THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par*Institut National Polytechnique de Toulouse*
Discipline ou spécialité :*Dynamique des Fluides*

Présentée et soutenue par *Stéphan Jauré*
Le 13 *Décembre 2012*

Titre :
Methodology for conjugate heat transfer simulations relying on Large Eddy Simulations in massively parallel environments

---

JURY

Marc-Paul Errera	*Rapporteur*
Denis Veynante	*Rapporteur*
Laurent Gicquel	*Directeur de thèse*
Florent Duchaine	*Codirecteur de thèse*
Luc Giraud	*Examinateur*
Jens-Dominik Mueller	*Examinateur*
Ronan Daviot	*Examinateur*

Ecole doctorale :*Mécanique, Energétique, Génie civil, Procédés*
Unité de recherche :*CERFACS*
Directeur(s) de Thèse :*Laurent Gicquel, Florent Duchaine*
Rapporteurs :*Marc-Paul Errera, Denis Veynante*

Contents

Disclaimer

Some of the material used/shown/discussed within this thesis was considered confidential and hence not suitable for a public release. Therefore some parts of the original manuscript have been modified/hidden or stripped to produce this public release. This includes stripping entire chapters, namely 4,5 and 15.

Je tiens tout d'abord à remercier mon encadrement (un grand merci à Florent, Laurent et Gabriel) pour m'avoir accompagné durant ces quelques années de thèse. J'ai énormément appris durant ces quelques années et c'est en grande partie grâce à vous. Mais également un grand merci à l'ensemble de l'équipe CFD du CERFACS: aux seniors ainsi qu'aux thésards/postdocs, notamment ceux qui sont partis Alex, Ignacio, JB, Thomas, Anthony, Damien, Jorge ceux qui sont encore David, Jean-Phi, Remy, Damien...). Merci au département des méthodes aérothermique de Snecma Villaroche qui a financé mes travaux (tout particulièrement merci à Eric Mercier, Juan Carlos Larroya, Emilie Lachaud, Sebastien Roux, Ronan Daviot). Merci au jury pour avoir pris le temps d'étudier ce manuscrit (ce qui ne devait pas être facile). Finalement merci à ceux qui m'ont supporté hors laboratoire: ma famille et Marie!

Poets say science takes away from the beauty of the stars - mere globs of gas atoms. I, too, can see the stars on a desert night, and feel them. But do I see less or more?

Richard P. Feynman

Chapter 1

Scientific Context

1.1 Introduction

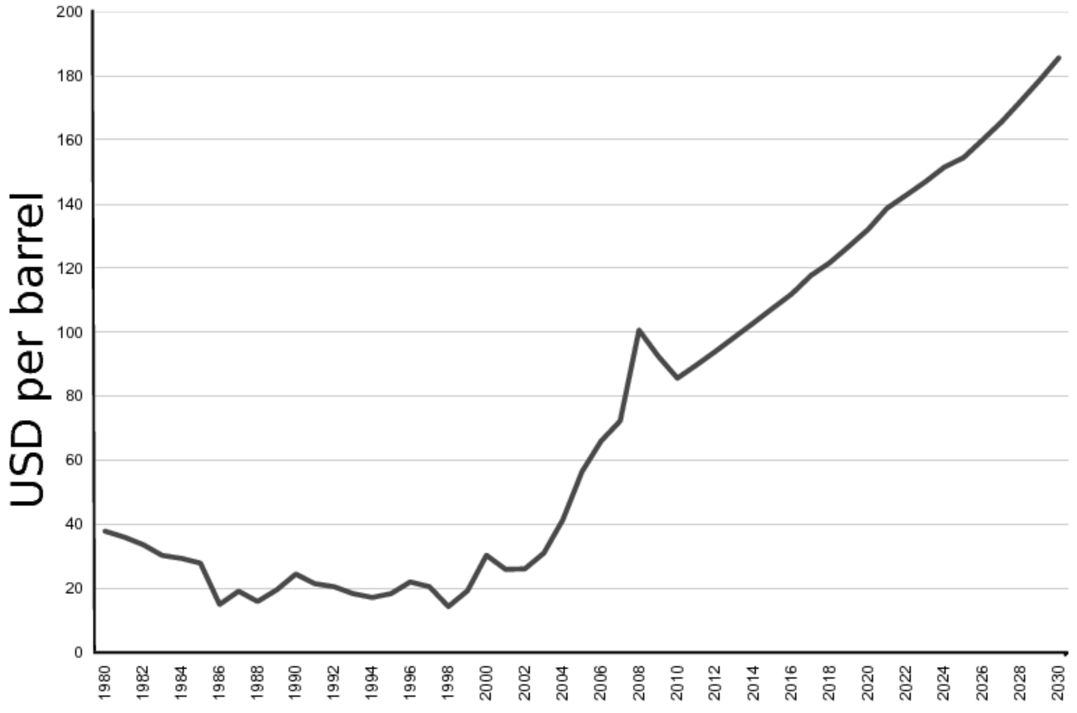


Figure 1.1: Source: Energy Information Administration (2008) International Energy Outlook 2008, June 2008

A difficult challenge is facing aeronautical engineering: due to oil shortage fuel price is rising, Fig 1.1, concerns about pollution and global warming are increasing while the FAA forecasts that air travel should continue to grow on average of 2% per year [1]. In order to cope with this situation, improvements in aeronautical engine design is necessary. The objective is to increase the engine overall efficiency, that is to say, the produced power in relation to the input fuel. Looking at a basic thermodynamic model, namely the Brayton-Joule cycle [102], Fig 1.2, it is possible to deduce a simple model for the gas turbine engine efficiency. The Brayton-Joule cycle divides the thermodynamic evolution of the gas in the engine into 4 steps:

1 isentropic compression of the gas in the compressor ($a \rightarrow b$ on Fig 1.2),

2 isobar combustion ($b \rightarrow c$),

3 isentropic expansion of the gas in the turbine ($c \rightarrow d$),

4 isobar heat rejection in the atmosphere ($d \rightarrow a$).

The Brayton-Joule cycle efficiency η is defined by the ratio of the net work $Q_{in} - Q_{out}$ and the input heat Q_{in}:

$$\eta = \frac{Q_{in} - Q_{out}}{Q_{in}} \tag{1.1}$$

Q_{in} being the heat brought in the system by combustion and Q_{out} the heat lost through exhaust. The input and exhaust heat can be expressed by enthalpy differences, the definition is chosen to be positive :

$$Q_{in} = h_c - h_b \qquad Q_{out} = h_d - h_a \tag{1.2}$$

Assuming that the specific heat capacity c_p remains constant, the efficiency can hence be expressed by:

$$\eta = \frac{c_p\left((T_c - T_b) - (T_d - T_a)\right)}{c_p\left(T_c - T_b\right)} = 1 - \frac{T_d - T_a}{T_c - T_b} = 1 - \frac{T_a\left(\frac{T_d}{T_a} - 1\right)}{T_b\left(\frac{T_c}{T_b} - 1\right)} \tag{1.3}$$

Because the transformation between $a \rightarrow b$ and $c \rightarrow d$ are isentropic we can write that

$$\left(\frac{T_a}{T_b}\right)^{\gamma} = \left(\frac{P_b}{P_a}\right)^{1-\gamma} \qquad \left(\frac{T_c}{T_d}\right)^{\gamma} = \left(\frac{P_d}{P_c}\right)^{1-\gamma} \tag{1.4}$$

Noting that the combustion and exhaust process are isobar processes, $P_a = P_d$ and $P_b = P_c$, we obtain $\frac{T_a}{T_b} = \frac{T_d}{T_c} \Rightarrow \frac{T_c}{T_b} = \frac{T_d}{T_a}$. The Brayton-Joule cycle can hence be simplified to

$$\eta = 1 - \frac{T_a}{T_b} = 1 - \frac{1}{\frac{P_b}{P_a}}^{\frac{\gamma-1}{\gamma}} \tag{1.5}$$

Hence increasing engine efficiency requires increasing the compressor pressure ratio $\frac{P_b}{P_a}$, this also means higher temperature levels.

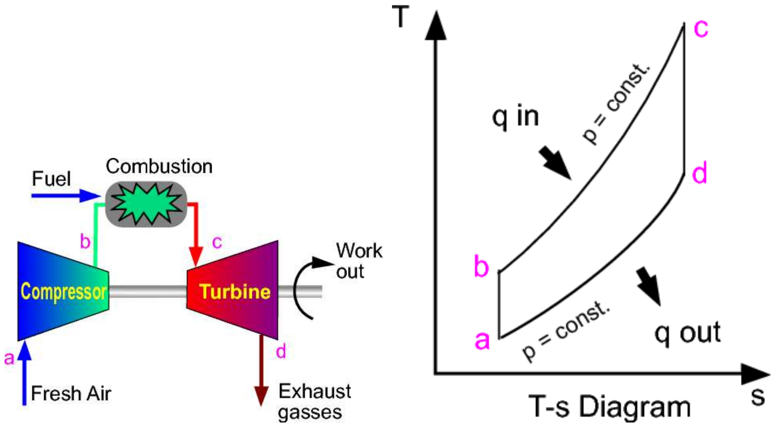


Figure 1.2: Thermodynamic cycle of gas turbine engines

Historically engines have been designed with rather low compressor pressure ratios. At low compressor ratios engines are not very efficient but because of the low combustion temperature levels the design remains relatively simple. With the need to increase engine efficiency the compressor pressure ratio has increased, Fig. 1.3, leading to higher temperatures, notably combustion temperatures. Indeed looking at the first industrial usage of a gas turbine engine, in 1939 at Neuchâtel Switzerland [125]

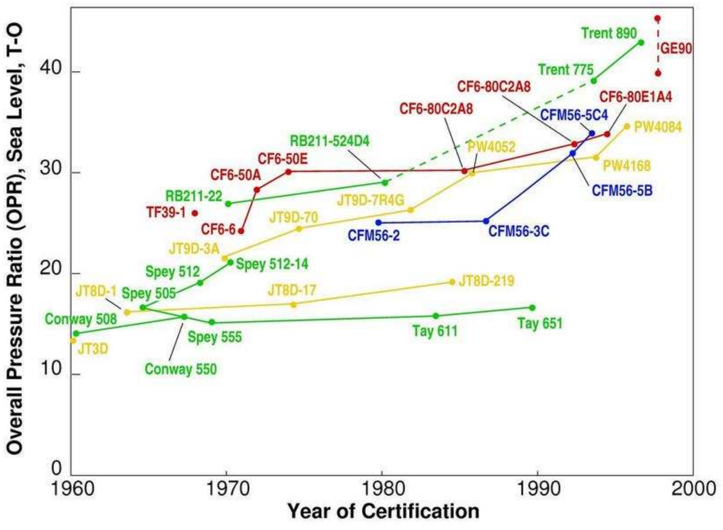
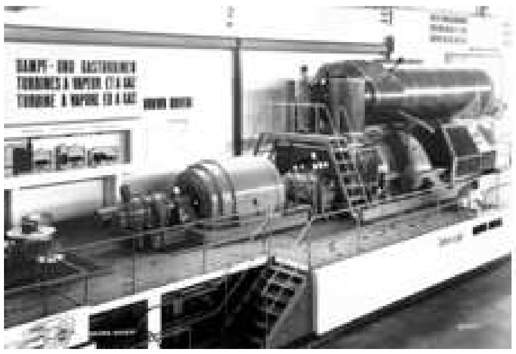


Figure 1.3: Gas turbine engine pressure ratio trends (www.mit.edu - Jane's Aeroengines, 1998)

for electrical power generation, Fig. 1.4. That gas turbine had an output power of 4MW and its highest temperature obtained within the thermodynamic cycle was 538°C [125]. At that time, engineers willing to improve the generator's efficiency increased the highest temperature to 648°C [125]. The efficiency increased from 18% to 23% [125]. By contrast, modern gas turbines such as industrial generators or aeronautical engines, Fig. 1.5, generally burn kerosene at temperatures above 2000°C.



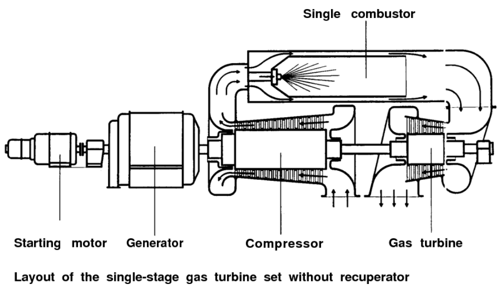

Single combustor
Starting motor
Generator
Compressor
Gas turbine
Layout of the single-stage gas turbine set without recuperator

Figure 1.4: Neuchâtel first industrial gas turbine

Unfortunately such temperatures are not compatible with the materials used to build engines: at these temperatures the metallic parts of the engine melt. To prevent the combustor walls and the turbine from having their life span shortened, the engine parts have to be cooled down. Designing efficient cooling systems relies on knowing where the hot spots are. This knowledge has long been based on engineer intuitions and expensive experiments with trial and error tests.

Today, turbine experts commonly acknowledge that computer simulation is a very promising path for optimization, which can reduce costs and diminish the duration of the design process. Up to today most conjugate heat transfer simulations have essentially relied on Reynolds Average Navier Stokes (RANS) [114, 90]. While RANS simulations become more and more accessible, its accuracy remains limited by the quality of its models. On the contrary, Large Eddy Simulation [91, 99] (LES) accuracy is far less limited by its models provided that the meshes used are fine enough [33]. LES can therefore be seen as a good trade off between high accuracy Direct Numerical Simulation [82] (DNS) and low computational cost RANS, Fig 1.6. LES computations however remain a great challenge notably in the High Performance Computing context (HPC) since LES is far more expensive than RANS.

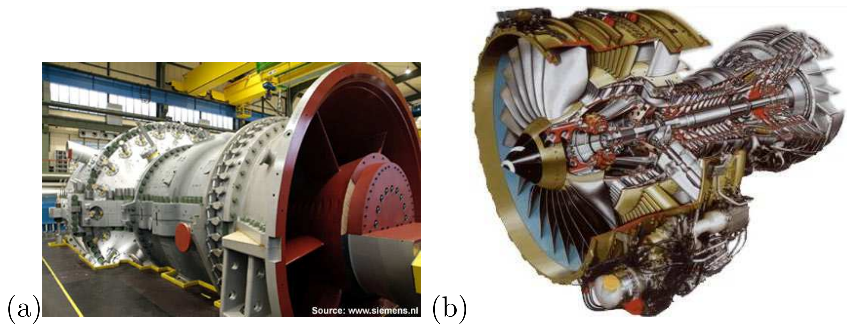


Figure 1.5: (a) An industrial gas turbine(Siemens), (b) The cfm-56 Turbofan Engine (Snecma)

But noting that LES is less limited by closure models it can take advantage of massively parallel super computers to increase prediction accuracy, Fig 1.7, which is not the case for RANS simulation. With the increase of computational power, LES simulations become accessible for specific components of gas turbines [43, 13, 14, 106]. However these stand-alone simulations and solutions now face a new challenge: to improve the quality of the results, new physics must be introduced with specific and distinct numerical models. For example, in the context of multi-component simulations, further improving the accuracy of turbine wall models is of limited interest if wall temperature boundary conditions are still set approximately. Hence the next milestone to improve simulation accuracy is multi-physics simulation.

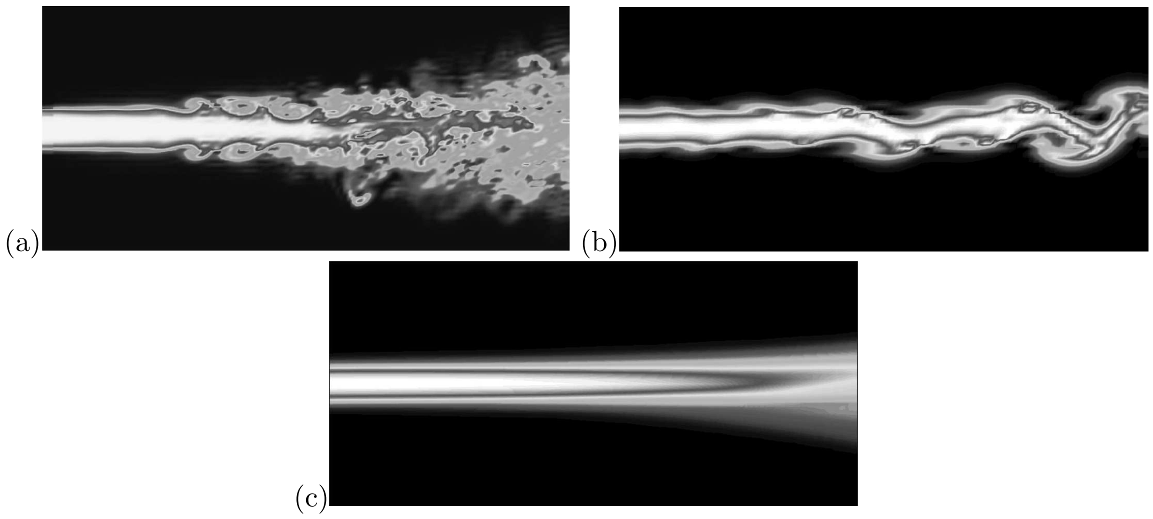


Figure 1.6: Comparison of flames using DNS (a), LES (b), RANS (c)

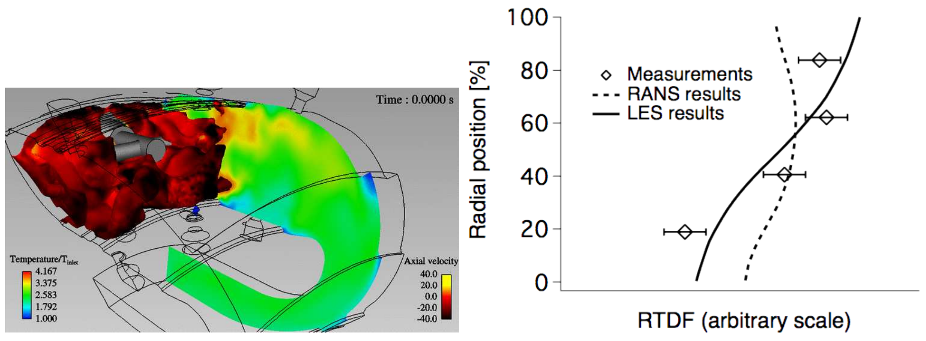


Figure 1.7: Example of LES outperforming RANS in terms of accuracy

1.2 Multi-physics in computer simulation

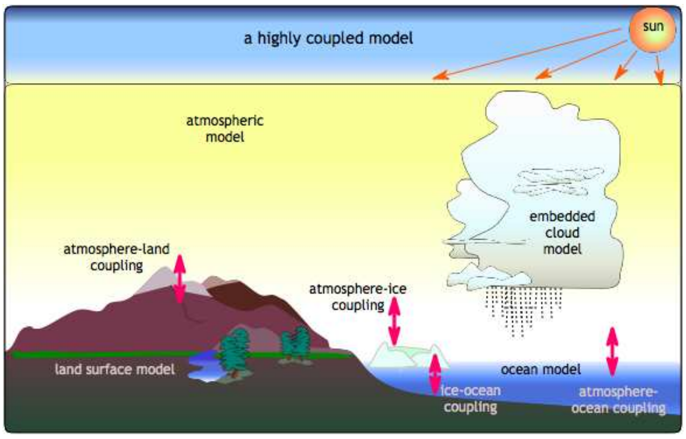


Figure 1.8: Coupled climate model

Multi-physics is a recent transverse discipline which has been pioneered essentially by the climate community. Climate models comprise many interdependent models: one for each key component of climate modeling, e.g. the atmosphere, the ocean, the biosphere, the cryosphere, etc... (Fig 1.8) Each of these models require boundary conditions from another model and in turn provide data to the other models input. At the origin these models were developed independently by different specialists. Those models were thus run in isolation using data known a priori for the boundary conditions: measurements, results from other simulations or rough approximations of other models. Further improving each model's accuracy independently was of limited interest as long as the boundary conditions were fixed approximately. On the contrary coupling the different models could lead to improved predictability of the global model. One of the first notable coupled simulations was performed by Boville and Gent [16] which showed more realistic variability in simulation results. Similar work is carried out within CERFACS GLOBC team [95].

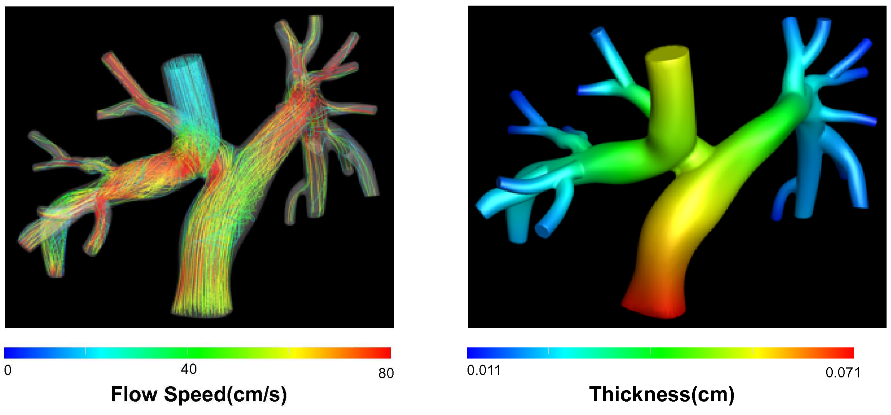


Figure 1.9: FSI in Hemodynamics: investigating the interaction between deformable arteries and blood flow [49]

An other field where many different physics interact is the new field of computational biology. An illustration of the multi-disciplinary aspect of this field is in hemodynamics, Fig 1.9: Fluid Structure Interaction (FSI) may be used to study the interaction between the deformable walls of the arteries and the blood flow. This new field of study is clearly at the cross roads of many very different disciplines: biology, fluid and structural dynamics. Mathematical models are currently being developed to solve this problem [67, 123, 49].

FSI is a fundamental problem for mechanical and structural engineering. Indeed failing to evaluate the interaction between the fluid flow and the mechanical structures can impair the time of life of those structures, and in the worse case even result in their destruction: a notable example is the accident of the Tacoma narrow bridge in 1940 [18]. Another notable FSI problem is the Pogo effect which appears in liquid engine rockets: a vicious circle between combustion instability in the engine, the rocket structure vibration and the engine fuel input generates a self exciting vibration mode. This almost resulted into complete failure on the unmanned Apollo 6 mission. A more contemporary application of FSI is in wind turbine rotor design: atmospheric turbulence generates a time varying pressure field across the blades, Fig 1.10. Hence generating vibrations on the blades leading to increased noise and reduce life span of the wind turbine components [85, 9]. Therefore investigating this matter is of great importance to improve design methods in mechanical engineering.

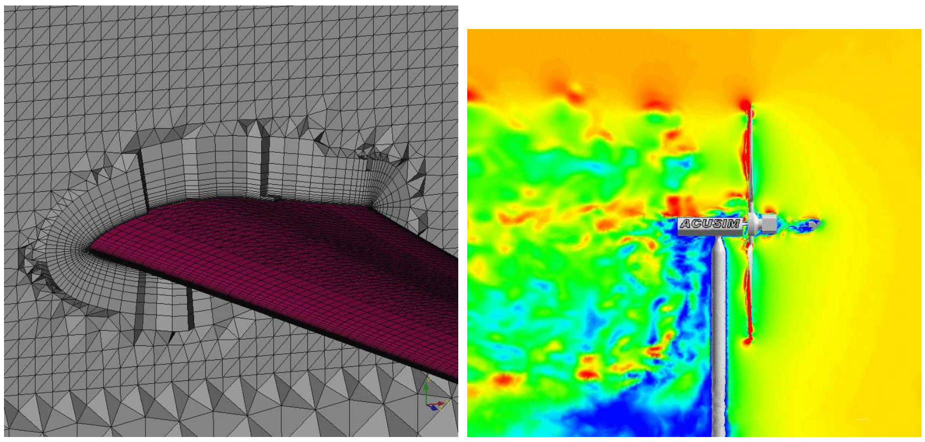


Figure 1.10: FSI for wind turbine design (ACUSIM - http://www.acusim.com/html/apps/windTurbFSI.html)

Considering design optimization, influence of heat stresses on metallic structures is of fundamental importance: in the context of gas turbine engineering, a critical problem is to predict the temperature levels within the combustor or the turbine blades. Even though the combustor walls and turbine blades may use special thermal coating, they can not withstand the heat levels generated by the flame without cooling systems. These cooling systems rely on complex dilution jets, cold air films, and porous materials. Maintaining the life span of the engine components requires evaluating the efficiency of these cooling systems by solving the interactions between the flow and solid domain, i.e. solving the conjugate heat transfer problem. In other contexts incorrect prediction of the temperature levels may have critical consequences. An application where the conjugate heat transfer problem is even more critical is hypersonic space reentry [81].

1.3 State of the art of multi-physic simulations and solutions

Multi-physics simulation was first obtained by manual transfers of simulation results into input parameters for other simulations and iterating this process until convergence. Slightly more advanced methods have then been developed in order to automate this process using basic tools such as shell scripts, python ,etc... Such methods remain very popular in the industry, however these methods are generally developed for a target application using ad-hoc tools and methods to transfer data from the different formats and numerical discretizations which make them difficult to upgrade and maintain on the long-term, Fig 1.11.

Therefore software dedicated to multi-physics has been developed. These software can be categorized into two main classes: on the first hand there are all-in-one solvers, on the other hand there are multi-physic couplers. The first method implies building a set of solvers for each physic which all share the same computational structure and can therefore run all together within the same process.

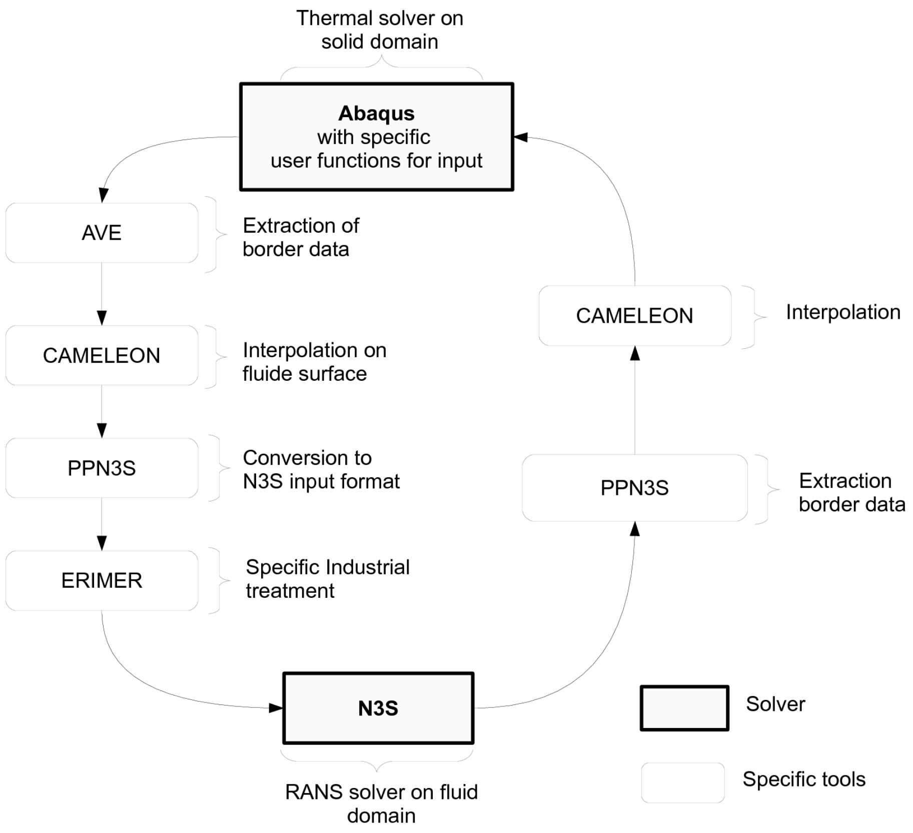

Thermal solver on solid domain
Abaqus with specific user functions for input
AVE
Extraction of border data
CAMELEON
Interpolation on fluide surface
PPN3S
Conversion to N3S input format
ERIMER
Specific Industrial treatment
N3S
RANS solver on fluid domain
PPN3S
Extraction border data
CAMELEON
Interpolation
Solver
Specific tools

Figure 1.11: N3S-ABAQUS Coupled chain developed at Snecma

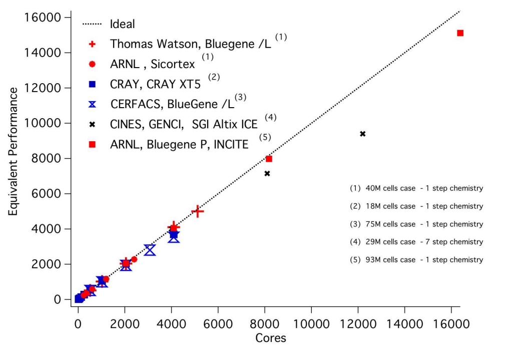
Generally these solvers are based on finite element discretization since it is the most general way to describe a large range of physical phenomena. While this concepts seems appealing its cost is generally loss of performance and scalability for specific physical modules such as unsteady LES solvers which rely on specific data structures and solvers for optimal performance. A clear example is the popular solver COMSOL [25] which contains finite element method based modules for physics ranging from fluid dynamics, acoustics, structural mechanics, heat transfer to electro-chemistry. But this all-in-one solver has not demonstrated interesting scalability properties yet: roughly 80% efficiency on 24 processors [26]. Therefore this tool is not suited for intensive computational tasks such as LES simulation.

The other path is to reuse already existing state of the art solvers and to interface them using a dedicated software named a code coupler. The most popular coupler used actually in the CFD community is MpCCI [53]. This coupler is already interfaced to the most popular commercial solvers used in industry. Considering MpCCI from a HPC perspective it is clear that flexibility is the key choice driving its development: MpCCI communicates with the simulations solvers using standard TCP/IP sockets [112] instead of using the more HPC specific oriented communication library MPI [44]. Therefore MpCCI is almost not intrusive and can handle heterogeneous computing environments, on the other hand using standard TCP/IP communications instead of MPI reduces greatly communication performance in terms of bandwidth and latency. Also MpCCI is based on a client server structure which is limiting for massively parallel applications: each process communicating with MpCCI within a simulation code needs a dedicated MpCCI server. In a massively parallel context a simulation running on a thousand processors would require a thousand MpCCI servers which is clearly not the most suited choice. Another method is to add internal communications within the solver and transfer all the data to a master processor which communicates with MpCCI. This solution however implies centralization which is generally a limitation to scalability (see part 11).

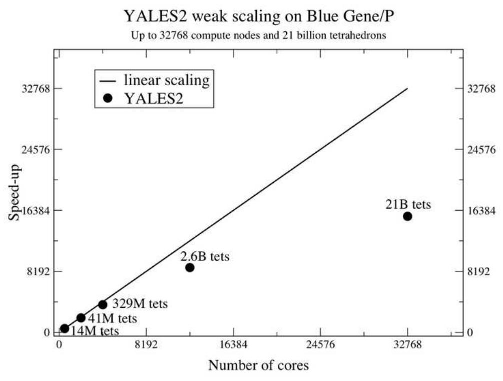
A popular open source coupler is Open-Palm [86] developed by CERFACS and ONERA. Some parts of the methods and concepts established during this thesis are transferred progressively to this coupler. Open-Palm has been designed differently than MpCCI: all the solvers run in the same MPI environment, therefore allowing higher communication performance. A consequence is that it is harder to interface a code to this coupler because sharing the same MPI environment generally requires compiling the code for a specific environment which is only possible if the source code is available. New developments allowing to communicate with closed source codes have been added. However in its actual version this coupler relies on a client server structure which may be seen as a limitation for massively parallel environments.

These different solutions imply different types of computations. All-in-one solvers are generally reserved for relatively small cases such as prototype simulations. Couplers such as MpCCI have been extensively used with steady state solvers such as Fluent [5], Abaqus [46], StarCD [24], etc... The goals of such simulations can be obtaining a converged solution or in some more advanced cases solving a multi-physical optimization problem. The low communication performance of such couplers is not a great penalty because generally steady state solvers have long computation times for each iteration and the aggregate converged solution is obtained after a relatively low count of coupling iterations.

The introduction of unsteady solvers such as LES solvers modifies clearly the problem. Such solvers generally require much more computational power and rely essentially on parallelization. Solvers such as AVBP [104] or YALES [76, 75] can scale linearly over thousands of cores almost perfectly with relatively low iteration execution times, Fig 1.12. Also because they are unsteady solvers different strategies have to be imagined to couple them: their response to a set of boundary conditions is not unique and depends greatly on time. Signal sampling issues have thus to be considered when coupling such solvers. A scalable coupler with low latency data transfers is mandatory to perform code coupling using such solvers without degrading their performance.



(a)



(b)

Figure 1.12: Scaling curves of AVBP (a) and YALES (b)

Note that simulations have already been attempted using Open-Palm's predecessor Palm [37, 4]. The two main contributions are a turbine blade calculated by Duchaine et al. [32], and a full combustion chamber by Amaya et al. [2]. Both simulations remain essentially demonstrator applications. In the first case the simulation was performed by loosely coupling the solid and the fluid simulation, this methodology was then applied to the combustion chamber. However the simulation demonstrated convergence problems of the conjugate heat transfer problem due to the unsteadiness brought by combustion. Converging a conjugate heat transfer problem while undergoing unsteady temperature fluctuations produced by a flame is investigated within this thesis. Also in that version of Palm [37, 4] no unstructured grid interpolation support was provided, hence a solution to this problem is proposed. Finally the communication schemes used are based on a centralized client server scheme, this is a bottleneck for massively parallel applications on the long term. A direct processor to processor coupling communication scheme is hence developed.

1.4 The goal: Coupling unsteady LES solver in massively parallel environments, application to a conjugate heat transfer case

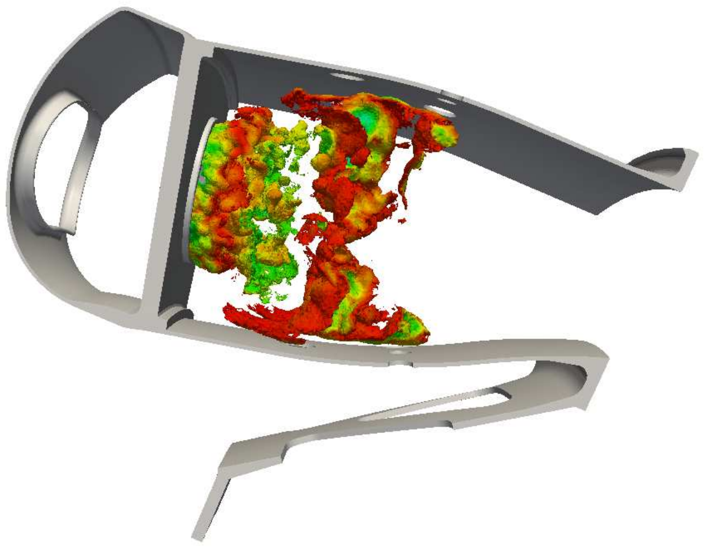


Figure 1.13: An aeronautical burner

The work presented throughout this thesis attempts to propose a complete methodology to perform multi-physic calculations relying on unsteady solvers in the context of *high performance computing* (HPC). The methodologies developed in this thesis are applied to a conjugate heat transfer problem relying on LES but should be extensible to other types of code coupling problems. Efforts have been focused to challenge the numerical and computational problems inherent to code coupling in the HPC context. The solutions to these problems are presented within this work and have been implemented within a multi-physic coupling library presented in appendix A.2. This library has been used to produce the demonstrator application. This implies complex development because such software must be able to adapt to the solver's numerical needs, maintain solvers scalability in HPC environments, and be portable.

Also the issues specific to industrial calculations such as complex geometry handling has been taken into account: the work has been developed with a target application which is the aeronautical combustion chamber, Fig 1.13. Indeed setting up a standard industrial LES means managing a complex mesh with approximately 70 boundary conditions and hundreds of parameters. Coupling such a simulation to a thermal solver implies definition and matching of geometrical interfaces, matching boundary conditions and even more parameters. This brings to a less scientific but nonetheless important aspect of this work which should also be taken into account: complex computation setup. In order to ease this task (added precomputation checks and diagnostic tools) a tool capable of visualizing different multi-physical computations and their specific features has been developed. This tool is briefly presented in A.3.

Due to the nature of this industrial application, no experimental data was available to validate the target simulation calculations so it can only be seen as a demonstrator case. The results of this thesis are more in the methods and algorithms developed to produce the demonstrator application than in actual the computational results.[1]

[1]Indeed software based on the methods explained throughout this thesis have been used by others for different multiphysic problems.

The outline of this document is

- Part I First the different physics and their solvers, namely AVBP and AVTP, are presented and validated on some basic test cases.
- Part II The conjugate heat transfer problem relying on an unsteady LES is investigated. A methodology is proposed and its convergence and stability are assessed.
- Part III The numerical methods used for unstructured grid interpolation are explained. The discussion starts by the fundamentals of interpolation and explains the actual methods implemented within the coupled application.
- Part IV The issues specific to the HPC aspect are treated and a scalable method for code coupling is proposed.
- Part V The results obtained on the demonstrator are presented.

Part I

Physical modelisation

Table of Contents

Nomenclature

α	Material heat diffusivity
ϵ	Rate of energy dissipation
λ	Material heat conductivity
μ	Dynamic viscosity
ν	Kinematic viscosity
ρ	Fluid density
c	Material specific heat capacity
C_p	Specific heat capacity and constant pressure
C_S	Smagorinsky constant
C_v	Specific heat capacity and constant volume
E	Energy
h_s	Sensible enthalpy of specie k
K	Von Karman constant
L	Length of domain
L_T	Turbulent length scale
P	Pressure
Pr	Molecular Prandtl number
Pr_t	Turbulent Prandtl number
Q_j	Reaction rate of chemical reaction j
R	Ideal gas constant
r	Specific ideal gas constant
S_L	Flame laminar velocity
S_{ij}	Rate-of-strain tensor
Sc_k^t	Turbulent Schmidt number Sc_k^t of specie k
T	Temperature
u_i	Velocity in direction i
W_k	Molecular weight of specie k
X_k	Molar fraction of specie k
Y_k	Mass fraction of specie k

In this thesis multiphysics simulation relies on independent solver coupling: well proven and optimized solvers are used to simulate each physic and additional code allows them to communicate. Before considering the coupled problem it is first important to consider each solver. Also to understand the different issues brought by the target application, the configuration for each physic is also presented.

