INTERNSHIP REPORT

May 16th to August 5th 2011

A library to solve partial differential equations with the finite element method

Baptiste MORIN
RICM4

Internship master :	Christophe PRUD'HOMME
Internship supervisor :	Stéphane VEYS
Internship tutor :	Dominique DUVAL

Laboratoire Jean Kuntzmann
51 rue des Mathématiques
38400 - SAINT-MARTIN D'HÈRES

Contents

Abstract

Within the scope of the computing science studies I currently follow at Polytech Grenoble, I carried out a twelve weeks long internship in the Laboratory Jean Kuntzmann (LJK), which offers me the possibility to combine both Information Technologies (IT) and mathematics strengths. This report shows the work done during this internship, detailling technologies I learned or improved as well as the methods used to carry out the mission that was entrusted to me : to create an application which can analyse the temperature variations of a heat sink, depending on its dimensions, materials and components it is made of.

To perform such a task, I developed an application which enables a total parametrization of CPU's heat sinks and also worked on the FEEL++ library, a C++ library for solving partial differential equations using the finite element method. This method is explained further in this document with a student point of vue, which has the advantage to go to the basics and gives the key to the understanding of the chosen solutions.

Résumé

Dans le cadre de ma formation d'ingénieur RICM à Polytech Grenoble, j'ai effectué un stage de douze semaines au sein du laboratoire Jean Kuntzmann à Grenoble. Ce rapport rend compte des différents travaux auxquels j'ai pu participer pendant cette période. Tout d'abord, le large éventail de compétences que propose le LJK m'a tout de suite attiré car je savais que j'y aurais la possibilité de combiner les sciences de l'informatique avec la puissance des mathématiques. J'ai notamment travaillé sur la librairie FEEL++ qui est une librairie développée en C++ qui permet de résoudre les équations aux dérivées partielles par la méthode des éléments finis. Cette méthode est, entre les autres outils présentés ici, abordée avec le regard d'un étudiant, ce qui a l'avantage d'aller directement aux fondamentaux et de donner les clés pour la bonne compréhension des problèmes rencontrés.

Outre les différents travaux sur la documentation et le tutoriel, j'ai été ammené à réaliser une application complète. Pour ce faire, j'ai mis en place une application qui a pour but de simuler la dissipation de chaleur entre microprocesseurs et dissipateur de chaleur. Ce simulateur permet donc une paramétrisation complète du problème, vous pourrez analyser les variations de température avec différents matériaux, ventilateurs, dimensions ou flux de chaleurs imposés. En plus de la refonte partielle de la documentation distribuée avec la librairie, j'ai également apporté quelques améliorations quant à la prise en compte des différents formats pour les maillages sur lesquels les calculs sont effectués.

Ce rapport montre le travail réalisé durant mon stage, en détaillant les technologies que j'ai pu apprendre ou améliorer mais aussi les méthodes employées pour mener à bien la mission qui m'a été confiée.

Acknowledgment

I would especially thank Mr Prud'homme, my internship master, who gave me the opportunity to make my 2nd year ingeneer internship at Laboratory Jean Kuntzmann. Altought mainly based in Strasbourg while the internship took place in Grenoble, he has been able to direct me properly and most of all succeed in trusting me. Despite his heavy schedule, my questions would be answered within a short time. His management was exemplary and has greatly contributed to the smooth functioning of my internship.

I also would like to thank Mr Veys, my internship supervisor and day-to-day interlocutor. He has always been available even if he is currently carrying out a thesis on multiscale methods. He has known how to help me and answered all the questions I could have have about mathematical theories.

I would thank Mrs Duval who was able to ensure the smooth running of my internship.

Finally, I would thank the Joseph Fourier University, Polytech Grenoble and the RICM department to make my internship at Jean Kuntzmann laboratory possible.

CHAPTER 1

The laboratory and the project

1.1 Introduction

The Laboratory Jean Kuntzmann is named after Jean Kuntzmann (1912-1992), a precusor in applied mathematics and informatics in Grenoble. Established in the city in January 2007, the LJK is an Applied Mathematics and Computer Science laboratory. Joint research unit of the UJF, the INPG, the UPMF, the CNRS and the INRIA, it combines the strengths of applied mathematicians and statisticians from the former laboratories LMC and LabSAD with graphics and computer vision experts from the former laboratory GRAVIR. Its expertise is focus on the computational and statistical sciences and their uses in analysing natural phenomena, with applications ranging from environmental modelling to mathematical finance.

1.2 Departments

The laboratory is structured into three scientific departments, each containing a number of research teams :

- Geometry and Images
  The Geometry-Image department conducts research in Geometric Design, Image Analysis, Computer Graphics and Computer Vision. The common framework of the research is the computer processing of geometry and images. Applications include Computer Aided Design systems for the manufacturing industry, the creation of animation movies for the leisure industry, or the indexation and mining of large image databases for the Information and Communication Technologies. This rare combination of computer sciences expertise in image synthesis and analysis, vision and geometry is the ideal blending for the development of innovative research towards a complete insertion of 3D geometry and images in the Information Society.

- Probability and Statistics
  The activities of the Department of Probability and Statistics are centered around probability, statistics, financial mathematics and image and signal processing. The Department is constituted of six research teams.

- Deterministic Models and Algorithms
  The activities of the MAD members are centered around the design and implementation of numerical and symbolic tools for the resolution of ODE's (Ordinary Differential Equations) and PDE's (Partial Differential Equations), with applications to control and optimization problems. I was attached to the PDE team regarding the goal of the library I have worked on which is detailed below.

1.3 The FEEL++ library

1.3.1 Points of interest

FEEL++ (formerly known as Life) is a C++ library for the Finite Element Method and is actually suitable for generalized Galerkin methods in 1D, 2D and 3D. The library supports nodal and modal basis at any order in the three dimensions.

The library is the fruit of a collaboration between three universities which are Université Joseph Fourier (Grenoble, France), Ecole Polytechnique Fédérale de Lausanne (Switzerland) and University of Coimbra (Portugal). It is a free software and is distributed under GNU General Public License (GPL).

FEEL++ is currently supported by two ANR (French National Research Agency) and one FNRAE (Research Foundation for Aviation and Space) projects. It is also supported by the region Rhônes-Alpes thanks to the cluster ISLE and CHPID project since 2009.

1.3.2 Two aspects

While I was seeking for an internship, the combination of two main aspects of FEEL++ have motivated my choice : mathematics and computer sciences skills I would have to improve in order to reach out the aim of this internship. This is a positive opportunity because it represents two of the essential skills that have to be part of an engineer's spectrum.

The mathematical point is obvious because it's the main goal of FEEL++ but solving such big equations requires powerful computers. What is interesting is that FEEL++ is not specifically intended to clusters or calculators but also to simple users who own personal computers.

To make it possible, the library takes advantage of C++ language : the developers could build the library with powerful tools such as BOOST, PETSC or SLEPC which enable a huge backend for huge memory needs or enormous mathematical calculations. Here is a short description of these powerful tools used by FEEL++.

BOOST
This important library provides free peer-reviewed portable C++ source libraries. The BOOST libraries aimed a wide range of C++ users and applications domain. They range from general purpose libraries to operating system abstractions. BOOST make extensive use of templates, it has been a source of extensive work and research into generic programming and metaprogramming in C++ . We use in FEEL++ many of its libraries such as `shared_ptr`, `numeric` or `timer`. With the `shared_ptr` library, each entity (such as matrix, vector, mesh, and so on) can be accessed thanks to a shared pointer. This structure is not as simple as the well-known pointers in C programs. A shared pointer stores a pointer to dynamically allocated object, typically with a C++ new expression. This guarantee that the pointed object will be deleted when the last shared pointer pointing to it is destroyed or reseted.

