Geometric Tools for Computer Graphics*. Morgan Kaufmann Publishers, 2003.

[SHB07] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision*. CL Engineering, 2007, p. 872. ISBN: 049508252X. URL: http://www.amazon.com/Image-Processing-Analysis-Machine-Vision/dp/049508252X.

[Tau91] G. Taubin. "Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with applications to edge and range image segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.11 (1991), pp. 1115–1138. ISSN: 01628828. DOI: 10.1109/34.103273. URL: http://www.computer.org/csdl/trans/tp/1991/11/i1115-abs.html.

[TO99] G. Turk and J. F. O'Brien. "Shape transformation using variational implicit functions". In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH '99*. New York, New York, USA: ACM Press, July 1999, pp. 335–342. ISBN: 0201485605. DOI: 10.1145/311535.311580. URL: http://dl.acm.org/citation.cfm?id=311535.311580.

[Wen10] H. Wendland. *Scattered Data Approximation (Cambridge Monographs on Applied and Computational Mathematics)*. Cambridge University Press, 2010, p. 348. ISBN: 0521131014. URL: http://www.amazon.com/Scattered-Approximation-Monographs-Computational-Mathematics/dp/0521131014.

[Wik12] Wikipedia. *Superscalar - Wikipedia, the free encyclopedia*. http://en.wikipedia.org/wiki/Supers 2012. URL: http://en.wikipedia.org/wiki/Superscalar.

[ZPBG01] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. “Surface splatting”. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH '01*. New York, New York, USA: ACM Press, Aug. 2001, pp. 371–378. ISBN: 158113374X. DOI: 10.1145/383259.383300. URL: http://dl.acm.org/citation.cfm?id=383259.383300.

Mathematical Foundations

We start to introduce some concepts that will be widely used under this work. We do not intend to exhaustively explain all this mathematical foundations, the interested reader should look at specific books and texts about those concepts, [MAT] or [OO85] are two good examples.

A.1 Linear Algebra and Matrices

Under this scope the reader should also be familiarized with vector spaces, linear transformations and linear subspaces. Linear Algebra is a branch of mathematics that covers vector spaces and their mappings. It have its focus on systems of linear equations that are represented with matrices and vectors. Giving us the tools to solve systems of equations that are assembled using those matrices and vectors. Such tools are central to several branches of mathematics and physics. They are used for example on analytic geometry, computer science, economics and many others.

A.1.1 System of Linear Equations and Matrices

A system of m linear equation in n unknowns $x_1, x_2, ..., x_n$ is a family of linear equations

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$

$$\vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m.$$

Usually one is interested to determine if such system has a solution, which satisfy all the equations simultaneously. This system can be written concisely as

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \;\; i = 1, 2, \cdots, m.$$

In order to ease the determination of such systems they can be written in a matrix equation as $Ax = b$ with

$$A = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, x = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \text{ and } b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

The geometric interpretation resulting from the solving a system of linear equations in two or three unknowns is equivalent to determine a family of lines or planes as a point of intersection.

Determinant On this work will be defined as $det(A)$ and by definition is a value associated to a square matrix A, that when used over the coefficients of a system of linear equations could tell us the system has a unique solution if its value is different than zero. The general notation over an $n \times n$ matrix is

$$det(A) = \sum_{\delta \in S_n} sgn(\delta) \prod A_i, \delta_i$$

A.1.2 Eigenvalues and Eigenvectors

Can only be found on square matrices. Not every square matrix has and eigen-vector, and given an $n \times n$ matrix having eigen-vectors, there are only n eigenvalues and eigenvectors that can be computed, having each eigen-value one eigen-vector associated. Given such square matrix A , an eigen value λ and its associated eigenvector v are a pair obeying the relation (A.1.2), with k variables

$$(A - \lambda I)^k v = 0$$

The eigenvectors are orthogonal, and those vectors if used as the basis vectors allow to express the data in terms of these orthogonal vectors.

A.1.3 Solving Systems of Linear Equations

Matrix Factorization It is a technique to decompose a matrix into a product of matrices in a way that the two are equivalent. There are several ways to factorize a matrix such as LU or QR factorization. Those are techniques for dividing a matrix A onto two or more matrices.

Eigen Analysis Eigenvalues are important in the analysis of the convergence characteristics of iterative methods for solving linear systems and as so they are very important in many areas of science and engineering.

On this work for some analysis we will try to represent data in a way such that the mutual independence between components may be exposed. Linear algebra will give us the right tools for such representation, the data is represented on a linear subset that will have some natural basis vectors allowing the data to be expressed as a linear combination on another coordinate system consisting of orthogonal basis vectors. These basis vectors are the *eigen vectors* and the inherent orthogonality of the *eigen* vectors assures the mutual independence.[SHB07] For an $n \times n$ square regular matrix A, eigen vectors are solution of the equation

$$Ax = \lambda x,$$

where λ is called and *eigen value*. A system of linear equations may be expressed in a matrix form as $Ax = b$, where A is the matrix of the system.

Any monic polynomial is the characteristic polynomial of some matrix, and as so, there are algorithms for finding the eigenvalues that could also be used to find the roots of the polynomials.