Chapter 2

Fluid Simulation

In this chapter the equations governing the flow in the fluid domain are presented. First the equations governing a compressible reactive multi-specie flow are introduced, then the Large Eddy Simulation concept is introduced with the filtered version of those equations. Finally basic validation cases using the CERFACS/IFP fluid solver AVBP are described. AVBP is a compressible multi-specie DNS/LES solver capable of simulating accurately turbulent reactive flows on hybrid unstructured grids. A more thorough description of AVBP is available in the official AVBP handbook.

2.1 Compressible Navier Stockes equations of multi species flows

The equations presented throughout this section govern a compressible fluid. They are expressed in Einstein's summation convention, except for the k index which represents the species index. Species summations are expressed explicitly.

$$\frac{\partial \rho\, u_i}{\partial t} + \frac{\partial}{\partial x_j}(\rho\, u_i\, u_j) = -\frac{\partial}{\partial x_j}[P\,\delta_{ij} - \tau_{ij}], \tag{2.1}$$

$$\frac{\partial \rho\, E}{\partial t} + \frac{\partial}{\partial x_j}(\rho\, E\, u_j) = -\frac{\partial}{\partial x_j}[u_i\,(P\,\delta_{ij} - \tau_{ij}) + q_j] + \dot{\omega_T}, \tag{2.2}$$

$$\frac{\partial \rho Y_k}{\partial t} + \frac{\partial}{\partial x_j}(\rho Y_k\, u_j) = -\frac{\partial}{\partial x_j}[J_{j,k}] + \dot{\omega_k}. \tag{2.3}$$

These equations describe the conservation of respectively momentum, energy and mass of specie k and depend on the following variables:

- u_i the velocity in direction i,
- ρ the density of the mixture,
- E the total energy of per unit mass of the mixture,
- Y_k the mass fraction species k.

And on the following quantities which are detailed in the following paragraphs:

- $J_{j,k}$ is the diffusive flux of specie k in direction i,

- τ_{ij} is the viscous stress tensor,
- q_j is the heat flux,
- P the pressure which is obtained using the equation of state,
- $\dot{\omega}_T$ and $\dot{\omega}_k$ are respectively the heat and chemical production rates.

The diffusive flux of species

Mass conservation in multi-species flows implies that:

$$\sum_{k=1}^{N} Y_k V_i^k = 0 \tag{2.4}$$

Where Y_k is the mass fraction of specie k and V_i^k are the components of the diffusion velocity of species k. They are often expressed as a function of the species gradients using the Hirschfelder Curtis approximation:

$$X_k V_{k,i} = -D_k \frac{\partial X_k}{\partial x_i} \tag{2.5}$$

Where X_k is the molar fraction of species k: $X_k = Y_k \frac{W}{W_k}$ and D_k is the diffusion coefficient of species k. In terms of mass fraction, the approximation may be expressed as:

$$Y_k V_i^k = -D_k \frac{W_k}{W} \frac{\partial X_k}{\partial x_i} \tag{2.6}$$

Summing Eq.(2.6) over all species shows that the Hirschfelder Curtis approximation does not conserve mass. In order to achieve this, a correction diffusion velocity V_c is added to the convection velocity to ensure global mass conservation [89] as:

$$V_i^c = \sum_{k=1}^{N} D_k \frac{W_k}{W} \frac{\partial X_k}{\partial x_i} \tag{2.7}$$

The diffusive species flux for each species k is hence modeled by:

$$J_{i,k} = -\rho \left(D_k \frac{W_k}{W} \frac{\partial X_k}{\partial x_i} - Y_k V_i^c \right) \tag{2.8}$$

The viscous stress tensor and rate of strain tensors

The viscous stress tensor τ_{ij} is modeled by

$$\tau_{ij} = 2\mu \left(S_{ij} - \frac{1}{3}\delta_{ij} S_{ll} \right) \tag{2.9}$$

Where μ is the shear viscosity and S_{ij} is the rate-of-strain tensor

$$S_{ij} = \frac{1}{2} \left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j} \right) \tag{2.10}$$

The heat flux vector

The heat flux vector for a multi-species is composed by the sum of the conductive flux and a flux due to the heat transport by species diffusion. Hence the heat flux writes:

$$q_i = -\lambda \frac{\partial T}{\partial x_i} - \rho \sum_{k=1}^{N} J_{i,k} h_{s,k} \tag{2.11}$$

Where λ is the heat conduction coefficient of the mixture and $h_{s,k}$ the sensible enthalpy of species k.

The equation of state

The model is closed using the perfect gas equation of state which reads:

$$P = \rho r T \tag{2.12}$$

Where r is defined by $r = \frac{R}{W}$ with W being the mixture's molecular weight (where $R = 8.314 J.mol^{-1}.K^{-1}$). W can be expressed using

- the molar fractions X_k and the molecular weight W_k of species k by

$$W = \sum_{k=1}^{N} X_k W_k \tag{2.13}$$

- the mass fractions Y_k and the molecular weight W_k of species k by $\frac{1}{W} = \sum_{k=1}^{N} \frac{Y_k}{W_k}$,

The heat capacities C_p and C_v of the gas mixture depend on the composition.

$$C_p = \sum_{k=1}^{N} Y_k C_{p,k} \tag{2.14}$$

$$C_v = \sum_{k=1}^{N} Y_k C_{v,k} \tag{2.15}$$

The heat and chemical source terms and reaction rate

The source terms in the energy and mass fraction conservation equations are respectively ω_T and ω_k. These source terms are linked via Eq.(2.16)

$$\dot{\omega_T} = -\sum_{k=1}^{N} \dot{\omega_k} \Delta h_{f,k}^0 \tag{2.16}$$

where $\Delta h_{f,k}^0$ is the formation enthalpy of species k.

The species k source term is given by Eq.(2.17):

$$\dot{\omega_k} = W_k \sum_{j=1}^{M} (\nu_{kj}'' - \nu_{kj}') Q_j \tag{2.17}$$

where ν'_{kj} and ν''_{kj} are the stochiometric coefficients and Q_j the reaction rate of chemical reaction j, Eq. (2.18).

$$\sum \nu'_{kj} R_k \rightleftarrows \sum \nu''_{kj} R_k \tag{2.18}$$

With R_k the species involved in this reaction. The reaction rate in AVBP is given by an Arrhenius law.

2.2 Large Eddy Simulation

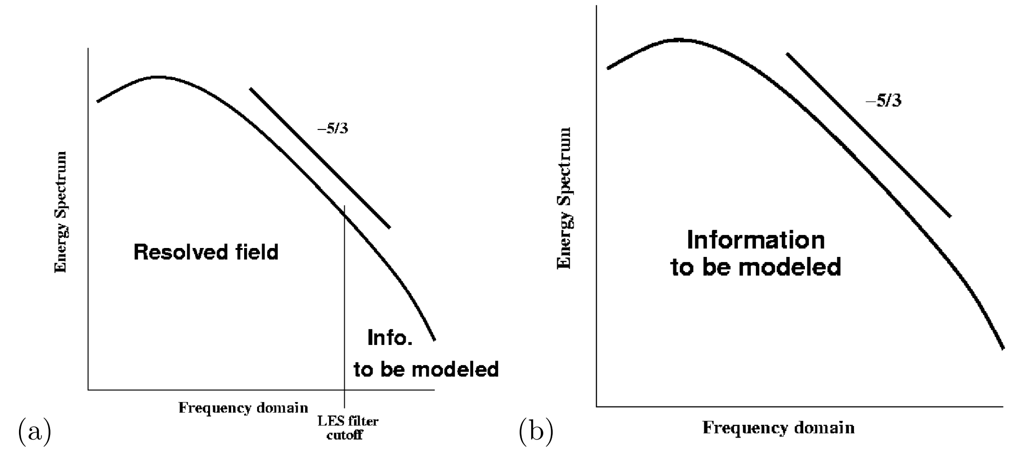


Figure 2.1: Idealized spectrum comparison LES (a) and RANS (b)

The equations governing LES are obtained by applying spatial low pass filtering operators to the governing equations. Mathematically this can be described by calculating the convolution of the exact quantity Q using a filter noted here G_Δ.

$$\bar{Q} = \int_D Q(x')G_\Delta(x - x')dx' \tag{2.19}$$

This operation filters out the small scales while keeping the large scales. In this example G_Δ filters out the turbulent motions of wave number higher than $k_c = \frac{\pi}{\Delta}$[1].

The small scales are thought to be independent of the macroscopic features of the flow [92], meaning that models used for the small scales can be applied to complex flows without particular tuning. The large scales still need to be resolved, but this clearly reduces the simulation cost compared to a DNS while maintaining high accuracy, Fig 2.1(a).

This is a fundamental difference with RANS simulation: in RANS simulation all the turbulence scales are modeled, Fig 2.1(b), hence allowing turbulent simulation at relatively cheap computational costs. But because the large scales are clearly dependent on the macroscopic properties of the flow, notably the flow geometry, RANS simulations generally require complex model tuning for each case.

For variable density flows applying directly the filtering operator to the governing equations yields products of fluctuations between density and other variables which is complex to solve. In order to avoid these terms a mass weighted filtering procedure is applied called Favre filtering:

$$\bar{\rho}\tilde{Q} = \int_D \rho Q(x')G_\Delta(x - x')dx' \tag{2.20}$$

[1] In solvers such as AVBP the filtering procedure is implicitly provided by the mesh, hence the local grid spacing determines the filter cutoff length Δ.

Using this filtering procedure the LES equations obtained are:

$$\frac{\partial \overline{\rho}\, \widetilde{u_i}}{\partial t} + \frac{\partial}{\partial x_j}(\overline{\rho}\, \widetilde{u_i}\, \widetilde{u_j}) = - \frac{\partial}{\partial x_j}[\overline{P}\, \delta_{ij} - \overline{\tau_{ij}} - \overline{\tau_{ij}}^{sgs}] \tag{2.21}$$

$$\frac{\partial \overline{\rho}\, \widetilde{E}}{\partial t} + \frac{\partial}{\partial x_j}(\overline{\rho}\, \widetilde{E}\, \widetilde{u_j}) = - \frac{\partial}{\partial x_j}[\overline{u_i\,(P\,\delta_{ij} - \tau_{ij})} + \overline{q_j} + \overline{q_j}^{sgs}] + \overline{\omega_T} \tag{2.22}$$

$$\frac{\partial \overline{\rho}\, \widetilde{Y_k}}{\partial t} + \frac{\partial}{\partial x_j}(\overline{\rho}\, \widetilde{Y_k}\, \widetilde{u_j}) = - \frac{\partial}{\partial x_j}[\overline{J_{j,k}} + \overline{J_{j,k}}^{sgs}] + \overline{\omega_k} \tag{2.23}$$

This filtering operation yields unclosed terms due to the nonlinearity of the Navier Stockes equations noted with the super script *sgs*.

The filtered viscous stress tensor is approximated as follows:

$$\overline{\tau}_{ij} = \overline{2\mu s_{ij} - \frac{2}{3}\mu s_{ll}\delta_{ij}} \approx 2\overline{\mu}\widetilde{s_{ij}} - \frac{2}{3}\overline{\mu}\widetilde{s_{ll}}\delta_{ij} \tag{2.24}$$

where

$$\widetilde{s}_{ij} = \frac{1}{2}\left(\frac{\partial \widetilde{u}_j}{\partial x_i} + \frac{\partial \widetilde{u}_i}{\partial x_j}\right) \tag{2.25}$$

Likewise the species diffusive flux and heat flux are approximated:

$$\overline{J_{i,k}} = -\overline{\rho\left(D_k \frac{W_k}{W}\frac{\partial X_k}{\partial x_i} - Y_k V_{k,i}^c\right)} \approx -\overline{\rho}\left(\overline{D_k}\frac{W_k}{W}\frac{\partial \widetilde{X_k}}{\partial x_i} - \widetilde{Y_k}\widetilde{V_{k,i}}^c\right) \tag{2.26}$$

$$\overline{q}_i = \overline{-\lambda\frac{\partial T}{\partial x_i} + \sum_k J_{j,k}h_{s,k}} \approx -\overline{\lambda}\frac{\partial \widetilde{T}}{\partial x_i} + \sum_k \overline{J_{i,k}}\widetilde{h_{s,k}} \tag{2.27}$$

2.2.1 Sub-grid closures

Turbulence increases mixing of momentum, heat and species. A basic modeling idea then consists in representing the unclosed terms as diffusive contributions with an associated turbulent viscosity μ_t (eddy-viscosity models). Under this assumption, the sub-grid stress tensor may be rewritten as:

$$\overline{\tau}_{ij}^{sgs} = -\overline{\rho}(\widetilde{u_i u_j} - \widetilde{u}_i\widetilde{u_j}) = 2\overline{\mu}_t\widetilde{s_{ij}} - \frac{2}{3}\overline{\mu}_t\widetilde{s_{ll}}\delta_{ij} \tag{2.28}$$

This supposes that the principal axes of the strain rate tensor are aligned with those of the sub-grid stress tensor which is not fulfilled in general [101]. The turbulent viscosity may be derived from algebraic relations or through the resolution of additional transport equations. A model for turbulent viscosity is detailed in subsection 2.2.2.

The sub-grid species flux is modeled in an analogous manner to the sub-grid stress tensor:

$$\overline{J}_{i,k}^{sgs} = \overline{\rho}\left(\widetilde{u_i Y_k} - \widetilde{u}_i\widetilde{Y_k}\right) \tag{2.29}$$

$$\overline{J}_{i,k}^{sgs} = -\overline{\rho}\left(D_k^t\frac{W_k}{W}\frac{\partial \widetilde{X_k}}{\partial x_i} - \widetilde{Y_k}\widetilde{V_{k,i}}^{c,t}\right) \tag{2.30}$$

with:

$$\widetilde{V_{k,i}}^{c,t} \approx \sum_k D_k^t \frac{W_k}{W} \frac{\partial \widetilde{X}_k}{\partial x_i} \tag{2.31}$$

The turbulent species diffusion is deduced from a turbulent Schmidt number Sc_k^t:

$$D_k^t = \frac{\mu_t}{\overline{\rho} Sc_k^t} \tag{2.32}$$

The constant value $Sc_k^t = 0.7$ is chosen for all species.

For the sub-grid heat flux, one obtains:

$$\overline{q_i}^{sgs} = \overline{\rho} \left(\widetilde{u_i E} - \widetilde{u}_i \widetilde{E} \right) \tag{2.33}$$

$$\overline{q_i}^{sgs} = -\overline{\lambda_t} \frac{\partial \widetilde{T}}{\partial x_i} + \sum_k \overline{J_{i,k}}^{sgs} \widetilde{h_{s,k}} \tag{2.34}$$

with:

$$\lambda_t = \frac{\nu_t \overline{c_p}}{Pr_t} \tag{2.35}$$

The turbulent Prandtl number $Pr_t = 0.6$ is also assumed constant [72].

2.2.2 Sub-grid scale models

The main task of the sub-grid scale model is to correctly reproduce the energy fluxes between resolved and unresolved turbulent scales. This involves interactions among the whole turbulence spectrum, that is to say the sub-grid scale model must ideally account for interactions between turbulent structures of different sizes as well as between structures of comparable sizes. Due to the difficulty of this task, one may only expect sub-grid scale models to be correct in the statistical sense.

Eddy-viscosity sub-grid scale models require the determination of a turbulent viscosity. As the kinetic viscosity is the product of a length and velocity and that the most energetic unresolved scales are found at the cut-off frequency k_c of the LES filter, the filter width Δ is a natural choice for the length scale of the turbulent viscosity. The characteristic velocity scale is determined from the sub-grid scale energy. The models based on an eddy viscosity assumption make different levels of simplification to obtain an estimate for this energy.

Many different sub-grid scale models exist (filtered Smagorinsky [79], dynamic Smagorinsky [11], WALE [78, 79], ...) but only the Smagorinsky model [29, 111] is presented here because it is one of the first sub-grid scale models and is probably the most popular sub-grid scale models due to its simplicity. It assumes equilibrium between production and dissipation of turbulent kinetic energy at the sub-grid scales. This assumption is justified in regions of isotropic turbulence for which the Smagorinsky model reproduces correct dissipation levels. In regions of anisotropy however, the model shows to be over dissipative as it cannot predict the occurrence of back-scatter, i.e. the instantaneous and localized back-flow of turbulent energy from smaller to larger scales. Piomelli et al. [88] showed that the failure to reproduce this phenomenon may result in wrong prediction of perturbation growth in transitional flows. It writes:

$$\nu_t = \left(C_S \overline{\Delta} \right)^2 \sqrt{2 \widetilde{s_{ij}} \widetilde{s_{ij}}} \tag{2.36}$$

Smagorinsky determined an analytical value of 0.18 for the constant C_S. However, C_S is often adjusted to the given application case and values ranging between 0.1 and 0.18 may be found in the literature.

2.2.3 Wall law model

Wall bounded flows are difficult to solve using LES. Indeed, in order to accurately compute the wall shear stresses and heat fluxes, high resolution meshes are required. An alternative commonly used is to model the wall shear stress and heat fluxes using Wall law models. In this document the basics of the law of the wall model are presented, a more thorough description of the wall models used by AVBP has been done by Schmitt [103].

First the wall units are introduced, using the wall shear stress τ_w and the fluid density ρ the friction velocity can be defined Eq. (2.37).

$$u_\tau = \sqrt{\frac{\tau_w}{\rho}} \tag{2.37}$$

The non-dimensional wall distance is defined comparing the wall distance y with the wall viscous length $\frac{u_\tau}{\nu}$, Eq. (2.38).

$$y+ = \frac{y u_\tau}{\nu} \tag{2.38}$$

The non-dimensional velocity $u+$ is defined by comparing the velocity parallel to the wall u to the wall velocity, Eq. (2.39).

$$u+ = \frac{u}{u_\tau} \tag{2.39}$$

The log-law model establishes a relation between $u+$ and $y+$:

$y+ \leq 7$ This region near the wall is called the viscous sub-layer, the variation of $u+$ is almost proportional to $y+$: $u+ = y+$

$y+ > 30$ This is the Log law region, the evolution $u+$ can be described by:

$$u+ = \frac{1}{K}\ln(y+) + 5.5 \tag{2.40}$$

where K is the Von Karman [117] constant determined experimentally to 0.41.

$7 \leq y+ \leq 30$ This region is called the buffer region, this is where the two equations combine. Strictly speaking neither the linear nor logarithmic laws can be applied here, but in practice the linear law may be used up to $y+ \simeq 11$ and then the log law is used.

Hence the kinematic model is defined with $u+$, but for conjugate heat transfer simulation a thermal model is also needed. This model is based on the Kader law [55] and the Van Driest transformation [31]. The non-dimensional temperature $T+$ follows a smooth function between the linear and the turbulent parts.

$$T+ = (\mathrm{Pr}\ y+)\, e^{\Gamma} + (\mathrm{Pr_t}\ u + +K)\, e^{\frac{1}{\Gamma}} \tag{2.41}$$

where Pr is the molecular Prandtl number, $\mathrm{Pr_t}$ is the turbulent Prandtl number ($\mathrm{Pr_t} = 0.85$), K is a constant depending exclusively on the molecular Prandtl number $K(\mathrm{Pr}) = 2(3.85\mathrm{Pr}^1/3 - 1.3) + 2.12\ln(\mathrm{Pr})$ and Γ is the Kader smooth function $\Gamma = -\frac{10^{-2}(\mathrm{Pr}\ y+)^4}{1+5\mathrm{Pr}^3 y+}$.

2.2.4 The thickened flame model for LES

Combustion simulation adds an additional difficulty: generally the laminar flame thickness δ_L^0 is smaller than the mesh size Δx. To cope with this situation a thickened flame model is introduced, i.e. the real flame (non solvable on the current mesh because it is too coarse) is replaced by an equivalent thicker flame. For laminar flows, the diffusion is increased and the reaction rates are decreased proportionally using $\mathcal{F}$ the thickening flame factor. Because of these two reciprocal modifications the proper flame speed of the non thickened laminar premixed flame is guarantied. For turbulent flows the problem is slightly more complex because turbulence wrinkles the flame front and hence increases the flame surface. The thickened flame model is not able to take into account this phenomenon at the sub-grid scale level, the reaction rate is hence underestimated. To correct this effect, an efficiency function $\mathcal{E}$, based on DNS results, has been added by Colin et al. [22]. This model modifies the species diffusion flux, energy flux, and source term equations:

- The species diffusion flux $\overline{J_{i,k}}$:

$$\overline{J_{i,k}} = -\mathcal{E}\mathcal{F}\bar{\rho}\left(\overline{D_k}\frac{W_k}{W}\frac{\partial\widetilde{X_k}}{\partial x_i} - \widetilde{Y_k}\widetilde{V_i}^c\right) \tag{2.42}$$

- The energy flux $\overline{q_i}$:

$$\overline{q_i} = -\mathcal{E}\mathcal{F}\left(\overline{\lambda}\frac{\partial\widetilde{T}}{\partial x_i} + \sum_{k=1}^{N}\bar{\rho}\left(\overline{D_k}\frac{W_k}{W}\frac{\partial\widetilde{X_k}}{\partial x_i} - \widetilde{Y_k}\widetilde{V_i}^c\right)\widetilde{h_{s,k}}\right) \tag{2.43}$$

- The reaction rate $\overline{\dot{\omega}_k}$ of the species k:

$$\overline{\dot{\omega}_k}(\widetilde{Y_k},\widetilde{T}) = \frac{\mathcal{E}\overline{\dot{\omega}_k}(\widetilde{Y_k},\widetilde{T})}{\mathcal{F}} \tag{2.44}$$

- The heat released by combustion $\overline{\dot{\omega}_T}$:

$$\overline{\dot{\omega}_T}(\widetilde{Y_k},\widetilde{T}) = \frac{\mathcal{E}\overline{\dot{\omega}_T}(\widetilde{Y_k},\widetilde{T})}{\mathcal{F}} \tag{2.45}$$

2.3 Basic validation cases

The purpose of code coupling is to use already validated solvers. It is therefore necessary to asses the capacity of AVBP to solve fluid dynamics and more specifically combustion problems. Several basic validation tests are briefly presented in this section, each demonstrating the capacity of AVBP to solve a key aspect necessary for a coupled combustion simulation in an industrial burner. The validation cases presented in this section are issued from a series of basic validation cases, named Quality Program Form (QPF), which are carried out at each major release of AVBP.

2.3.1 Turbulence validation

Flows in industrial burners are essentially turbulent, a first key aspect to investigate in AVBP is turbulence simulation. First the ability of AVBP to simulate properly turbulence has been investigated using direct numerical simulation to solve a homogeneous isotropic turbulence (HIT) case. Then AVBP Large Eddy Simulation models are tested on a lower resolution HIT case.

The following quantities are introduced:

- The two point correlation between points A and B

$$Q_{ij}(A,B) = \overline{u_i'(A)u_j'(B)} \tag{2.46}$$

- The correlation coefficients R_{ij}:

$$R_{ij}(A,B) = \frac{Q_{ij}(A,B)}{\sqrt{\overline{u'_i(A)^2}}\sqrt{\overline{u'_i(B)^2}}} \tag{2.47}$$

- The longitudinal, lateral and transverse integral length scales L^1_{11}, L^2_{22}, L^3_{33}:

$$L^l_{ij} = \int_0^\infty R_{ij}(x_l, 0, 0)dx_l \tag{2.48}$$

- The corresponding Reynolds number (linked to the integral length scale):

$$Re_{L^i_{ii}} = \frac{u' L^i_{ii}}{\nu} \tag{2.49}$$

- The characteristic time scale based on kinetic energy dissipation:

$$\tau_\epsilon = \frac{k}{\epsilon} \tag{2.50}$$

For more details on the various quantities used for HIT computations readers are referred to Boughanem [15].

For the DNS HIT a 3D box of length $L = 2.78510^{-4}m$ is discretized by 64^3 hexahedral cells. The initial field is generated using a Passot Pouquet spectrum [83] with $Re_{L^i_{ii}} = 42$. The solver is then applied and the temporal evolution of statistical quantities are extracted:

- the longitudinal, lateral and transverse integral length scales,
- the dissipation rate ϵ and kinetic energy k.

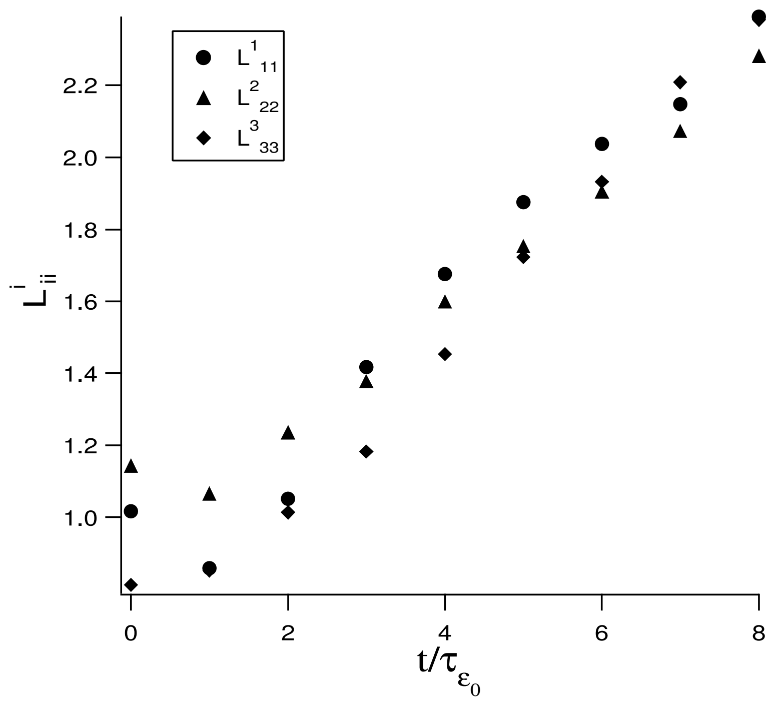


Figure 2.2: Temporal evolution of the integral scales of auto-correlation

First looking at the the evolution of the different integral length scales, Fig 2.2, it is clear that the isotropic property of turbulence is preserved:

$$L^1_{11} \simeq L^2_{22} \simeq L^3_{33} \tag{2.51}$$

Also the monotonic growth of the length scales agrees with the evolution of the turbulent length scale L_T predicted by the $k - \epsilon$ model [52, 64], Eq (2.52).

$$\frac{L_T}{L_{T0}} = \left[1 + (C_{\epsilon_2} - 1)\frac{t}{\tau_{\epsilon 0}}\right]^{\frac{2C_{\epsilon_2}-3}{2(C_{\epsilon_2}-1)}} \tag{2.52}$$

Where C_{ϵ_2} is a $k - \epsilon$ model constant [122].

The evolution of the dissipation and kinetic energy in the HIT case has been evaluated using AVBP, NTMIX [8, 15] a well proven DNS solver and the $k-\epsilon$ model. Figures 2.3 and 2.4 show good agreement between NTMIX and AVBP and a rather good agreement with the $k-\epsilon$ model for which the evolution reads (for large Reynolds number flows):.

$$\frac{k}{k_0} = \left[1 + (C_{\epsilon_2} - 1)\frac{t}{\tau_{\epsilon 0}}\right]^{-\frac{1}{C_{\epsilon_2}-1}} \tag{2.53}$$

$$\frac{\epsilon}{\epsilon_0} = \left[1 + (C_{\epsilon_2} - 1)\frac{t}{\tau_{\epsilon 0}}\right]^{-\frac{C_{\epsilon_2}}{C_{\epsilon_2}-1}} \tag{2.54}$$

Initial quantities are marked with the subscript $_0$.

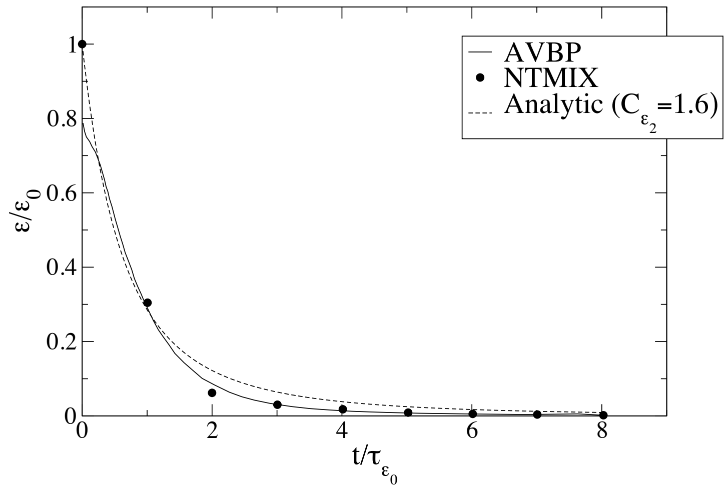


Figure 2.3: Comparison of the evolution of dissipation between AVBP, NTMIX and analytical predictions from $k - \epsilon$ model. C_{ϵ_2} is chosen to 1.6 which is a typical value for low resolution DNS

For real geometries direct numerical simulation is not yet affordable, it would need extremely refined meshes. On the other hand solving turbulence by modeling the smallest eddies and simulating the largest eddies allows to use coarser meshes. This is implemented in AVBP by several LES models, namely: Smagorinsky, Dynamic Smagorinsky, WALE. In the configuration presented in this thesis (chapter 4) the Smagorinsky model is chosen for its simplicity and because of wall models are used instead of solving the wall boundary layers. It is therefore important to at least validate AVBP's LES Smagorinsky model. A basic validation test has been carried out by comparing HIT performed using DNS to LES on a coarser mesh (32^3 points), Fig 2.5. A third simulation used to control the impact of LES models is performed using no models on the coarse grid.

This test clearly shows that the LES models in AVBP are capable of providing an accurate simulation using a coarser mesh than a DNS would require.

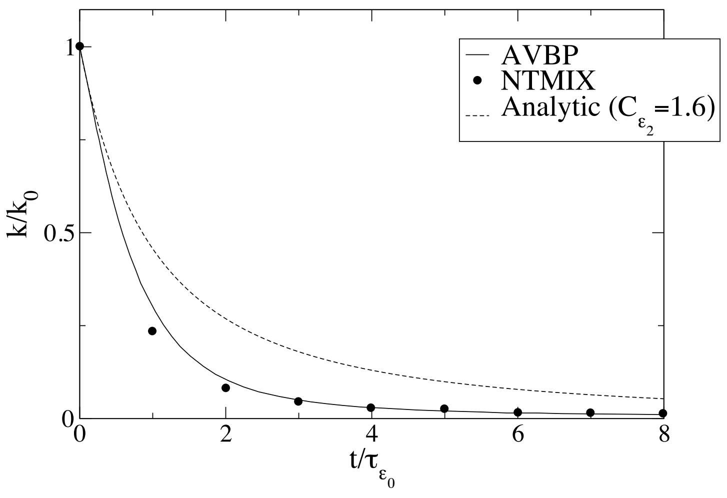


Figure 2.4: Comparison of the evolution of kinetic energy between AVBP, NTMIX and analytical predictions from $k - \epsilon$ model. C_{ϵ_2} is chosen to 1.6 which is a typical value for low resolution DNS

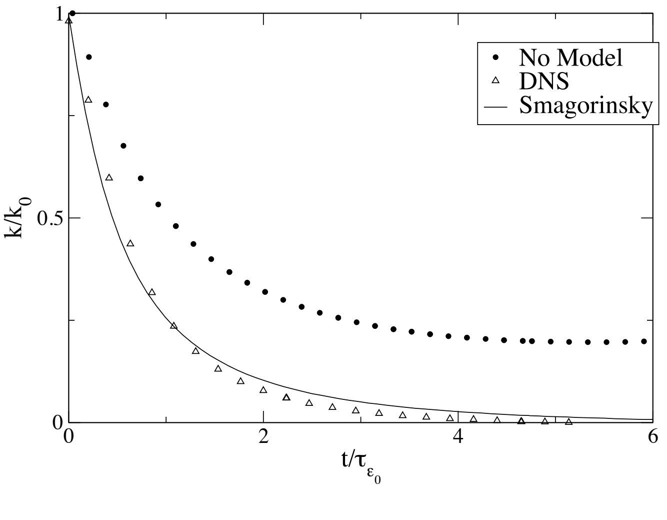


Figure 2.5: Comparison of LES and DNS on a HIT case

2.3.2 Wall model validation

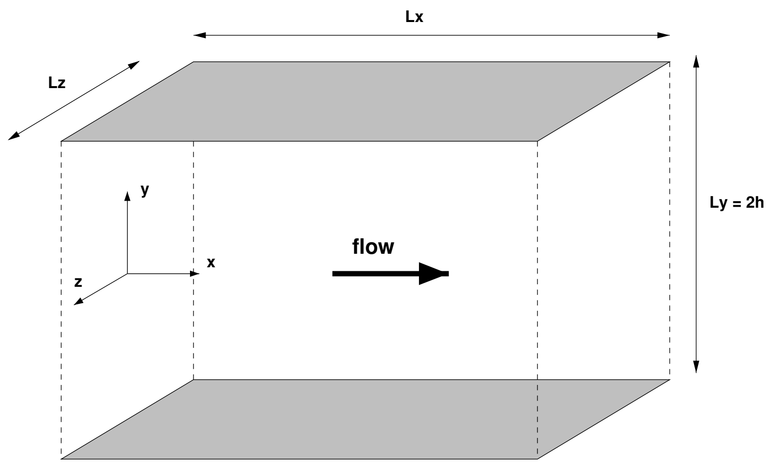
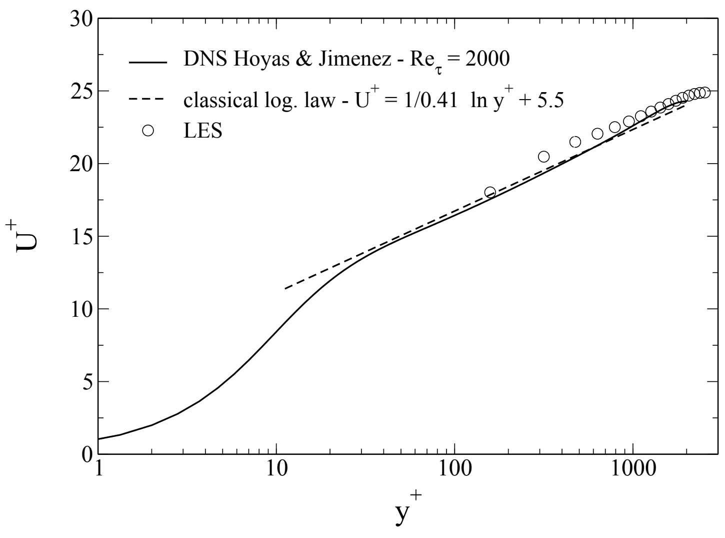


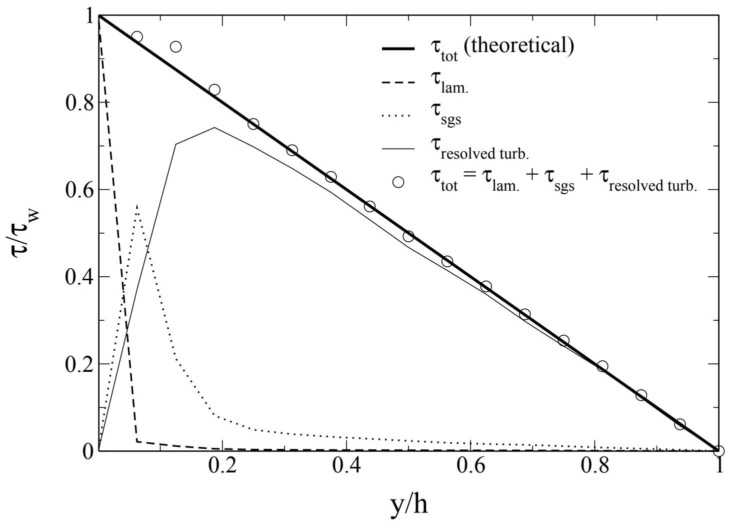
Figure 2.6: Turbulent channel diagram

Conjugate heat transfer simulation requires the ability to compute the wall heat flux from the fluid. This work relies on wall laws for this task, therefore their behavior should be investigated. A simple configuration which has the advantage of having both analytical and DNS results is the turbulent channel configuration. The channel is composed of a fluid domain between two walls separated by the distance $2h$ (each wall is at $y = \pm h$), and periodic boundary conditions in the streamwise and spanwise directions, Fig 2.6. The flow motion is obtained by imposing a pressure gradient on the streamwise direction. Wall laws are used for each wall and key wall quantities are extracted: velocity profile and wall friction, Fig 2.7, on one hand and on the other temperature profile and wall heat flux, Fig 2.8. These quantities are then compared to values obtained through DNS and analytical models: the logarithmic wall model for velocity and the Kader law for temperature.

The results presented in figure 2.7 and 2.8 are for the Lax Wendroff [65] numerical scheme. They show that the wall quantities computed with LES and DNS have a rather good agreement, for the cinematic and thermal laws. Of course this configuration is far simpler than an industrial configuration but this does show that the wall law model produces rather good predictions.