PETSC
PETSC stands for Portable Extensible Toolkit for Scientific Computation and is a suite of data structures and routines for the parallel solution of scientific applications modeled by partial differential equations. It

employes the MPI standard for parallelism. Each FEEL++ application owns a PETSC backend to optimize the solution calculation.

SLEPC
SLEPC stands for Scalable Library for Eigenvalue Problem Computations and is a software library for the solution of large scal sparse eigenvalue problems on parallel computers. It is an extension of PETSC which can be used for standar or generalized eigenproblems with real or complex arithmetic.

1.4 The project

The main author of the project is my internship tutor Christophe Prud'homme, who started to build the library in September 2004.

The project is under Subversion and is hosted on `forge.imag.fr` which is a collaboration plateform shared between several laboratories in Grenoble. Actually the current version is 0.91.0 and the deposit version is the 7436^{th} thanks to all commits and add that the developers have done. FEEL++ is currently developed by 9 people including a six months trainee and myself.

My internship would be focused on 3 key parts: learn the finite element method, improve the entire manual and build an application with the library. From the start, we had build a Grantt chart in order to monitor my work's progress. The estimated is given in the appendices on page 29. In practise, the "specific tools" part has been dropped to make a full parametrized application. To be able to work properly on the library, I first had to appropriate myself the finite element method. This method is explained in the following section.

CHAPTER 2

Finite Elementh Method

All mathematical notions and definitions employed in this chapter are available in the appendices on page 22.

2.1 Introduction

A differential equation or a partial differential equation owns inifinite solutions. To get only one, you have to solve the limits issue which means to impose boundary conditions.

Limits issues are differential problems which are set on an open interval with several dimensions $\Omega \subset \mathbb{R}^k$ ($k = 1, 2$ or 3) where values of the unknown or its derivatives are attached to the end a and b (or on the edge $\partial\Omega$ in the multidimensional case). However, define only one boundary conditions (on the derivatives for example) is sometimes not enough to get the uniqueness. Here are some classical equations of limits issues (the bondaries conditions are noted BC here and are described in 2.1):

- Poisson's equation

$$\left\{\begin{array}{ccc} -\Delta u & = & f \quad \text{in } \Omega \\ + \quad BC & & \end{array}\right. \tag{2.1}$$

  where f is a known-function and Δ is the $laplacian$.

- Heat equation

$$\left\{\begin{array}{l} \dfrac{\partial T}{\partial t} - D(T)\Delta T = 0 \quad \text{in } \Omega \\ + \quad BC \end{array}\right. \tag{2.2}$$

  where Ω is a domain of $\mathbb{R}^n$ (n=1, 2 or 3), $\partial\Omega$ its edge, D is the thermal diffusivity's coefficient depending on T, ∇ the $nabla$ operator and finally $\vec{n}$ the unit normal vector at the x point.

- Wave equation

$$\left\{\begin{array}{l} \dfrac{\partial^2 u(\mathbf{x}, t)}{\partial t^2} - c\Delta u(x, t) = 0, \quad \forall x \in \Omega, \forall t > 0 \\ + \quad BC \end{array}\right. \tag{2.3}$$

  where c is a positive constant.

To obtain the uniqueness of those equations, boundary conditions have to be fixed. These conditions make values specification that the solution has to verifiy on the domain's edge possible. The solution's uniqueness relies on correct boundaries conditions. Furthers boundaries conditions are possible, let's see the well-known conditions: Dirichlet, Neumann and Robin.

Boundaries conditions
One of the simplest one is the Dirichlet's condition: it refers directly to the function we are interested in (u for example). Take a look at some Dirichlet's conditions above :

- Dirichlet's conditions
  A classical non-homogeneous could look like

  $$u \quad = \quad g \quad \text{on } \partial\Omega$$

  To obtain a homogeneous Dirichlet's condition, you will obtain it with $g = 0$ which leads to

  $$u \quad = \quad 0 \quad \text{on } \partial\Omega$$

Another well-known boundary condition is the Neumann's condition. It refers to values that solution's derivatives must verify on the domain's borders (often with $gradient$, for example $\vec{\nabla} u \cdot \vec{n} = g$).

- Neumann's condition
  A non-homogeneous condition with Neumann is such as

  $$\kappa \vec{\nabla} u \cdot \vec{n} = g \quad on \quad \Gamma_N$$

  and a homogeneous Neumann's condition would give

  $$\kappa \vec{\nabla} u \cdot \vec{n} = 0 \quad on \quad \Gamma_N$$

Another well-known is a combination of the two others : the Robin's condition (for example $u + \vec{\nabla} u \cdot \vec{n} = g$ on $\partial\Omega$).

2.2 General example

The finite elements method, on which is based the FEEL++ library, is used to numerically solve partial differential equation. The resolution of such equations makes the representation of complex system's dynamic behavior possible.

Let's considerate the equation to solve with boundary conditions where $u \in \Omega$ is the unknown

$$\begin{cases} -\Delta u & = f \\ \quad u & = u_D \quad on \quad \Gamma_D \\ \nabla u.n & = g \quad on \quad \Gamma_N \end{cases} \tag{2.4}$$

where $\Gamma = \Gamma_D \cup \Gamma_N$ is the border of Ω. By integrating by parts with a test function (called v) supposed picewise regular, we obtain :

$$\int_\Omega \nabla u \cdot \nabla v - \int_\Gamma (\nabla u \cdot n) v = \int_\Omega f v \quad u \in \Omega \quad \forall v \in V_{\Gamma_D}$$

We have $u = u_D$ on Γ_D, we consequently take $v = 0$ on Γ_D and we got:

$$\int_\Omega \nabla u \cdot \nabla v - \int_{\Gamma_N} g v = \int_\Omega f v \quad u \in \Omega \quad \forall v \in V_{\Gamma_D}$$

where $V_{\Gamma_D} = \{v \in H^1(\Omega), v = 0 \text{ on } \Gamma_D\}$ with f and g which are known functions belonging to $C^0(\Omega)$.

2.3 Notions

2.3.1 Mesh

A mesh represents the spatial discretization of a continous domain. Its goal is to simplify a system with a model representing this system in view of simulations or graphical representations. Each calculation launched on a mesh requires boundary condition and parameters to respect checking. The finite element method relies on space's division with a mesh, it allows us a non-regular mesh (the step of the mesh is not constant). As a consequence, it enables us to tighten this mesh around the interests points. The more fine the mesh will be, the more precise the approached solution will be. The mesh is at the basis of the implementation of the method (after making the mathematical analysis and its writing under variational form). Create a mesh consists in covering the domaine with geometric elements as accurately as possible.

For example on $\mathbb{R}$ and considering $\Omega =]a, b[$ a mesh of Ω is a set of $N + 2$ points $(x_i)_{0 \leq i \leq N+1}$ such as $x_0 = a < x_1 < ... < x_i < .... < x_N < x_{N+1} = b$. The step of the mesh is apparently not regular, that is to say $h_i = x_{i+1} - x_i$ (for a regular step, $h = \frac{b-a}{N+1}$).

2.3.2 Finites elements

Définition

A finite element is a triplet $(\widehat{K}, \widehat{\Sigma}, \widehat{P})$ such as :

- $\widehat{K}$ is a geometric element of $\mathbb{R}^n$ ($n = 1, 2$ or 3).
- $\widehat{\Sigma}$ is a set of linear forms $(\widehat{\sigma}_{1_f}, ..., \widehat{\sigma}_{n_f})$ on $\widehat{P}$ called degrees of freedom.
- $\widehat{P}$ is a finite dimensional vector space of functions defined on $\widehat{K}$.