Singular value decomposition It is a generalization of the definition A.1.3, on regular matrices. It allows the factorization of a real matrix, where a non-negative real number σ is a singular value of a matrix A if and only if it exists unit-length vectors u and v such that

$$Av = \sigma u \text{ and } A^*u = \sigma v.$$

The vectors u and v are called left-singular and right-singular vectors for σ.[SHB07] The definition for *SVD* is achieved by noting that any $m \times n$ matrix A, $m \geq n$, can be decomposed into a product of three matrices,

$$A = UDV^T,$$

where U is an $n \times n$ orthonormal columns, D is a non-negative diagonal matrix, and V^T has orthonormal rows.

SVD has many practical uses such as resolving the least squares fitting of data or solving homogeneous linear equations.

Jacobi eigenvalue algorithm It is an iterative algorithm for the calculation of the *eigenvectors* and *eigenvalues* of a real symmetric matrix. It has many practical uses such as the computation of the singular value decomposition from a covariance matrix of a *three-dimensional* dataset. On this work is used to extract the principal components for the dataset.

A.2 Statistics

Under the statistical concepts we will use the

Covariance Is a statistical analysis tool measured among two variables, representing a measure of how two variables varies together.

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{n}$$

Being $\bar{X}$ the mean value for the X variable, one should also be aware that $cov(X,Y) = cov(Y,X)$ and $cov(X,X) = var(X)$.

It is common to represent the covariance in a matricidal form and for a *3-dimensional* data set the covariance matrix is 3×3. This is matrix is a symmetric matrix, as so, one should compute the values for the $cov(X,X)$, $cov(Y,Y)$, $cov(Z,Z)$, $cov(X,Y)$, $cov(X,Z)$ and the $cov(Y,Z)$ having with that all the possible covariance values on the data set.

$$C = \begin{bmatrix} cov(X,X) & cov(X,Y) & cov(X,Z) \\ cov(Y,X) & cov(Y,Y) & cov(Y,Z) \\ cov(Z,X) & cov(Z,Y) & cov(Z,Z) \end{bmatrix}$$

For a n-dimensional data set the matrix have $n \times n$ and is $C^{n\times n} = (C_{i,j}, C_{i,j} = cov(Dim_i, Dim_j))$, where $C^{n\times n}$ is a matrix with n rows and n columns, and the Dim_x is the x-th dimension.

A.3 Geometric and Analytical Measures

A.3.1 Mathematical Foundations

Besides some matricial algebra introduced early on Section 2.2, one should be familiar with the next mathematical concepts.

Surface Representation In mathematical foundation a generic manifold on dimension n, is a topological space that near each sampled point resembles n-dimensional Euclidean space. Ideally it can be seen as the decomposition of a set in several pieces of the same kind, in a way that they will fit together. An surface for example, is an 2-dimensional manifold.

A.3.2 Distance Metrics

Euclidean Distance Given two points $p = \{p_x, p_y, p_z\}$ and $q = \{q_x, q_y, q_z\}$ sampled from a surface S, one could define the *Euclidean distance* between these two points as:

$$\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2}$$

the distance among one point $p \in S$, is the defined as the distance between the point and the closest point of the surface from p as a minimization problem as:

$$\min_{q \in S} \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2}$$

This is an expensive operation, since it needs to iterate among all the points on the surface, this kind of computation has some drawbacks, since it works with absolute distances, it doesn't take into account the information from the surface.

Geodesical Distance For two points p, q sampled on the surface M the geodesical distance can be seen as the length of the shortest path from p to q on the surface M.

Algebraic Distance It is commonly used for the computation of the distance on the implicit surface reconstruction because of its simple form. For a given surface S, defined by $f(x, y, z) = 0$, one could define the *algebraic distance* of a point p to the surface S as,

$$f(p) = d(p, s)$$

conceptually the algebraic distance could be seen as the same that the euclidean distance for surfaces but on some cases could have significant error. This error could be avoided by using a variant of this formula called presented by Taubin on [Tau91], where the algebraic distance is divided by it's gradient, such as:

$$d(s, p) = \frac{f(p)}{\|\nabla(p)\|}.$$

Distance Field representation For a given surface $S \subset \mathbb{R}^3$ a volume representation consists of a scalar valued function $f : \mathbb{R}^3 \to \mathbb{R}$ such that

$$[x, y, z] \in S \longleftrightarrow f(x, y, z) = 0.$$

Assuming that f is a continuous function, it isn't uniquely defined for a given surface S, a common choice tends to use the signed distance field function which assigns for every point $[x, y, x] \in \mathbb{R}^3$ its distance

$$f(x, y, z) := dist([x, y, z], S])$$

with a positive sign for points outside the region enclosed by S and a negative sign for points inside S. By using this notation many operations are quite efficiently implemented, and the standard way to store the distance field f for a surface S is to sample f in an uniform spatial grid $g_{i,j,k} = f[ih, jh, kh]$, and the sampled distances $d_{i,j,k} = f(ih, jh, kh)$ can be interpolated on each grid cell such as

$$C_{i,j,k}(h) = [ih, (i+1)h] \times [jh, (j+1)h] \times [kh, (k+1)h]$$

by a tri-linear function obtaining a piecewise tri-linear approximation f^* to the original distance field f and a corresponding surface S^* defined by $f^*(x, y, z) = 0$ which approximates S.[KBSS01]