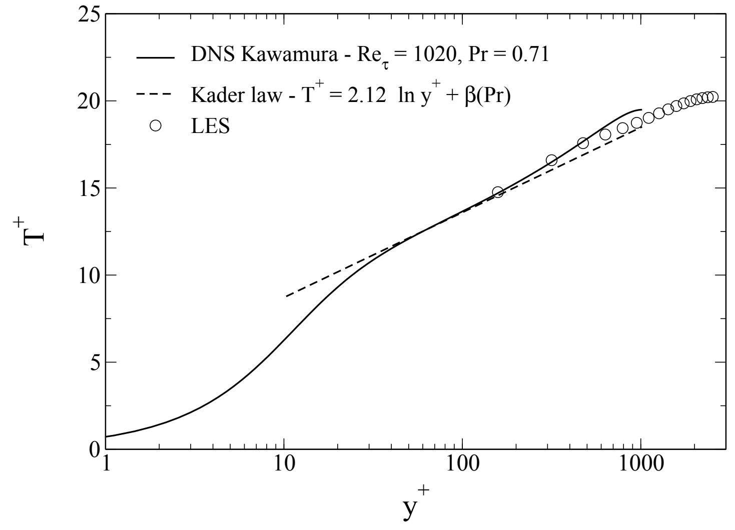


(a)

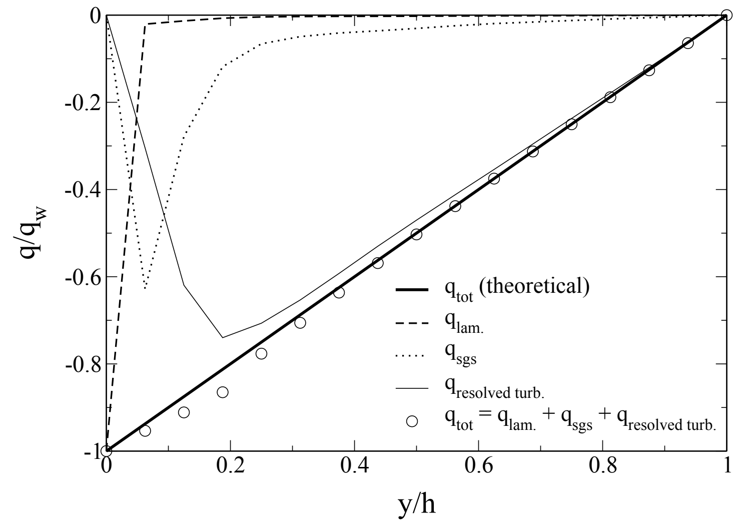


(b)

Figure 2.7: Turbulent channel diagram



(a)



(b)

Figure 2.8: Turbulent channel diagram

2.3.3 Combustion validation

An other fundamental aspect to validate for industrial burner simulation is combustion. Before considering turbulent combustion of industrial burners the first step is to validate the code at least for laminar combustion. A test has been carried out comparing results obtained through the solver PREMIX[57] from the CHEMKIN[30] package and AVBP on a laminar one dimensional premixed methane air flame. The domain is discretized by 200x1 quads with symmetry conditions on the top and bottom sides to reduce to a one dimensional problem. A two step chemical scheme is chosen to check the chemical equilibrium. The chemical species considered are O_2, N_2, CH_4, CO_2, CO, H_2O. The two reactions are:

- R1:

$$CH_4 + \frac{3}{2}O_2 \rightarrow CO + 2H_2O \tag{2.55}$$

- R2:

$$CO + \frac{1}{2}O_2 \rightleftarrows CO_2 \tag{2.56}$$

First a calculation is performed using PREMIX, the velocity, temperature and composition profiles are extracted and used as initial conditions for AVBP. Also the flame velocity calculated by PREMIX is used as inlet velocity for AVBP, $S_L^{PREMIX} = 0.263ms^{-1}$. The flame is then calculated using AVBP until the profiles are stabilized (this is obtained after 15ms of simulation) and the profiles are extracted and compared to the profiles given by PREMIX, Fig 2.9.

The results show a slight shift to the left of the profiles which can be explained because AVBP slightly over predicts the flame speed $S_L^{AVBP} = 0.27ms^{-1}$, therefore the flame front is slowly moving upstream. Despite this small difference (the difference of flame velocity is less than 3%) it is clear that AVBP and PREMIX are in good agreement.

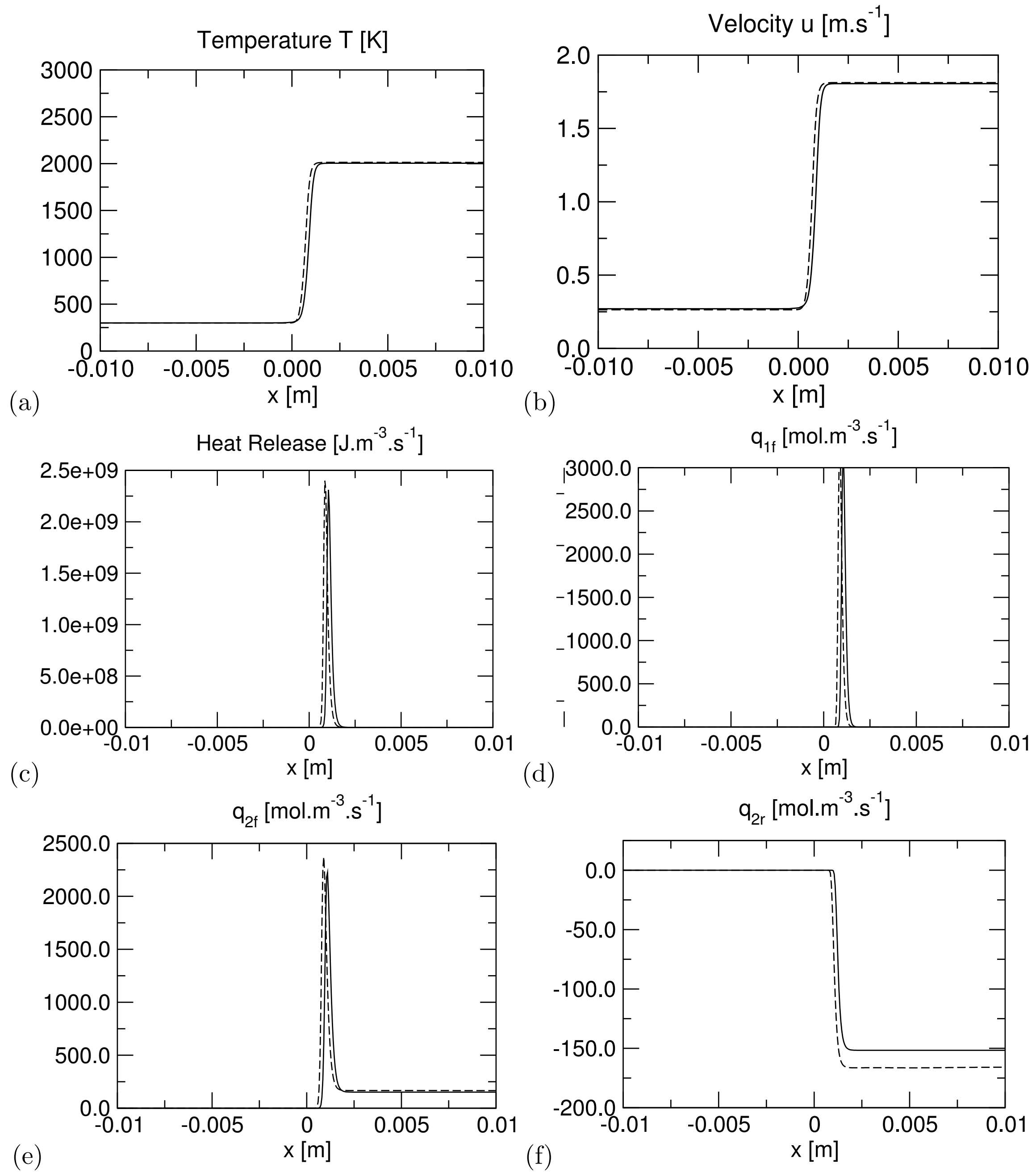


Figure 2.9: Comparison between AVBP (Lax Wendroff) and PREMIX profiles (Dashed line AVBP, continuous line PREMIX)

Chapter 3

Solid thermal conduction

AVTP has been derived from AVBP, therefore it shares the computational structure of AVBP, however its numerical methods are explicit which is clearly not optimal for solving the heat equation. Recently an implicit scheme has been added to the solver but this scheme was not available during this thesis, hence only the explicit scheme is used.

3.1 The equation solved: the unsteady heat equation

The unsteady heat equation solved reads:

$$\frac{\partial T}{\partial t} = \frac{1}{\rho c}\left[\frac{\partial}{\partial x_i}\left(\lambda \frac{\partial T}{\partial x_i}\right) + Q\right] \tag{3.1}$$

In AVTP λ, ρ and c depend on the material used and the local temperature. Q is a heat source. Equation (3.1) shows almost explicitly how the calculation is performed within AVTP:

Step	Description
1	Calculate the λ,ρ,c from the temperature and the material tables
2	Compute the temperature gradient
3	Compute the heat flux using the gradient and λ
4	Sum an eventual source term
5	Apply boundary conditions (Neumann boundaries)

The gradient computation on the unstructured grid is in fact the only difficult operation, it is explained in N. Lamarque PhD thesis [62].

At the end of the entire Runge-Kutta integration the Dirichlet boundary conditions are set. It is clear that using an explicit time integration scheme is not the best adapted method to solve the heat equation, this is why during this thesis an implicit solver has been added to AVTP. In this thesis only the explicit solver is used because the implicit solver was implemented too late. However as it is shown in chapters 4 and 5 the thermal solver is still much faster than the LES solver. Also there is no point in advancing in time the thermal solver too much compared to the LES solver, this can lead to instabilities (see chapter 7).

3.2 Validation

A basic configuration for which an analytical solution exists is used to validate AVTP. The configuration consists of a square domain at initial temperature $300K$ which is heated using Dirichlet boundaries at $400K$ on the left and right sides. In order to obtain a one dimensional problem symmetries are applied on the top and bottom boundaries, Fig 3.1.

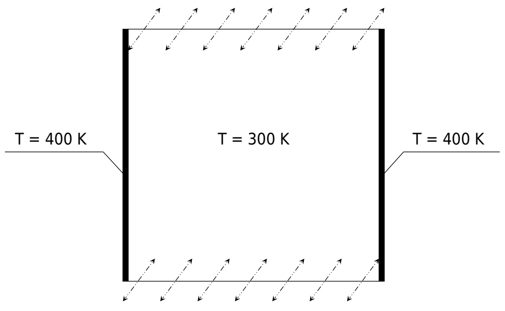


Figure 3.1: Heating setup

First an analytical solution is established, then this reference solution is compared to the AVTP solution.

3.2.1 Analytical resolution of the problem

The one dimensional problem can be described by the following system:

$$\begin{cases} \frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2} \\ T(x,0) = T_I, \forall x \\ T(0,t) = T(L,t) = T_{bc}, t > 0 \end{cases} \tag{3.2}$$

Where $\alpha = \frac{\lambda}{\rho C_p}$ is the heat diffusivity, L the length of the domain, T_I the initial temperature and T_{bc} the Dirichlet boundary temperature.

First it is clear that the stationary solution $T = T_{bc}$ satisfies the partial differential equation and the boundary conditions, therefore to simplify resolution a function $\tilde{T} = T - T_{bc}$ is introduced. System (3.2) thus becomes (3.3) which is homogeneous allowing to use the separation of variables method[80], i.e. the solutions searched are in the form $\tilde{T} = v(x)w(t)$.

$$\begin{cases} \frac{\partial \tilde{T}}{\partial t} = \alpha \frac{\partial^2 \tilde{T}}{\partial x^2} \\ \tilde{T}(x,0) = T_I - T_{bc}, \forall x \\ T(0,t) = T(L,t) = 0, t > 0 \end{cases} \tag{3.3}$$

Injecting $\tilde{T} = v(x)w(t)$ into the unsteady heat equation yields

$$vw' = \alpha v'' w \Leftrightarrow \frac{1}{\alpha}\frac{w'}{w} = \frac{v''}{v} \tag{3.4}$$

where the w' and v'' stand for $\frac{\partial w}{\partial t}$ and $\frac{\partial^2 v}{\partial x^2}$. Because Eq. (3.4) is true $\forall x, t$, $\frac{1}{\alpha}\frac{w'}{w} = \frac{v''}{v}$ remains constant, noted here k. Hence the following system should be solved:

$$\begin{cases} v'' - kv = 0 \\ w' - k\alpha w = 0 \end{cases} \tag{3.5}$$

Considering the boundary conditions, the only non null solutions to $v'' - kv = 0$ are obtained with $k < 0$. For $k < 0$ the form of the solution for v is:

$$v(x) = A\cos\left(\sqrt{-k}x\right) + B\sin\left(\sqrt{-k}x\right) \tag{3.6}$$

The boundary condition at $x = 0$ implies that $A = 0$. The boundary condition at $x = L$ implies that either

- $B = 0$ which would lead to a trivial solution,
- or if we assume that $B \neq 0$ then $\sin\left(\sqrt{-k}L\right) = 0$ which leads to $\sqrt{-k} = \frac{n\pi}{L}$ with $n = 1, 2, 3, ...$

Therefore $v_n(x) = B\text{sin}\left(\frac{n\pi}{L}x\right)$ are solutions of $v'' - kv = 0$. Using the expression for k, we can calculate a homogeneous solution for $w' - k\alpha w = 0$ depending on n:

$$w_n' + \frac{n^2\pi^2\alpha}{L^2}w_n = 0 \Rightarrow w_n(t) = D_n e^{-\frac{n^2\pi^2\alpha}{L^2}t} \tag{3.7}$$

Where D_n are constants. Noting $C_n = BD_n$, $\tilde{T_n}(x,t) = v_n(x)w_n(t) = C_n\text{sin}\left(\frac{n\pi}{L}x\right)e^{-\frac{n^2\pi^2\alpha}{L^2}t}$ with $n = 1, 2, 3, ...$ are solutions to system (3.3).

Hence also a solution to system (3.3) is

$$\tilde{T_g}(x,t) = \sum_{n=1}^{\infty} C_n\text{sin}\left(\frac{n\pi}{L}x\right)e^{-\frac{n^2\pi^2\alpha}{L^2}t} \tag{3.8}$$

Using the initial condition we obtain Eq. (3.9).

$$\tilde{T_g}(x,0) = \sum_{1}^{\infty} C_n\text{sin}\left(\frac{n\pi}{L}x\right) = T_I - T_{bc} \tag{3.9}$$

Noticing that Eq. (3.9) is in fact a decomposition of $T_I - T_{bc}$ using Fourier series the coefficients can be obtained :

$$\begin{aligned} C_n &= \frac{1}{L}\int_{-L}^{+L}(T_I - T_{bc})\text{sin}\left(\frac{n\pi}{L}x\right)dx = \frac{2(T_I - T_{bc})}{L}\int_{0}^{+L}\text{sin}\left(\frac{n\pi}{L}x\right)dx \\ &= -\frac{2(T_I - T_{bc})}{n\pi}\left[\cos(n\pi) - 1\right] \end{aligned} \tag{3.10}$$

Which can be expressed for n odd or even:

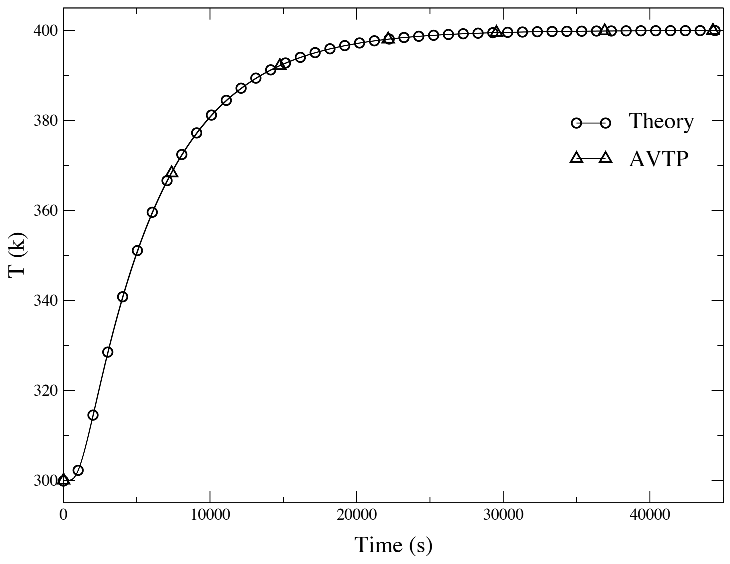
$$\begin{cases} n = 2p, C_{2p} = 0 \\ n = 2p+1, C_{2p+1} = \frac{4(T_I - T_{bc})}{(2p+1)\pi} \end{cases} \tag{3.11}$$

Finally the solution to the original problem (3.2) is obtained:

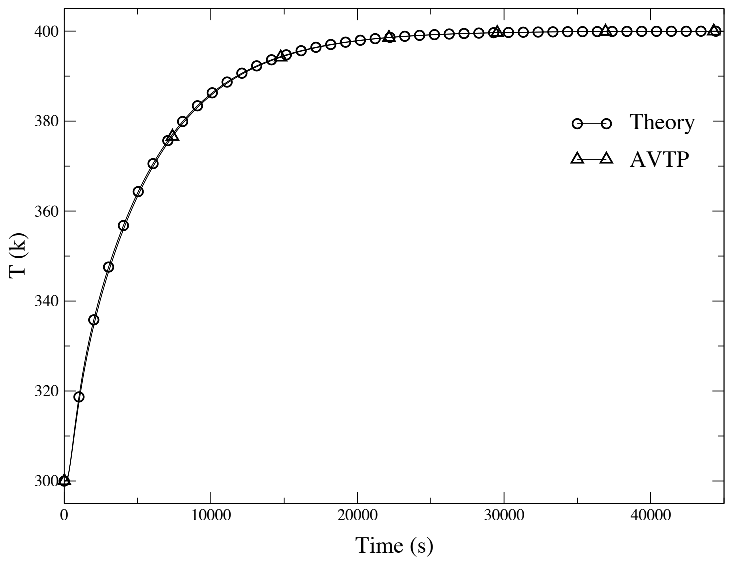
$$T(x,t) = T_{bc} - \frac{4}{\pi}(T_{bc} - T_I)\sum_{p=1}^{\infty}\frac{1}{(2p+1)}\text{sin}\left(\frac{(2p+1)\pi}{L}x\right)e^{-\frac{(2p+1)^2\pi^2\alpha}{L^2}t} \tag{3.12}$$

3.2.2 Comparison with AVTP results

The calculation is performed using the explicit forward Euler temporal scheme on a square uniform triangular mesh with 32 nodes per side (of size 1). Two sensors at locations ($x = 0.5, y = 0.5$) and ($x = 0.74, y = 0.5$) record the evolution of the simulated temperatures. Figure 3.2 shows the comparison of the simulated and analytical temperatures. These results clearly show agreement between the analytical and simulated solutions therefore demonstrating the ability of AVTP to solve thermal conduction problems.



(a)



(b)

Figure 3.2: Comparison of analytical and resolved temperature temporal evolution

Chapter 4

LES computation

Due to confidential material this chapter has been stripped from the public release of this manuscript.

Chapter 5

The solid domain

Due to confidential material this chapter has been stripped from the public release of this manuscript.

Conclusion

The objective of this thesis is to investigate methods for conjugate heat transfer relying on LES and to apply these methods on a target configuration: an aeronautical burner. The fluid and the thermal solver, namely AVBP and AVTP, as well as their respective physics have been presented in chapters 2 and 3. These solvers have been validated using simple test cases. Finally an aeronautical burner fluid and thermal configurations and uncoupled computations have been presented. The two configurations are based on a complex geometry which has been discretized according to each problems specific needs.

The LES results show complex unsteady structures in the fluid domain, notably for the flame. This implies unsteady heat release and hence unsteady fluid wall temperature and heat flux. Since the wall heat flux and temperature may be used to couple the thermal solver, and that the coupling procedure may be executed at a given frequency, the effect of such temporal variations should be investigated.

Part II

Coupling stability and convergence analysis

Table of Contents

Nomenclature

μ	Dynamic viscosity $kg.m^{-1}.s^{-1}$
ω	Pulsation $rad.s^{-1}$
ϕ	Thermal flux $W.m^{-2}$
ρ	Density $kg.m^{-3}$
$\rho(M)$	Spectral radius of matrix M
τ	Characteristic time
a	Convergence acceleration parameter
c	Thermal capacity $J.K^{-1}$
c_m	Thermal specific capacity $J.kg^{-1}.K^{-1}$
D	Thermal diffusivity $m^2.s^{-1}$
d	Fourier Number
f	Frequency s^{-1}
H_c	Convective exchange coefficient
k	Thermal conductivity $W.m^{-1}.K^{-1}$
P_r	Prandtl number
T	Temperature K
T_c	Convective temperature

As has been seen in chapter 4, combustion incorporates strong unsteady components and LES has already proved to be a powerful tool to simulate them accurately. Promising results have been obtained not only on academic combustion chambers [63] but also on industrial ones [13, 58]. LES is therefore a natural choice for burner simulation, however coupling an unsteady LES raises questions about the coupled simulation's convergence, specially when the two solvers are coupled at a given frequency which may be very different than the frequencies of the unsteady combustion components. In the case of conjugate heat transfer simulation it is the temperature fluctuations which are essentially of interest. Such fluctuations can be the result of different mechanisms but in a combustion chamber the main producer of such fluctuations is the flame.

In this part the effect of such temperature fluctuations on the convergence of the conjugate heat transfer application is investigated. Based on this analysis, a coupling methodology relying on LES to compute stationary thermal solutions is proposed, investigated and its stability is verified. Simple systems, Fig. 5.1, are used to study these more fundamental aspects of coupling neglecting the complex geometrical issues inherent to industrial applications. These aspects will be treated in parts III and part IV. Throughout this study it is considered that from an unsteady point of view the conjugate heat transfer can be studied as a one way coupling from the LES solver to the conduction solver. Granted this is a strong hypothesis but it is supported by the difference in the characteristic times of both problems.

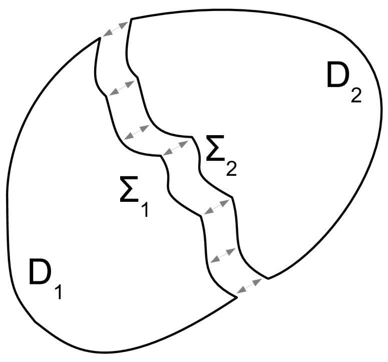


Figure 5.1: A simple partitioned domain

Chapter 6

Influence of fluid instabilities on solid's temperature convergence

Finding a stationary solution to a heat conduction problem corresponds to solving a Laplace equation and thus has a nice property: for a set of fixed boundary conditions (Dirichlet boundaries), there is only one solution possible. In other words, the solution within the domain is entirely defined by its boundary conditions. This point is fundamental to assess the quality of the solutions found with the proposed coupling strategy: if the boundary conditions transferred from the fluid solver to the conduction solver are *physical* then the thermal solver will converge to *the physical solution*. Unfortunately the coupling process is an iterative process with an unsteady fluid solver providing boundary conditions which may vary greatly from a coupling step to an other. For example a turbine blade located after a combustion chamber can see a succession of hot and cold structures impacting on it due to the instabilities within the combustion chamber and the mixing with the dilution jets.

coupling frequency can thus lead to an incorrect prediction of the temperature field within the blade, e.g. if due to the coupling frequency the blade only sees hot or cold spots.

In RANS this problem does not exist because a RANS solver does not resolve the instantaneous temperature fields but only to their statistical mean which is also thought to be the average over an infinite period of time Eq. (6.1).

$$T_{RANS}\left(x\right) = \lim_{\tau_{avg}\mapsto\infty} \frac{1}{\tau_{avg}} \int_{0}^{\tau_{avg}} T\left(x,t\right) dt \tag{6.1}$$

LES on the other hand only gives access to instantaneous fields or averaged fields over a finite period of time τ_{avg}.

$$T_{LES}\left(x,t_0,\tau_{avg}\right) = \frac{1}{\tau_{avg}} \int_{t_0}^{t_0+\tau_{avg}} T\left(x,t\right) dt \tag{6.2}$$

Adapting existing RANS methodology to LES would hence impose to choose τ_{avg} big enough such that T_{LES} only depends on space, x. LES being much more expensive than RANS, having to reconverge and average the LES after each coupling iteration may in some situations be very expensive.

On the other hand if τ_{avg} is not big enough then the averaged temperature depends on t_0 and the averaging period $T_{LES} = T_{LES}\left(x,t_0,\tau_{avg}\right)$. Since the averaging period is linked to the coupling frequency and averaging over a finite time segment is a sampling process: it obeys to sampling theory [10, 70, 71, 100]. Special care must therefore be taken to avoid aliasing effects. Typically if LES reaches a limit cycle of period τ_L, if $\tau_{avg} > \frac{1}{2}\tau_L$, $T_{LES}(x, t_0, \tau_{avg})$ can oscillate at low frequencies. To avoid such problems a simple condition is to impose $\tau_{avg} < \frac{1}{2}\tau_L$, of course this requires that τ_L is known a priori.

To investigate these issues implied by the use of a LES solver, the effect of an unsteady fluid coupled to a solid is first studied. The influence of the coupling frequency on the convergence of a conjugate heat transfer problem when subject to an unsteady fluid is in this context specifically assessed since of clear importance for the target problem.

Prior to the theoretical analysis of the aforementioned problem several aspects need to be clarrified or defined. In this section, strong coupling refers to two systems which are synchronized continuously whereas loose coupling refers to two systems which synchronize at a given frequency. In order to simplify the following discussion we will assume that there is a linear relation between the two simulation times such as

$$t_s = N_s \Delta t_s \tag{6.3}$$

$$t_f = N_f \Delta t_f \tag{6.4}$$

$$N_s = a N_f \tag{6.5}$$

With N_f and N_s being the iterations of each solver between two coupling updates (subscript f for the fluid side, s for the solid), Δt_f and Δt_s their respective time steps (enforced by the stability of the individual solvers). As long as the goal of the coupled simulation is to obtain a stationary solution in the solid, the ratio a has no physical meaning and can be chosen arbitrarily as long as the coupled simulation remains stable. This means that it can be used to improve load balancing for example. Since $t_s = a\frac{\Delta t_s}{\Delta t_f} t_f$, throughout the remaining we will work in the solid's time space $t = t_s$ (all flow frequencies are apparent frequencies for the solid taking into account the scaling rule).

6.1 Influence of fluid unsteady features on a 1D solid's temperature domain

In this case an unsteady fluid is in contact with a 1D solid domain at $x = 0$, the coupling is modeled by imposing the fluid temperature to the solid's border temperature. The fluid unsteady temperature is modeled by a sinusoidal wave, hence at $x = 0$ the temperature applied at $x = 0$ is a sinusoidal signal $T(t) = \Delta T \cos(\omega t)$. Only the harmonic response of the solid is considered here, to simplify the solid temperature at $x = \infty$ is 0. The unsteady heat equation reads:

$$\frac{\partial T}{\partial t} = D \frac{\partial^2 T}{\partial x^2} \tag{6.6}$$

Where $D = \frac{k}{\rho c_m}$ is the thermal diffusivity. To solve Eq. (6.6) a complex notation $\frac{\partial \bar{T}}{\partial t} = D\frac{\partial^2 \bar{T}}{\partial x^2}$ is introduced. The solutions searched are in the following form $\bar{T} = \Delta T e^{i(\omega t - kx)}$. In order to respect Eq. (6.6), $k^2 = -i\frac{\omega}{D}$ must be satisfied.

Noting that[1]

$$-i = e^{\frac{3\pi}{2}} = \left(e^{\frac{3\pi}{4}} e^{p\pi}\right)^2 = \left(\frac{(1-i)}{\sqrt{2}} e^{p\pi}\right)^2 \text{ with } p \in \mathbb{Z} \tag{6.7}$$

We can write that

$$k = \sqrt{\frac{\omega}{2D}}(1-i)e^{p\pi} \tag{6.8}$$

[1]This resolution of the problem can be found i[61].

By inserting this into $\bar{T}$ and writing $T = \Re(\bar{T})$:

$$T = \Delta T e^{-\sqrt{\frac{\omega}{2D}} e^{p\pi} x} \cos(\omega t - \sqrt{\frac{\omega}{2D}} e^{p\pi} x) \tag{6.9}$$

There are two solutions, one for p odd, one for p even, the only physical solution is for p even, thus $e^{p\pi} = 1$, the final solution is:

$$T = \Delta T e^{-\sqrt{\frac{\omega}{2D}} x} \cos(\omega t - \sqrt{\frac{\omega}{2D}} x) \tag{6.10}$$

Eq. (6.10) shows that the amplitude of the perturbation set within $x = 0$ decreases exponentially and the higher the frequency, the faster the decay is. From this expression we can derive the perturbation length at 1%:

$$x_{1\%} = \ln(100) \sqrt{\frac{2D}{\omega}} \tag{6.11}$$

Table 6.1(a) shows perturbations depths at 1% for different frequencies and for a metallic (iron) domain $k = 50W.m^{-1}.K^{-1}$, $c_m = 440J.kg^{-1}.K^{-1}$, $\rho = 8000kg.m^{-3}$ giving $D = \frac{k}{\rho c_m} = 14.10^{-6}m^2.s^{-1}$. These values show that low frequency excitations do have an impact on the solid's temperature. Also due to the acceleration of convergence methodology the apparent frequencies of the fluid unsteady features are lowered, hence the perturbation depths are over predicted. Therefore convergence acceleration can yield unrealistic instantaneous solutions. However at each point in space the temperature oscillates in time around the average solution at the forcing frequency, meaning that averaging over a period of the forcing frequency should yield the converged time independent solution.

$f_f(Hz)$	$f_s(Hz)$	$x_{1\%}(mm)$
1	9.67E-04	315
100	9.67E-02	31.5
500	4.83E-01	14.1
1000	9.67E-01	9.96
2000	1.93	7.04
5000	4.83	4.45
10000	9.67	3.15

Table 6.1: Perturbation depth at 1% for a metallic solid, for different frequencies considering the conversion from the fluid time line to the solid time line $f_s = \frac{1}{a}\frac{\Delta t_f}{\Delta t_s} f_f$ with N_f=5, N_s=15, Δt_f=3.8E-8s, Δt_s=1.3E-5s

6.2 Influence of the coupling frequency on the convergence of an unsteady conjugate heat transfer problem

Having investigated the influence of fluid unsteadinesses on a solid's temperature field it is now important to investigate the effect of a key parameter of a coupling methodology: the coupling frequency. First a 0D conjugate heat transfer model will be considered where instantaneous values are exchanged. The response of the solid to the unsteady fluid system will be investigated in different circumstances:

- first monochromatic harmonic forcing will be considered for strong and then loose coupling,
- then solution for polychromatic forcing will be considered for loose coupling

Finally extension to averaged quantities will be considered.

The physical setup

The physical model (Fig 6.1) is the composition of 2 zero dimensional domains:

- a fluid which has a pulsating temperature, this represents the effect of high frequency phenomena inside a reactive fluid, for example hot pockets of gas which are convected out of a pulsating flame.
- a solid domain which is in contact with the fluid.

The aggregate system is considered to be isolated, the coupling condition between the fluid and the solid is a simple convective flux, Eq. (6.12), applied to the solid.

$$\phi_S = H_c\left(T_F - T_S\right) \tag{6.12}$$

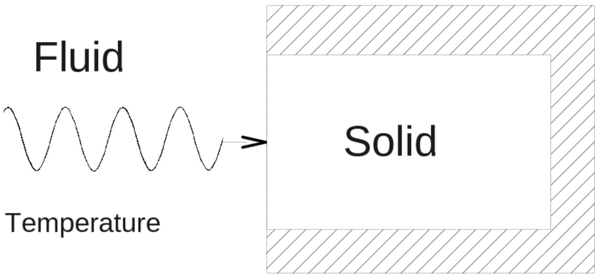


Figure 6.1: Conjugate heat transfer convergence analysis setup

Only one way coupling of instantaneous quantities is studied at this stage to illustrate the difficulty introduced by the LES solver: i.e. from the fluid to the solid.

Taking the fluid's temperature signal to be modeled by a sinusoidal function:

$$T_F(t) = T_{fluc}\cos(\omega_0 t) + T_{F0} \tag{6.13}$$

where $\omega_0 = 2\pi f_0$ is the pulsation of the temperature signal, T_{F0} the average temperature within the fluid.

The solid is modeled by a simple thermal capacity which receives a flux ϕ_S continuously. Since we consider coupling at a given frequency (potentially different from ω_0) the value of ϕ_S of the model is updated at each coupling iteration according to Eq. (6.12). At each local time step the solid's temperature hence satisfies Eq. (6.14).

$$c\frac{dT_S}{dt} = \phi_S \tag{6.14}$$

Strong coupling analytical model

With c being the solid's thermal capacity($J.K^{-1}$) and H_c being a convection coefficient ($W.K^{-1}$), the solid's temperature T_S satisfies:

$$T_S(t) = \int_0^t \frac{\phi(t)}{c}dt + T_{S0} \tag{6.15}$$

With T_{S0} the initial temperature of the solid. And

$$\phi(t) = H_c\left(T_F(t) - T_S(t)\right) \tag{6.16}$$

With

$$T_F(t) = T_{fluc}\cos(\omega_0 t) + T_{F0} \tag{6.17}$$

Combining Eq. (6.15), (6.16), and (6.17) and taking the derivative leads to:

$$\frac{dT_S(t)}{dt} = \frac{T_{fluc}}{\tau}\cos(\omega_0 t) + \frac{T_{F0}}{\tau} - \frac{1}{\tau}T_S(t) \tag{6.18}$$

with $\tau = \frac{c}{H_c}$. Solving for Eq. (6.18), knowing that $T_S(0) = T_{S0}$, leads to:

$$T_S(t) = \left(T_{S0} - T_{f0} - \frac{T_{fluc}}{1+\omega_0{}^2\tau^2}\right)e^{\frac{-t}{\tau}} + \frac{T_{fluc}}{\sqrt{1+\omega_0{}^2\tau^2}}\cos\left(\omega_0 t - \arcsin\left(\frac{\omega_0\tau}{\sqrt{1+\omega_0{}^2\tau^2}}\right)\right) + T_{f0} \tag{6.19}$$

This expression shows that the system never completely obtains a stationary solution, instead it continues to oscillate at a pulsation ω_0. It is interesting to see that the amplitude of the oscillating part is in fact the gain of a first order linear low pass filter (with a cutoff frequency of $\frac{1}{2\pi}\frac{H_c}{c}$).

Loose coupling model

The loose coupling of instantaneous quantities can be viewed as a process which samples the fluid temperature every τ_{cpl}. This can be modeled by introducing a new function ψ which transforms the continuous time into a discrete time of granularity τ_{cpl}. Hence ψ is defined by

$$\psi(t) = E\left(\frac{t}{\tau_{cpl}}\right)\tau_{cpl} \tag{6.20}$$

with $E(x)$ the integer part of x. The loose coupling effect is implemented by replacing the flux $\phi(t)$ by $\phi(\psi(t))$ where τ_{cpl} is the period in physical time between flux updates. The solid temperature thus reads:

$$T_S(t) = \int_0^t \frac{\phi(\psi(t))}{c}dt + T_{S0} \tag{6.21}$$

Coupling can in this case be seen as a sampling procedure. Therefore to ensure a proper convergence, the Nyquist-Shannon theorem should be respected, meaning that the coupling frequency f_c should be at least twice the frequency of the pulsating phenomenon in the fluid f_0. Not respecting this constraint leads to aliasing effects appearing as low frequency oscillations in the solid temperature. This is important because as demonstrated in 6.1, low frequencies penetrate further inside the solid domain than higher frequencies. The lowest low frequency mode is called principal alias and can be estimated using sampling theory [119] to

$$f_{alias} = \min_{l\in\mathbb{Z}}\left(|f_0 + l f_c|\right) \tag{6.22}$$

Solving analytically such a differential system is difficult because ψ is non continuous, this is why a numerical approach is used instead. The analysis has been carried out using a simple FORTRAN code where the temporal integrations are discretized using a simple first order Euler explicit scheme. Several observations are underlined in Fig 6.2 which shows the temperature evolution and convergence of the model for different coupling frequencies:

- In the first case(Fig 6.2(a)), the coupling frequency is chosen to be twice the frequency of the physical phenomenon in the fluid, as expected, the solid oscillates around the stationary solution with a frequency corresponding to the original forcing frequency. We can see that this case agrees with the analytical or exact resolution (Case-4), Fig 6.2(b).

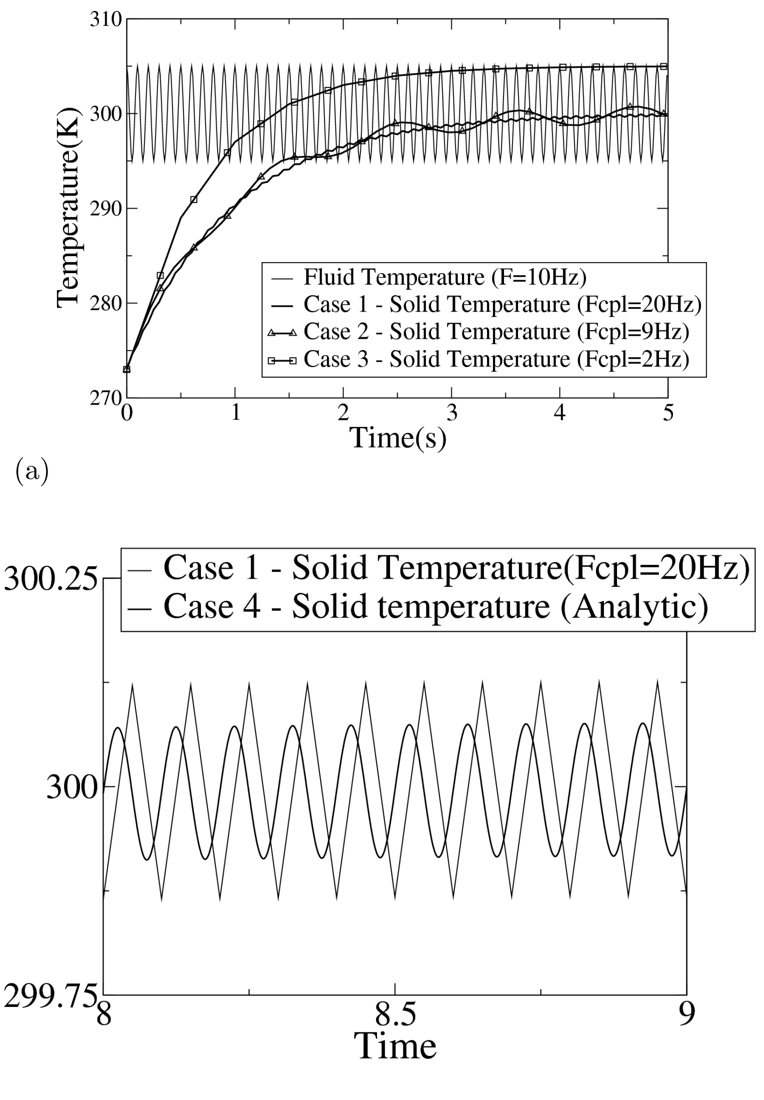


Figure 6.2: Evolution of solid temperature in loose coupling environment: (a) cases 1, 2, 3 and 4 (b) comparison of 1 and 4

- The second case illustrates a coupling which is not respecting the Nyquist-Shannon's theorem: a low frequency mode appears within the solid, its frequency can be predicted using Eq. (6.22) with $f_0 = 10Hz, f_c = 9Hz$ we obtain $f_{alias} = 1Hz$

- The third case is a particular case where the coupling frequency is very badly chosen: the coupling period is a multiple of the forcing period, meaning that at each coupling exchange the fluid is at the same phase. In this case the solid sees a stationary fluid at 305K. There again the alias frequency can be predicted using Eq. (6.22) with $f_0 = 10Hz, f_c = 2Hz$ we obtain $f_{alias} = 0Hz$.