The dual basis of $\widehat{P}$ (associated to $\widehat{\Sigma}$ as it's a set of linear forms) noted $(\widehat{\theta}_i)_{1 \leq i \leq n_f}$ is such as

$$\widehat{\sigma}_i(\widehat{\theta}_j) = \delta_{ij} \quad 1 \leq i, j \leq n_f$$

where δ_{ij} is the Kroneker's term which is equal to 1 if $i = j$, 0 else. It is said the triplet $(\widehat{K}, \widehat{\Sigma}, \widehat{P})$ is unisolvent, that is to say the function

$$\begin{aligned} \mathcal{L} : \quad & \widehat{P} \longrightarrow \mathbb{R}^{n_f} \\ & \widehat{p} \longrightarrow (\widehat{p}(\widehat{\sigma}_1, ..., \widehat{p}(\widehat{\sigma}_{N_f})) \end{aligned}$$

is bijective (or, in another way, dim $\widehat{P}$ = card $\widehat{\Sigma}$ and it exists a unique element $\widehat{p}$ of $\widehat{P}$ such as $p(\widehat{\sigma}_i) = \alpha_i \quad i = 1, ..., n_f$).

Family of finite elements

This notion is necessary to link problem resolution and mesh notion. First of all, it is said that two finite elements $(\hat{K}, \hat{\Sigma}, \hat{P})$ and (K, Σ, P) are affine-equivalents if and onfly if it exists an invertible eaffine function F (where a_i are the degrees of freedom) such as

- $K = F(\hat{K})$
- $a_i = F(\hat{a}_i) \qquad i = 1, .., N$
- $P = \{\hat{p} \circ F^{-1}, \quad \hat{p} \in \hat{P}\}$

We call then an **affine family of finite elements** a family where all finite elements are affine-equivalents to a same finite element $(\hat{K}, \hat{\Sigma}, \hat{P})$ called **the reference element**. In practise, work with a family of affine elements allow us to bring back all the calculations of integrals in calculous on the reference element. The problem's unisolvance ensures us that from the reference element, we are able to describe in a unique way each finite elements of our family.

You can find several examples of such meshes or finite elements's families in the appendices on page 23.

Link

Here we would like to establish the link between the resolution with finite elements method and the mesh by Lagrange's finite elements. Let's consider a partial differential equation to solve on a domain Ω with H the Hilbert's space in which we are searching for a solution of the variationnal form. Thanks to the unisolvence property, the approached solution (which is u_h) will be fully defined on each elements (K_i, Σ_i, P_i) with its values on the degrees of freedom of the mesh.

By construction, a node will be shared by several adjacent elements. In meshes vocabulary, we say that a mesh is conform when the intersection between two elements is either empty or reduced to a vertex or an edge in dimension 2, or a vertex, edge or face in dimension 3. A non-conforming mesh can be handled by the finite elements method (thanks to Mortar's method, not described here) but we will prefer working with conform meshes.

Let's note $a_{1_h}, ..., a_{N_h}$ thoses degrees of freedom, the approach problem is reduced to determinate the values of u_h at the points a_i. Those degrees of freedom will be usefull to build basis functions, because to each a_i is associated a basis vector. That's that way we define global basis functions φ_i by :

$$\varphi_{i|K_l} \in \mathbb{P}_k, \quad l = 1, ..., N_e \quad and \quad \varphi_i(a_j) = \delta_{i,j}, \quad 0 \leq i, j \leq N_h$$

where N_e is cells number, N_h the degrees of freedom number and k is a fixed integer. This basis trully create V_h which is the inner approximation space. The $(\varphi_i)_{1 \leq i \leq N_h}$ are null everywhere, except on the elements where a_i is a node.

2.4 General principle

We have a partial differential equation (PDE) that we have to solve on Ω, with the method describe above we write this PDE under the variationnal form : we make the scalar product of the equation with a test function v in the space V to precise and then we integrate by parts the terms of highest degree (this integration will take account of the boundaries conditions). We obtain the weak formulation :

$$\text{Find } u \in \Omega \text{ such as } a(u, v) = l(v), \quad \forall v \in V$$

where $a(.,.)$ is a bilinear form on V^2 and $l(.)$ is a linear form on V. Then we make an internal approximation, so we have to define a mesh of Ω. This mesh is going to define the approximation space V_h (which is, by construction, a linear subspace of V). The problem is then equivalent to :

$$\text{Find } u_h \in V_h \text{ such as } a(u_h, v_h) = l(v_h), \quad \forall v_h \in V_h$$

Now we have to provide a basis $(\varphi_j)_{1 \leq j \leq N_h}$ for V_h to decompose any vector of this domain in this basis. By decomposing u_h and v_h in this basis and with a's linearity, the problem is brought to :

$$\text{Find } u_1, ..., u_{N_h} \text{ such as } \sum_{i=1}^{N_h} u_i a(\varphi_i, \varphi_j) = l(\varphi_j) \quad \forall j = 1...N_h$$

The problem can be delay in matrix, which leads us to this linear system :

$$\begin{pmatrix} a(\varphi_1, \varphi_1) & ... & a(\varphi_{N_h}, \varphi_1) \\ \vdots & & \vdots \\ a(\varphi_1, \varphi_{N_h}) & ... & a(\varphi_{N_h}, \varphi_{N_h}) \end{pmatrix} \begin{pmatrix} u_1 \\ \vdots \\ u_{N_h} \end{pmatrix} = \begin{pmatrix} l(\varphi_1) \\ \vdots \\ l(\varphi_{N_h}) \end{pmatrix}$$

Finally, the system to be solved is :

$$Au = b$$

There are several ways to solve this linear system, even if the matrix A could probably be full. The choice of the basis $(\varphi_j)_{1 \leq j \leq N}$ is therefore crucial to make each function φ_j be null except at few meshes. Finally, the $a(\varphi_i, \varphi_j)$ term will be the most of the time null, which will simplify the calculations (in practise, there will be a "band" on the matrix).

CHAPTER 3

Building and tutorial

In this section, I will present the different works done on FEEL++ and its manual.

3.1 Build

At the beginning, the tutorial which includes the Build section was a bit outdated. From the time it was written, furthers things have moved on. So I had to update this part but also add a Mac OS X section. Indeed, the exlanation was only available for Unix systems. I had just bought a new macbook pro so the opportunity was too good for me to add this section.

3.1.1 Shape

I started by gaving a "refresh" style to the manual by giving a new structure to make the reading more clear to any reader. We decided to split the manual into three parts :

- 𝕀 Tutorial : this part covers the building of the library, how to compile and how to create elementary applications.
- 𝕀𝕀 Learning by examples : this one shows furthers concrete examples, such as the heat sink application I have made.
- 𝕀𝕀𝕀 Progamming with FEEL++: this last one aims to present the keywords for programming with the library.

This work gave me the chance to discover LaTeX: a powerfull document markup language and document preparation system. It also makes me discover CMAKE which is a unified, cross-platfrom, open-source build system that enables developers to build, test and package softwares by specifying build parameters in simple and portable text files. This tool can create libraries, generate wrappers, compile source code or build executables in arbitrary combinations.

3.1.2 Mac OS X

Being a beginner with Macintosh systems, adding this section gave me the opportunity to be directly confortable with this operating system. Actually, the installation process is quite different on Macintosh systems than on Unix ones.

FEEL++ is working with many dependencies such as :

Required packages	Optional packages
g++	Superlu
Mpi : openmpi or mpich	Suitesparse
Boost	Metis
Petsc	Trilinos
Cmake	Google perftools
Libxml2	Paraview
	Python

To make it easier, we used MACPORTS by creating a port for FEEL++ but furthers issue could still happened, so I had to explain how to fix them. The major concerns were linked with `boost` and `mpi`. When I left the laboratory, FEEL++ was also working with the newest operating system Lion.