The major drawback from this computation is that samples S^* aren't close to the surface S on the neighbor of sharp features. To avoid this is proposed in [KBSS01], a different discretization of the distance field called *directed distance field*, that based on the fact that the Marching Cubes algorithm only computes surface samples on cell edges, as so, it is not necessary to generate a continuos function f^* which approximates f in the interior of the cells. For each grid point $g_{i,j,k}$ three directed distances, instead of the scalar valued distances $d_{i,j,k}$, i.e.,

$$C = \begin{pmatrix} dist_x \\ dist_y \\ dist_z \end{pmatrix}$$

The processing of this directed distances is identical to the scalar distances, and although storing the directed distances $d_{i,j,k}$ increases the memory consumption by a factor of three, they give the advantage that the sample points lying exactly on the surface S are available for later isosurface extraction, improving the quality of the reconstruction around sharp features.

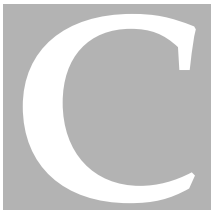
B

SciRun Integration

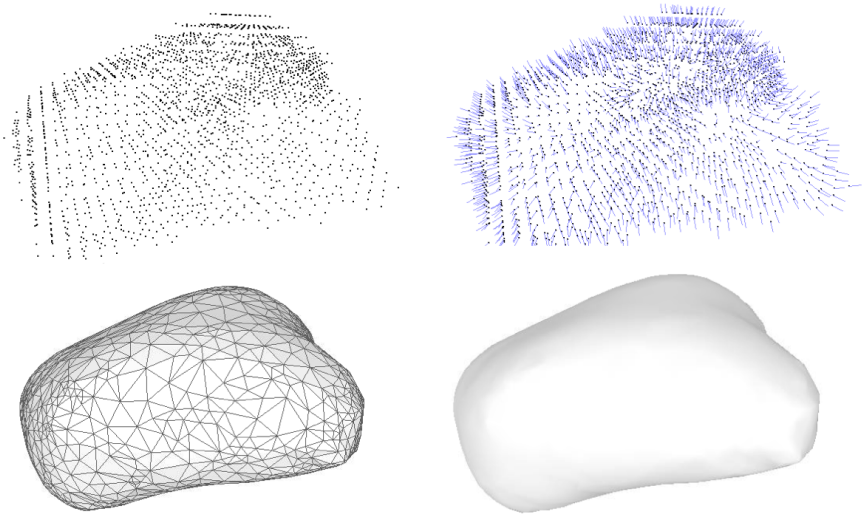
Here we show how the implementation of the *SCIRun* integration may be achieved.

```
1
2  namespace Tomo {
3    using namespace SCIRun;
4
5    class ObjectCharacterization : public Module {
6      public:
7        ObjectCharacterization(GuiContext *context);
8        virtual ~ObjectCharacterization();
9    };
10
11 // ....
12   void ObjectCharacterization::execute()
13   {
14     FieldHandle input, output;
15     MeshHandle inputmesh, meshout;
16
17     // Variable initialization //
18     int* matrix;
19     int* matrix_out;
20 // ....
21         if(get_input_handle("Input",input,true))
22     {
23       FieldInformation fi(input);
24       VMesh*  inputmesh  = input->vmesh();
25       VMesh::dimension_type dims;
26       inputmesh->get_dimensions(dims);
27       int size = dims[0];
28
```

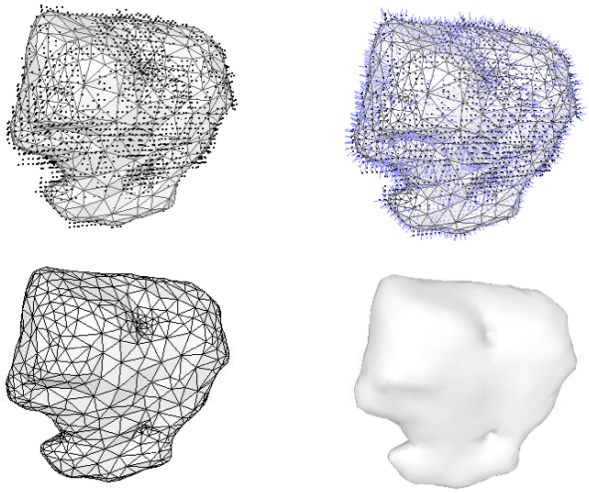
```
29      // matrix containing all the data in the described format
30      matrix = (int*)input->vfield()->get_values_pointer();
31
32      update_state(Executing);
33
34      //  ... Proccess the data from the dataset
35      //      and write data to output
36
37      send_output_handle("Output",  output);
38      send_output_handle("Output2", output2);
39    }
```



Reconstruction Examples



(a) Reconstruction from an identified object with 5k points



(b) Reconstruction from an identified object with 30k points

Figure C.1: Reconstruction of two different objects