In practice, respecting the Nyquist-Shannon theorem may impose a lot of coupling exchanges and thus increase the coupled computation cost. On the other hand not respecting the Nyquist-Shannon may have an impact on the solid's core temperature depending on the perturbation depth (Eq. (6.11)) for the principal alias frequency f_{alias}. If this perturbation depth can not be neglected then the solid's temperature should be averaged over at least $\frac{1}{f_{alias}}$ and not over $\frac{1}{f_0}$.

This conclusion is valid for a monochromatic harmonic excitation coming from the fluid, however

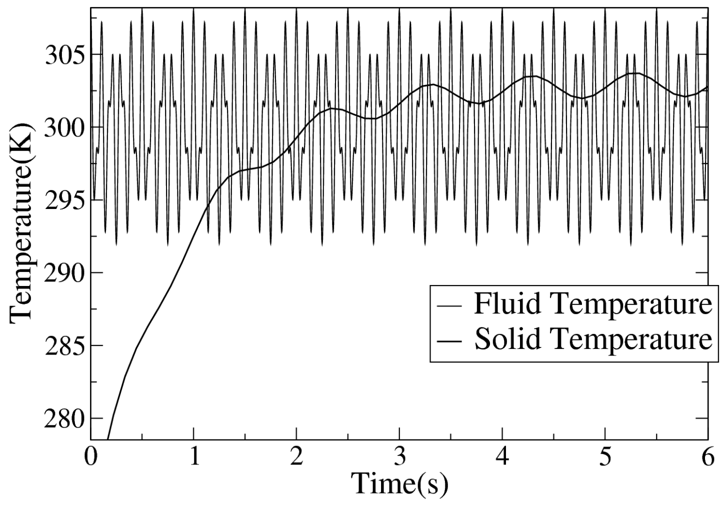


Figure 6.3: An example of a setup which can not converge

it is not directly generalizable to more complex signals. Figure 6.3 shows a case where the solid mean temperature will never converge to T_{F0}: a fluid with a temperature signal composed of two harmonics at $10Hz$ and $18Hz$ with amplitudes of $5K$ and $3K$ is coupled at a frequency of $9Hz$. This sampling process would yield respectively two aliases at $1Hz$ and $0Hz$. This $0Hz$ alias means that the solution will be shifted by a constant value. In this example the solid temperature oscillates around $303K$ instead of $300K$ with a frequency of 1Hz and an amplitude of $1K$. This also shows that the relative amplitudes of the two signals between the input and the response have been modified. In order to couple real applications with complex signals the polychromatic case must be studied to approach a fully turbulent unsteady flow.

Extension to a polychromatic signal

To extend the monochromatic results to polychromatic signals it is important to consider the response of the system to multiple frequencies. Considering only the harmonic forcing, the system's response can be modeled by a sampler linked to a low pass filter. Hence the transfer function of the filter should be established, first in continuous form then in discrete form.

Once the limit cycle is established, the systems obeys the following equation (only the harmonic components are kept):

$$T_S(t) = \int_0^t \frac{1}{c}\phi(t)dt = \frac{1}{\tau}\int_0^t \left(T_F(t) - T_S(t)\right)dt \tag{6.23}$$

where $\tau = \frac{H_c}{c}$. Using the Laplace transform we can write

$$T_S(s) = \frac{1}{\tau}\frac{1}{s}\left(T_F(s) - T_S(s)\right) \tag{6.24}$$

Hence

$$H_{cont}(s) = \frac{T_S(s)}{T_F(s)} = \frac{1}{1+\tau s} \tag{6.25}$$

Note that by taking the amplitude $A_{cont}(\omega) = \|H_{cont}(j\omega)\| = \frac{1}{\sqrt{1+\tau^2\omega^2}}$ which is the amplitude of the oscillating part in Eq. (6.19). This indicates that in the continuous time space, frequencies higher than τ^{-1} will be damped by the first order filter. However in the discrete time space the conclusion is different. To illustrate the discretization, the z-transform [54, 45] is introduced[2]. In the numerical method used to solve the loosely coupled problem, the continuous integral is solved using the forward Euler method.

Discretizing Eq. (6.23) with forward Euler[3] yields:

$$T_S(t+\Delta t) = T_S(t) + \frac{\Delta t}{\tau}\left(T_F(t) - T_S(t)\right) \tag{6.26}$$

The discrete notation $u^k = u(k\Delta t)$ is introduced:

$$T_S^{k+1} = T_S^k + \frac{\Delta t}{\tau}\left(T_F^k - T_S^k\right) \tag{6.27}$$

Applying the z-transform gives

$$zT_S(z) = T_S(z) + \frac{\Delta t}{\tau}\left(T_F(z) - T_S(z)\right) \tag{6.28}$$

Hence the discrete transfer function H_{disc} writes:

$$H_{disc}(z) = \frac{T_S(z)}{T_F(z)} = \frac{\Delta t}{\Delta t - \tau + \tau z} \tag{6.29}$$

The gain $A_{disc}(\omega) = \|H_{disc}(e^{j\omega})\|$ reads:

$$A_{disc}(w) = \frac{1}{\sqrt{1 + 4\frac{\tau(\tau-\Delta t)}{\Delta t^2}\sin\left(\frac{\omega\Delta t}{2}\right)^2}} \tag{6.30}$$

In the following ω_N is the Nyquist frequency, which is defined as half of the sampling frequency ω_s. It is important to understand the fundamental difference between $A_{cont}(\omega)$ and $A_{disc}(\omega)$. $A_{cont}(\omega)$ is the gain of a standard first order low pass filter. $A_{disc}(\omega)$ is similar to $A_{cont}(\omega)$ on the interval $[0, \omega_N]$, however it is mirrored on $[\omega_N, \omega_s]$ (Fig. 6.4(a)). The pattern between $[0, \omega_s]$ is then repeated infinitely (Fig. 6.4(b)). This can be seen as a consequence of the Shannon Theorem: the spectrum of a sampled signal is composed of the repetition of the continuous signal's spectrum translated by the multiples of the sampling frequency. However it has an important consequence if the input $T_F(t)$ contains frequencies which are higher than ω_N:

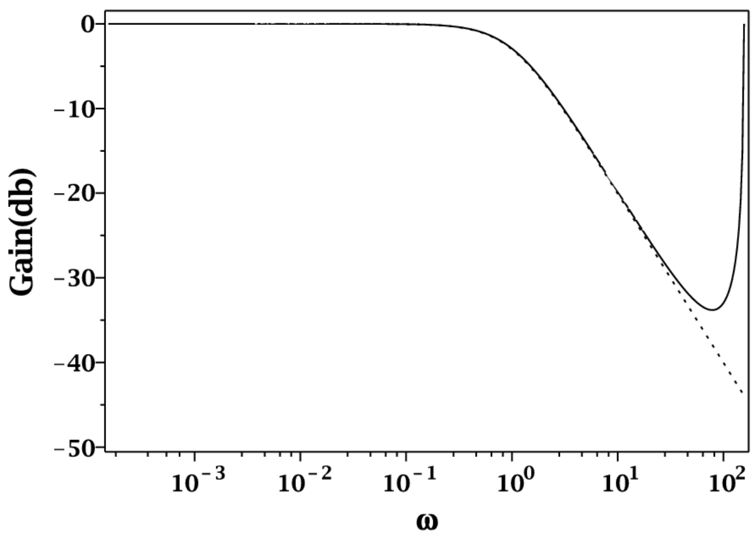
- not only will these frequencies be aliased to low frequencies,
- but the closer they will be to a multiple of ω_s, the less they will be damped.

Hence it is important to investigate the effects of a real world polychromatic signal provided by an unsteady LES solver coupled to an unsteady solid solver. In this work we are interested in the temperature fluctuations in a combustion chamber. Such fluctuations may be produced by combustion instabilities or turbulence [74]. Combustion instabilities provide through complex mechanisms high temperature fluctuations at a relatively low frequency whereas turbulence provides low temperature fluctuations over a larger frequency range.

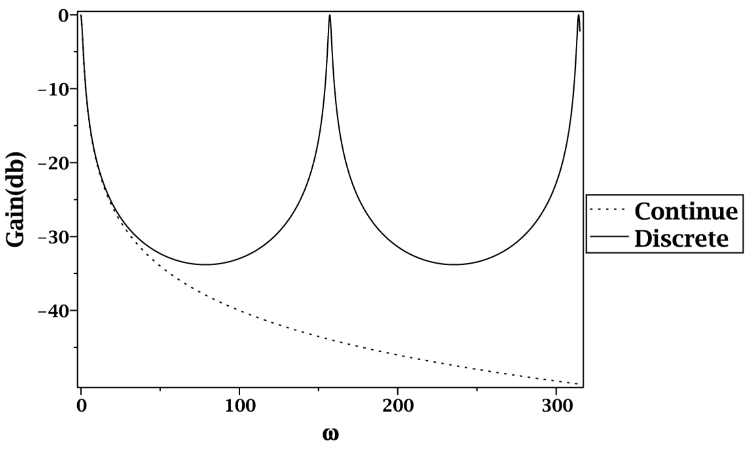
To study the response of this coupled system, a simple model for $T_F(t)$ is proposed: the combustion instability is modeled by a monochromatic signal at low frequency and high amplitude, while

[2]The z-transform is commonly used in digital signal processing in the same way the Laplace transform is used on continuous systems.

[3]Forward Euler is used here because it simplifies the equations, similar results can be obtained using higher order integration methods.



(a)



(b)

Figure 6.4: Comparison between the continuous and the discrete low pass/integrator filter: (a) on a $[0, \omega_s]$ period on a log-log diagram, (b) on $[0, 2\omega_s]$ on a semi-log diagram.

turbulence on the other hand is modeled by a polychromatic signal. The amplitudes of the different harmonics are given by a Passot-Pouquet [83] spectrum[4]. The hypothesis used to build $T_F(t)$ is to consider these two characteristics to be independent, hence $T_F(t)$ is composed of the sum of the combustion instability signal and the turbulence signal. The ratio between the combustion instability amplitude and the maximum amplitude for the Passot-Pouquet signal is chosen arbitrarily to be 10.

Figure 6.5 shows the spectrum of the input signal $T_F(t)$. Based on this input model, the frequency response is studied for 5 different coupling frequencies. For easier understanding the Nyquist frequency (half the sampling frequency) for each case is shown in Fig 6.5:

- A - the Nyquist frequency is at 70Hz(point A in Fig. 6.5). At this frequency the entire spectrum of $T_F(t)$ is almost correctly sampled.
- B - the Nyquist frequency is at 45Hz. At this frequency almost half of the turbulent part of $T_F(t)$'s spectrum is not correctly sampled. But the turbulent most energetic peak and the combustion instability peak frequencies are correctly sampled.
- C - the Nyquist frequency is at 32Hz. Point C is similar to point B. However a greater portion

[4]In reality the temperature fluctuation spectrum and the velocity fluctuation spectrum are different, this has been investigated by Corrsin [28]. The choice of a Passot-Pouquet spectrum is to obtain a spectrum representative of turbulence.

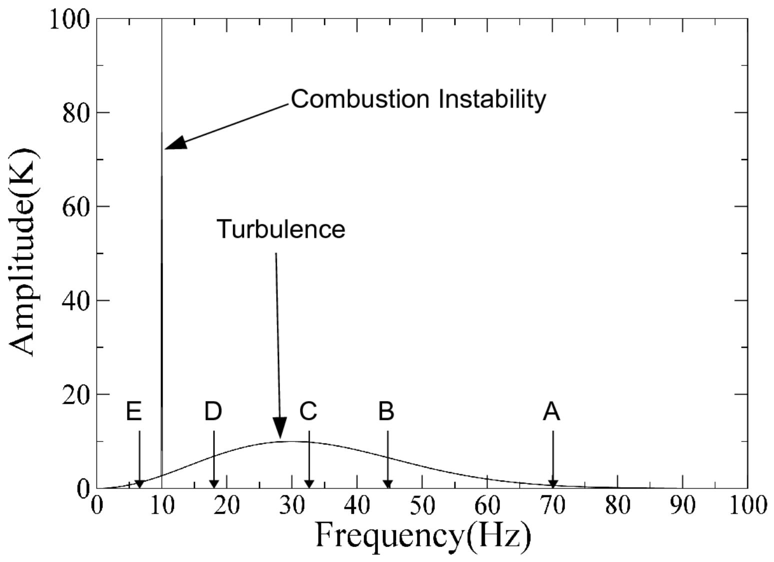


Figure 6.5: The spectrum of $T_F(t)$

of the turbulence spectrum is under-sampled.

- D - the Nyquist frequency is at 17Hz. The turbulent input spectrum is almost entirely under-sampled, but the combustion instability remains correctly sampled.
- E - the Nyquist frequency is at 6Hz. Both the combustion and the turbulence parts of $T_F(t)$ spectrum are under-sampled.

For each case A,B,C,D and E, the output signal T_S is calculated using the numerical code described in 6.2. The spectral responses for each case are then calculated using the Fourier transform, these responses are then grouped into two groups:

- Cases A, B and C are shown on Fig. 6.6. These three cases show almost no aliasing effects. Hence averaging T_S over a period corresponding to the combustion instability should yield a converged result. In cases A and B turbulence undersampling can be neglected because the portion of the spectrum aliased contains very little energy. However in case C the portion of the turbulent spectrum undersampled is sufficient to have a slight impact on the converged temperature, in this case it is shifted by 1K.
- Cases D and E are shown on Fig. 6.7. Both cases show important aliasing effects which would not lead to convergence as fast as cases A, B and C. The combustion instability is still correctly sampled in case D: the peak component of the spectrum remains at the combustion instability frequency. However in case E, the combustion instability harmonic is under-sampled, hence the main peak is aliased to a lower frequency. It is also very interesting to notice that in the input signal the ratio between the amplitude of the temperature fluctuations due to turbulence compared to combustion instabilities is 10%, whereas in the output signal the ratio is almost 50%. This non intuitive behavior is due to the symmetrical shape of the gain of the discrete low-pass filter.

Of course these results depend on the cutoff frequency $\frac{1}{2\pi}\frac{h}{c}$ of the low pass filter and on the ratio of temperature fluctuations between the combustion instability and turbulence. However, such results

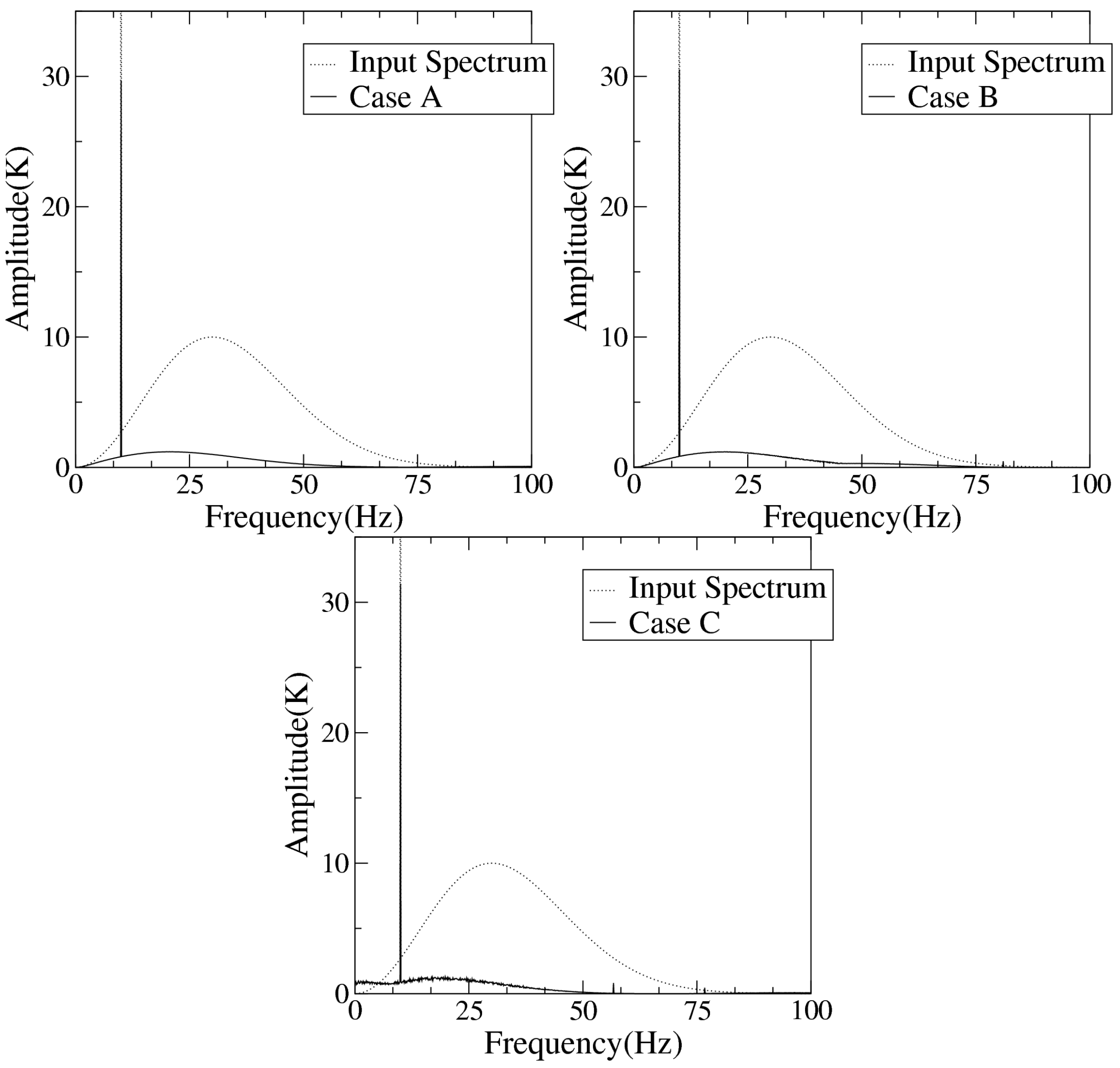


Figure 6.6: Comparisons of the spectrums of T_F and T_S for cases without important aliasing effects

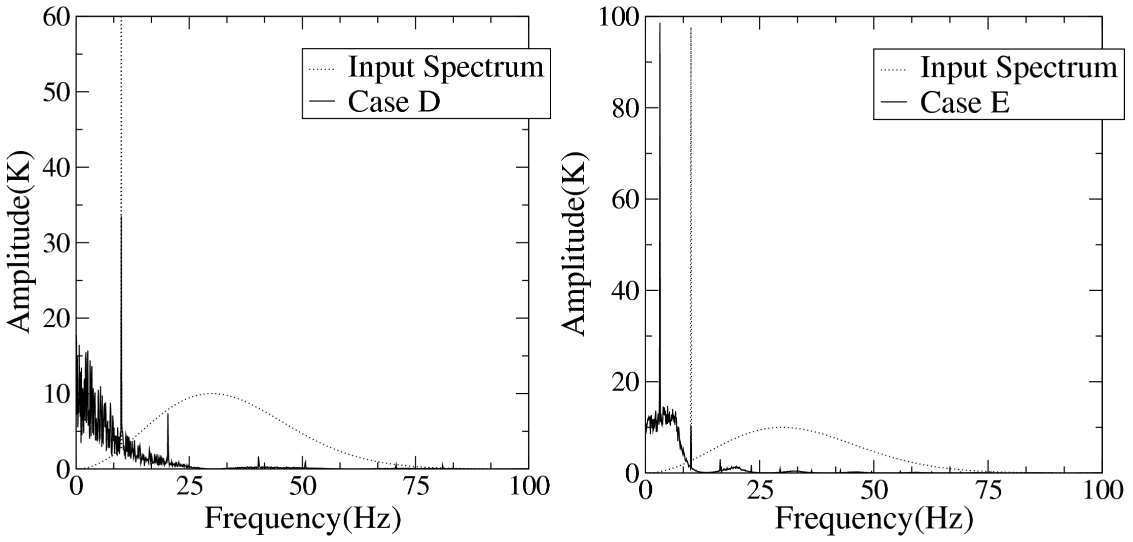


Figure 6.7: Comparisons of the spectrums of T_F and T_S for cases with important aliasing effects.

show that the relative amplitude of the aliased signals compared to the correctly sampled signals may be much higher than in the input. Hence secondary effects such as turbulence may be amplified if undersampled.

Extension to averaged quantities obtained from a finite time interval integration

It may seem intuitive to average the coupled quantities over an inter-coupling interval in order to filter out the high frequencies which may cause the aliasing problems identified previously or simply by trying to recover a RANS type conjugate heat transfer approach. Introducing a sliding integration window to obtain mean quantities from LES, the temporal average of $T_F(t)$ on the interval $[t, t+\tau_{cpl}]$ is:

$$\frac{1}{\tau_{cpl}} \int_t^{t+\tau_{cpl}} T_F(t)dt = \frac{2T_{fluc}}{\tau_{cpl}\omega_0} \sin\left(\frac{\omega_0 \tau_{cpl}}{2}\right) \cos\left(\omega_0 t + \frac{\omega_0 \tau_{cpl}}{2}\right) + T_{F0} \tag{6.31}$$

It is important to note that this function maintains its pulsating component at ω_0 with an amplitude of $2T_{fluc}\frac{\sin\left(\frac{\omega_0\tau_{cpl}}{2}\right)}{\omega_0\tau_{cpl}}$. Comparing τ_{cpl} and $\frac{2\pi}{\omega_0}$ yields:

- if $\tau_{cpl} \gg \frac{2\pi}{\omega_0}$ then the oscillating component of the averaged signal can be neglected, meaning that the aliasing problems may be neglected.
- if τ_{cpl} is a multiple of $\frac{2\pi}{\omega_0}$ (an entire period is averaged), then the integrated signal remains constant, hence no aliasing problems can exist. This is a special case which is very unlikely in real applications, notably when the signal is composed of several harmonics.
- if $\tau_{cpl} \sim \frac{2\pi}{\omega_0}$ then the averaged signal contains an oscillating component at pulsation ω_0 meaning that if the Nyquist-Shannon theorem is not respected ($\tau_{cpl} < \frac{1}{2}\frac{2\pi}{\omega_0}$), aliasing will occur and may not be negligible.

Such procedures are clear alternatives especially if ω_0 is known a priori. The associated computing cost needs however to be quantified.

6.3 Conclusion: what should we do?

The previous study shows that there are at least two different paths which lead to an accurate converged solution if coupling relies on a fully unsteady flow solver:

- the first one corresponds to the RANS like coupling strategy: averaged quantities are exchanged every $\tau_{cpl} \gg \frac{2\pi}{\omega_0}$ which means that we have to converge the signals provided by the unsteady solvers and then exchange them. This is well suited for stationary solvers, however for unsteady solvers this may be very expensive since the primary constraint scales inversly with ω_0.
- On the other hand, an accurate solution may be obtained by averaging the unsteady solid temperature obtained through a tightly[5] coupled simulation. The draw back is that it requires a lot of inter solver communications. The main advantage however is that no apriori knowledge of the flow physics is required.

Generally coupling very tightly is assumed to be expensive, however a methodology capable of tightly coupling two solvers without loss of performance is developed in this work. Now the stability of this methodology needs to be assessed.

[5]The Nyquist frequency associated to the coupling frequency should be chosen inorder to ensure that aliasing can be neglected.

Chapter 7

Numerical stability of a tightly coupled algorithm

Using the Godunov Ryabenkii[41] method Giles[39] studied the stability of a basic one dimensional conjugate heat transfer problem. Different coupling schemes were studied, notably the explicit time marching scheme which is of interest to our problem. For this scheme, Giles[39] demonstrates that the coupled problem is stable if at the interface a Dirichlet condition is applied to the fluid and a Neumann condition to the solid, Fig 7.1. This result is proved for a simulation with equal time steps in each domain, i.e. the fluid and solid simulations. It is then generalized to the case where each solver has its own time step (see Giles conclusion[39]). However this result is true as long as the two domains are very tightly coupled, in this study the domains exchange information at each time step. Unfortunately when several solvers are involved this is not always possible due to exchange overhead.

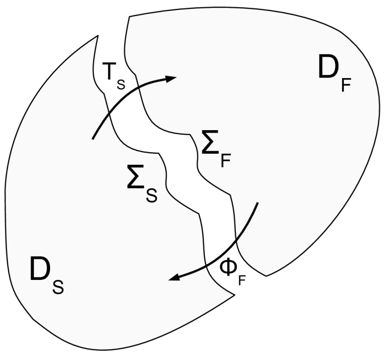


Figure 7.1: Dirichlet Neumann coupling

To remedy stability issues due to weak coupling, Chemin et al.[20] and Duchaine et al.[32] introduced a mixed boundary condition[1], Fig 7.2, where relaxation coefficients are used to stabilize the coupling scheme.

However, finding the best relaxation coefficient to remain stable while maintaining a good convergence rate is not obvious. To solve this optimization problem, Duchaine et al.[32] uses a numerical method called the amplification matrix[47] to compute stability domains for these simulations. A clear alternative to that methodology, is to exchange information far more often between the solvers without loosing computational performance. However Giles' result is only valid if we couple at every time step, which is a very restrictive condition. This is why the stability of a very tightly coupled Dirichlet/Neumann coupling for conjugate heat transfer simulation still needs to be investigated.

[1]A mixed boundary condition, also known as Robin boundary condition is in fact a linear combination of a dirichlet and a Neumann boundary condition.

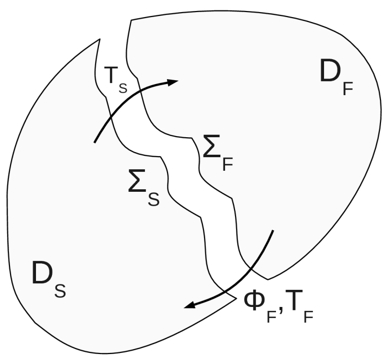


Figure 7.2: Dirichlet Robin coupling

The model used here is the the explicit time marching algorithm described in sect 4.1 of Giles[39]. For the analysis, the heat equation is discretized using finite differences on two one dimensional domains. These domains are coupled using a Dirichlet/Neumann interface at j=0. The negative domain ($j < 0$) receives the flux computed from the positive domain ($j > 0$). In the original analysis, each domain is semi-infinite and keeping Giles notations the following equations are obtained:

$$T_j^{n+1} = T_j^n + d_- \left(T_{j+1}^n - 2T_j^n + T_{j-1}^n\right), j < 0 \tag{7.1}$$

$$T_0^{n+1} = T_0^n - 2d_- \left(T_0^n - T_{-1}^n\right) + 2rd_+ \left(T_1^n - T_0^n\right) \tag{7.2}$$

$$T_j^{n+1} = T_j^n + d_+ \left(T_{j+1}^n - 2T_j^n + T_{j-1}^n\right), j > 0 \ , \tag{7.3}$$

$$\tag{7.4}$$

with

$$d_\pm = \frac{k_\pm \Delta t}{c_\pm {\Delta x_\pm}^2} \tag{7.5}$$

$$r = \frac{c_+ \Delta x_+}{c_- \Delta x_-} \ . \tag{7.6}$$

Using the Godunov Ryabenkii method[41] Giles then proves that this system remains stable provided that

$$r < r_{lim} \ , \tag{7.7}$$

with

$$r_{lim} = \frac{\sqrt{1 - d_-}}{1 - \sqrt{1 - d_+}} \ . \tag{7.8}$$

However this model is not compatible with a situation where each solver runs independently and exchanges information with a different iteration count between two coupling events. A looser model (Fig 7.3) with different time steps $\Delta t_\pm$, different interface temperatures T_{0+}, T_{0-} and different iteration count N_+ and N_- between each coupling update would write:

- In the domain (-), the iteration number is n_-:

$$T_j^{n_-+1} = T_j^{n_-} + d_- \left(T_{j+1}^{n_-} - 2T_j^{n_-} + T_{j-1}^{n_-}\right), j < 0 \tag{7.9}$$

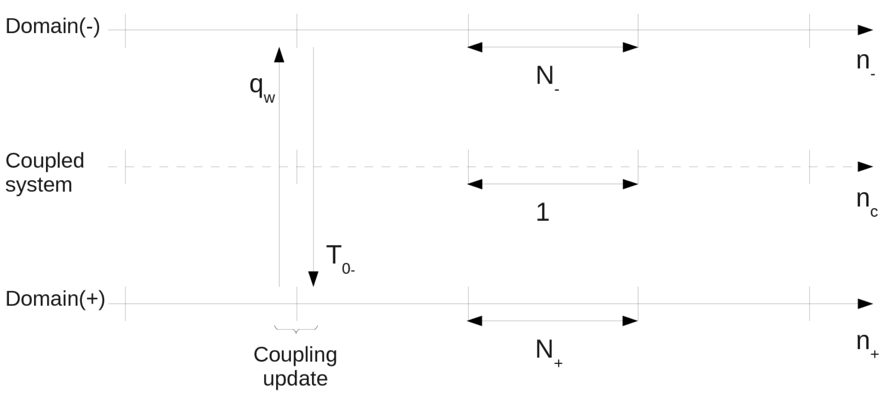


Figure 7.3: Loose coupling model, 3 temporal lines (in iteration count) are presented: one for the domain(-) one for the domain(+) and one for the coupled system.

- In Giles paper the equation (4.1) describing the evolution of the temperature at j=0 reads

$$\frac{c_{-}\Delta x_{-}}{2\Delta t}\left(T_0^{n+1}-T_0^n\right)=-q_w-\frac{j_{-}}{\Delta x_{-}}\left(T_0^n-T_{-1}^n\right) \tag{7.10}$$

  with

$$q_w=-\frac{k_{+}}{\Delta x_{+}}\left(T_1^n-T_0^n\right) \tag{7.11}$$

  Because in Giles case $\Delta t_{+}=\Delta t_{-}=\Delta t$ combining Eq. (7.10) and Eq. (7.11) resulted in Eq. (7.2). However with two separate time steps the equation obtained is slightly different:

$$T_{0_{-}}^{n_{-}+1}=T_{0_{-}}^{n_{-}}-2d_{-}\left(T_{0_{-}}^{n_{-}}-T_{-1}^{n_{-}}\right)+2\frac{\Delta t_{-}}{\Delta t_{+}}rd_{+}\left(\tilde{T}_1^{n_c}-\tilde{T}_{0_{+}}^{n_c}\right) \tag{7.12}$$

  The Temperatures marked with a $\tilde{T}$ are the values which are updated at each coupling iteration n_c, i.e. these values are exchanged between the solvers every N_{+} for the domain (+), N_{-} for the domain (-).

- In the domain (+), the iteration number is n_{+}:

$$T_{0+}^{n_{+}+1}=\tilde{T}_{0+}^{n_c} \tag{7.13}$$

$$T_j^{n_{+}+1}=T_j^{n_{+}}+d_{+}\left(T_{j+1}^{n_{+}}-2T_j^{n_{+}}+T_{j-1}^{n_{+}}\right),j>0 \tag{7.14}$$

To study the stability of this new system the same numerical approach as Duchaine et al.[32] has been used. The domains have now a finite length m, they both have imposed temperatures on the borders. The entire system is thus written in matrix form. To represent the inter-coupling iterations and the coupling iterations, a matrix is built for each domain so that:

$$T_j^{n+1}= M_{-}T_j^n, for\ domain\ (-) \tag{7.15}$$

$$=\begin{pmatrix} 1 & & & & & & \\ d_{-} & 1-2d_{-} & d_{-} & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & d_{-} & 1-2d_{-} & d_{-} & & \\ & & & -2d_{-} & 1-2d_{-} & -2\frac{\Delta t_{-}}{\Delta t_{+}}rd_{+} & +2\frac{\Delta t_{-}}{\Delta t_{+}}rd_{+} \\ & & & & & 1 & \\ & & & & & & 1 \end{pmatrix}\begin{pmatrix} T_{-m} \\ T_{-(m-1)} \\ \vdots \\ T_{-1} \\ T_{0-} \\ \tilde{T}_{0+} \\ \tilde{T}_1 \end{pmatrix}$$

$$\begin{aligned}
T_j^{n+1} &= M_+ T_j^n, for\ domain\ (+) \\
&= \begin{pmatrix} 1 & & & & \\ d_+ & 1-2d_+ & d_+ & & \\ & \ddots & \ddots & \ddots & \\ & & d_+ & 1-2d_+ & d_+ \\ & & & & 1 \end{pmatrix} \begin{pmatrix} T_{0+} \\ T_1 \\ \vdots \\ T_{m-1} \\ T_m \end{pmatrix}
\end{aligned} \tag{7.16}$$

These matrices are then raised to the power of the domain inter-coupling iteration count $N_\pm$ and assembled as follows:

$$M_{assemble} = \begin{pmatrix} \left(M_-\right)^{N_-} & \\ & \left(M_+\right)^{N_+} \end{pmatrix} \tag{7.17}$$

Using a new matrix M_{cpl} we can write that domain(-) transfers its current wall temperature to domain(+) and domain(+) transfers its current wall flux, through two border temperatures, to domain(-). This matrix is defined by

$$M_{cpl} = \begin{array}{c} \\ T^o_{-m} \\ \vdots \\ T^o_{0-} \\ \tilde{T}^o_{0+} \\ \tilde{T}^o_1 \\ T^o_{0+} \\ T^o_1 \\ \vdots \\ T^o_m \end{array} \begin{array}{c} \begin{array}{ccccccccc} T^i_{-m} & \dots & T^i_{0-} & \tilde{T}^i_{0+} & \tilde{T}^i_1 & T^i_{0+} & T^i_1 & \dots & T^i_m \end{array} \\ \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & 0 & & 1 & & & \\ & & & & 0 & & 1 & & \\ & & 1 & & & 0 & & & \\ & & & & & & 1 & & \\ & & & & & & & \ddots & \\ & & & & & & & & 1 \end{pmatrix} \end{array} \tag{7.18}$$

where the input values and output values to this linear transform are noted respectively with the superscripts ^{i} and ^{o}.

The complete system is then built by multiplying $M_{assemble}$ and M_{cpl} resulting in the amplification matrix of the system M_{amp}.

$$M_{amp} = M_{cpl} \cdot M_{assemble} \tag{7.19}$$

The system is unstable if the spectral radius of M_{amp} is greater than 1. Using a numerical code it is possible to compute the eigen values of M_{amp} and thus deduce the stability of the system.

To validate this numerical approach, the simple test case of Giles is recomputed and compared to its analytical result. In this case the amplification matrix for the system which is tightly coupled (at each iteration) is built. This procedure is applied for varying values of the physical properties of domain (+) as described in Table 7.1, where u is a scalar which varies linearly between 0 and 1 (500 steps). Hence at each time step the Fourier values r and r_{lim} vary with u. The spectral radius M_{amp} is also computed for each u. The physical constants used for the computation are summarized in Table 7.1.

The results showed on Fig 7.4 show that both methods do agree:

- when $r < r_{lim}$ the system remains stable $\rho(M_{amp}) \leq 1$,
- when $r > r_{lim}$ the system becomes unstable $\rho(M_{amp}) > 1$.

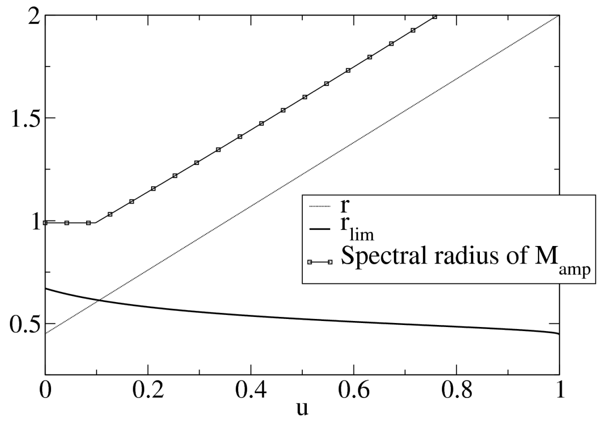


Figure 7.4: This graph shows two stability conditions together we can see that when $r > r_{max}$, $\rho(M) > 1$.

Domain	k	c	Δx	Δt
-	0.2	2000	0.005	0.1
$+$	$0.1 + 0.4u$	$900 + 1300u$	0.005	0.1

Table 7.1: Parameters used for the validity test

In this work we are interested in applying this methodology to an industrial application (part V). Hence all the numerical values used such as time step, grid size, capacity and conductivity are characteristic values for the target application (Table 7.2). The spectral radius of M_{amp} has been calculated for N_+ varying between 1 and 100 and N_- varying between 1 and 500 and results are presented on Fig. 7.5. It shows that for all points calculated the spectral radius is below 1, meaning that the simulation is stable within these limits. This result is coherent with Duchaine et al.[32] which showed that the Dirichlet/Neumann is stable for very tightly coupled systems.

Hence thightly coupling the flow and the conduction solver ensures that:

- the coupling scheme remains stable,
- the aliasing problems are avoided, meaning that the average of the coupled problem can converge over a minimal time interval.

The main constraint is however to minimize the coupling overhead in a massively parallel environment. To do so a methodology targeting modern hardware treating coupled problems is developed.