3.2 Tutorial

3.2.1 Getting started

This part intends to give users the basic tools to create elementary applications. The explanations and the code was a bit outdated so I update these two parts to keep the manual up to date. In the tutorial, this part provides a lot of code and exaplanations, you can find an extract in the appendix 6.3.

3.2.2 More meshes

As it stands, FEEL++ was able to load many meshes but essentially the Gmsh mesh file format. It provided also some classes to manipulate `.geo` files and generate `.msh` files. We wanted to introduce new types represented by medit meshes (`.mesh` format) which is very used in the scientific domain, and STL meshes (format `.stl`) which are very used in stereolithography. To make it possible, I had to analyse the difference between those formats and implement how to differentiate them.
To make it clear, GMSH is an automatic 3D finite element mesh generator with build-in pre- and post-processing facilities. The standard for `.msh` file format is described in the appendix 6.2.

We were here interested in adding 2 new formats for FEEL++:

- Medit
  A Medit reader is integrated into GMSH but this one is not fully compatible with our library, so many improvements have to be done. The medit reader of GMSH is able to read medit meshes, the issue comes from markers for areas of the edges were we want to apply different boundaries conditions. GMSH is currently using the Physical Entities (physical line, area, volume). Unfortunetly, the medit reader of Gmsh considers the physical flag as null. To make it possible, the Gmsh importer has to be slightly modified. Once the modifications were brought, we were able to call our physical entities to make calculation on it. Concretely, once the `.msh` mesh has been produced, we found the same entities that were present on the `.medit` original mesh.

  To be sure that our `medit` mesh was correctly read, double check has been realised. The first one was easy : I have to see if Gmsh found the same entities in the `.msh` than in the `.mesh`. The second one was to make calculation on it, so I applied the Gaussian theorem which states that

  $$\int_{\Omega} div(u) = \int_{\partial\Omega} u \cdot n \tag{3.1}$$

  which means that the outward flux of a vector field through a closed surface is equal to the volume integral of the divergence of the region inside the surface. So I choose a vector $u = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ because

$div(u) = 0$. The check was consequently to calculate $\int_{\partial\Omega} u \cdot n$, which is in FEEL++ language :

```
integrate( _range=boundaryfaces(mesh),
        _expr=trans(vec(cst(1.),cst(1.),cst(1.)))*N() ).evaluate()(0,0);
```

I was pleased because I obtained the result $6.53344e - 12$ which is what I expected.

- STL

We also add the `stl` files, those files are native to the stereolithography CAD software created by 3D Systems. These files describe only the surface geometry of a three dimensional object without any representation of color, texture or other common attributes.

To use FEEL++ with `stl` files, a `geo` script has to be created to enable gmsh to remesh the file. The `stl` file used must be a volume mesh. The script is very small, all informations to make one is on Gmsh/slt section on their web site. Once it's done, I just have to type

```
gmsh stl_file_name.geo -3
```

with `stl_file_name.stl` in the same directory. That command will produce the correct `.msh` mesh that I could now use as usual without any modification in a FEEL++ application. Take a look above how the remesh has produced a complete mesh with the file `pelvis.stl` and `pelvis.geo`:

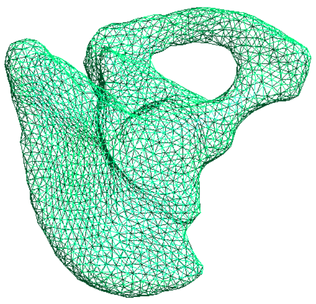


Figure 3.1: Pelvis before remesh (stl)

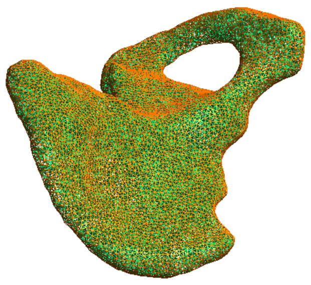


Figure 3.2: Pelvis after remesh (msh)

CHAPTER 4

Heat sink application

In this section, we will consider in this section the performance of a heat sink designed for the thermal management of high-density electronic components. The heat sink is comprised of a base/spreader which in turn supports a number of plate fins exposed to flowing air. We model the flowing air through a simple convection heat transfer coefficient. From the engineering point of view, this issue illustrates the conduction analysis of an important class of cooling problems: electronic components and systems.

Our interest is in the conduction temperature distribution at the base of the spreader. The target is to study how the heat transfer occures with different parameters on our heat sink. The heat generated by high-density electronic components is such that it's very expensive to cool large structures (data center). The cooling optimization is consequent in the financial race for decreasing operating costs. A classical thermal CPU cooler looks like this

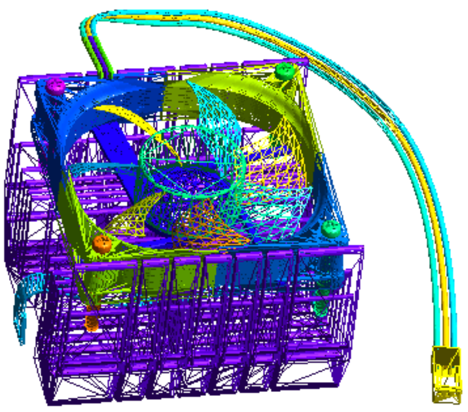


Figure 4.1: Mesh of a classical CPU cooler

I have considered a classical "radiator" which is a CPU heat sink. Those types of coolers are composed with a certain number of plate fins exposed to flowing air or exposed to a ventilator. Regarding the periodicity and geometry of our concern, I have made my study on a characteristical element of the problem : a half cell of a thermal fin of a heat sink with its spreader at the basis. Let's take a look at the geometry of our problem :

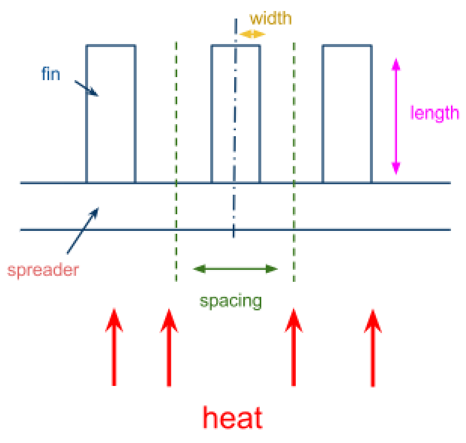


Figure 4.2: Geometry of a heat sink

The application would be parametrized, so the mesh I had to create had to be generated with FEEL++ thanks to a simple header attached to the application. In that way I could add many inputs to my application to parameterize it as I wanted. So I created the corresponding meshes in 2D or 3D. I applied what I have learned with the `.geo` scripts and obtained :

Figure 4.3: 2D mesh

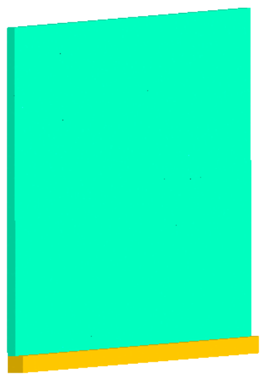


Figure 4.4: 3D mesh

I was now able to start with the theory and write the equations I will have to code with the library.