Domain	$k(W.m^{-1}.K^{-1})$	$c_m(J.kg^{-1}.K^{-1})$	$\rho(kg.m^{-3})$	$\Delta x(mm)$	$\Delta t(s)$
(-) - solid	50	440	7900	0.3	1e-5
(+) - fluid	0.1	1000	1.3	0.3	1e-7

Table 7.2: Characteristic parameters used in the simulation

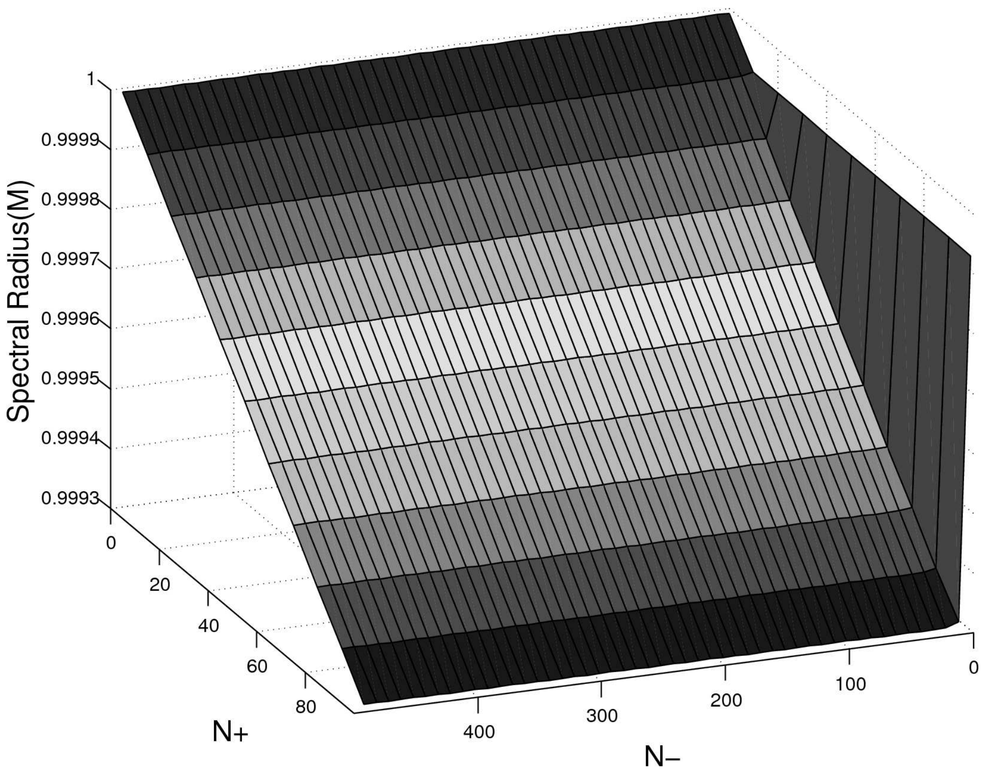


Figure 7.5: Spectral radius of the system function of N_+ and N_-

Conclusion

In this part a methodology for coupling an unsteady LES solver to a thermal solver has been established, its accuracy and stability have been assessed. Also some numerical alternatives to handle unsteady coupling remain to be investigated: the aliasing problem illustrated may be handled by adding a tuned low pass filter inside the unsteady fluid solver and exchange the filtered quantities. However the problems considered in this part have been reduced to simple zero or one dimensional problems. Hence in these simplistic problems exchanging boundary conditions between the solvers is trivial, i.e. one only needs to transfer one or two scalars. Yet to be able to handle complex geometries, such as those used in an aeronautical burner, the problem is somewhat more complex. Indeed the boundary conditions are discretized on complex non matching grids. This leads to the next topic investigated in this thesis.

Part III

Interpolation methods for unstructured grid coupling

Table of Contents

Nomenclature

λ_i	Barycentric coordinates
ω	Signal frequency
$dist(A, B)$	Euclidean distance between points A and B
E_i^d	Element i of destination mesh
E_i^s	Element i of source mesh
E_i	Edge E_i
$Edge(T, i)$	i^{th} edge of traingle T
G^d	Destination grid
G_i^d	Vertex i destination grid
G_i^d	Vertex i of destination mesh
G^s	Source mesh
G_i^s	i^{th} vertex of source mesh
h_Box	Impulse response of the box filter
h_tent	Impulse response of tent filter
N^d	Number of vertices of the destination mesh
N^s	Number of vertices of source mesh
$N_v(E)$	Vertex count of element E
S_i^k	k^{th} point of segment S_i ($k = 0$ or 1)
T_i^B	i^{th} vertex of triangle B
T_j^s	Triangle j of source mesh
T_ij	Transformation matrix
U	Field to transport

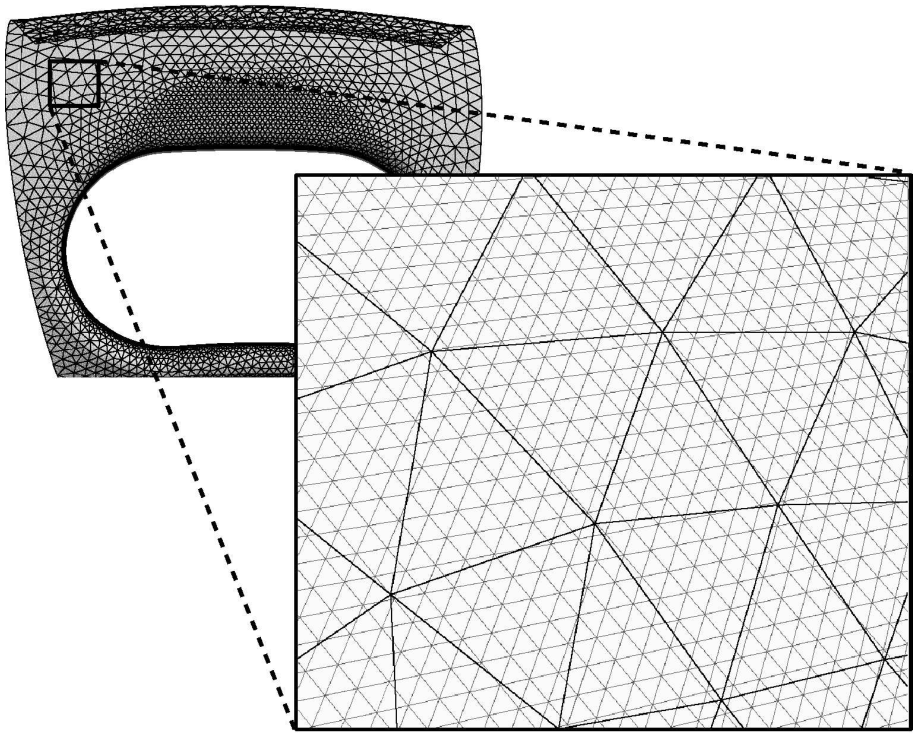


Figure 7.6: An example of an interface discretized by two separate grids

Using multiple codes to perform multi-physical simulations generally implicates using several computational grids since each solver focuses on a particular physic and therefore has different numerical constraints. In such a situation the grid interfaces can be computed in order to match in which case transferring data between the grids is straightforward. However this is not always possible, therefore when the grids do not match, Fig 7.6, the data fields have to be interpolated in order to be exchanged.

Using structured solvers interpolation can be relatively easily computed because there is generally an explicit relation between the points spatial positions and their memory locations[2]. With unstructured solvers performing interpolation is harder because this relation is not explicit, complex geometrical search algorithms are required. Considering the geometrical search problem solved and that for each point of the destination mesh the nearby source mesh data points are known, the actual procedure to build the destination data has still to be defined. Knowing that the computations in this thesis use simplicial elements[3], only interpolation methods relying on linear transforms have been considered. The available methods are hence compared on simple basic cases and the differences are explained using basic signal theory. Then the construction of the linear transforms from a source grid G^s to a destination grid G^d is treated. Finally the geometrical algorithms used are presented.

[2]For a 3D problem the relation between the location i, j, k of each point in a 3 dimensional buffer and the spatial coordinates x, y, z can be given by a set of polynomials.

[3]Elements composed of $n+1$ points in a n dimensional space: a line segment is simplex in a 1D, a triangle is a simplex in 2D, a tetrahedron in 3D...

Looking at the summary problem, Fig 7.7, this part corresponds to the interpolation row.

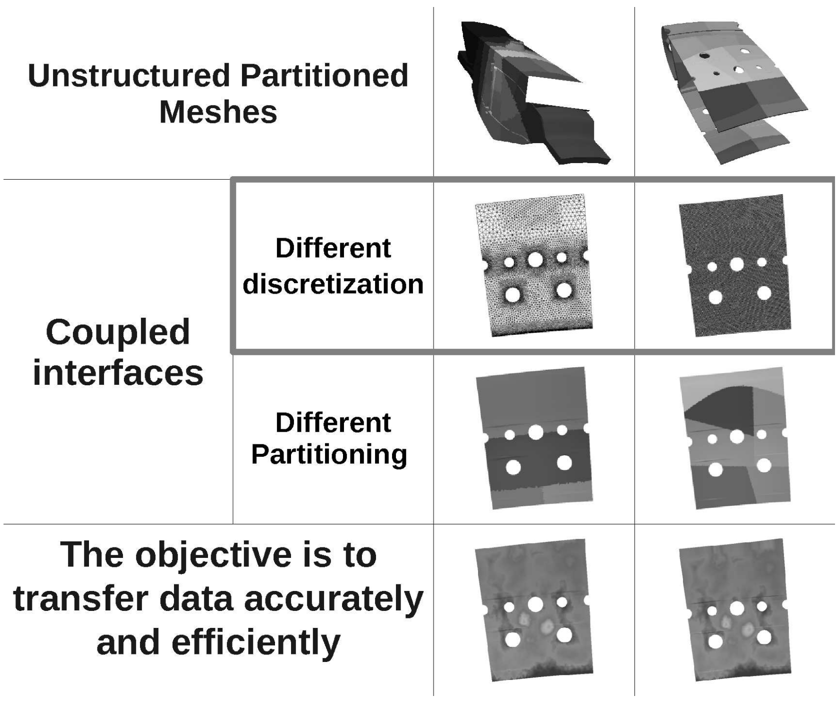

Unstructured Partitioned Meshes
Coupled interfaces
Different discretization
Different Partitioning
The objective is to transfer data accurately and efficiently

Figure 7.7: Summary of the technical issues for data coupling. In the context of this thesis coupling involves two unstructured partitionned meshes which are connected at geometrical interfaces. These interfaces may have different discretizations and partitionning. The goal is to transfer data between them efficiently while preserving the original signals.

Chapter 8

The basics of interpolation

To understand the complex algorithms and interpolation methods it is first necessary to understand the fundamental aspects behind interpolation, notably sampling theory.

8.1 Sampling based interpolation

In the coupling community and more generally in the CFD community *interpolation* refers to the process which transfers data fields from a grid to an other. The interpolation process can be broken down into several steps:

- the first is signal reconstruction: a signal is reconstructed from the sampled data on grid G^s through convolution with a low pass filter, resulting into an interpolated signal.
- this interpolated signal is sampled using the destination grid G_i^d

In order to understand this process, the basics of signal reconstruction will be discussed, then the entire process of signal interpolation will be illustrated through detailed examples. Note that in this discussion the function are interpolated in space therefore the time variable t which is present within many similar discussions is replaced by the space variable x.

8.1.1 Signal reconstruction

In the strict sense interpolation is the construction of a continuous signal $f_i(x)$, Fig. 8.2, from a sampled signal $f_s(x)$, Fig. 8.1.

To understand the properties of interpolation it is important to introduce the continuous signal $f(x)$. Sampling $f(x)$ on the source grid points $G_n^s = n\Delta x$ yields the sampled signal $f_s(t)$:

$$f_s(x) = \sum_{n=-\infty}^{+\infty} f(G_n^s)\delta(x - G_n^s) = \sum_{n=-\infty}^{+\infty} f(n\Delta x)\delta(x - n\Delta x) = f(x) \sum_{n=-\infty}^{+\infty} \delta(x - n\Delta x) \tag{8.1}$$

where $\delta(x)$ is the Dirac function and $\sum_{n=-\infty}^{+\infty} \delta(x - n\Delta x)$ a Dirac comb. As states the first part of the Shannon theorem, because of the sampling process the spectrum of $f_s(x)$ is the composed of an infinite number of duplicates of the spectrum of $f(x)$ spaced by the sampling frequency called here $f_{G^s} = \frac{1}{\Delta_x}$. Hence to be able to reconstruct the original signal the duplicates of $f(x)$'s spectrum must

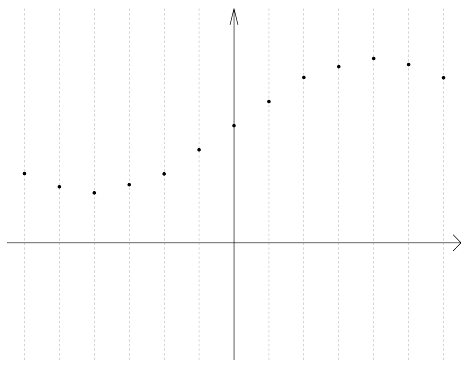


Figure 8.1: A sampled signal

be eliminated. This can be performed in frequency domain by multiplying the spectrum of $f_s(x)$ by a function G_{ideal} defined by:

$$G_{ideal}(f) = \begin{cases} 1 \ on \ \left[\frac{-f_{G^s}}{2}, \frac{f_{G^s}}{2}\right] \\ 0 \ elsewhere \end{cases} \tag{8.2}$$

Hence G_{ideal} is an ideal low pass filter.

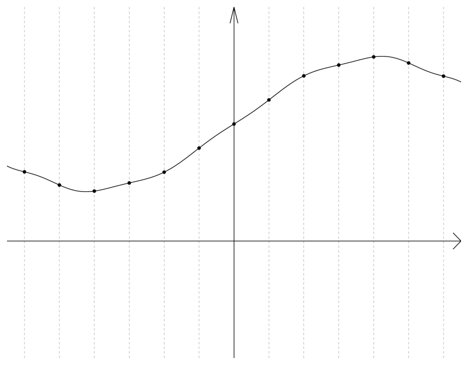


Figure 8.2: The reconstructed continuous signal through interpolation

However instead of doing the low pass filtering in the frequency domain which would require a Fourier transform and inverse transform to pass from the time domain to the frequency domain and vice versa, the low pass filtering is obtained by calculating the convolution in spatial domain of the spatial function associated to $G_{ideal}(f)$ called the impulse response $h_{ideal}(x)$.

$$f_i(x) = h_{ideal}(x) * f_s(x) = \sum_{n=-\infty}^{+\infty} f(G_n^s) h_{ideal}(x - G_n^s) \tag{8.3}$$

The impulse response of G_{ideal} is a *sinc* function ($sinc : x \mapsto \frac{sin(x)}{x}$), this is unfortunate because *sinc*'s support is not bounded, i.e. there is no segment S such as $x \notin S \Rightarrow sinc(x) = 0$. Meaning that in space the stencil used to compute the convolution between h_{ideal} and $f_s(x)$ is infinite. In order to compute this convolution, approximations to the ideal filter are used. In this discussion only the box and the tent filter will be considered because they are the only filters which are useful when dealing with non structured grids composed of P1 elements[1]. Also studying basic interpolation problems using these filters is enough to demonstrate the fundamentals of signal reconstruction.

[1]Although the *sinc* function converges rapidly to 0, interpolation with G_{ideal} would require a stencil larger than one element which is very complex to implement on unstructured grids.

8.1.1.1 The Box filter

The box filter [10], Fig. 8.3, is defined in space by $h_{Box}(x)$:

$$h_{Box}(x) = \begin{cases} 1 \ x \in \left[\frac{-1}{2}, \frac{1}{2}\right] \\ 0 \ x \notin \left[\frac{-1}{2}, \frac{1}{2}\right] \end{cases} \tag{8.4}$$

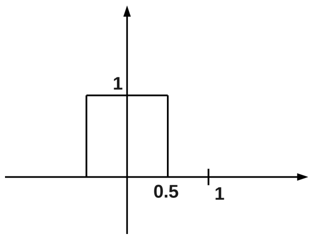


Figure 8.3: The impulse response of the box filter

This filter can be used to produce the *nearest neighbor* interpolation. The signal obtained through this interpolation is composed of a succession of steps. In the frequency domain the spectrum of h_{Box} is $\Delta x sinc(\pi f \Delta x)$, Fig. 8.4. The spectrum of the impulse response of the box filter h_{Box} can be considered as the filter's frequency domain transfer function.

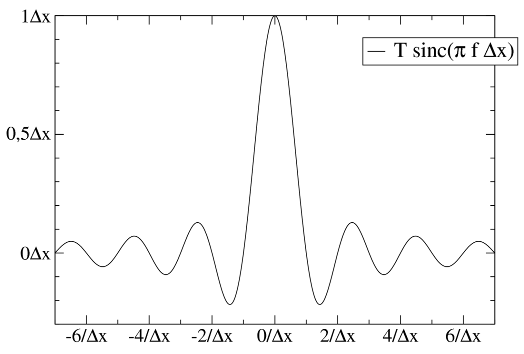


Figure 8.4: The Fourier transform of the box filter's impulse response

8.1.1.2 The tent filter

The tent filter [10], Fig. 8.5, is defined in space by $h_{tent}(x)$:

$$h_{tent}(x) = \begin{cases} 0, \ x \in]-\infty, -1] \\ x+1, \ x \in [-1, 0] \\ -x+1, \ x \in [0, 1] \\ 0, \ x \in [1, \infty[\end{cases} \tag{8.5}$$

This filter can be used to produce the *linear* interpolation. The signal is obtained by linking the sampled points with straight lines.

It is interesting to notice that the impulse response of the tent filter, Fig. 8.6, is obtained by the convolution of a box filter with it self, hence in the frequency domain the spectrum of h_{tent} is the square of the box filter's spectrum [10]: $(\Delta x sinc(\pi f \Delta x))^2$.

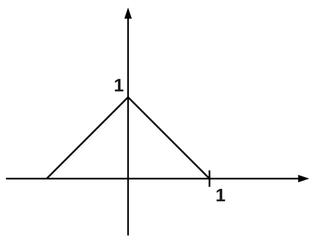


Figure 8.5: The impulse response of the tent filter

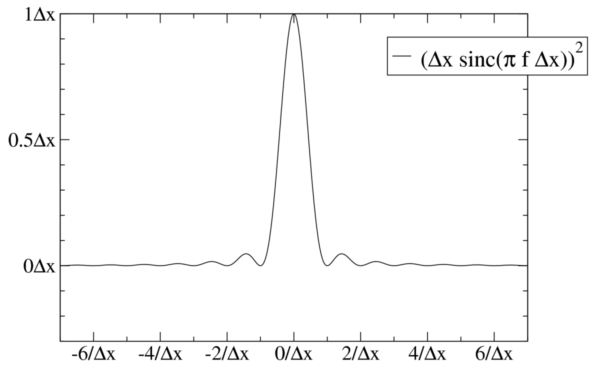


Figure 8.6: The Fourier transform of the tent filter's impulse response

8.1.2 Grid to Grid Interpolation example

In order to fully illustrate the interpolation from a grid G^s to a grid G^d, the process from the original continuous signal to the final continuous signal is detailed in 3 steps. In this example the original continuous signal is a sinusoid of period 5, the source grid is a regular grid spaced by $\Delta x^s = 0.66$, the destination grid is also a regular grid spaced by $\Delta x^d = 0.3$.

1 at first the continuous signal, Fig. 8.7, is sampled on to the source grid G^s, yielding a sampled signal which is only defined at the source grid points G^s_i, Fig. 8.8,

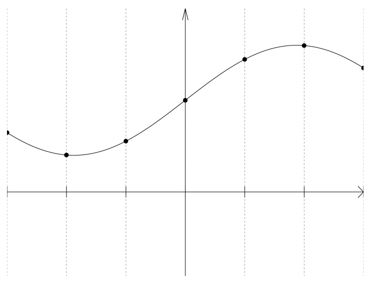


Figure 8.7: The continuous signal

2 in order to calculate the values of the signal on the destination grid points G^d_i, a new signal defined everywhere must be reconstructed, hence the sampled signal is convoluted with a chosen low pass filter, Fig. 8.9. Therefore a new signal called the interpolated signal is defined for every point in space, Fig. 8.10.

3 The interpolated signal is sampled on the destination grid points G^d_i, Fig. 8.10, yielding a signal defined only at G^d_i, Fig. 8.11.

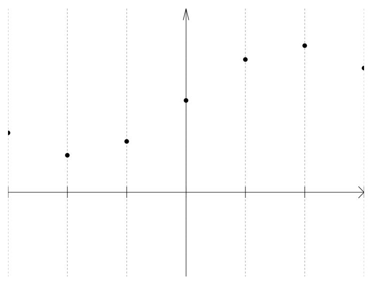


Figure 8.8: The sampled signal on source grid

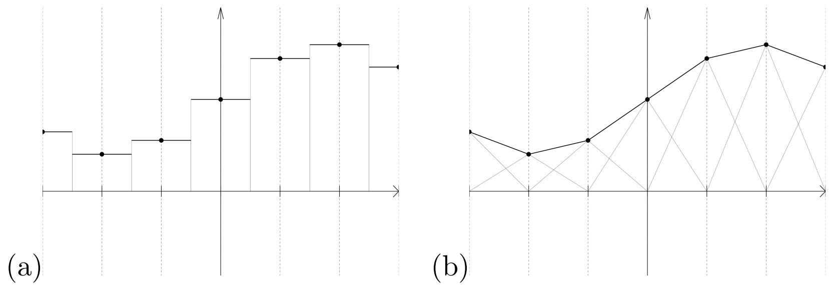


Figure 8.9: The convolution with the low pass filter: (a) box filter, (b) tent filter

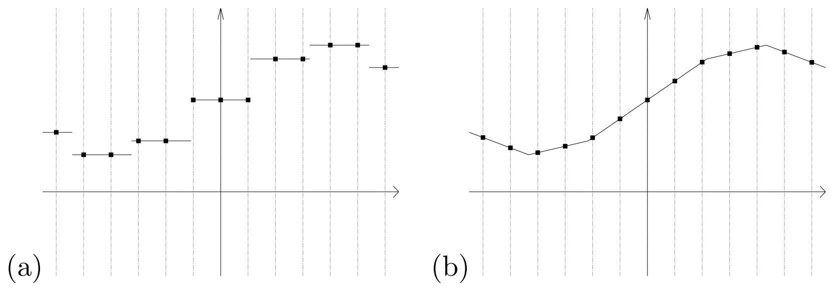


Figure 8.10: The reconstructed signal: (a) box filter, (b) tent filter

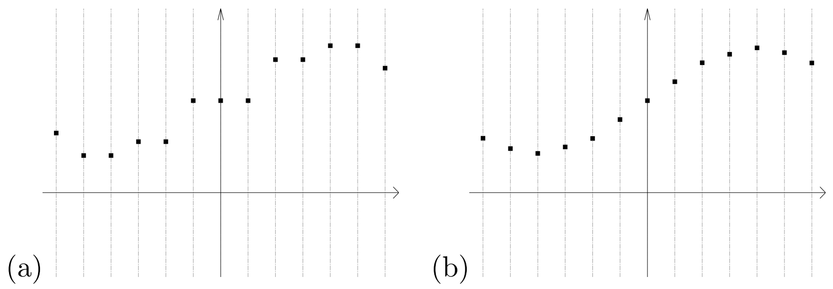


Figure 8.11: The sampled signal on destination grid: (a) box filter, (b) tent filter

4 Finally to visualize the signal represented on the destination mesh a continuous signal is reconstructed using polynomial interpolation, Fig. 8.12. This view is added to compare with the original continuous signal in, Fig. 8.7.

It is important to understand that the grid to grid interpolation process is represented by the figures 8.9, 8.10, 8.11.

This example shows the clear difference between the two interpolation methods. It is important to notice that in this example the Nyquist frequency is respected for the sampling processes, however the reconstruction process does not respect the Shannon theorem: the perfect signal reconstruction is approximated using either the box or the tent filter instead of the perfect low pass filter. Therefore

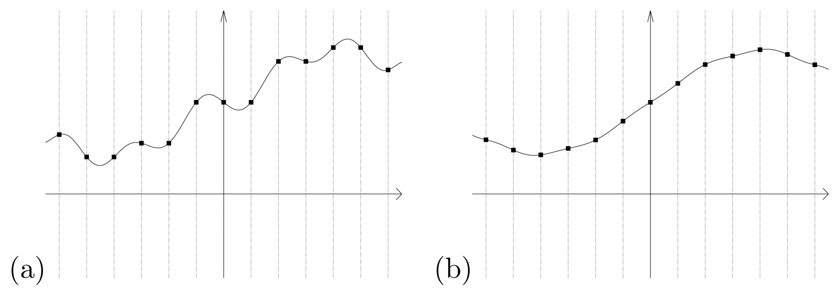


Figure 8.12: The final continuous signal: (a) box filter, (b) tent filter

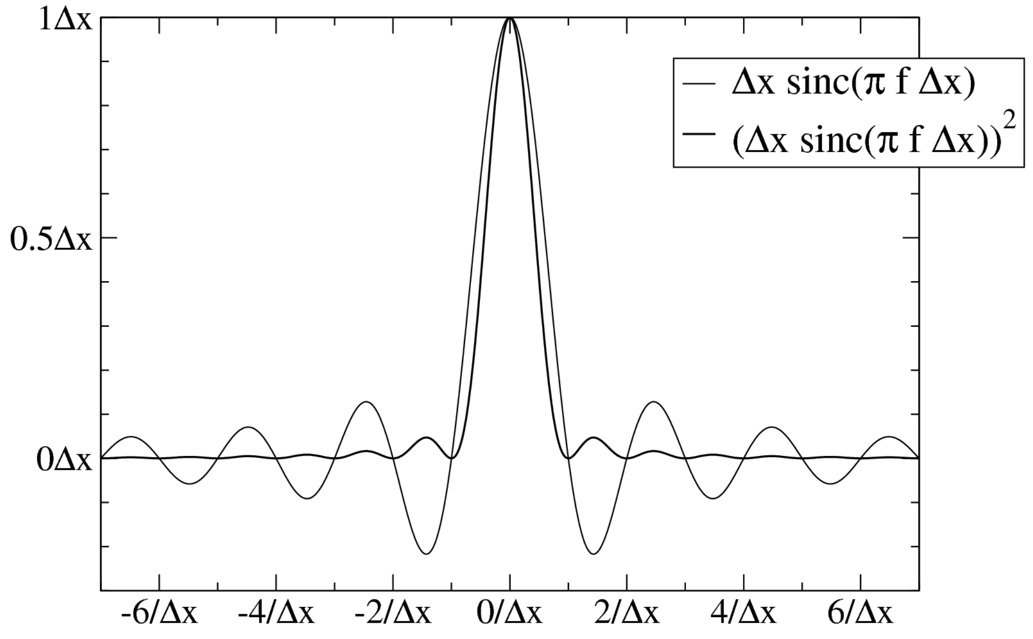
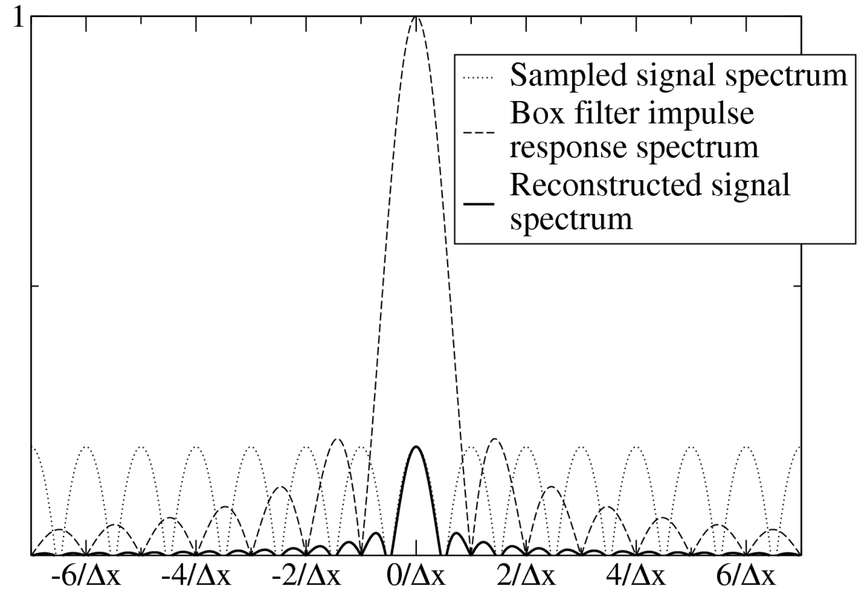
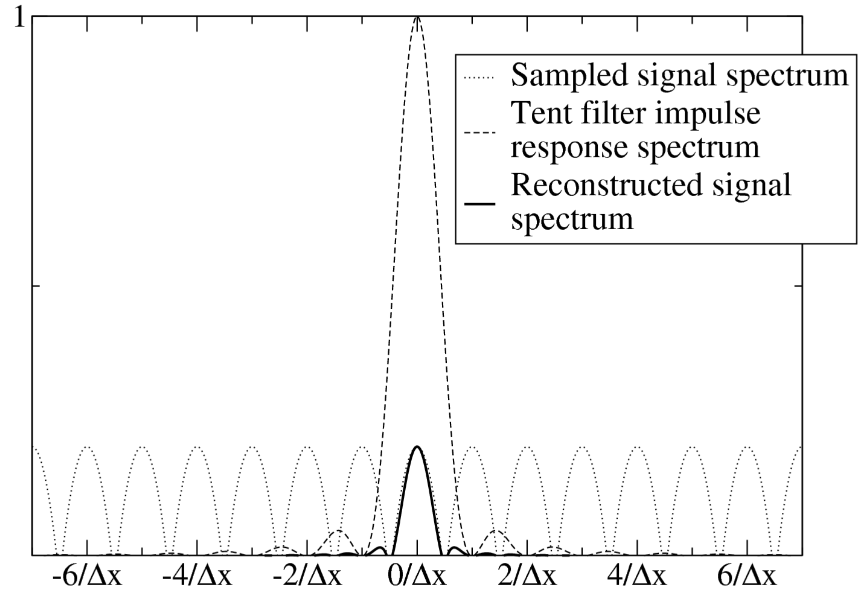


Figure 8.13: Comparisons of the Fourier transform of the tent and box filter's impulse response

even if the spectrum of the continuous signal does not contain components at frequencies higher than the Nyquist frequency, during the signal reconstruction phase the replicates of the continuous signal spectrum are not eliminated by the non-perfect low pass filter. Better results are obtained using the tent filter which can be simply explained by looking at the Fourier transform of the filters impulse response (it's transfer function): the tent filter's transfer function is the square of the box filter's, Fig. 8.13. Hence the tent filter damps more the replicates of the continuous signal spectrum than the box filter, Fig. 8.14. On this example a continuous signal having a parabolic spectrum is interpolated on a regular grid. The figures shows the sampled signal spectrum, the filter transfer function (Fourier transform of the impulse response) and the reconstructed signal's spectrum. Figure 8.15 shows the comparison of the spectrum of the two reconstructed signals.



(a)



(b)

Figure 8.14: Frequency view of signal reconstruction using: (a) box filter, (b) tent filter

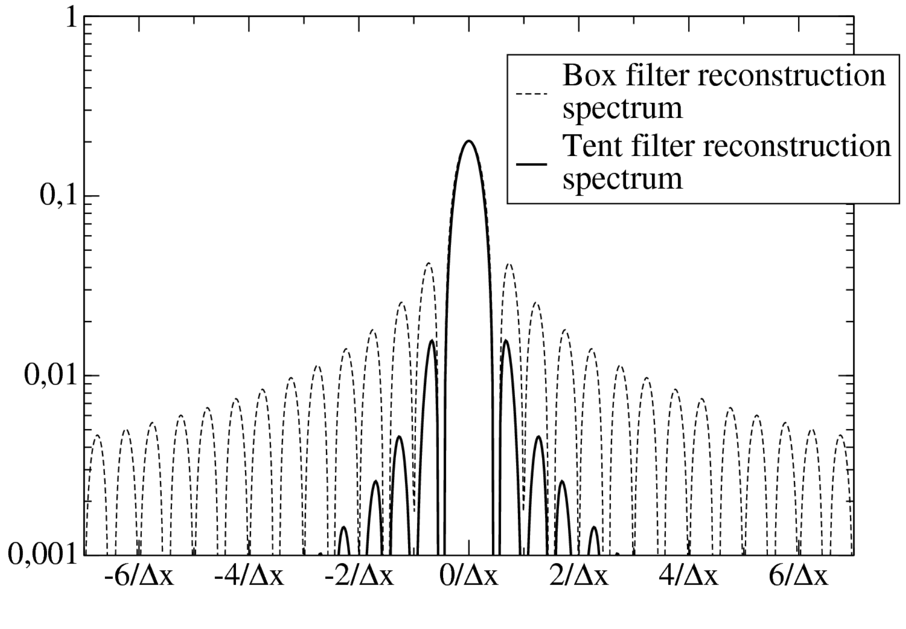


Figure 8.15: Comparisons of reconstructed signal spectrum using box and tent filters

8.2 Conservative interpolation

In some problems such as flux transport between grids, it is necessary to add an extra constraint to the interpolation process: the interpolation process should preserve the integral of the signal from G^s to G^d:

- for 1D methods:

$$\int_{G^d} x^d du = \int_{G^s} x^s du \tag{8.6}$$

- for surface methods:

$$\int_{G^d} x^d ds = \int_{G^s} x^s ds \tag{8.7}$$

- for volumetric methods:

$$\int_{G^d} x^d dv = \int_{G^s} x^s dv \tag{8.8}$$

Methods respecting this constraint are referred in this thesis as *conservative interpolation* methods. Different approaches exist to obtain conservative interpolation: a first approach is to consider the interpolation coefficients as a set of free scalars and then derive a set of constraint equations which ensure that the interpolation method remains conservative[21]. An other approach consists in computing mesh-to-mesh intersections (Cell-Intersection-Based Donor-cell method) or by approximating the mesh-to-mesh intersection (Simplified Face Based Donor-Cell)[69]. A third approach (still based on mesh-tomesh intersections) is called the supermesh approach, a mesh containing the set of intersection elements between two meshes[35] is built and used to transfer data from and to the input meshes. A conservative interpolation method has been developed during this thesis, it is based on element over element projection (Similar to the Cell-Intersection-Based Donor-cell[69]). Projecting elements over elements in the three dimensions is a complex task which is described in 9.3.1.

8.2.1 Conservative interpolation in 1D

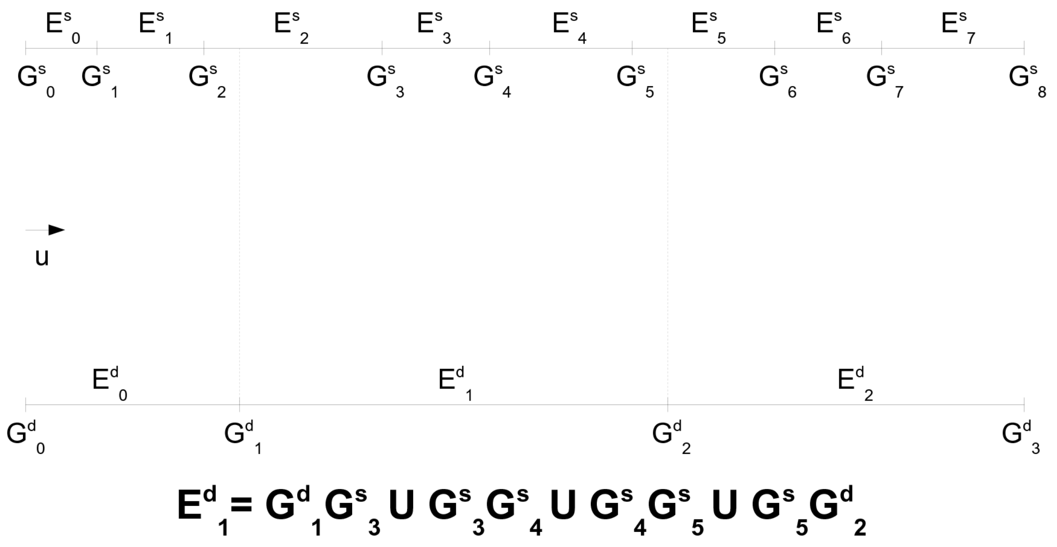

E^s_0 E^s_1 E^s_2 E^s_3 E^s_4 E^s_5 E^s_6 E^s_7
G^s_0 G^s_1 G^s_2 G^s_3 G^s_4 G^s_5 G^s_6 G^s_7 G^s_8
u
E^d_0 E^d_1 E^d_2
G^d_0 G^d_1 G^d_2 G^d_3
$E^d_1 = G^d_1 G^s_3 \cup G^s_3 G^s_4 \cup G^s_4 G^s_5 \cup G^s_5 G^d_2$

Figure 8.16: Conservative interpolation example

In order to introduce the concepts of conservative interpolation, the one-dimensional case is presented. In this discussion a signal x^s known on the source grid G^s is transferred to the destination grid

G^d, Fig. 8.16 shows an example of this setup. The source grid has N points the destination grid has M points. The axis is called $(O, \vec{u})$, the coordinates of the source grid points are noted $G_i^s = \vec{u} \cdot \vec{OG_i^s}$, the destination grid points are noted $G_i^d = \vec{u} \cdot \vec{OG_i^d}$. The grids G^s and G^d are supposed to be arbitrary. The points are indexed such as $G_1^s < G_2^s < \ldots < G_N^s$ and $G_1^d < G_2^d < \ldots < G_M^d$. Both grids discretize the same portion of space, hence $G_1^s = G_1^d$ and $G_N^s = G_M^d$.

The elements of the source grid are segments which link the points G_i^s and G_{i+1}^s, the segment between G_i^s and G_{i+1}^s is noted E_i^s. Therefore the segment of the destination grid between G_i^d and G_{i+1}^d is noted E_i^d. The destination grid has $M-1$ segments, the source has $N-1$ segments.

The projection step is applied for each segment of the destination grid E_i^d. The idea is to decompose E_i^d in a set of elements defined using G^s points. The projection of E_i^d on G^s is noted $P(E_i^d \mapsto G^s)$ and $P(E_i^d \mapsto G^s) = \cup_j S_j$ where S_j are segments. The points defining the segments S_j are linear combinations of points of the source grid G^s. For convenience the notation $S(A, B)$ is the segment defined by the points A and B, S_i^0 and S_i^1 are the two points defining the segment S_i with $S_i^0 < S_i^1$.

In a one dimensional case $P(E_i^d \mapsto G^s)$ can be simply expressed:

- if E_j^s satisfies $G_i^d < G_j^s$ and $G_{j+1}^s < G_{i+1}^d$ then $S(G_j^s, G_{j+1}^s) \in P(E_i^d \mapsto G^s)$, Fig. 8.17(a),
- if E_j^s satisfies $G_j^s < G_i^d$ and $G_{i+1}^d < G_{j+1}^s$ then noting that $G_i^d = \alpha G_j^s + (1-\alpha)G_{j+1}^s$ and $G_{i+1}^d = \beta G_j^s + (1-\beta)G_{j+1}^s$, with $\alpha, \beta \in [0,1]$ we can write $S(\alpha G_j^s + (1-\alpha)G_{j+1}^s, G_j^s + (1-\beta)G_{j+1}^s) \in P(E_i^d \mapsto G^s)$, Fig. 8.17(b),
- if E_j^s satisfies $G_j^s < G_i^d$ and $G_i^d < G_{j+1}^s$ then noting that $G_i^d = \alpha G_j^s + (1-\alpha)G_{j+1}^s$, with $\alpha \in [0,1]$ we can write $S(\alpha G_j^s + (1-\alpha)G_{j+1}^s, G_{j+1}^s) \in P(E_i^d \mapsto G^s)$, Fig. 8.17(c),
- if E_j^s satisfies $G_j^s < G_{i+1}^d$ and $G_{i+1}^d < G_{j+1}^s$ then noting that $G_{i+1}^d = \beta G_j^s + (1-\beta)G_{j+1}^s$, with $\alpha \in [0,1]$ we can write $S(G_j^s, \beta G_j^s + (1-\beta)G_{j+1}^s) \in P(E_i^d \mapsto G^s)$, Fig. 8.17(d),

A method to conserve the integral during the interpolation process is to break the integral over the destination grid into the integral over each element of the destination grid:

$$\int_{G^d} x^d(u)du = \sum_{i=1}^{M-1} \int_{E_i^d} x^d(u)du \tag{8.9}$$

Where the integral over E_i^d can be calculated using $P(E_i^d \mapsto G^s)$:

$$\int_{E_i^d} x^d(u)du = \sum_{S_j \in P(E_i^d \mapsto G^s)} \int_{S_j} x^s(u)du \tag{8.10}$$

The integral can be calculated using different methods, here two are considered.

The first method considers that the field value is uniform on each source element. Therefore the integral is calculated using the ratio of the intersection segment length $|S_j|$ over the source element length $|E_k^s|$.

$$\int_{S_j} x^s(u)du = \frac{|S_j|}{|E_k^s|} \frac{x(G_k^s) + x(G_{k+1}^s)}{2} \tag{8.11}$$

This method is called in this document the *conservative interpolation* method. While simple this method does not preserve the element gradients which can be problematic notably when data is interpolated from low resolution grids to high resolution grids.

In this work a more accurate method is considered, since the elements are P1 the integral can be calculated using a trapezoidal integration rule. Therefore the gradient on the element is preserved

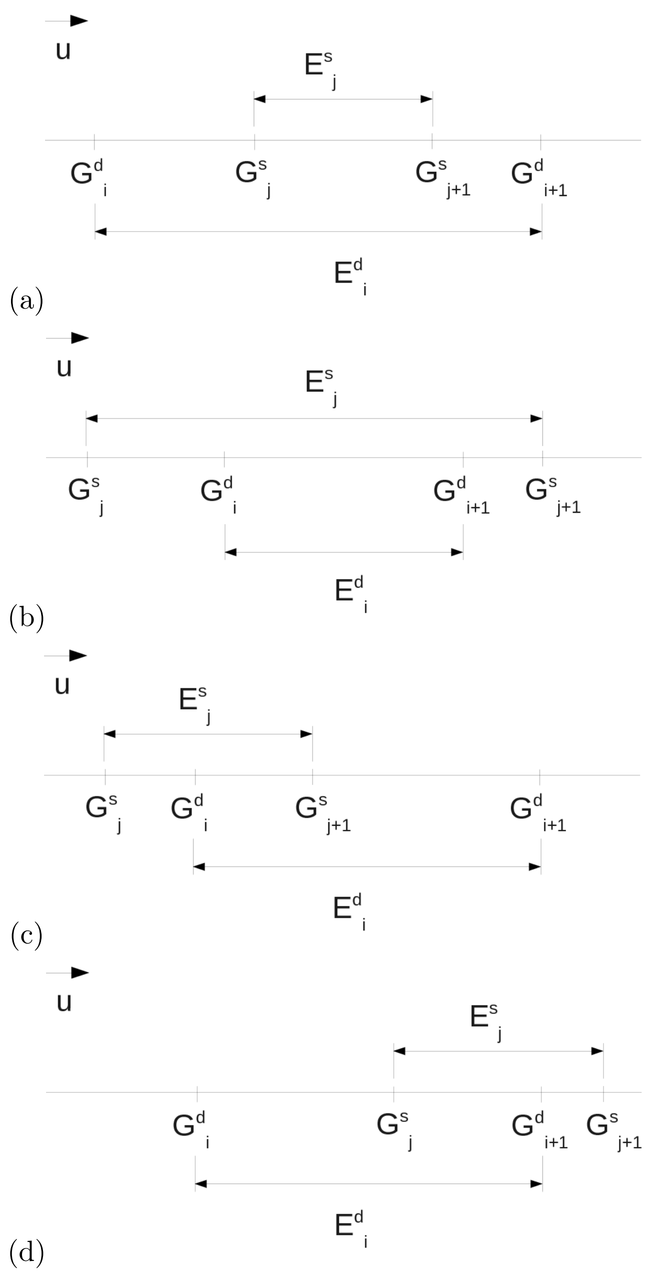


Figure 8.17: (a) E^s_j is contained in E^d_i, (b) E^d_i is contained in E^s_j, (c) E^s_j has intersection with E^d_i on its left, (d) E^s_j has intersection with E^d_i on its right

and considered in integral calculation. This method is called in this document *linear conservative interpolation*. The integral $\int_{S_j} x^s(u)du$ is calculated using the trapezoidal rule which is natural choice for P1 elements, hence in 1D:

$$\int_{S_j} x^s(u)du = \left(S^1_i - S^0_i\right) \frac{x^s(S^0_i) + x^s(S^1_i)}{2} \tag{8.12}$$

If S^k_i is a point of G^s then $x^s(S^k_i)$ is known since x^s is defined on G^s. On the other hand if S^k_i is linear combination of G^s points, $S^k_i = \sum_j \alpha_j G^s_j$, then by supposing that x^s varies linearly over each element (valid for P1 elements) of G^s, we can write that $x^s(S^k_i) = \sum_j \alpha_j x^s(G^s_j)$. Therefore the integral over each element of the destination grid G^d can be calculated using element on grid projection and the trapezoidal integration rule. This method can be viewed as a procedure to establish a mesh aware quadrature to calculate element integrals.

The value obtained is defined for the element (or cell), not for the vertices. In this thesis the solvers used store their data at the vertices therefore the cell-centered integral value must be distributed to the vertices in a conservative way, i.e. the integral over G^d must be preserved, Fig. 8.18. The distribution

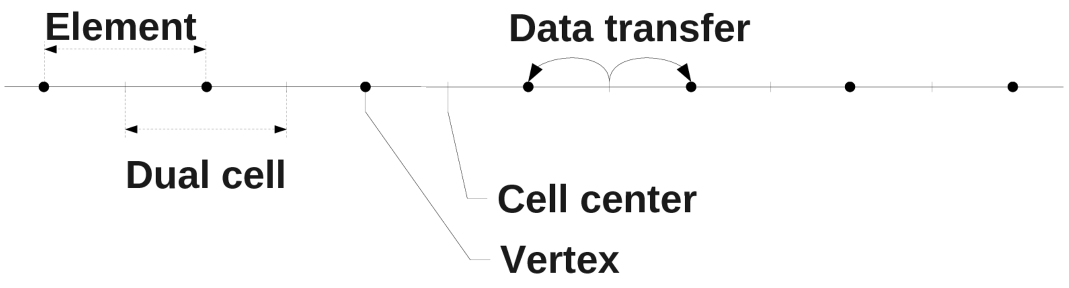


Figure 8.18: Conservative transfer from cells to vertices

method uses the dual cell concept to transfer conservatively the integral from the cells to the vertices. The dual cell concept is presented for complex elements in 9.3.1, in 1D the dual cells are simply defined by cutting each element E_i^d into two equal parts. The union of the parts which are adjacent to the vertex G_i^d are noted $Adj(G_i^d)$ which is the dual cell centered on G_i^d. The left and right halves of E_i^d are noted $lh(E_i^d)$ and $rh(E_i^d)$.

For the destination grid we can write that

$$Adj(G_1^d) = \{lh(E_1^d)\} \tag{8.13}$$

$$Adj(G_i^d) = \{rh(E_{i-1}^d), lh(E_i^d)\}, \text{ for } 1 < i < M \tag{8.14}$$

$$Adj(G_M^d) = \{rh(E_{M-1}^d)\} \tag{8.15}$$

$$\tag{8.16}$$

x^d is considered to be uniform on $Adj(G_i^d)$, therefore the vertex centered value is calculated by dividing the integral of x^d over the dual cell by the dual cell's weight (length, area, volume) noted $w(G_i^d) = w(Adj(G_i^d)) = \int_{Adj(G_i^d)} du$:

$$x^d(G_i^d) = \frac{\int_{Adj(G_i^d)} x^d(u)du}{w(G_i^d)} \tag{8.17}$$

Calculating the integral of x^d using the vertex centered values yields

$$\sum_{i=1}^{M} x^d(G_i^d)w(G_i^d) = \sum_{i=1}^{M} \int_{Adj(G_i^d)} x^d(u)du = \sum_{i=1}^{M} \sum_{E \in Adj(G_i^d)} \int_E x^d(u)du \tag{8.18}$$

Considering the definition of $Adj(G_i^d)$:

$$\sum_{i=1}^{M} \sum_{E \in Adj(G_i^d)} \int_E x^d(u)du = \int_{lh(E_1^d)} x^d(u)du + \int_{rh(E_1^d)} x^d(u)du + \int_{lh(E_2^d)} x^d(u)du + \ldots + \int_{rh(E_{M-1}^d)} x^d(u)du \tag{8.19}$$

Hence

$$\sum_{i=1}^{M} \sum_{E \in Adj(G_i^d)} \int_E x^d(u)du = \sum_{i=1}^{M-1} \int_{E_i^d} x^d(u)du = \int_{G^d} x^d(u)du \tag{8.20}$$

Therefore Eq 8.17 preserves the integral over G^d while transporting the data from the cells to the vertices. It is important to understand that this operation is a linear filter which diffuses information while conserving the global integral.

8.3 Basic comparison of the interpolation methods

The interpolation methods considered in this chapter have been compared on basic test cases. The tests have been carried out on simple two dimensional meshes[2] using the interpolation code developed in the coupling library.

The domain is a square with x and y ranging from -30 to 30 which is discretized by two uniform grids: a low resolution and a high resolution one. The grid vertices are joined using triangles, Fig. 8.20 in order to avoid favoring a particular discretization the triangles are arranged using a diamond scheme. A first test has been carried out to measure the accuracy of each interpolation method. A reference

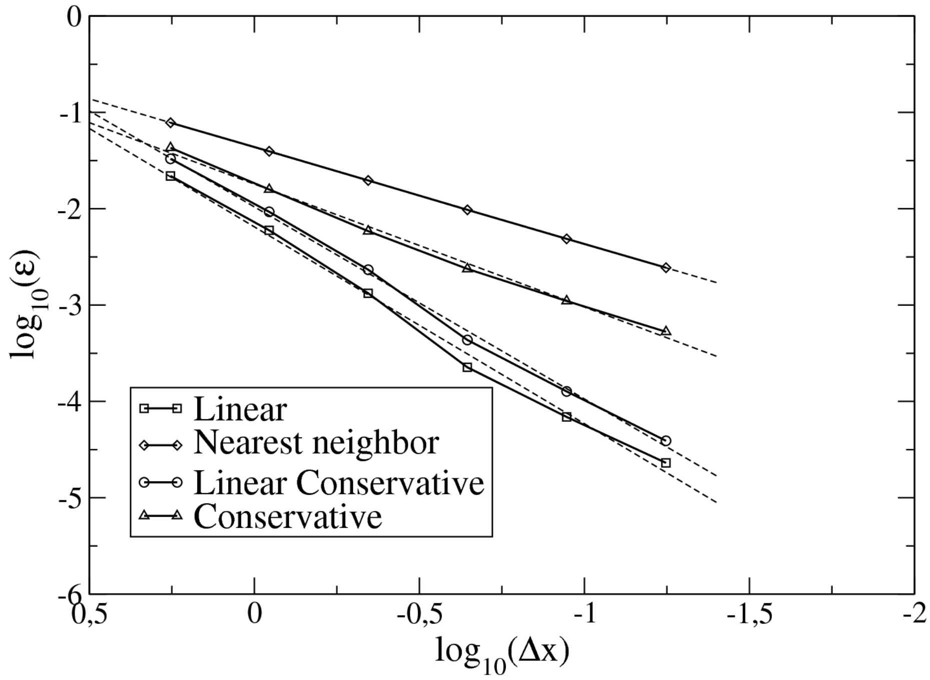


Figure 8.19: Error comparison of interpolation methods

sinusoidal signal $f(x, y) = |cos(\omega_0 x)cos(\omega_0 y)|$ is sampled on a grid G^s, this signal is then interpolated onto a grid G^d yielding a signal g. The interpolated values on grid G^d are then compared to the values from the original sinusoidal signal yielding an error value $\epsilon(G^d_i) = |g(G^d_i) - f(G^d_i)|$ for each point of G^d. Also the distance between the closest point of G^s is calculated for each point of G^d yielding $\Delta x(G^d_i)$. These values are then averaged for the entire mesh. This procedure is then applied for several sets of source and destination meshes. The destination mesh element size is chosen be half the source element size.

Figure 8.19 shows the evolution of $log_{10}(\epsilon)$ function of $log_{10}(\Delta x)$ for the different interpolation methods considered in this document. These results show that the error for the linear and linear conservative methods scale with Δx^2 whereas the nearest neighbor method (and conservative method) scale with Δx. Note also that the linear conservative method is less accurate than the linear method.

A second test illustrating both the aliasing conclusions of this chapter and the conservation properties of the different interpolation schemes has been carried out. The nearest neighbor, linear and linear conservative methods have been tested in two situations: in the first case the data is transferred

[2]The extension of the different interpolation methods to surfaces is described in chapter B.3.3.

from a coarse grid to a fine grid, Fig. 8.22, in the second data is transferred from a fine grid to a coarse grid, Fig. 8.22. The responses are analyzed in two ways:

- the first is qualitative: does the resulting data field look like the source data field?
- the second is quantitative: how much of the integral is lost through the interpolation process?

Interpolation Method	fine to coarse	coarse to fine
Nearest Neighbor	14.9%	9.37e-02%
Linear	12.6%	9.56e-04%
Conservative	5.7e-13%	5.0e-12%
Linear conservative	1.04e-12%	7.2e-13%

Table 8.1: Global integral conservation results for different interpolation methods

Description of the test:

- in the coarse to fine test, the coarse grid is a 21x19 vertex grid, the fine grid is a 200x200 vertex grid,
- in fine to coarse test, the fine grid is a 200x200 vertex grid, the coarse is a 51x49 vertex grid,

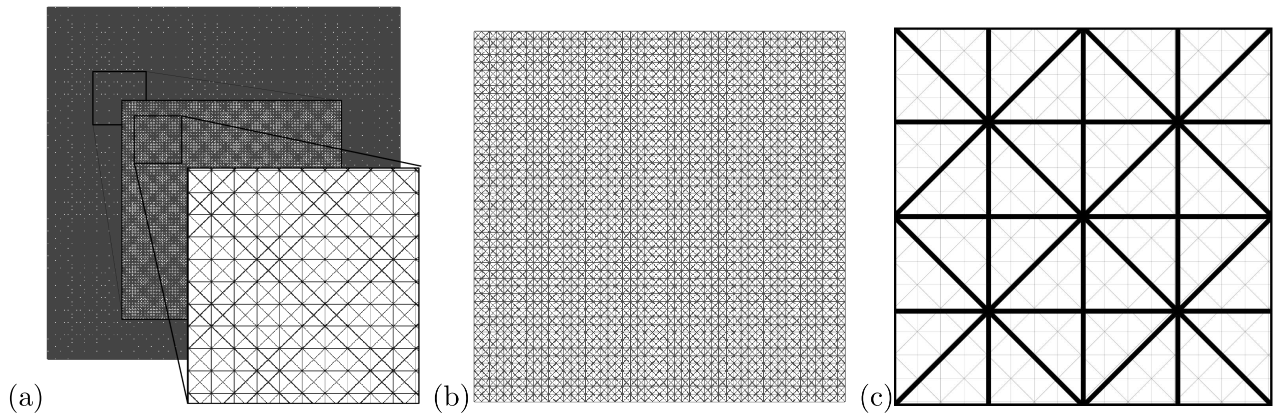


Figure 8.20: (a) Fine grid(200x200), (b) coarse Grid(51x49), (c) a section of the fine and coarse grids overlapped.

The source signal is generated using the following ad-hoc formulas:

- For the coarse to fine case the source signal is a low frequency signal:

$$f(x,y) = |cos(\omega_0 x)cos(\omega_0 y)| \tag{8.21}$$

  Where pulsation $\omega_0 = 0.1$ has been chosen to be clearly lower than the maximal frequency that the coarse grid may support.

- For the fine to coarse case the source signal is the sum of a low frequency signal and a high frequency signal:

$$f(x,y) = |cos(\omega_0 x) * cos(\omega_0 y)| + 0.4cos(\omega_1 x) * cos(\omega_1 y) \tag{8.22}$$

  Where $\omega_1 = \frac{5\pi}{3} \simeq 5.236$ has been chosen to be clearly higher than the highest frequency the coarse grid may support.

The global Integral results presented are relative differences between interpolated integral and source integral, Table 8.1.

Looking at the results for the coarse to fine interpolation, Fig. 8.21, it is obvious that the nearest neighbor method clearly provides bad results compared to the linear method. This is explained by the comparison of the low pass filtering properties of the box filter and the tent filter. Both linear and linear conservative methods take into account the element gradient hence they have similar results on the coarse-to-fine interpolation case. However quantitatively the linear conservative method provides a much better integral conservation then the linear method, Table 8.1.

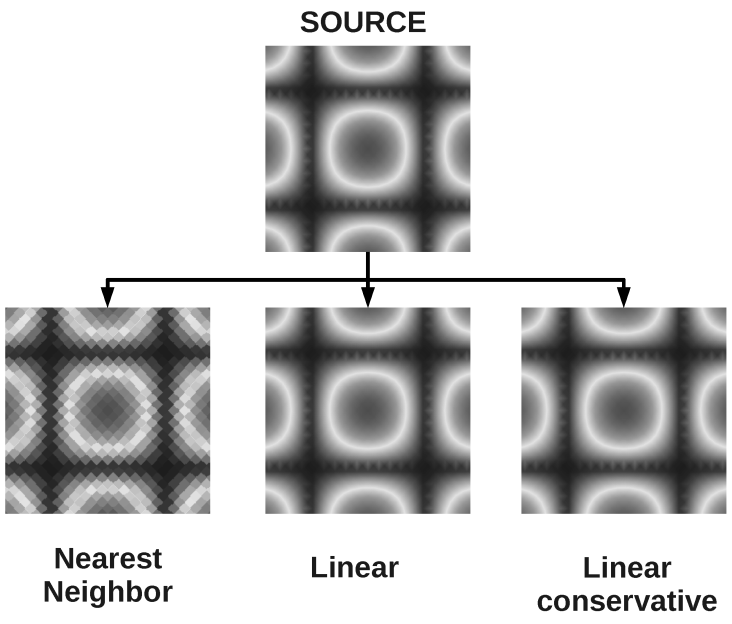


Figure 8.21: Interpolation from coarse to fine grid

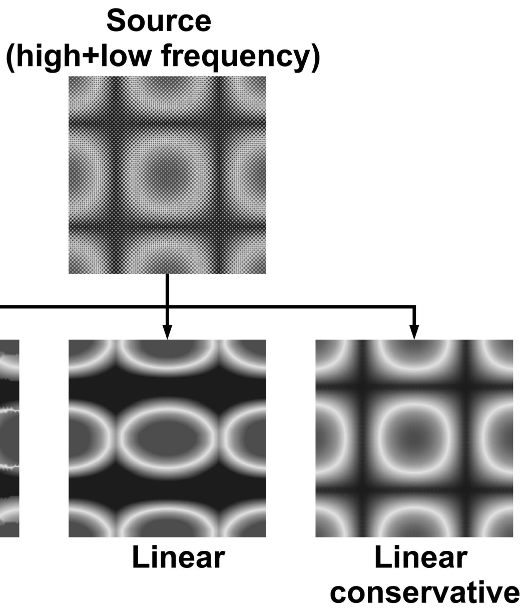


Figure 8.22: Interpolation from fine to coarse grid

For the fine to coarse interpolation results, Fig. 8.22, it is clear that the nearest neighbor and the linear methods demonstrate aliasing, however the linear conservative does not show this phenomenon. It is also clear that the aliasing is stronger with the nearest neighbor interpolation than with the linear interpolation, there again the low pass filtering properties of the box and tent filters explain these differences. Quantitatively the aliasing has a severe impact on the nearest neighbor and linear methods integral conservation, whereas the linear conservative method maintains very good integral conservation.

An other characteristic of the result is that the linear and nearest neighbor interpolated fields show clear anisotropy. To verify that this property comes from the mesh choice (51x49), different target meshes have been tried, Fig. 8.23. This test shows that the anisotropy is clearly a consequence of the target mesh, i.e. in the cases (c) and (d) on Fig. 8.23 the isotropy is conserved. Similar results can be viewed for the linear interpolation method, Fig. 8.24. However for the linear conservative method, the interpolated field does not show as much dependence on the target mesh, Fig. 8.25.

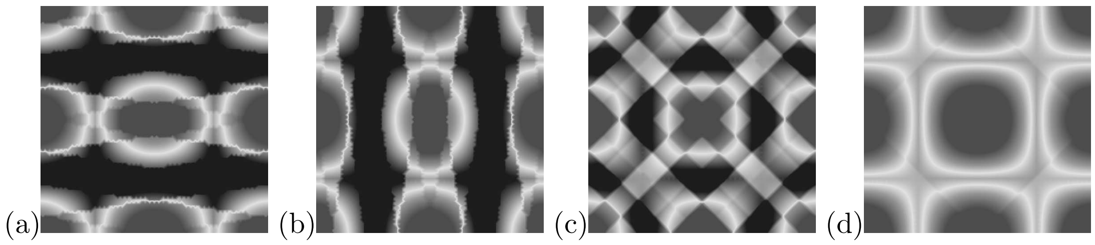


Figure 8.23: Nearest neighbor results for different meshes: (a) 51x49, (b) 49x51, (c) 49x49, (d) 51x51

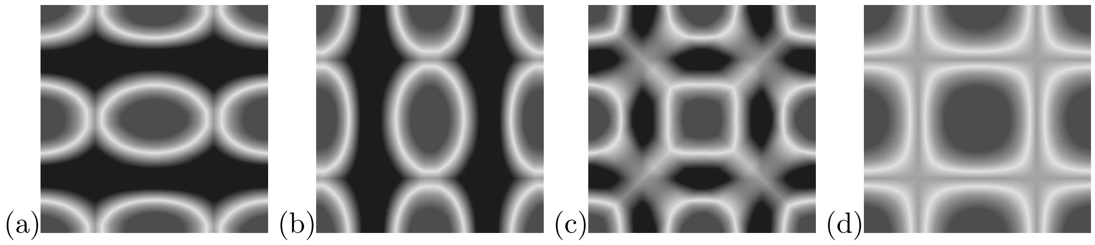


Figure 8.24: Linear interpolation results for different meshes: (a) 51x49, (b) 49x51, (c) 49x49, (d) 51x51

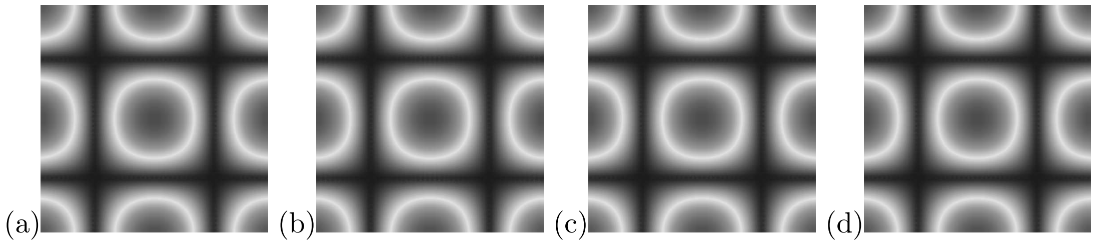


Figure 8.25: Linear conservative interpolation results for different meshes: (a) 51x49, (b) 49x51, (c) 49x49, (d) 51x51

To further investigate the results, one dimensional profiles are extracted for each interpolation method. These profiles are extracted for the target mesh 51x49, at locations described on figure 8.26.

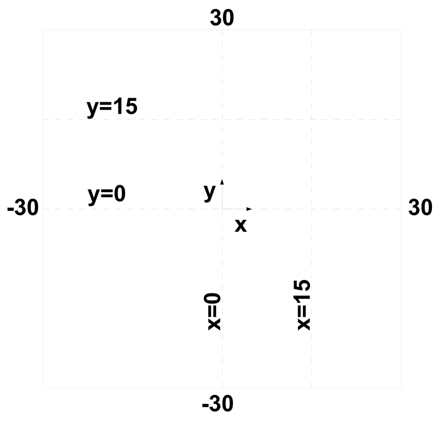


Figure 8.26: Position of the profiles

The profiles are shown on Fig. 8.27, 8.28, 8.29, 8.30. Also the low frequency signal, corresponding to the reference signal, is plotted on each graph. The results show that the interpolated signal calculated using the conservative interpolation methods clearly match the low frequency signal. On the other hand the nearest neighbor and linear interpolation methods fail to reproduce the low frequency signal. This can also be seen looking at the signals spectrum: Fig. 8.31 shows the source signal spectra, the low frequency signal spectrum and the spectrum obtained through nearest neighbor and linear conservative interpolations. To better visualize the results, the difference between the nearest neighbor and linear conservative spectra with the reference signal are plotted in Fig. 8.32. The linear conservative interpolation method is capable of correctly filtering out the high frequency peak of the source signal, yielding the almost the same spectrum as the reference signal. On the contrary the nearest neighbor interpolation method shows aliasing responsible for the high differences on Fig. 8.32.

The results shown in this discussion emphasize the differences between the interpolation methods on basic signals. These results have been summarized in Table 8.2. It is clear that looking at these results the best interpolation method is the linear conservative method: it maintains second order error growth, extremely low integral conservation errors and is almost not prone to aliasing problems. However the signals and meshes used in these tests have been chosen voluntarily as worst case scenarios in order to point out the weaknesses of the more standard methods: nearest neighbor and linear interpolation. For signals containing essentially low frequencies (relative to the maximal frequency supported by the meshes), linear interpolation remains a good approximation, simple to implement. On the other hand the nearest neighbor interpolation should be avoided because of its higher vulnerability to aliasing artifacts[3].

Interpolation Method	Error growth order	Integral conservation	Aliasing vulnerability
Nearest Neighbor	First order	mesh dependent	very high
Linear	Second order	mesh dependent	moderate
Conservative	First order	almost exact	very low
Linear conservative	Second order	almost exact	very low

Table 8.2: Summary of interpolation methods comparison results

[3]In this comparison the cost of the interpolation methods is not discussed. The interpolation cost greatly depends on implementation choices which are discussed in chapter B.3.3 and chapter 10.

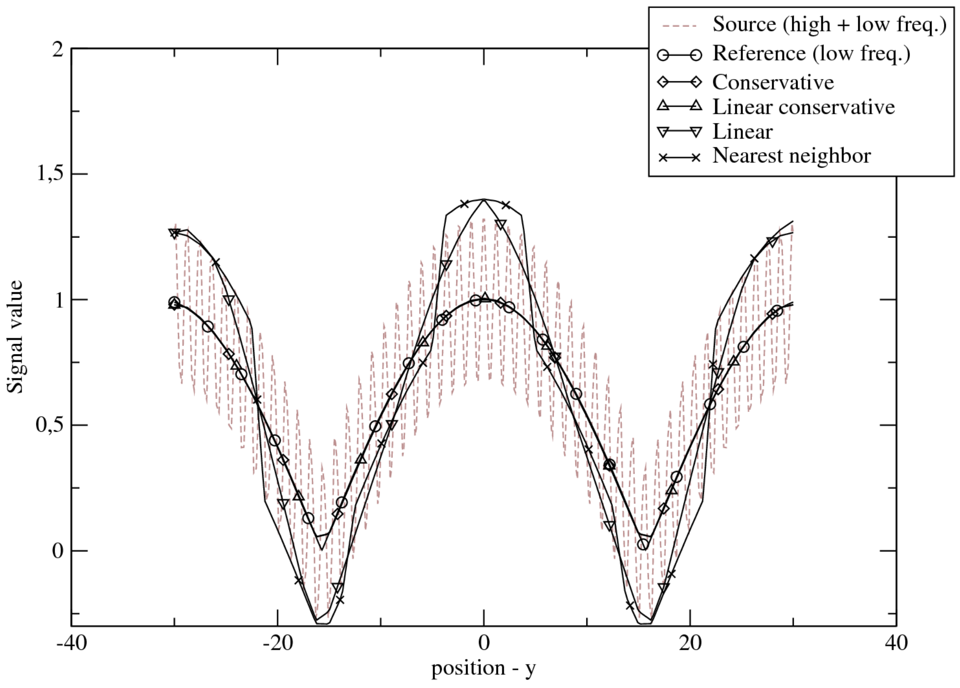


Figure 8.27: Interpolation profile on line $x = 0$

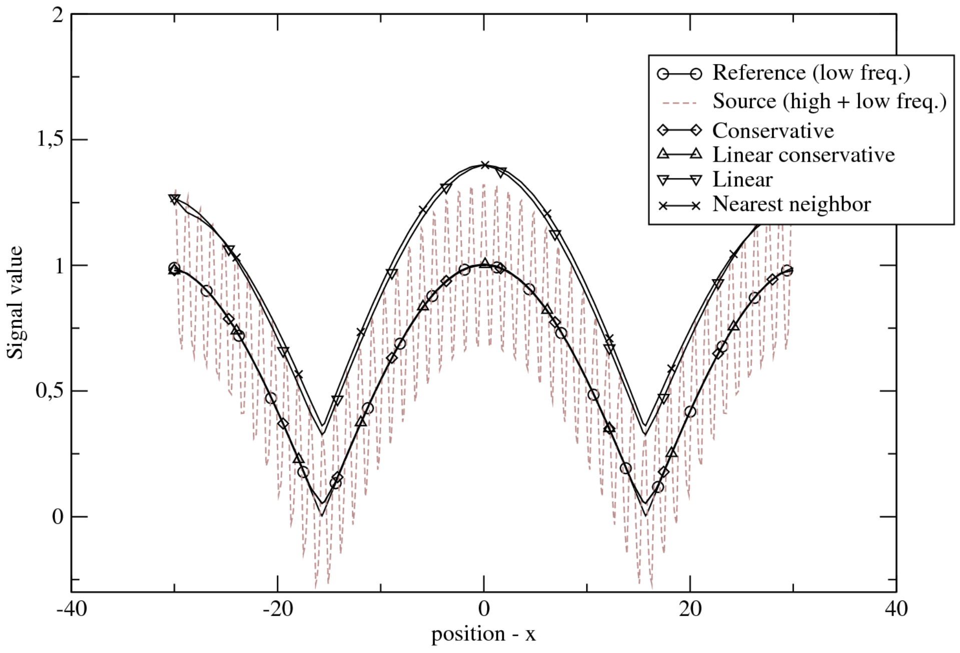


Figure 8.28: Interpolation profile on line $y = 0$

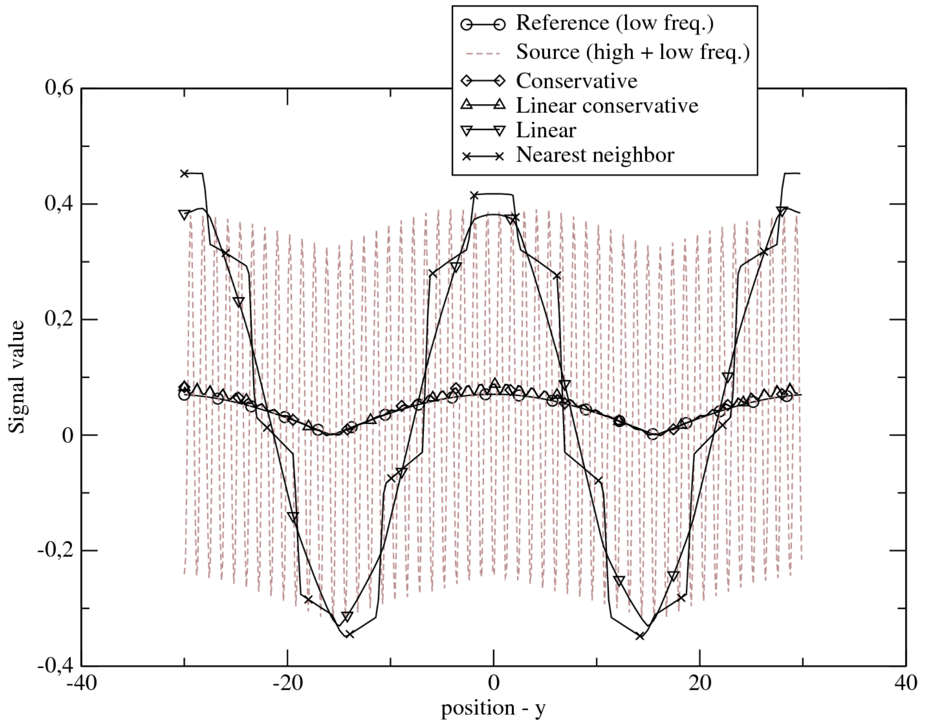


Figure 8.29: Interpolation profile on line $x = 15$

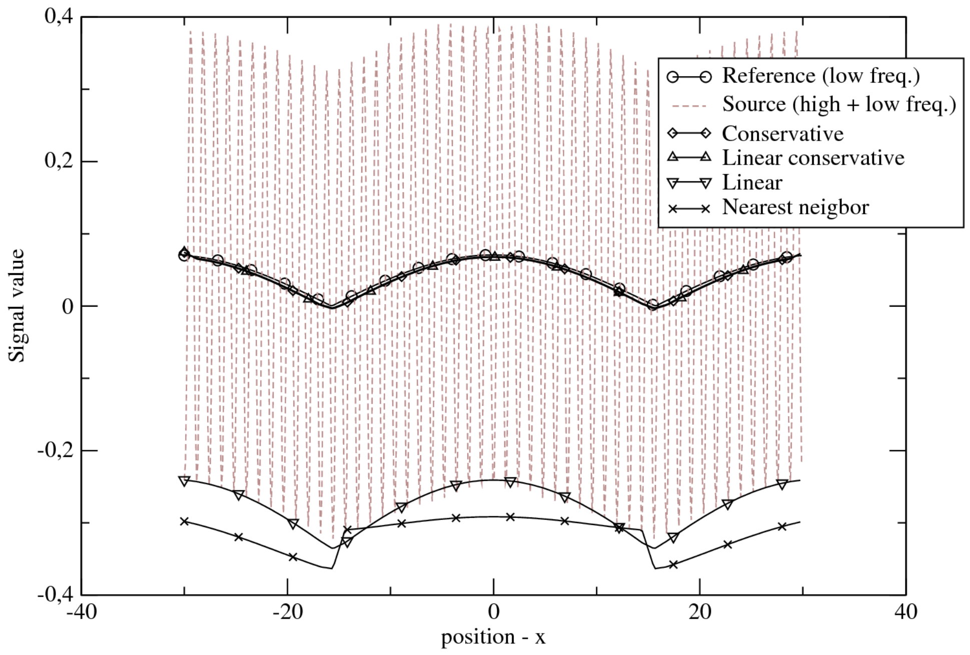


Figure 8.30: Interpolation profile on line $y = 15$

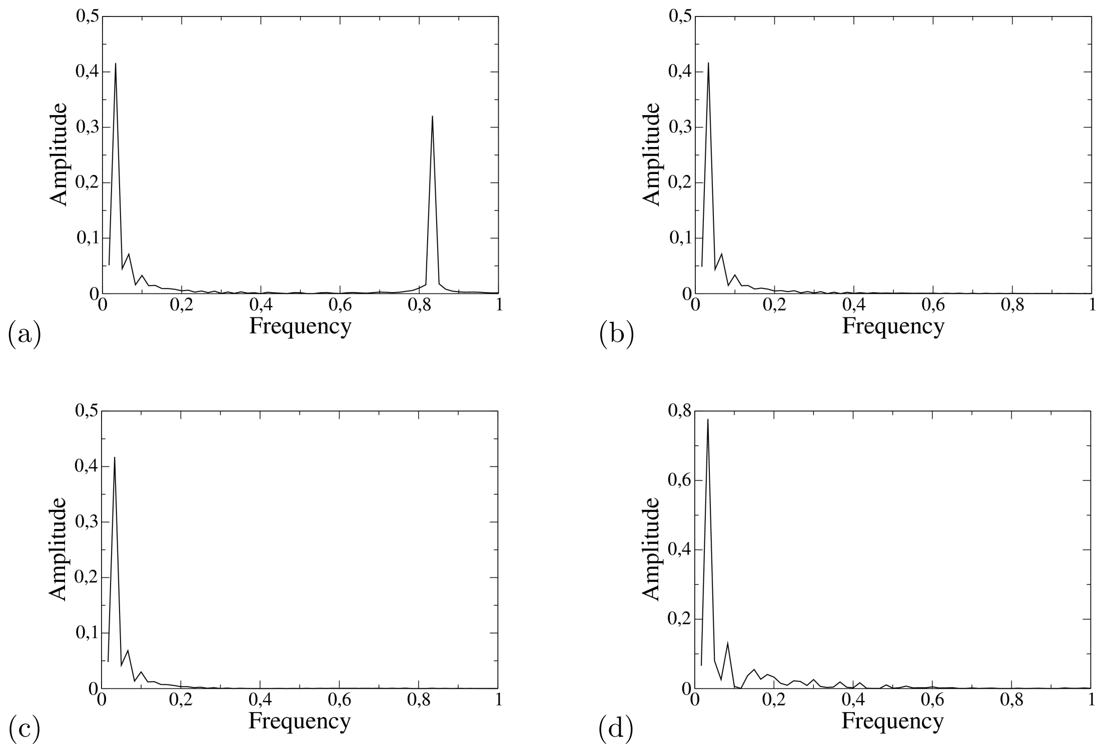


Figure 8.31: $x = 0$ profile spectrum for different signals: (a) Source (low+high freq.), (b) Low frequency reference, (c) Linear conservative, (d) Nearest neighbor

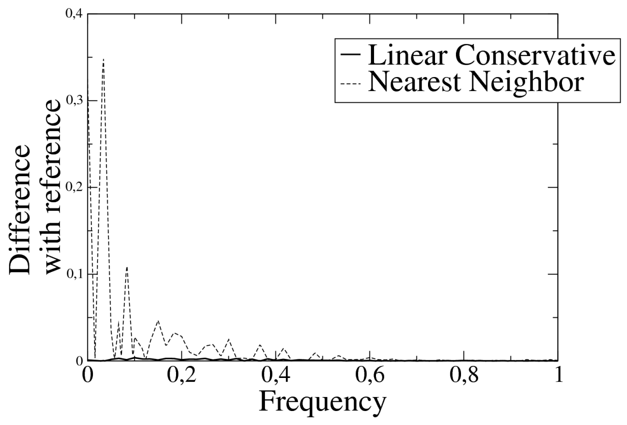


Figure 8.32: Difference between spectra of nearest neighbor interpolated and linear conservative signals with reference signal on profile at $x = 0$

Chapter 9

Interpolations based on linear transforms

To use the interpolation methods introduced in chapter 8 on realistic unstructured geometries more complex algorithms and methods have to be introduced. In order to explain these algorithms and methods in a comprehensive way a set of notation are introduced:

- The destination and source grids are respectively noted G^d and G^s which have respectively N^d and N^s vertices.
- The notation G_i^d refers to the i^{th} vertex of the destination grid, and likewise for G_i^s.
- U is the field to transport, U^d and U^s are respectively the fields defined on the destination and source grids.
- The notation U_i^d refers to the value of the field U^d at the vertex i of G^d: $U_i^d = U^d(G_i^d)$. Likewise $U_i^s = U^s(G_i^s)$.
- The i^{th} element of the destination grid is noted E_i^d, likewise for the source grid.
- The vertex count of E_i^d is noted $N_v(E_i^d)$, likewise for E_i^s.
- The j^{th} vertex index of E_i^d is noted $E_{i,j}^d$, likewise for E_i^s.
- The element count of G^d and G^s is noted respectively M^d and M^s.