4.1 Theory

My main interest here was the temperature, and every physical temperature has to obey to the heat equation. To obtain the right output, I fixed furthers boundaries conditions linked with the heat sink concern. The equations are :

$$\sum_{i=1}^{2} \kappa_i \Delta T - \rho_i C_i \frac{\partial T}{\partial t} = 0 \tag{4.1}$$

$$\kappa_1 \nabla T \cdot n = 0 \quad on \quad \Gamma_2 \quad and \quad \Gamma_6 \tag{4.2}$$

$$\kappa_2 \nabla T \cdot n = 0 \quad on \quad \Gamma_5, \Gamma_7 \quad and \quad \Gamma_8 \tag{4.3}$$

$$\kappa_1 \nabla T \cdot n = -h(T - T_{amb}) \quad on \quad \Gamma_1 \tag{4.4}$$

$$\kappa_2 \nabla T \cdot n = Q(1 - e^{-t}) \quad on \quad \Gamma_4 \tag{4.5}$$

$$T_{|\Omega_1} = T_{|\Omega_2} \quad on \quad \Gamma_3 \tag{4.6}$$

$$\kappa_1 \nabla T \cdot n = \kappa_2 \nabla T \cdot n \quad on \quad \Gamma_3 \tag{4.7}$$

with $i = 1$ for the fin and $i = 2$ for the base and where κ_i is the thermal conductivity, ρ_i is the material's density ($kg.m^{-3}$ in the SI unit), C_i the heat capacity and T the temperature at a precise point (in 2D or 3D). The corresponding diagrams are :

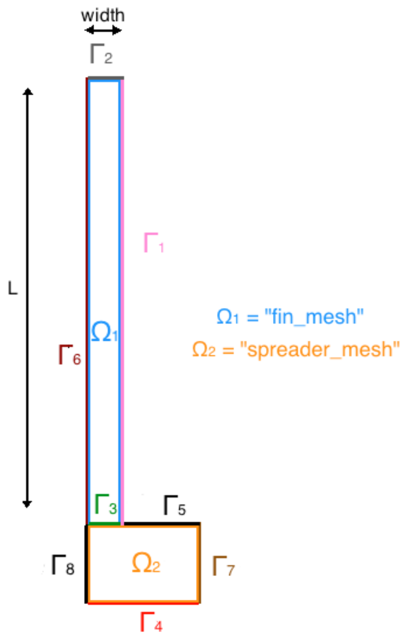


Figure 4.5: 2D geometry details

Starting from the equation and its boundaries conditions, I could make the calculation, which resuts in the final equation that is coded :

$$\boxed{h\int_{\Gamma_1} vT + \sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v\frac{T^{n+1}}{\delta t} + \kappa_i \int_{\Omega_i} \nabla v \cdot \nabla T = \int_{\Gamma_4} vQ(1 - e^{-t}) + hT_{amb}\int_{\Gamma_1} v + \sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v\frac{T^n}{\delta t}} \tag{4.8}$$

The calculation details are described in the appendices at the page 27.

4.2 Implementation

I had to create a complete application, so I carried out what I have learned during the tutorial's improvements. I started by making an adimensional application but after furthers iterations between S.Veys and

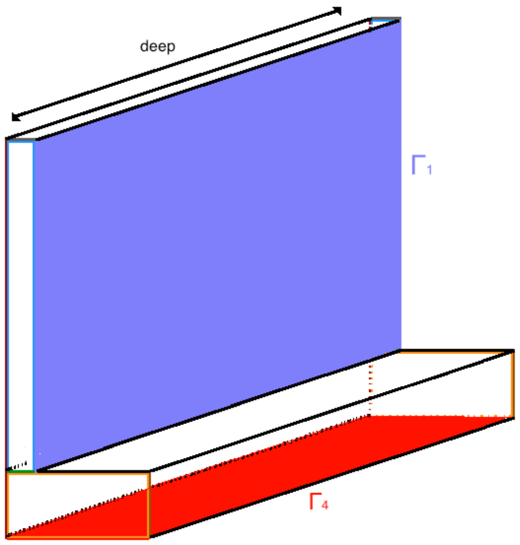


Figure 4.6: 3D geometry details

C.Prud'homme we decided to build a dimensional one. You can check all possible inputs in the appendices on page 27

4.3 The results

Thanks to the GMSH exporter, we could observe in a nice way the results of the calculation. Here is an example in 3D at the steady state considering that the heatsink is totally made in copper, with a heat flux $Q = 1e6\ W.m^{-2}$ and a thermal coefficient $h = 1e3$. The figures 4.8 and 4.9 represent the temperature evolution during the transient state, the graphics have been released thanks to OCTAVE :

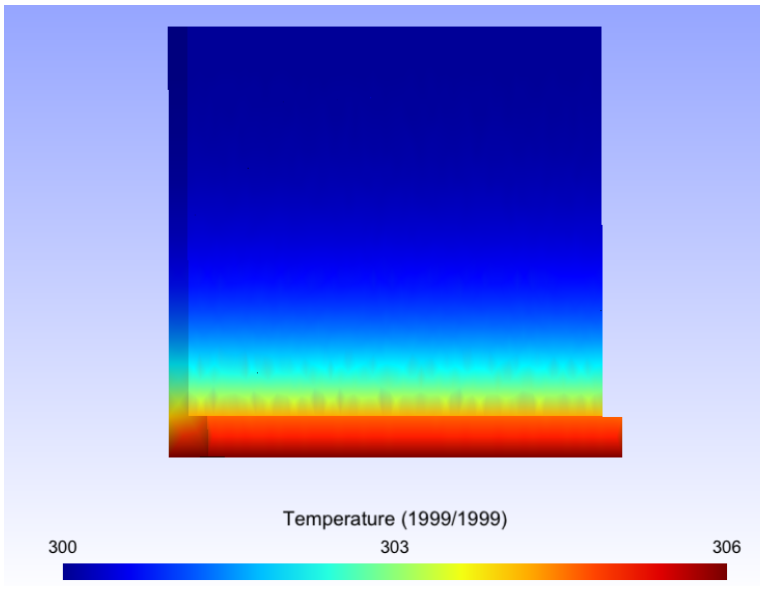


Figure 4.7: Steady state: spreader and fin in copper, $Q = 1e6$ and $h = 1e3$

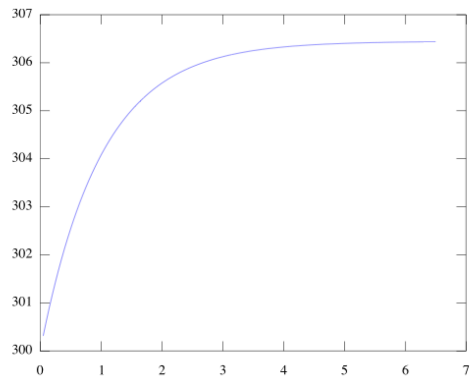


Figure 4.8: Transient state on Γ_4

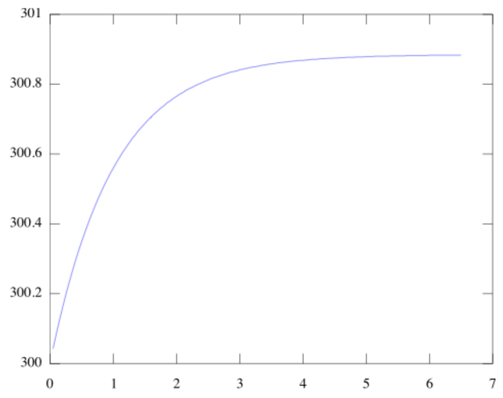


Figure 4.9: Transient state on Γ_1

CHAPTER 5

Conclusion

It was my first internship in the computer science domain and it has been a really good experience for me. This is a new step for my ingeneer graduation and this work has brought me several experiences that I'm glad to have acquired. My work at the LJK is part of a strong desire to develop a complete and efficient tool in applied mathematics. The stages of FEEL++ evolution make the library becoming more and more functional. It is important to see how far we can push the limits.