The interpolation methods described in 8 rely on linear transformations, therefore each destination value can be written as a linear combination of the source values:

$$U_i^d = \sum_{j=1}^{N^s} \alpha_{i,j} U_j^s \tag{9.1}$$

Hence computing all the destination values can be written in matrix form:

$$U^d = TU^s \tag{9.2}$$

Where T is the transformation (interpolation) matrix

$$T = \begin{array}{c} \\ U_1^d \\ \vdots \\ U_i^d \\ \vdots \\ U_{N^d}^d \end{array} \begin{array}{c} \begin{array}{cccccc} U_1^s & \dots & U_{j-1}^s & U_j^s & U_{j+1}^s & \dots & U_{N^s}^s \end{array} \\ \begin{pmatrix} & & & & & & \\ & & & & & & \\ \alpha_{i,1} & \dots & \alpha_{i,j-1} & \alpha_{i,j} & \alpha_{i,j+1} & \dots & \alpha_{i,N^s} \\ & & & & & & \\ & & & & & & \end{pmatrix} \end{array} \tag{9.3}$$

The matrix T has N^d rows and N^s columns, therefore storing T in dense form is extremely expensive. Fortunately the interpolation methods that are described in this chapter are local, i.e. each interpolated value depends on a rather small amount of points. Hence each row of T contains very few non zero values, implying that the storage cost of T in sparse form is reasonable.

The objective of this chapter is to explain the computation of the interpolation matrix T. The algorithms presented hereafter are naive brute force algorithms and hence should only be considered for comprehension, not for implementation. The algorithms implemented in the geometric part of the coupling library are presented in 10.

9.1 Nearest neighbor interpolation

This method is the most basic interpolation method, easy to implement: for each vertex of the destination grid G_i^d find the closest vertex in the source grid G^s.

The matrix T is a Boolean matrix. For every G_i^d compute the closest G_j^s, and set $T_{ij} = 1$. The rest of the matrix should be filled with zeros.

```
initialize the matrix T the null matrix (0 everywhere);
$T \leftarrow 0$;
for $i \leftarrow 1$ to $N^d$ do
    $j \leftarrow FindClosest(G_i^d)$;
    $T_{i,j} \leftarrow 1$
end
```

Algorithm 1: Nearest neighbor interpolation matrix calculation

Where the $FindClosest(V)$ is a function which returns the index of the closest vertex to V in the grid G^s. This can be calculated using this brute-force algorithm:

```
initialize the search;
$closest\_index \leftarrow 1$;
$closest\_dist \leftarrow \|V - G_1^s\|$;
for $i \leftarrow 2$ to $N^s$ do
    if $\|V - G_i^s\| < closest\_dist$ then
        $closest\_index \leftarrow i$;
        $closest\_dist \leftarrow \|V - G_i^s\|$;
    end
end
return $closest\_index$
```

Algorithm 2: Brute force algorithm for nearest neighbor search

Since the complexity of this brute force algorithm is $O(N^s)$, the overall complexity of the nearest neighbor method is $O(N^d N^s)$ which is far from optimal. Therefore this algorithm should only be considered in cases where the grid sizes are very small.

More efficient algorithms are presented in 10.1.

9.2 Linear interpolation

Linear interpolation can be derived quite simply from a Taylor development of a function f:

$$f(x_0 + h) = f(x_0) + h\frac{df(x)}{dx}|_{x0} + \frac{h^2}{2}\frac{d^2f(x)}{dx^2}|_{x0} + ... + \frac{h^n}{n!}\frac{d^nf(x)}{dx^n}|_{x0} + O(h^n) \tag{9.4}$$

if x is a vector we can write:

$$f(x_0 + h) = f(x_0) + \nabla f(x_0) \cdot h + \frac{1}{2}h^T\mathbb{H}(x_0)h + o(||h||^2) \tag{9.5}$$

The Linear interpolation method is derived from this formula, using only the first order term ∇f.

For linear elements (segments in 1D, triangles in 2D, tetrahedrons in 3D) linear interpolation can be performed by calculating barycentric coordinates, the calculation of such coordinates is presented in B.1 and can be implemented by using explicit formula's.

The only computational difficulty is to find the element containing the vertex V. What is more when interpolating on surfaces in 3D the vertex V must first be projected onto the surface before the vertex-in-element test is performed.

In some cases no element may contain the vertex V even after projection, in such case a simple fix is to fall back to nearest neighbor interpolation, therefore assign the value of the closest vertex.

```
initialize the matrix $T$ the null matrix (0 everywhere);
$T \leftarrow 0$;
for $i \leftarrow 1$ to $N^d$ do
    $j \leftarrow FindContainingElement(G^d_i)$;
    if $j < 0$ then
        Fallback to nearest neighbor;
        $j \leftarrow FindClosest(G^d_i)$;
        $T_{i,j} \leftarrow 1$ ;
    end
    else $E^s_j$ contains $G^d_i$
        Calculate barycentric coordinates of $G^d_i$ in $E^s_j$;
        $\lambda_k \leftarrow ComputeBarycentricCoords(G^d_i, E^s_j)$ ;
        for $k \leftarrow 1$ to $N_v(E^s_j)$ do
            $col \leftarrow E^s_{j,k}$ ;
            $T_{i,col} \leftarrow \lambda_k$ ;
        end
    end
end
```

Algorithm 3: Linear interpolation matrix calculation

The $FindContainingElement(V)$ returns an integer j which is the index of the element of G^s which contains V ($V \in E^s_j$), if no element of G^s contains V then $FindContainingElement$ returns a negative value.

The function $ComputeBarycentricCoords(V, E)$ calculates the barycentric coordinates of the vertex V in the element E. It returns a vector of scalars Λ_k, one for each vertex of the element E. The implementation of this function is straightforward for P1 elements and only requires selecting the correct formulas for the correct element type. Hence this function is extremely cheap in terms of execution time since no sub iterations are required.

An interesting property of barycentric coordinates is that the barycentric coordinates calculation can be used for vertex in element testing: the vertex V is contained in E as long as its barycentric coordinates remain between 0 and 1, hence $TestVertexInElement(V, E)$ can be defined by:

```
Calculate barycentric coordinates of $V$ in $E$;
$\lambda \leftarrow ComputeBarycentricCoords(V, E)$ ;
if $\forall k \in [\![1, N_v(E)]\!], \lambda_k \in [0, 1]$ then
    return $True$
end
return $False$
```

Algorithm 4: Vertex in Element test using barycentric coordinates

Therefore a simple $FindContainingElement$ algorithm may be written

```
for $i \leftarrow 1$ to $M^d$ do
    if $TestVertexInElement(G^d_i, E^s_j)$ then
        return $i$
    end
end
return $-1$
```

Algorithm 5: Find containing element brute force algorithm

Of course this algorithm is clearly not optimal since for each vertex of the destination grid, all the elements of the source grid have to be processed resulting in a complexity in $O(N^d M^s)$.

9.3 Conservative interpolation

In some problems integrals have to be conserved during the interpolation process, Fig 9.1, for example when transferring a flux, the flux should be the same on the source and on the destination mesh. However by using interpolation methods based on sampling, there is no way to guarantee that the integral is conserved. A method to do conservative interpolation is presented here (similar to the Cell-Intersection-Based Donor-Cell method[69]). The method is based on intersection calculation between both meshes. Once the intersection polygons are calculated, the integral on each polygon is evaluated using a shape function. The resulting integral is the sum of the integrals on all the intersection polygons.

9.3.1 Conservative Interpolation on surface meshes

In this problem we have two grids G^s and G^d discretizing the same surface S but in a different way, Fig 9.2, i.e. the cell sizes may differ. The field U must be transferred from G^s to G^d without integral loss over the surface S. The methodology presented here is based on element to element intersection. To simplify we will consider triangular surface meshes, but the methodology can be extended to other element types.

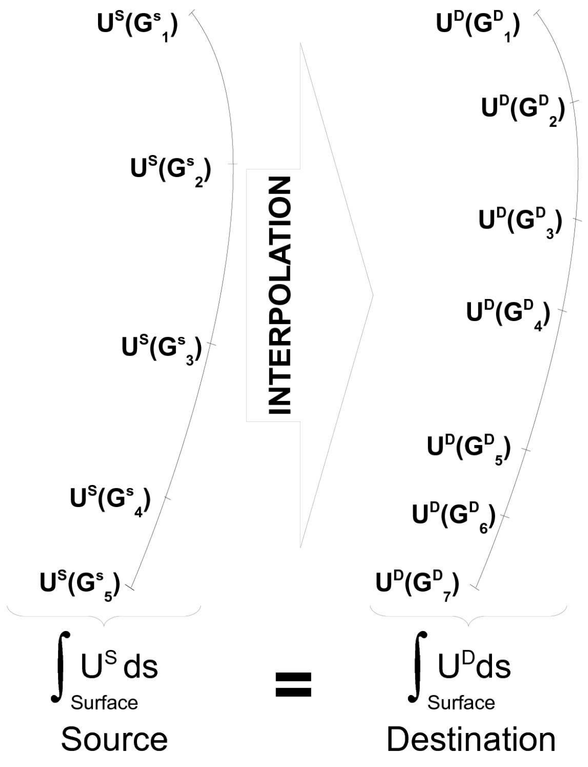


Figure 9.1: Conservative interpolation preserves global integrals

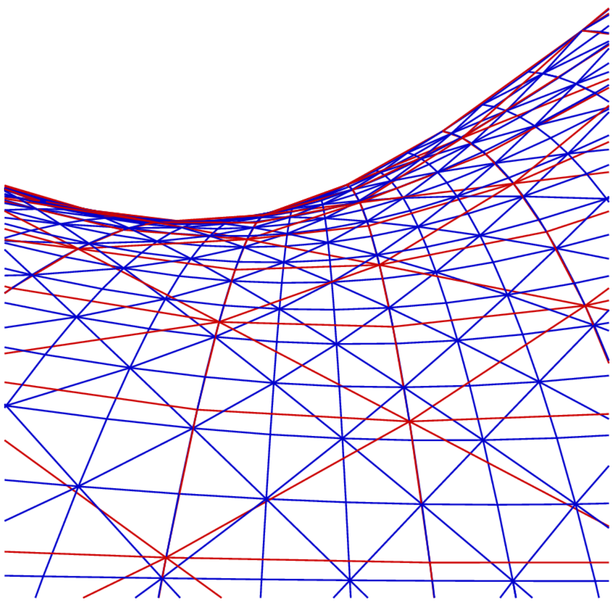


Figure 9.2: Surface to surface interpolation problem

9.3.1.1 Step 1: Projection step

For each triangle E_i^d of G^d find the set of elements $I = \left\{E_j^s\right\}$ on G^s such as $E_i^d \cap E_j^s \neq \emptyset$.

In the 3D case S may be curved, Fig 9.2, therefore in order to obtain true intersection polygons, the elements E_i^d and E_j^s are projected onto the plane obtained by averaging the support planes of E_j^s and E_i^d, Fig 9.3. The intersection calculation is then performed in 2D allowing to calculate intersection polygons (if the calculation was done in 3D most intersection calculations would result in obtaining either no intersection or intersection lines). The projection is a source of integral conservation loss if the support planes of E_j^s and E_i^d are very different: the area of the projected elements may be very different than the area of the original elements. This is one of the limits of the method: important deviations between the discretizations of S may induce conservation loss.

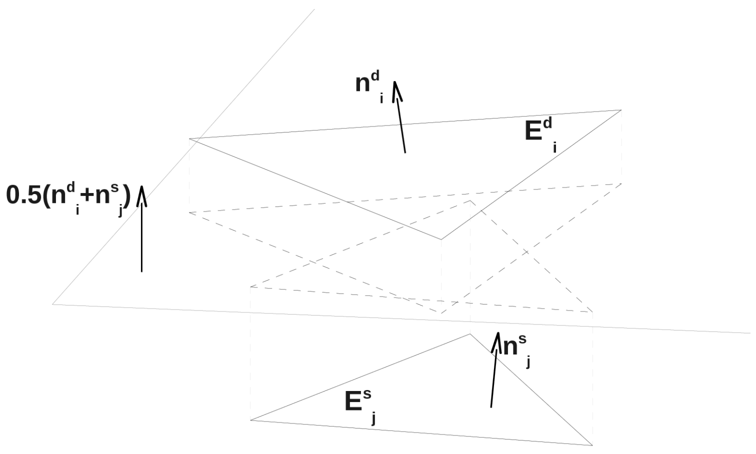


Figure 9.3: Before calculating the intersection polygons the source and destination elements are projected on a average plane

Reducing the set of elements of G^s to calculate I is the key to accelerate this method, for now the reader can consider applying the following procedure to all the elements of G^s therefore obtaining a naive but functional method. A more efficient way is presented in chapter 10.

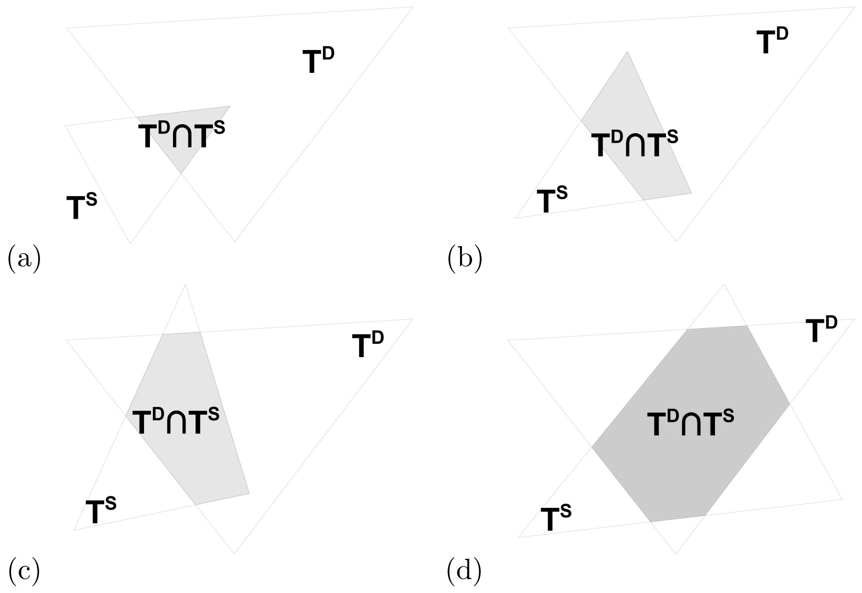


Figure 9.4: Examples of intersections between destination and source triangles T^d and T^s: (a) 3 point intersection, (b) 4 point intersection, (c) 5 point intersection, (d) 6 point intersection

9.3.1.2 Step 2: Intersection step

After the projection on the mean plane the intersection polygon of the two triangles T_j^s has to be calculated T_i^d, Fig 9.4, a general procedure is described in procedure 6.

```
$intersect_points = \emptyset$
Test points of $T^A$ in $T^B$;
for $i \leftarrow 1$ to 3 do
    Calculate barycentric coordinates of $T_i^A$ in $T^B$;
    $\lambda \leftarrow ComputeBarycentricCoords(T_i^A, T^B)$ ;
    if $\forall k \in [\![1,3]\!], \lambda_k \in [0,1]$ then
        $intersect_points \leftarrow intersect_points \cup T_i^A$;
    end
end
Test points of $T^B$ in $T^A$;
for $i \leftarrow 1$ to 3 do
    Calculate barycentric coordinates of $T_i^B$ in $T^A$;
    $\lambda \leftarrow ComputeBarycentricCoords(T_i^B, T^A)$ ;
    if $\forall k \in [\![1,3]\!], \lambda_k \in [0,1]$ then
        $intersect_points \leftarrow intersect_points \cup T_i^B$;
    end
end
Add segment intersections $T^B$ in $T^A$;
for $i \leftarrow 1$ to 3 do
    for $j \leftarrow 1$ to 3 do
        $intersect_points \leftarrow intersect_points \cup SegmentIntersection(Edge(T^A, i), Edge(T^B, i))$ ;
    end
end
Filter point duplicates;
...
```

Algorithm 6: Intersection point set of triangles T^A and T^B in 2D

The test vertex in element has already been presented for the linear interpolation method, the next difficulty is properly computing segment intersection. Indeed degenerated intersection cases must be considered, i.e. the intersection of two segments may also be a segment. Such cases may seem unlikely but considering the mount of elements in each grid, such situations may happen. Also because the two meshes are discretizations of the same surface, their borders are thus defined using the same curves, leading to a high probability of coinciding segments on the borders. Hence the output of the procedure should be a variable size point set O:

- if $O = \emptyset$ then no intersection found between the two segments (this does not mean that the lines supporting the segments do not intersect)
- if O contains 1 point then the two segments intersect and have only 1 intersection point.
- if O contains 2 points then the two segments are parallel and have a common portion (which can be the entire segment), therefore the segments have an infinite number of intersection points defining an intersection segment. Only the extremal points of the intersection segment are returned.

Therefore a general procedure to calculate the intersection of two segments in 2D is presented in B.3.1. The intersection point set P_i is available. It is important to understand that this procedure only calculates a set of points which are on the intersection polygon's border, these points are not ordered and therefore the polygon formed by linking the points P_i by lines may be self intersecting, Fig 9.5.

Sorting an arbitrary set of points in order to obtain a non self intersecting polygon is a complex task. Yet if the target polygon is convex[1] then the problem can be solved using a convex hull algorithm [6, 93]. In this case the intersection of two triangles is considered, since triangles are convex polygons their intersection must also be convex [2]. Therefore using a convex hull algorithm it is possible to sort P_i to obtain the intersection polygon.

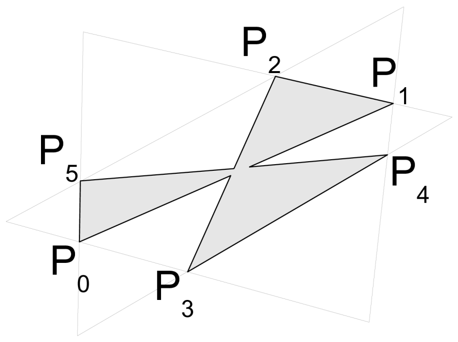


Figure 9.5: Unsorted intersection point set

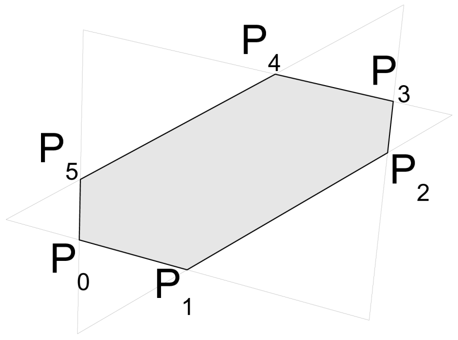


Figure 9.6: Sorted intersection point set

The algorithm used here is a Jarvis March algorithm [50, 27] (also called Gift Wrapping algorithm). The output of this algorithm is an ordered list of vertices which define the intersection polygon. Since the intersection polygon is convex it can then be easily cut into triangles. The new triangles can be formed by using a vertex V_0 as base by taking successively $V_0V_1V_2$, $V_0V_2V_3$,..., $V_0V_{N-1}V_N$ where N the number of vertices of the intersection polygon. This new set of triangles forms the intersection polygon's mesh.

The Jarvis March algorithm is presented in Appendix B.3.2[3].

9.3.1.3 Step 3: Intersection polygon triangulation and Integral calculation step

The intersection step has provided an intersection polygon noted here Ip. The intersection polygon vertex count is noted $N_v(Ip)$ each vertex is noted Ip_i. In order to calculate the integral over the

[1]if A convex polygon is a polygon P for which if $A \in P$ and $B \in P$ then $AB \subset P$

[2]The proof is simple: consider two triangles T_1 and T_2, their intersection $I = T_1 \cap T_2$, if $A \in I$ and $B \in I$ then $AB \subset T_1$ and $AB \subset T_2$ hence $AB \subset I$

[3]Sorting the intersection point set allows to triangulate the intersection polygon directly without needing to use more advanced algorithms such as Delaunay triangulation[36].

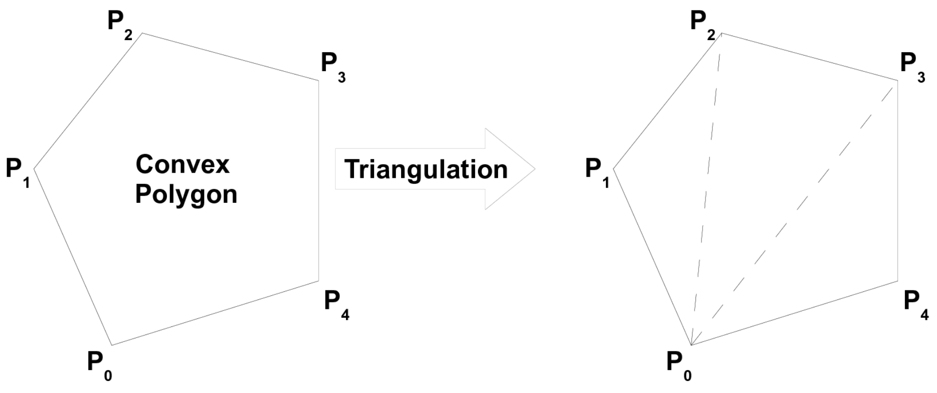


Figure 9.7: Triangulating a convex polygon into a triangle fan

intersection polygon, the intersection polygon is broken down into basic triangles. For a convex polygon this can be simply done by braking the polygon into a triangle fan, Fig 9.7.

A triangle fan is a term used in computer graphics. A triangle fan is a set of connected triangles which all share a base vertex. For example the triangle fan formed by the set of points P_0,P_1,P_2,P_3,P_4 is the set of triangles $P_0P_1P_2$, $P_0P_2P_3$, $P_0P_3P_4$.

A convex polygon P (its vertices are noted P_i with $i \in [\![0, N_v(P)-1]\!]$) can be decomposed using this scheme: Considering the edge E_i between $P_{E_i^0}$ and $P_{E_i^1}$, every point $M \in E_i$ is also in P, because P is convex the segment $S(M, P_0) \subset P$[4]. Therefore sliding M on E_i shows that the triangle $P_0P_{E_i^0}P_{E_i^1} \subset P$. Hence the triangle set $\left(\cup_{i=0}^{N_v(P)-1} P_0P_{E_i^0}P_{E_i^1}\right) \subset P$. The equality $\left(\cup_{i=0}^{N_v(P)-1} P_0P_{E_i^0}P_{E_i^1}\right) = P$ can be proven by supposing that $\left(\cup_{i=0}^{N_v(P)-1} P_0P_{E_i^0}P_{E_i^1}\right) \neq P$ therefore $\exists M$ such as $M \in P$ but $M \notin \cup_{i=0}^{N_v(P)-1} P_0P_{E_i^0}P_{E_i^1}$ which contradicts that P_i is the border of P.

Thus $\cup_{i=0}^{N_v(P)-1} P_0P_{E_i^0}P_{E_i^1} = P$.

The edges E_0 and $E_{N_v(P)-1}$ can be ignored because they produce degenerate triangles (flat triangles). Therefore P is decomposed into $\cup_{i=1}^{N_v(P)-2} P_0P_{E_i^0}P_{E_i^1}$.

Now that the intersection polygon has been triangulated calculating the integral over Ip can be decomposed into the integral calculation over each simplex (triangle) composing Ip.

The integral over a triangle of a linear function is presented in B.2.1 (it is similar to a trapezoidal integration in 1D). This integral only depends on the values of the function on the triangles summits $P_0, P_{E_i^0}, P_{E_i^1}$. Since we know that all the points P_i are vertices of Ip the intersection of T^d and T^s we know that $P_i \in T^s$, therefore the value of the field at $U^s(P_i)$ and be calculated by linear interpolation on the triangle T^s.

Noting $\lambda_{i,0}$, $\lambda_{i,1}$ and $\lambda_{i,2}$ the barycentric coordinates of P_i in T^s.

[4]For a convex polygon if two points $A, B \in C$ then the segment $S(A, B) \subset P$

The integral over $P_0 P_{E_i^0} P_{E_i^1}$ can be calculated:

$$\begin{aligned}
\int_{P_0 P_{E_i^0} P_{E_i^1}} U^s ds = \; & \tfrac{1}{6}|J_\Phi| \left(U^s(P_0) + U^s(P_{E_i^0}) + U^s(P_{E_i^1}) \right) \\
= \; & \tfrac{1}{6}|J_\Phi|(\lambda_{0,0} + \lambda_{E_i^0,0} + \lambda_{E_i^1,0}) U^s(T_0^s) + \\
& \tfrac{1}{6}|J_\Phi|(\lambda_{0,1} + \lambda_{E_i^0,1} + \lambda_{E_i^1,1}) U^s(T_1^s) + \\
& \tfrac{1}{6}|J_\Phi|(\lambda_{0,2} + \lambda_{E_i^0,2} + \lambda_{E_i^1,2}) U^s(T_2^s)
\end{aligned} \tag{9.6}$$

Hence the integral over $Ip = T^d \cap T^s$ can be expressed by:

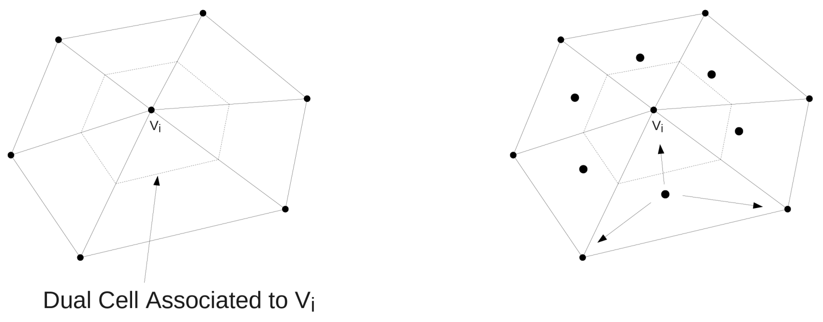
$$\int_{T^d \cap T^s} U^s ds = \sum_i \int_{P_0 P_{E_i^0} P_{E_i^1}} U^s ds \tag{9.7}$$

$$= \tfrac{1}{6}|J_\Phi| \sum_i \begin{bmatrix} \lambda_{0,0} + \lambda_{E_i^0,0} + \lambda_{E_i^1,0} \\ \lambda_{0,1} + \lambda_{E_i^0,1} + \lambda_{E_i^1,1} \\ \lambda_{0,2} + \lambda_{E_i^0,2} + \lambda_{E_i^1,2} \end{bmatrix} \cdot \begin{bmatrix} U^s(T_0^s) \\ U^s(T_1^s) \\ U^s(T_2^s) \end{bmatrix} \tag{9.8}$$

Finally the integral over T^d is expressed by summing the integrals over each intersection polygon of G^s and T^d:

$$\int_{T^d} U^s ds = \frac{1}{6} \sum_{\substack{T^s \in G^s \\ T^s \cap T^d \neq \emptyset}} |J_\Phi(T^s)| \sum_i \begin{bmatrix} \lambda_{0,0} + \lambda_{E_i^0,0} + \lambda_{E_i^1,0} \\ \lambda_{0,1} + \lambda_{E_i^0,1} + \lambda_{E_i^1,1} \\ \lambda_{0,2} + \lambda_{E_i^0,2} + \lambda_{E_i^1,2} \end{bmatrix} \cdot \begin{bmatrix} U^s(T_0^s) \\ U^s(T_1^s) \\ U^s(T_2^s) \end{bmatrix} \tag{9.9}$$

9.3.1.4 Step 4: cell to vertex data transfer



(a) (b)

Figure 9.8: (a) The dual-cell associated to vertex V_i, (b) Cell to vertex value transfer

The following notations are introduced:

- $N_C(G_i^d)$ is the number of cells associated to the vertex G_i^d in the grid G^d.

- $C(G_i^d, j)$ is the j^{th} cell associated to the vertex G_i^d in the grid G^d, cells are indexed from 1 to $N_C(G_i^d)$.
- $w(C)$ is the weight of the cell C. If the cell is a surfacic element then $w(C)$ is its area, likewise if C is a volumetric element then $w(C)$ is its volume.
- $w(G_i^d)$ is the weight of the dual-cell associated to G_i^d.
- $N_V(C)$ is the number of vertices defining the cell C, e.g. for a triangle 3, for a tetrahedron 4, etc...

The integral value calculated in the previous step must first be scaled by the inverse of the cell's area to remain homogeneous with the source signal. The cell value is therefore defined, Eq 9.10.

$$U_C(T^d) = \frac{1}{w(T^d)} \int_{T^d} U^s ds \tag{9.10}$$

Because the solvers work with fields defined at the cell vertices, this value must be distributed to the cell's nodes while preserving the global integral over the grid G^d. To do this the dual-cell concept is used. Each vertex G_i^d of G^d is assigned a dual-cell, Fig 9.8(a). The dual-cell of G_i^d is not defined explicitly. However it is built by aggregating a portion of each cell $C(G_i^d, j)$ associated to G_i^d. The cells $C(G_i^d, j)$ are distributed evenly to the different vertices which are associated to them, hence the portion of $C(G_i^d, j)$ distributed is divided by $N_V(C(G_i^d, j))$.

Thus the dual-cell of G_i^d's weight is defined by summing the weights of the portions of the associated cells:

$$w(G_i^d) = \sum_{j}^{N_C(G_i^d)} \frac{w(C(G_i^d, j))}{N_V(C)} \tag{9.11}$$

The vertex value $U_V(G_i^d)$ is defined by a weighted average of the associated cell values $U_C(C(G_i^d, j))$: each cell value is weighted by the ratio of the cell $C(G_i^d, j)$ contribution to the dual-cell of G_i^d over the weight of that dual-cell, Eq (9.12).

$$U_V(G_i^d) = \frac{1}{w(G_i^d)} \sum_{j=1}^{N_C(G_i^d)} U_C(C(G_i^d, j)) \frac{w(C(G_i^d, j))}{N_V(C(G_i^d, j))} \tag{9.12}$$

Noting that $U_i^d = U_V(G_i^d)$, the final interpolation matrix T_{ij} is built by expanding Eq (9.12) using Eq (9.10) and the quadrature issued from the projection step, Eq (9.9).

Chapter 10

Efficient geometrical search methods for unstructured grids

In chapter 8 the different interpolation methods used within this thesis have been explained, and in chapter B.3.3 basic algorithms allowing their implementation have been detailed. This chapter focuses on improving these methods by replacing the brute force search algorithms by more efficient ones. In most coupling problems relying on static meshes like in this thesis this problem may seem secondary since the geometrical search is done only once. Nevertheless algorithmic efficiency in these geometrical search problems remains a fundamental problem to consider when one wants to deal with large meshes. For example considering the nearest neighbor brute force search algorithm, it requires computing the distances between all the vertices of a mesh with all the vertices of an other, hence its execution time scales with the product of the two meshes vertices count. Considering for simplicity that the two meshes contain the same point count N the algorithm execution time would scale with N^2. On current hardware with well written code this algorithm can still perform with reasonable restitution times ($\simeq 1h$) for problems of sizes up to $N \simeq 10^5$. However because the execution time scales with the square of the problem size this is extremely limiting: if the execution time for a problem of size N is 1 hour, the execution time for a problem 10 times larger would be a little more than 4 days. And looking at the evolution of CFD other the past years, it is clear that mesh sizes keep increasing. It is therefore clear that to develop multi-physical applications capable of performing on the long term algorithmic efficiency should not be taken lightly.

The algorithms presented in this chapter have been implemented in the coupling library which is used for the target application, Appendix A.2, also the algorithm used for the nearest neighbor problem has been implemented in a tool named projector actually used at Snecma, Appendix A.1.

10.1 The Nearest neighbor problem

To find v_j^s such as $dist(v_j^s, v_i^d)$ is minimal, many different algorithms exist. This can of course be done by a direct computation of all the distances between vertices of G^s and v_i^d. Such an algorithm is very expensive, it requires N^s distance evaluations and comparisons. What is more if it has to be done for all vertices of G^d then the algorithm scales in $N^d N^s$, generally noted as $O(N^2)$. Fortunately there are more optimal solutions to this problem.

A very basic method is simply to use a coarse uniform grid to accelerate searches. Each cell C of the uniform grid maintains a list of the vertices which project onto C. The grid accelerates searches by mapping vertices to locations in space. The grid's resolution is the main parameter which drives the efficiency of this method: refining the mesh accelerates the searches but increases the memory required to store the grid structure. This method performs very well for vertices evenly distributed in space. However this method may loose efficiency for meshes which are not homogeneous such as meshes used in combustion unless the grid resolution is chosen to match the size of the smallest elements of the mesh. Hence the drawback of this method is the storage waste induced by over discretizing parts of space which would only need a low resolution grid. This adaptivity problem can be dealt by storing the grid in a sparse form (storing only the non empty zones of the grid), a more advanced method using this idea is presented in 13.1.1. In this chapter a different approach is considered: a tree based algorithm is used.

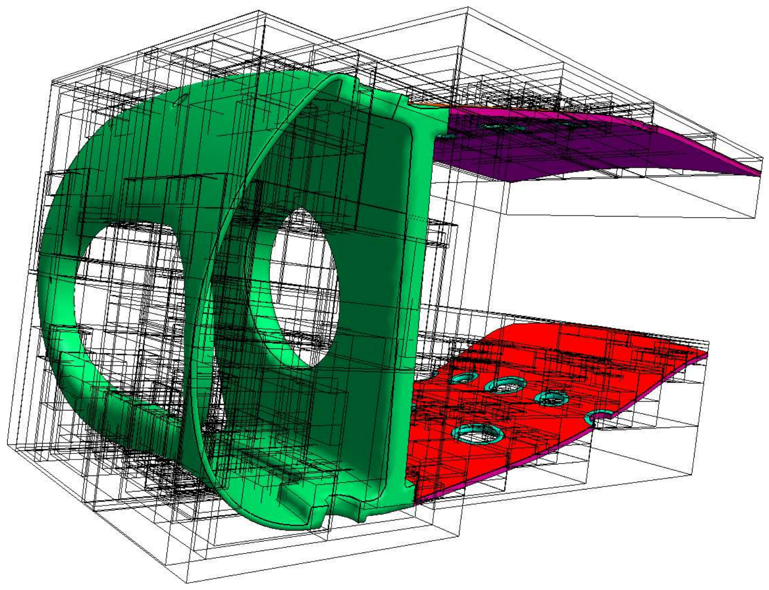


Figure 10.1: Kd-Tree graphical representation used for fast geometrical searches

10.1.1 Kd-Tree search algorithm

The Kd-Tree [107, 124] algorithm will be briefly introduced in this chapter. Kd-Trees allow to resolve nearest neighbor problems almost in logarithmic time, i.e. finding the closest element to M_{Target} in a set of points. A Kd-Tree is a Binary Space Partitioning (BSP) [38] tree for which the cutting planes are orthogonal to one of the basis directions($\vec{x}, \vec{y}, \vec{z}, ...$). Kd-trees are primarily used in computational graphics where they play an important part in video games or ray tracing algorithms. Kd-Trees are a more general tree structure than quad-trees or oct-trees [66], since they can handle K dimensions. They are also easier to balance than quad-trees or oct-trees, which ensures a better performance. A graphical representation of a Kd-Tree for a portion of an aeronautical burner solid domain is shown on figure 10.1.

In this section the Kd-Tree construction and search procedure are presented. In this discussion a K dimensional space E is considered, $(O, \vec{x_1}, \vec{x_2}, \ldots, \vec{x_K})$ is an orthonormal basis of E.

10.1.1.1 Constructing a Kd-Tree

The procedure to build a Kd-Tree for a set of N points $(M_1, \ldots, M_N)$. The building procedure is a recursive algorithm, at each step

1 a cut plane is calculated from the set of points, this cut plane should be chosen to cut the point set in two equal parts.

2 all the points on the left of the cut plane are put in a new set which will form the left branch of the tree.

3 all the points on the right of the cut plane are put in a new set which will form the right branch of the tree.