First of all, I had a mutli scale role so I had to be able to handle multiple tasks at the time which could be sometimes confusing but contributes to a personal satisfaction when the tasks are complete. These challenges conforts me in the idea of becoming an ingeneer.

Most of all, I have worked with my superiors trusting, which is greatfull and made me feel safe in my work progression. In my point of vue, that kind of feelings is an advantage for improving work progess.

Finally, this experience has given me a real insight of working life. The feedback about my work I could have collected from my coworkers during my internship can only make me progress. I trully think that practicing all the academic skills is necessary in order to acquire a solid formation.

Bibliography

[1] ASME. *Real-time reliable simulation of heat transfer phenomena*. G.Rozza, D. B. P. Huynh, N. C. Nguyen and A.T. Patera, 2009.

[2] Eric Blayo. Notes de cours sur la méthodes des éléments finis. 1(1), 2010.

[3] D. J. Knezevic D. B. P. Huynh and A.T. Patera. A static condensation reduced basis element method: approximation and a posteriori error estimation. 2011.

[4] Ken Martin et Bill Hoffman. *Mastering CMake*, volume 1. 2004.

[5] Patrick Ciarlet et Eric Lunéville. La méthode des éléments finis, de la théorie à la pratique. *Les presses de l'ENSTA*, 1(1), 2009.

[6] Alfio Quarteroni et Fausto Saleri. *Calcul scientifique, cours, exercices corrigés et illustrations en MATLAB et Octave*, volume 1. 2006.

CHAPTER 6

Appendices

6.1 FEM tools

- Gradient
  The gradient of a scalar field is a vector which points in the direction of the greatest rate of increase of the scalar field. It's a vector denoted $\vec{grad}$, for a scalar function $f(x_1, ..., x_n)$ we have :

$$\overrightarrow{grad}f = \vec{\nabla} f = (\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_n})$$

- Laplacian
  The Laplacien is a differential operator obtained by making the divergence of the gradient of the function. The usual symbol is Δ and the Laplacien represents the sum of all the second partial derivatives :

$$\Delta f = \sum_{i=1}^{N} \frac{\partial^2 f}{\partial x_i^2}$$

- Divergence
  The divergence is a vector operator which indicates the magnitude of a vector field's source. It's a scalar-valued function which characterizes a stream of particles coming from somewhere (which goes to the source or which comes from it). If the divergence is not equal to zero, that means there is a concentration around a point (increasing or decreasing, depending on the sign). In 3D, the divergence of a vector's field $\vec{F} = (F_x, F_y, F_z)$ represents the scalar :

$$div\mathbf{F} = \nabla \cdot \mathbf{F} = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z}$$

  In the general case, we have :

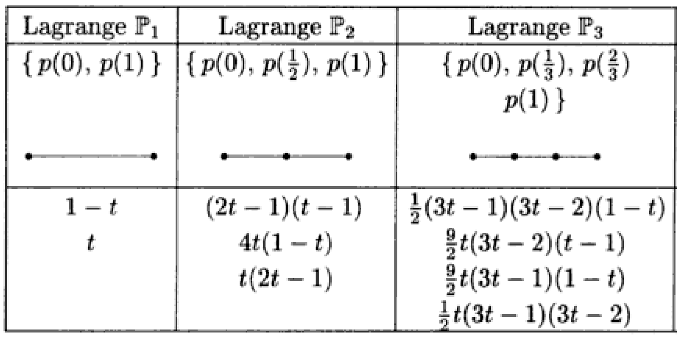
$$\nabla \cdot \mathbf{F} = \sum_{j=0}^{d} \frac{\partial F_j}{\partial x_j}$$

  where d is the dimension.

- Vector spaces
  Here is a classical vector space used in this document. $\mathbb{P}_k$ represents all real-valued polynomials of total degree less or equal to k.
  $$\mathbb{P}_k = Vect(1, X, ..., X^k)$$
  but it can also be writtent this way (in 1D)
  $$\mathbb{P}_k = \{p(x) = \sum_{0 \leq i \leq k} a_i x_i, a_i \in \mathbb{R}\}$$
  and this way (in 2D) :
  $$\mathbb{P}_k = \{p(x_1, x_2) = \sum_{0 \leq i+j \leq k} a_{ij} x_1^i x_2^j, \quad a_{ij} \in \mathbb{R}\}$$

6.2 Mesh

Finites elements examples
One dimension
The Lagrange finite element $\mathbb{P}_k$ can be formed of the segment $\widehat{K} = [0, 1]$, of $\widehat{P} = \mathbb{P}_k$ and of $\widehat{\Sigma}$ such as $\widehat{\sigma}_i(\widehat{p}) = \widehat{p}(\frac{i}{k}), 0 \leq i \leq k$. The basis functions associated to thoses elements are the classical Lagrange polynomials. Here are the degrees of freedom and the basis functions associated for $k = 1, 2$ and 3 :



Lagrange $\mathbb{P}_1$	Lagrange $\mathbb{P}_2$	Lagrange $\mathbb{P}_3$
$\{ p(0), p(1) \}$	$\{ p(0), p(\frac{1}{2}), p(1) \}$	$\{ p(0), p(\frac{1}{3}), p(\frac{2}{3})$ $p(1) \}$
$1-t$ t	$(2t-1)(t-1)$ $4t(1-t)$ $t(2t-1)$	$\frac{1}{2}(3t-1)(3t-2)(1-t)$ $\frac{9}{2}t(3t-2)(t-1)$ $\frac{9}{2}t(3t-1)(1-t)$ $\frac{1}{2}t(3t-1)(3t-2)$

Figure 6.1: Finite elements of Lagrange in 1D example

Three dimensions

The Lagrange finite element $\mathbb{P}_k$ can be formed of the tetrahedron of summits a_1, a_2, a_3 and a_4 with respective coordinates $(0,0,0), (1,0,0), (0,1,0)$ and $(0,0,1)$. The basis functions for $k = 1$ are given by $\widehat{\theta}_1 = 1 - x_1 - x_2 - x_3$, $\widehat{\theta}_2 = x_1$, $\widehat{\theta}_3 = x_2$ and $\widehat{\theta}_4 = x_3$. Here are their representations and basis functions associated :

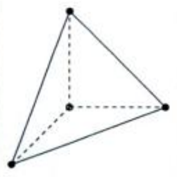
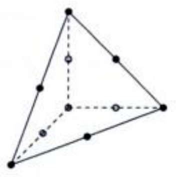
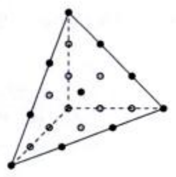
$\mathbb{P}_1$	$\mathbb{P}_2$	$\mathbb{P}_3$
		
$\widehat{\theta}_i, 1 \le i \le 4$	$\widehat{\theta}_i(2\widehat{\theta}_i - 1), 1 \le i \le 4$ $4\widehat{\theta}_i\widehat{\theta}_j, 1 \le i < j \le 4$	$\frac{1}{2}\widehat{\theta}_i(3\widehat{\theta}_i - 1)(3\widehat{\theta}_i - 2), 1 \le i \le 4$ $\frac{9}{2}\widehat{\theta}_i(3\widehat{\theta}_i - 1)\widehat{\theta}_j, 1 \le i,j \le 4, i \ne j$ $27\widehat{\theta}_i\widehat{\theta}_j\widehat{\theta}_k, i \ne j \ne k \ne i$

Figure 6.2: Finite elements of Lagrange in 3D example

New mesh

FEEL++ is using GMSH to load, create and save meshes. To make it clear, GMSH is an automatic 3D finite element mesh generator with build-in pre- and post-processing facilities. The standard for `.msh` file format is as follow :

```
$MeshFormat
version-number file-type data-size
$EndMeshFormat
$Nodes
number-of-nodes
node-number x-coord y-coord z-coord
...
$EndNodes
$Elements
number-of-elements
elm-number elm-type number-of-tags < tag > ... node-number-list
...
$EndElements
$PhysicalNames
number-of-names
physical-dimension physical-number "physical-name"
...
$EndPhysicalNames
$NodeData
number-of-string-tags
< "string-tag" >
...
number-of-real-tags
< real-tag >
...
number-of-integer-tags
< integer-tag >
...
node-number value ...
...
$EndNodeData
$ElementData
number-of-string-tags
< "string-tag" >
...
```

```
      number-of-real-tags
      < real-tag >
      ...
      number-of-integer-tags
      < integer-tag >
      ...
      elm-number value ...
      ...
      $EndElementData
      $ElementNodeData
      number-of-string-tags
      < "string-tag" >
      ...
      number-of-real-tags
      < real-tag >
      ...
      number-of-integer-tags
      < integer-tag >
      ...
      elm-number number-of-nodes-per-element value ...
      ...
      $ElementEndNodeData
```

Medit
Here the section in which we are interested if `number-of-tags`. This number gives the number of integer tags that follow for the n-th element. By default, the first tag is the number of the physical entity to which the element belongs; the second is the number of the elementary geometrical entity to which the element belongs; the third is the number of mesh partitions to which the element belongs, followed by the partition ids (negative partition ids indicate ghost cells).

The medit reader of Gmsh is able to read medit meshes, the issue comes from markers for areas of the edges were we want to apply different boundaries conditions. Gmsh is currently using the Physical Entities (physical line, area, volume). Unfortunetly, the medit reader of Gmsh considers the physical flag as null. To make it possible, the Gmsh importer has to be slightly modified. That is why the boolean parameter `physical_are_elementary_regions` has been introduced in the functions `loadGMSHMesh` and `createGMSHMesh`. It acts on the visit of the mesh and if this boolean is **`true`**, instead of adding

```
__et[__i].push_back(__physical_region);
__et[__i].push_back(__elementary_region);
```

we have to add :

```
__et[__i].push_back(__elementary_region);
__et[__i].push_back(__elementary_region);
```

because the `physical_region` was considered as null.
Once the modifications have been made, we were able to call our physical entities to make calculation on it. Concretely, once the `.msh` mesh has been produced, we found the same entities that were present on the `.medit` original mesh.

6.3 Getting started

Extract from the tutorial section I wrote :

" In the example, we provide the options `dt` which takes an argument, a **`double`** and its default value is `1` if the options is not set by the command line. Then we describe the application by defining a class `AboutData` which will be typically used by the `help` command line options to describe the application

```
inline
AboutData
makeAbout()
{
    AboutData about( "myapp" ,
                     "myapp" ,
                     "0.1",
                     "my first Feel application",
                     AboutData::License_GPL,
                     "Copyright (c) 2008 Universite Joseph Fourier");

    about.addAuthor("Christophe Prud'homme",
                    "developer",
                    "christophe.prudhomme@ujf-grenoble.fr", "");
    return about;
}
```

Now we turn to the class `MyApp` itself: it derives from `Feel::Application`. This class provides two constructors : one with only description and one with additionnal parameters which enables to add options `argc` and `argv`. This class `MyApp` has to redefine the `run()` method. It is defined as a pure virtual function in `Application`.

```
class MyApp: public Application
{
public:

    /**
     * constructor only about data and no options description
     */
    MyApp( int argc, char** argv, AboutData const& );

    /**
     * constructor about data and options description
     */
    MyApp( int argc, char** argv,
           AboutData const&,
           po::options_description const&  );

    /**
     * This function is responsible for the actual work done by MyApp.
     */
    void run();
};
```

The implementation of the constructors is usually simple, we pass the arguments to the super class `Application` that will analyze them and subsequently provide them with a `Feel::po::variable_map` data structure which operates like a `map`. Have a look at the document `boost::program_options`[1] for further details. Here our two constructors do nothing (because {}).

```
MyApp::MyApp(int argc, char** argv,
             AboutData const& ad )
    :
    Application( argc, argv, ad )
{}
MyApp::MyApp(int argc, char** argv,
             AboutData const& ad,
             po::options_description const& od )
    :
    Application( argc, argv, ad, od )
{}
```

The `run()` member function holds the application commands/statements. Here we provide the smallest code unit: we print the description of the application if the `--help` command line options is set. "

[1]http://www.boost.org/doc/html/program_options.html

6.4 Heat sink application

Equations
Here is the detail of the calculation with the finite element method. We apply the method by introducing the test function v and we integrate the main equation, which reads now as :

$$\sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v\frac{\partial T}{\partial t} - \kappa_i \int_{\Omega_i} v\Delta T = 0 \tag{6.1}$$

We integrate by parts, which leads to :

$$\sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v\frac{\partial T}{\partial t} + \kappa_i \int_{\Omega_i} \nabla v \cdot \nabla T - \kappa_i \int_{\partial\Omega_i} (\nabla T \cdot n)v = 0 \tag{6.2}$$

then, by decomposing the borders $\partial\Omega_i$, we obtain :

$$\begin{aligned} -\kappa_1 \int_{\Gamma_1} (\nabla T \cdot n)v - \kappa_2 \int_{\Gamma_4} (\nabla T \cdot n)v - \kappa_1 \int_{\Gamma_{2,6}} (\nabla T \cdot n)v - \kappa_2 \int_{\Gamma_{5,7,8}} (\nabla T \cdot n)v \quad + \\ \sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v\frac{\partial T}{\partial t} + \kappa_i \int_{\Omega_i} \nabla v \cdot \nabla T - \kappa_i \int_{\partial\Omega_i \cap \Gamma_3} (\nabla T \cdot n)v = 0 \end{aligned} \tag{6.3}$$

Now, we apply the conditions (4.2), (4.3), (4.4) and (4.5) which brings us to :

$$\int_{\Gamma_1} hv(T - T_{amb}) - \int_{\Gamma_4} vQ(1 - e^{-t}) + \sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v\frac{\partial T}{\partial t} + \kappa_i \int_{\Omega_i} \nabla v \cdot \nabla T - \underbrace{\kappa_i \int_{\partial\Omega_i \cap \Gamma_3} (\nabla T \cdot n)v}_{\text{=0 thanks to 4.7}} = 0 \tag{6.4}$$

Now we apply the boundary conditions (4.7) which results in :

$$h\int_{\Gamma_1} v(T - T_{amb}) - \int_{\Gamma_4} vQ(1 - e^{-t}) + \sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v\frac{\partial T}{\partial t} + \kappa_i \int_{\Omega_i} \nabla v \cdot \nabla T = 0 \tag{6.5}$$

We can now start to transform the equation by puting in the right hand the known terms :

$$h\int_{\Gamma_1} vT + \sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v\frac{\partial T}{\partial t} + \kappa_i \int_{\Omega_i} \nabla v \cdot \nabla T = \int_{\Gamma_4} vQ(1 - e^{-t}) + hT_{amb} \int_{\Gamma_1} v \tag{6.6}$$

We discretize $\frac{\partial T}{\partial t}$ where δt is the time step, such as:

$$h\int_{\Gamma_1} vT + \sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v\frac{T^{n+1} - T^n}{\delta t} + \kappa_i \int_{\Omega_i} \nabla v \cdot \nabla T = \int_{\Gamma_4} vQ(1 - e^{-t}) + hT_{amb} \int_{\Gamma_1} v \tag{6.7}$$

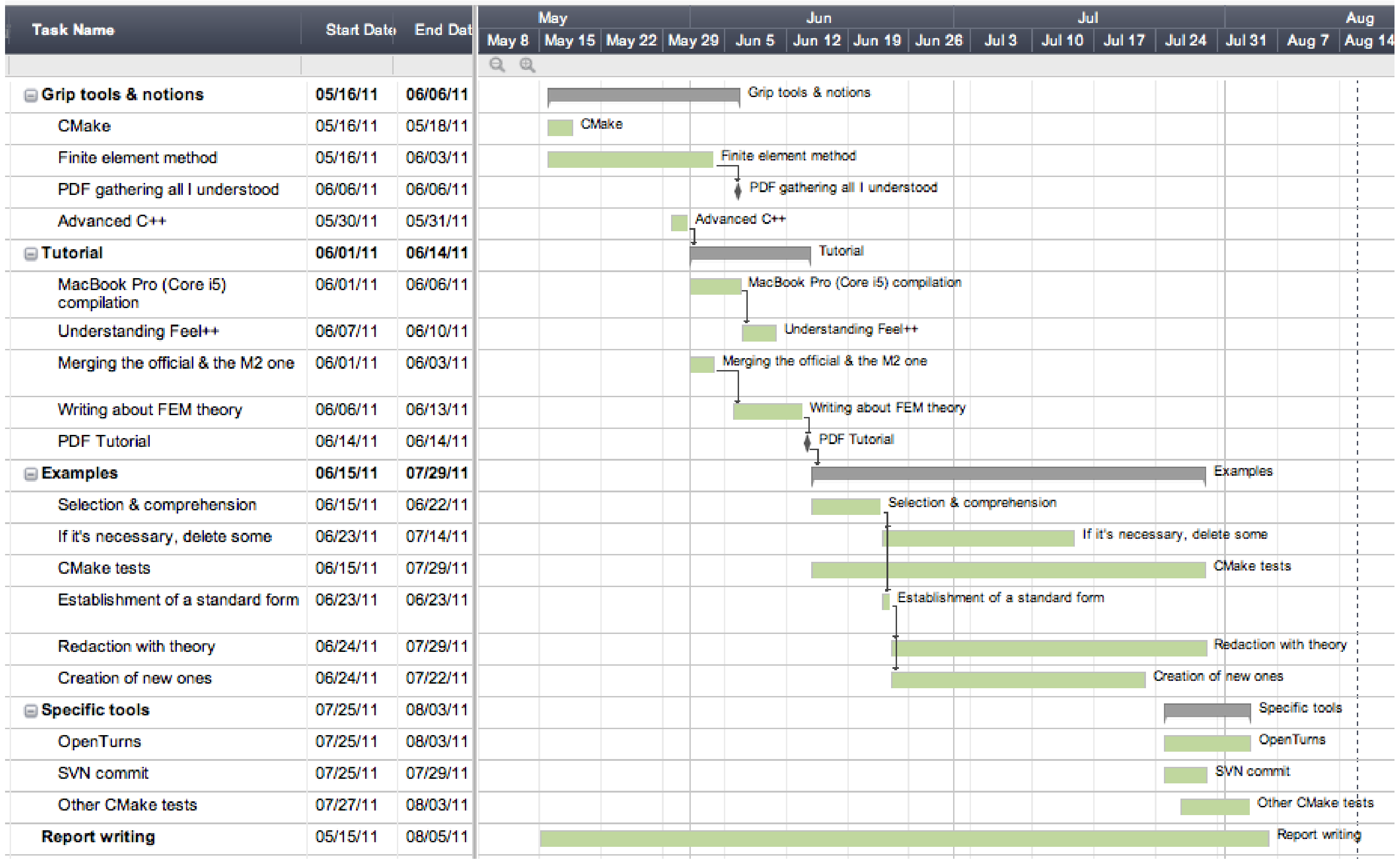
Finally we obtain :

$$\boxed{h\int_{\Gamma_1} vT + \sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v\frac{T^{n+1}}{\delta t} + \kappa_i \int_{\Omega_i} \nabla v \cdot \nabla T = \int_{\Gamma_4} vQ(1 - e^{-t}) + hT_{amb} \int_{\Gamma_1} v + \sum_{i=1}^{2} \rho_i C_i \int_{\Omega_i} v\frac{T^n}{\delta t}} \tag{6.8}$$

Inputs
The following table displays the various fixed and variables parameters of this application.

Name	Description	Nominal Value	Range	Units
	BDF parameters			
$time-initial$	begining	0		
$time-final$	end	50	$]0,1500]$	
$time-step$	time step	0.1	$]0,1[$	
$steady$	steady state	0	$\{0,1\}$	
$order$	order	2	$[0,4]$	
	Physical parameters			
L	fin's length	$2\cdot10^{-2}$	$[0.02,0.05]$	m
$width$	fin's width	$5\cdot10^{-4}$	$[10^{-5},10^{-4}]$	m
$deep$	heat sink depth	0	$[0,7\cdot10^{-2}]$	m
	Mesh parameter			
$hsize$	mesh's size	10^{-4}	$[10^{-5},10^{-3}]$	
	Fin Parameters			
κ_f	thermal conductivity	386	$[100,500]$	$W\cdot m^{-1}\cdot K^{-1}$
ρ_f	material density	8940	$[10^3,12\cdot10^3]$	$kg\cdot m^{-3}$
C_f	heat capacity	385	$[10^2,10^3]$	$J\cdot kg^{-1}\cdot K^{-1}$
	Base/spreader Parameters			
κ_s	thermal conductivity	386	$[100,500]$	$W\cdot m^{-1}\cdot K^{-1}$
ρ_s	material density	8940	$[10^3,12\cdot10^3]$	$kg\cdot m^{-3}$
C_s	heat capacity	385	$[10^2,10^3]$	$J\cdot kg^{-1}\cdot K^{-1}$
	Heat Parameters			
T_{amb}	ambient temperature	300	[300,310]	K
$heat\_flux$	heat flux Q	10^6	$[0,10^6]$	$W\cdot m^{-3}$
$therm\_coeff$	thermal coefficient h	10^3	$[0,10^3]$	$W\cdot m^{-2}\cdot K^{-1}$

6.5 Gantt chart



Task Name	Start Date	End Dat
⊟ **Grip tools & notions**	**05/16/11**	**06/06/11**
CMake	05/16/11	05/18/11
Finite element method	05/16/11	06/03/11
PDF gathering all I understood	06/06/11	06/06/11
Advanced C++	05/30/11	05/31/11
⊟ **Tutorial**	**06/01/11**	**06/14/11**
MacBook Pro (Core i5) compilation	06/01/11	06/06/11
Understanding Feel++	06/07/11	06/10/11
Merging the official & the M2 one	06/01/11	06/03/11
Writing about FEM theory	06/06/11	06/13/11
PDF Tutorial	06/14/11	06/14/11
⊟ **Examples**	**06/15/11**	**07/29/11**
Selection & comprehension	06/15/11	06/22/11
If it's necessary, delete some	06/23/11	07/14/11
CMake tests	06/15/11	07/29/11
Establishment of a standard form	06/23/11	06/23/11
Redaction with theory	06/24/11	07/29/11
Creation of new ones	06/24/11	07/22/11
⊟ **Specific tools**	07/25/11	08/03/11
OpenTurns	07/25/11	08/03/11
SVN commit	07/25/11	07/29/11
Other CMake tests	07/27/11	08/03/11
Report writing	05/15/11	08/05/11

Figure 6.3: Gantt chart