This procedure is then applied to the left and the right sets in a recursive way. The algorithm is stopped when the size of the sets are small enough (this is a parameter to the method). In practice there is no need to build the entire tree (recursing until the sets contain only one point), in the version implemented the algorithm is used until the sets contain less than 32 points. The Kd-Tree is constituted by the different cutting planes (direction $\vec{x_j}$ and pivot P_j), their relations and the leaves containing the points M_i. In practice only the indices of the points are stored.

An important difference between Kd-Trees and other types of tree such as quad-trees or oct-trees is that Kd-Trees are simple to balance, i.e. at each node of a balanced tree the left and right branches contain the same amount of elements. A balanced tree is the smallest binary tree possible, and therefore storage and search efficiency is maximal.

Therefore the choice of the cut plane is essential. The Kd-Trees are axis aligned binary space partitioning trees therefore the cut plane is defined by a direction $\vec{x_j}$ and a value on this axis P_j. For a set of points $S = \{M_1, M_2, \ldots\}$ a perfectly balanced tree is obtained by choosing $P_j = median_{M \in S}(\vec{OM} \cdot \vec{x_j})$. Choosing the direction of the cut plane is the next difficulty. A practical solution is to choose the direction which after projection on its axis yields the greatest distance between the extremal points, an other more accurate method is to calculate the root mean squared of the coordinates for each direction [1]. In the following examples and in the implementation the extremal distance method is used. In the implementation the median is approached by using a dichotomic procedure until the balance is good enough, i.e. the balancing is stopped when the relative difference of the left and right sets is smaller than a given parameter. The complexity of the construction of a Kd-Tree depends on the algorithms used to choose the direction or compute the pivot but it is of the order of $O(nlog(n))$.

10.1.1.2 Searching in a Kd-Tree

A Kd-Tree is searched by successively testing the target element M_{Target} with the different cutting planes $(\vec{x_j}, P_j)$ until the current position in the tree is a leaf. If the current position is not a leaf then

- if $OM_{Target} \cdot \vec{x_j} < P_j$ the search procedure is applied to the left child branch.
- if $OM_{Target} \cdot \vec{x_j} >= P_j$ the search procedure is applied to the right child branch.

If the current position is a leaf, a linear nearest neighbor search is performed. This is why the parameter defining the size of the leaves is an important parameter for performance. A first result is obtained M_{temp}. Ending the search here results in very fast look-ups (exactly $log_2(N)$ for a balanced tree) but the algorithm would not be not exact. M_{temp} may not be the closest element to M_{Target}. A simple test must be performed:

- if $\|M_{Target}M_{temp}\| > |\vec{OM_{Target}} \cdot \vec{x_j} - P_j|$ then the other branch of the cutting plane must be searched recursively.
- if $\|M_{Target}M_{temp}\| <= |\vec{OM_{Target}} \cdot \vec{x_j} - P_j|$ then the result is satisfying for this stage.

[1]Some authors simply specify to cycle through the directions, this can be problematic if there are parts of the mesh are flat and are axis aligned (this happens with boxes). If by cycling on the directions the cut plane's direction is orthogonal to the flat part of the mesh then that sorting stage will be inefficient.

In other words, the test compares the distance between the found point M_{temp} and M_{Target} with the distance between the M_{Target} and the current cutting plane. If that cutting plane is closer to the target than the temporary result, a better result might be on the other side of that cutting plane. Hence the search should continue on the other side. The search process is a recursive up-down process, in most cases there are few side changes (especially if the bucket size is not too small) keeping the average complexity near to $O(log(n))$. However if there are a lot of side changes then the search performance can be drastically degraded.

10.1.1.3 Example of a Kd-Tree in 2D

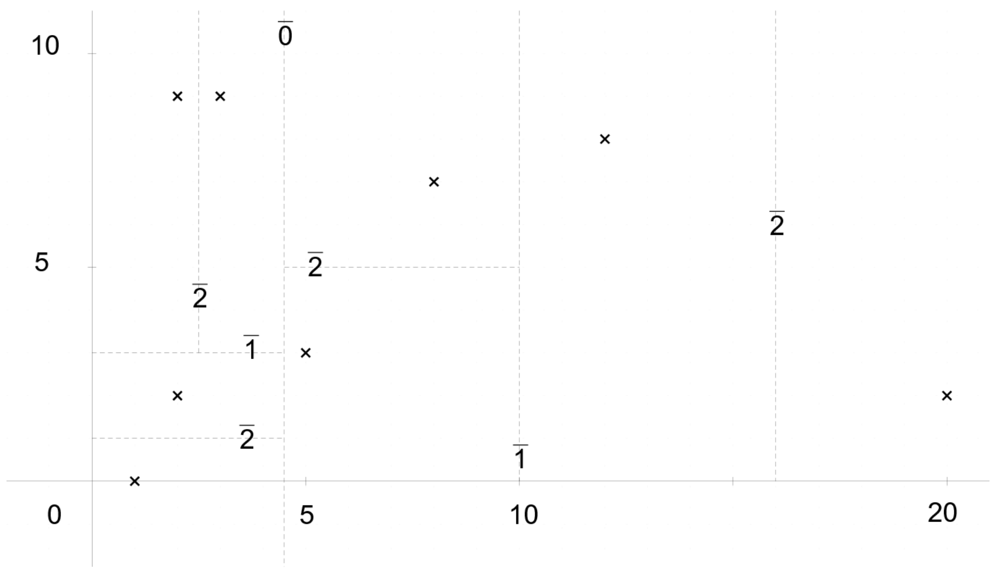


Figure 10.2: The point set with the different cut planes, the recursive level of the cut plane is noted by over-lined numbers

To help understand a simple example is detailed. The point set is $S_0 = ((2,2),(1,0),(3,9),(20,2),(5,3),(12,8),(2,9$
Fig 10.2.

1. Evaluate the extremal distance for each direction, $\Delta x = 20 - 1 = 19$, $\Delta y = 9 - 0 = 9$ so the first cut plane is orthogonal to $\vec{x}$.
2. Find a pivot for the x coordinates of the point set S. By writing them in ascending order $1, 2, 2, 3, 5, 8, 12, 20$, it is clear that 4.5 splits the set into two equal subsets. hence two subsets are created:

   S_0^{left} the left subset $((1,0),(2,2),(2,9),(3,9))$

   S_0^{right} the right subset $((5,3),(8,7),(12,8),(20,2))$.

The procedure is then applied recursively to S_0^{left} and S_0^{right}, the tree produced is shown on Fig. 10.3.

To illustrate the search process two different searches will be considered.

This first example shows a result obtained through a direct tree descent which is the best case, the target point is $M_{Target} = (1,-1)$. The search starts at $Node_0$, the successive steps are, Fig 10.4:

1. Test M_{Target} with plane $x = 4.5$, M_{Target} is on the left of $x = 4.5$ (1¡4.5). Continue on left branch, the current location in the tree is $Node_1$.

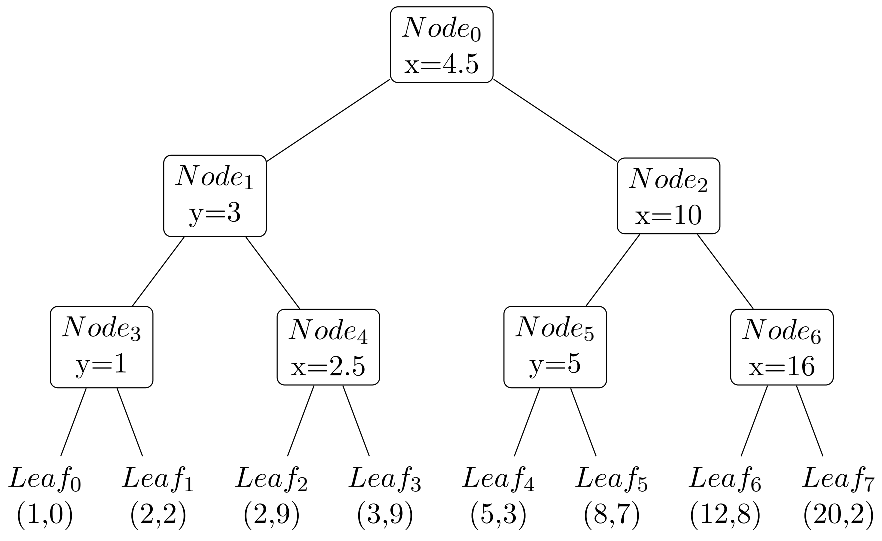


Figure 10.3: A Kd Tree in 2D (cut planes (pivots) are marked with rectangular boxes)

2. Test M_{Target} with plane $y = 3$, M_{Target} is on the left of $y = 3$ (-1¡3). Continue on left branch, the current location in the tree is $Node_3$.

3. Test M_{Target} with plane $y = 1$, M_{Target} is on the left of $y = 1$ (-1¡1). Continue on left branch, the current location in the tree is $Leaf_0$.

4. A leaf has been reached, a temporary result obtained through linear nearest neighbor search is $M_{Temp} = (1, 0)$.

5. Compare distance between $M_{Temp} = (1, 0)$ and the cut plane of the parent node $Node_3$: $y = 1$. $dist(M_{Temp}, M_{Target}) = 1$ smaller than $dist(M_{Temp}, y = 1) = 2$, therefore there is no need to check $Leaf_1$. The location in the tree moves up to $Node_3$.

6. Compare distance between $M_{Temp} = (1, 0)$ and the cut plane of the parent node $Node_1$: $y = 3$. $dist(M_{Temp}, M_{Target}) = 1$ smaller than $dist(M_{Temp}, y = 3) = 3$, therefore there is no need to check $Node_4$. The location in the tree moves up to $Node_1$.

7. Compare distance between $M_{Temp} = (1, 0)$ and the cut plane of the parent node $Node_0$: $x = 4.5$. $dist(M_{Temp}, M_{Target}) = 1$ smaller than $dist(M_{Temp}, x = 4.5) = 3.5$, therefore there is no need to check $Node_2$. The closest point of M_{Target} is therefore $(1, 0)$.

This second example shows a result obtained through a up-down tree walk, the target point is $M_{Target} = (0.5, 1.1)$. The search starts at $Node_0$, the successive steps are, Fig 10.5:

1. Test M_{Target} with plane $x = 4.5$, M_{Target} is on the left of $x = 4.5$ (0.5¡4.5). Continue on left branch, the current location in the tree is $Node_1$.

2. Test M_{Target} with plane $y = 3$, M_{Target} is on the left of $y = 3$ (1.1¡3). Continue on left branch, the current location in the tree is $Node_3$.

3. Test M_{Target} with plane $y = 1$, M_{Target} is on the right of $y = 1$ (1.1¿1). Continue on right branch, the current location in the tree is $Leaf_1$.

4. A leaf has been reached, a temporary result obtained through linear nearest neighbor search is $M_{Temp} = (2, 2)$.

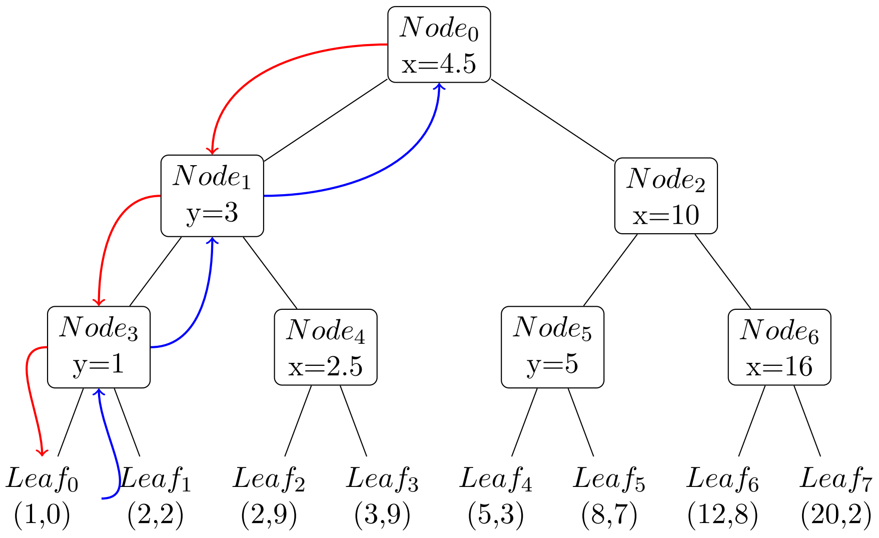


Figure 10.4: Search path for $M_{Target} = (1, -1)$. Descent is in red, Ascent in Blue

5. Compare distance between $M_{Temp} = (2, 2)$ and the cut plane of the parent node $Node_3$: $y = 1$. $dist(M_{Temp}, M_{Target}) \simeq 1.75$ greater than $dist(M_{Temp}, y = 1) = 0.1$, therefore $Leaf_0$ should be checked. The left leaf ($Leaf_0$) is searched using the normal recursive up-down algorithm. The new temporary target is $M_{Temp} = (1, 0)$. The location in the tree moves up to $Node_3$.

6. Compare distance between $M_{Temp} = (1, 0)$ and the cut plane of the parent node $Node_1$: $y = 3$. $dist(M_{Temp}, M_{Target}) \simeq 1.75$ smaller than $dist(M_{Temp}, y = 3) = 3$, therefore there is no need to check $Node_4$. The location in the tree moves up to $Node_1$.

7. Compare distance between $M_{Temp} = (1, 0)$ and the cut plane of the parent node $Node_0$: $x = 4.5$. $dist(M_{Temp}, M_{Target}) = 1$ smaller than $dist(M_{Temp}, x = 4.5) = 3.5$, therefore there is no need to check $Node_2$. The closest point of M_{Target} is therefore $(1, 0)$.

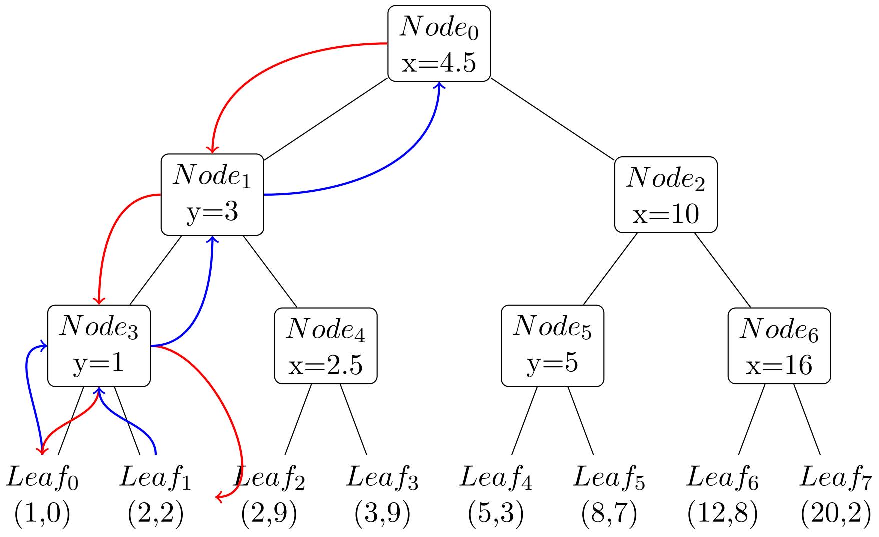


Figure 10.5: Search path for $M_{Target} = (0.5, 1.1)$. Descent is in red, Ascent in Blue

10.1.2 Validation of the Kd Tree implementation

The Kd Tree implementation has been tested on a simple problem. A program generates randomly two sets of N points in a three-dimensional space, these sets correspond to the vertices of two grids G^s and G^d. Then the Kd Tree code is used to find the nearest neighbor of each vertex of G^d in G^s. The same projection is also performed using the brute force algorithm. This procedure has been carried out for problem sizes varying between 2048 points to 262144 (problem sizes chosen here were powers of 2 but this is not mandatory). The first result, yet mandatory, obtained is that both methods returned exactly the same nearest neighbor for each point of G^d.

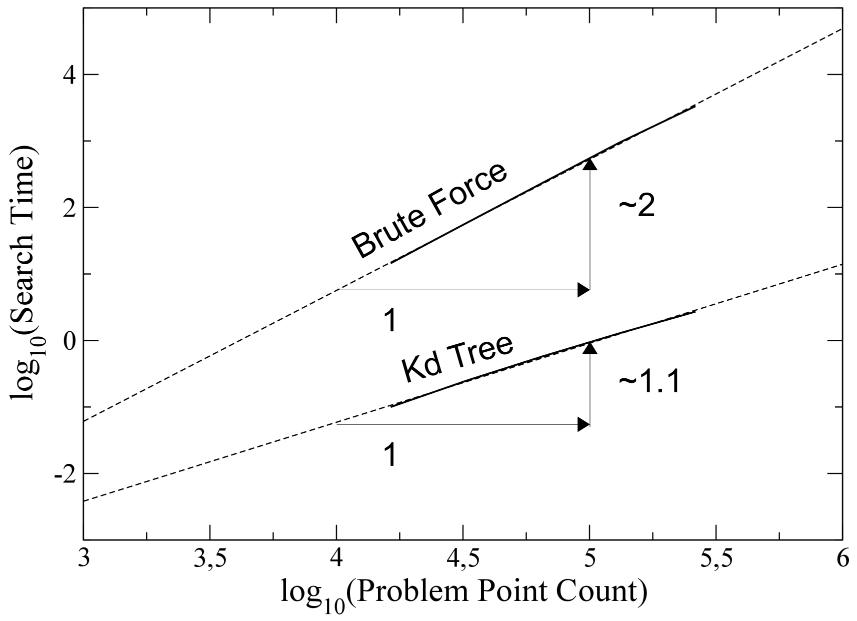


Figure 10.6: Scaling of the Kd Tree method versus the brute force method

Having validated that the Kd Tree implementation returned the correct results the methods have been timed to check the actual scaling of the implemented algorithm. The two methods performance is as expected: looking at figure 10.6 it is clear that the Kd Tree almost scales linearly to the problem size whereas the brute force scales with the square of the problem size.

As mentioned earlier a solution to the Kd Tree's drawback (the memory needed to store the tree) is to store more than 1 vertex per leaf (bag size> 1). However the effect on search performance needs to be investigated, therefore the same tests have been performed for different bag sizes. The interesting quantities are the search time, Fig 10.7, and the memory foot print of the method, Fig 10.8.

These results show that even the slowest Kd Tree solutions considered here clearly outperform the brute force approach, also choosing larger bag sizes is detrimental to performance but clearly decreases the memory required. It is difficult to actually decide which bag size is optimal, though the trends are clearly generalizable, the results obtained here are machine dependent (these results will depend on the processor type, frequency, cache size and memory latency). Furthermore the actual choice of the bag size depends on the problem to solve and the target machine:

- if the nearest neighbor searches have to be very fast (because they are part of a complex looping algorithm) and memory is not an issue then the bag size should be chosen to low values (generally

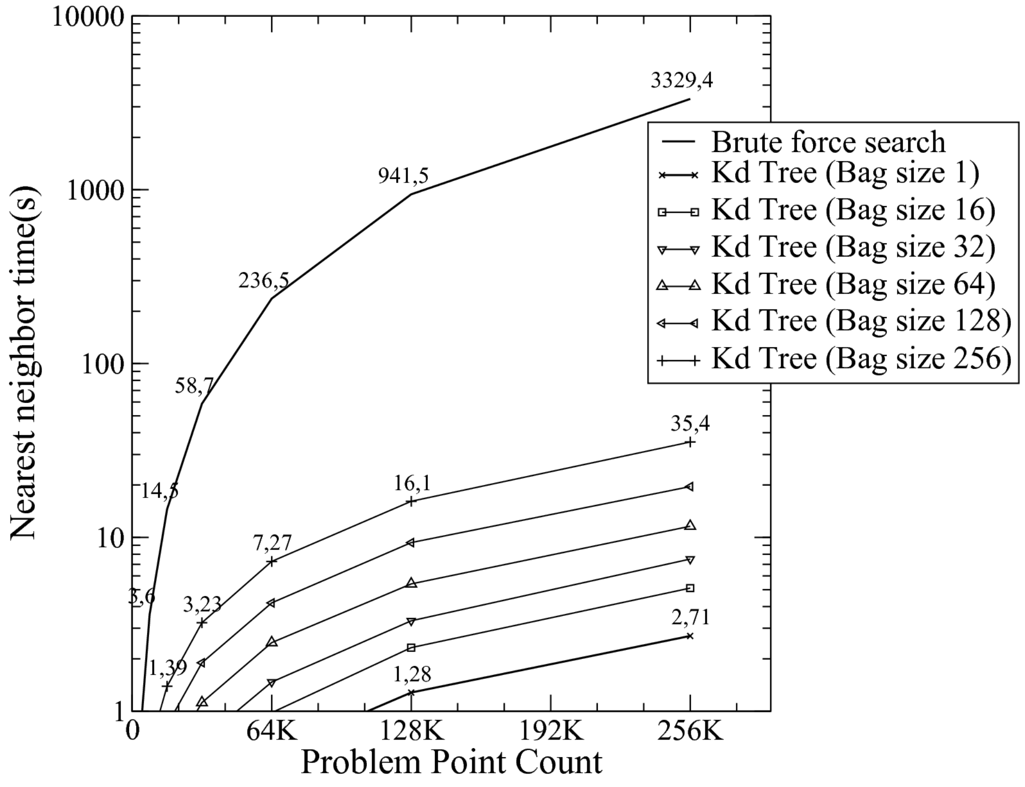


Figure 10.7: Timings for different nearest neighbor solutions

2 or 4 are as fast as 1)

- on the other hand if the CPU time is less critical than memory usage then choosing higher values is interesting (32 or 64)

In the coupling application the nearest neighbor calculation is done only once and memory may be an issue therefore the bag size is chosen to 32.

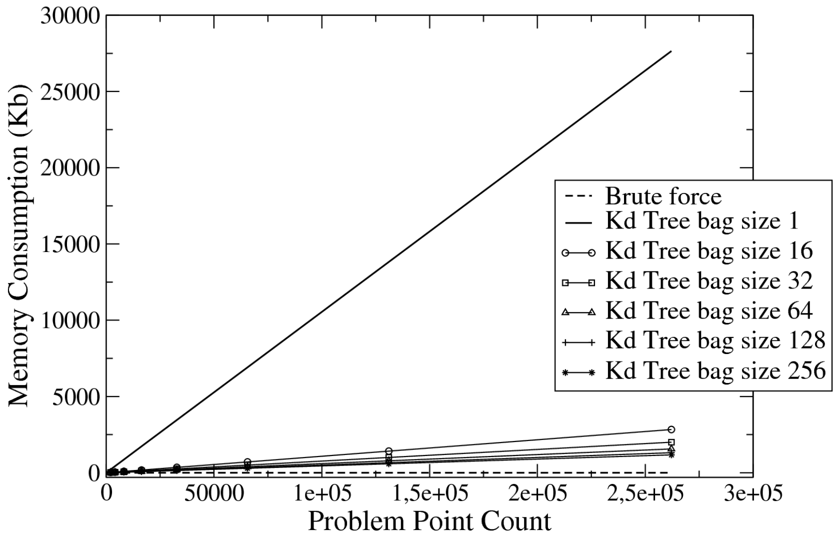


Figure 10.8: Kd-Tree memory consumption for different bag sizes

10.2 Element scan methods

Interpolation methods such as linear interpolation or linear conservative interpolation rely on geometry localized in small portions of space. Therefore reducing the search space to a small region can greatly optimize the search methods used. A simple idea is to scan a portion of space using the mesh adjacency, i.e. the relation between each mesh element and its neighbors [2].

Two procedures are described, each are used to accelerate a key process of the linear and linear conservative interpolation coefficient calculations.

10.2.1 For linear interpolation: Finding the containing element

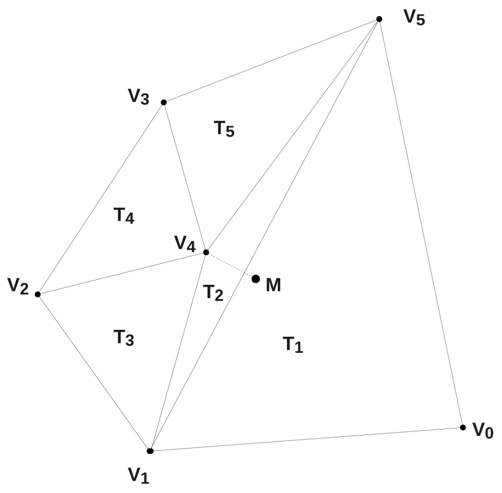


Figure 10.9: A situation where the list of elements associated to the closest vertex does not contain the target point M

Computing linear interpolation seems fairly simple, for each vertex G_i^d the source grid G^s is searched for the element $E_j^{G^s}$ such as $G_i^d \in E_j^{G^s}$. Then the barycentric coordinates of G_i^d in $E_j^{G^s}$ are calculated. Hence the difficulty resides in finding the element $E_j^{G^s}$ containing G_i^d. Two different approaches are described in this section. The first method uses the nearest neighbor algorithms to find the closest vertex of G_i^d in G^s, then an element to element scan is executed in order to find the element containing the vertex G_i^d.

An other approach is to use a binary space partitioning approach to sort elements, a method named

[2]Mesh adjacency can be computed using optimal sort algorithms therefore in almost linear time $O(nlog_2(n))$

AABB tree [118] search is described in 10.3.

```
Data: $Element\_Processed$ a Boolean array initialized to false marking if elements have
      already been processed
$recursive\_walk\_test(element\_index, point\_to\_test, recursion\_level)$ : begin
    Mark element as processed  $Element\_Processed[neighbor\_index] \leftarrow True$
    Retrieve element from element index  $E \leftarrow Element\_From\_Index(element\_index)$ ;
    Do the vertex in element test ;
    if $TestVertexInElement(point\_to\_test, E)$ then
        vertex is in element return element's index  return $element\_index$
    end
    else
        continue only if recursion level is not too high  if $recursion\_level < max\_rec\_level$ then
            retrieve adjacency neighbor_indices for element element_index
            $neighbor\_indices \leftarrow Get\_Element\_Neighbors(element\_index)$ ;
            for each neighbor check if it has already been processed  foreach $neighbor\_index$ in
            $neighbor\_indices$ do
                if not $Element\_Processed[neighbor\_index]$ then
                    element has not yet been processed apply recursion on this neighbor
                    $r \leftarrow recursive\_walk\_test(neighbor\_index,\ point\_to\_test, recursion\_level + 1)$;
                    if $r \geq$ then test was successful ;
                    if $r \geq 0$ then
                        return $r$
                    end
                end
            end
        end
    end
    no element found ;
    return $-1$
end
```

Algorithm 7: Recursive walking algorithm

For linear interpolation and higher order interpolation methods it is necessary to know in which element of G^s each point of G^d is projected to. A simple method is to use the Kd-Tree algorithm detailed above coupled with the node-to-element connectivity. For a target point M_{Target} of G^d:

1. using the Kd-Tree find the closest node of M_{Target} in G^s, noted $G^s_{closest}$,

2. use the node-to-element connectivity to obtain a list of elements associated to $G^s_{closest}$.

Hence this algorithm requires precomputing the list of elements associated to each vertex of G^s[3]. However in some cases the elements associated to the closest vertex may not contain the target point M_{Target}. Figure 10.9 shows an example situation where the closest vertex to M_{Target} is V_4, but the elements associated to V_4 namely T_2, T_3, T_4, T_5 do not contain M_{Target}.

In such a situation two strategies are possible: either fall back to nearest neighbor interpolation, or do a more exhaustive search to find the element containing M_{Target}. An efficient method to do this, provided that the mesh adjacency is calculated, is to march through the mesh moving from each face to its neighbors recursively (Fig 10.10). This may require storing the elements which have already been treated to avoid an infinite recursive loop[4] A strong stop condition, for example a maximum

[3]This can be computed in linear time, and for static meshes this needs to be done only once.

[4]In the example algorithm presented a Boolean array is used, however this requires resetting the array at each new

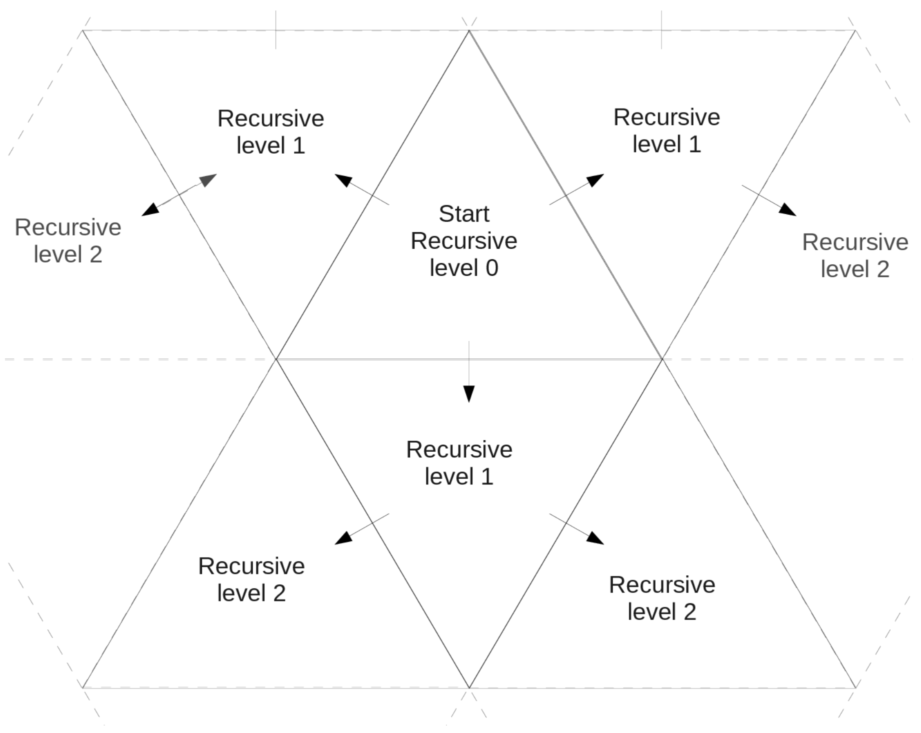


Figure 10.10: A recursive mesh marching algorithm (with maximum recursive level of 2)

recursion level or a maximal distance, is mandatory to prevent the algorithm to propagate too far from the interesting zone. Also choosing the correct propagation direction, i.e. propagate the recursive march towards the target point by selecting the correct propagation face first (this is possible using the face normal) is crucial for the performance of the algorithm.

10.2.2 For linear conservative interpolation: efficient projection algorithm

The essential part to optimize in this algorithm is not the intersection calculation but make sure that this procedure is executed on the smallest set of elements possible. The solution to optimize the intersection calculation of element E_i^d on the source grid G^s which has been implemented is also based on an element recursive march on the source grid. The march is carried out in two phases:

1. At first the algorithm propagates recursively from an initial guess element $E^s_{init\_search}$ until a non empty intersection between the target element E_i^d and the current source element E_j^s is obtained $E_i^d \cap E_j^s \neq \emptyset$. The first march is stopped, the current source E_j^s element is noted $E^s_{init\_intersect}$.
2. A second recursive march is propagated from $E^s_{init\_intersect}$, at each element the intersection with E_i^d is calculated. If the intersection is non empty then it is used for the integral calculation and the algorithm continues propagating from that element, otherwise the algorithm stops propagating.

The first phase is the actual search phase, in practice it should be accelerated by choosing an initial element $E^s_{init\_search}$ close to E^d (this can be done using nearest neighbor search and node to element connectivity). Also the propagation direction towards the element E_i^d should be privileged.

The second phase is the actual element on element projection phase.

search. A slicker way to do this is to use an integer array and identify each new search by a different integer. Each time an element is processed it is marked using the current search integer therefore the array does not need to be reinitialized at each new search.

10.3 Binary space partitioning applied to elements AABB trees

An alternative approach to solve the *find-the-containing-element* problem is presented here. The idea is to use an extended version of the Kd-Tree to sort elements directly, the method described here is the AABB tree [118] search method. This algorithm has not been implemented within the coupling library but in the graphical tool presented in A.3. The fundamental idea used in this algorithm is to sort the geometry using axis aligned bounding boxes. At first all the geometry is contained within a global box. Using a similar method to the Kd-Tree an axis aligned cut-plane is calculated splitting the geometry into two new balanced sets. An axis aligned bounding box is calculated for each new subset, if elements are intersected by the cut plane then they are added to the two sub-sets. This procedure is applied recursively until the sets of elements are small enough (controlled by a parameter). The search procedure is simpler because in this case if a point is contained within an bounding box the element is in that in that bounding box. Therefore there is no up-down motion like in the Kd-Tree method. If the balancing is properly done the algorithmic complexity is comparable to the Kd-Tree initial root to leaf search. However in this case the search is performed on the elements instead of the vertices. This can make a non negligible difference: in a mesh the element count can generally be several times greater than the vertex count. Also the storage needed for each node of the tree is greater than in the Kd-Tree case since a bounding box needs to be stored at each node of the tree instead of a cut plane. This algorithm allows to solve the *find-the-containing-element* problem without needing the vertex to element connectivity and the element to element scan search, hence simplifying the implementation code needed.

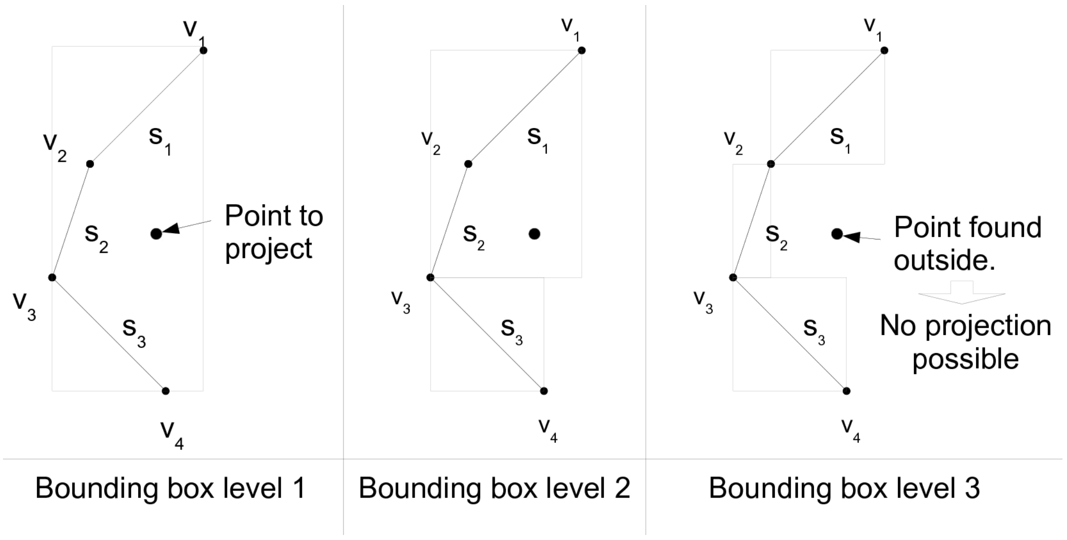


Figure 10.11: AABB-Tree projection process

There is however a fundamental difference between the two approaches. This algorithm is not suited for vertex projection onto meshes. As it can be seen on figure 10.11, if the point projected is outside of the meshes element bounding boxes then the point is considered to be outside of the domain. On the contrary the Kd-Tree considers infinite bounding volumes since it only uses cut planes to partition space, fig 10.12. This means that this algorithm can be very interesting to test if a location is contained within an arbitrary mesh. Yet using this algorithm for surface to surface projection in 3D cases is not a good choice because of the likelihood of having the source mesh points outside of the destination mesh element bounding boxes. This algorithm can be used for more than only interpolation, it can be used for contact computation or ray tracing. An example of usage of this algorithm is presented in appendix A.3.

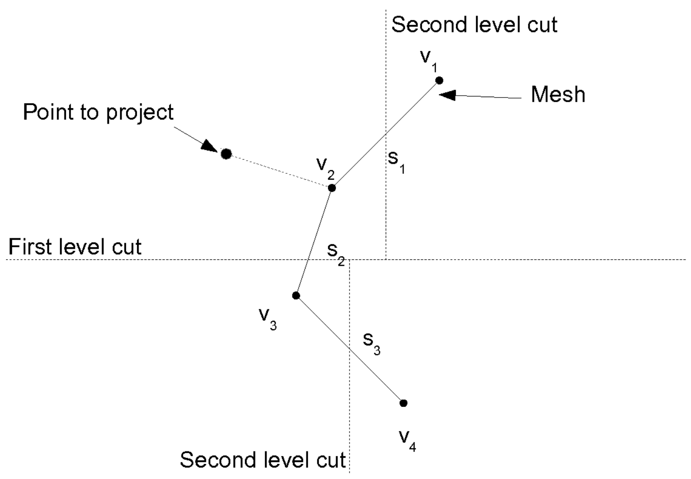


Figure 10.12: Kd-Tree projection process

Conclusion

In this part interpolation methods have been investigated. First two standard signal sampling based techniques have been considered: nearest neighbor interpolation, linear interpolation. Then more advanced methods called conservative methods capable of preserving signal integrals have been built, namely the conservative and linear conservative interpolation methods. These methods have been compared on basic problems involving simple harmonic based signals on simplicial meshes. Different characteristics of these methods were studied, their accuracy, their integral conservation and their sensitivity to aliasing. The best method obtained was the linear conservative method as it obtains the same accuracy as the linear interpolation method while maintaining high integral conservation and very low sensitivity to aliasing. Then geometrical search algorithms allowing efficient implementation of these interpolation methods have been explained.

However in this part a complex aspect of parallel computing has not been considered: mesh partitioning. Indeed as shown in part I the high computational power required by LES means are only met by massively parallel machines. On such machines the mesh is subdivided into smaller sub meshes which are solved individually by each processor. This partitioning procedure applies also to the coupling interfaces hence meaning that these interpolation problems have to be solved considering an additional difficulty: the geometrical searches have to consider geometry distribution. This leads to the next topic of this thesis, part IV.

Part IV

Code coupling methods designed for high performance computing

Table of Contents

Nomenclature

(p, q, r)	Integer grid coordinates on cartesian projection grid
Δt^A	Solver A iteration duration
Δt^B	Solver B iteration duration
$\Delta x, \Delta y, \Delta z$	Projection grid stride in x,y,z directions
Cc_n^j	Cell of index n on client processor of rank j
Cs_n^j	Cell of index n on server processor of rank j
G^c	Client grid
G^u	Uniform grid
N_{iter}^A	Iteration count between two couplng synchronizations for solver A
N_{iter}^B	Iteration count between two couplng synchronizations for solver B
P_i^C	Client process of rank i
P_i^S	Server process of rank i
T_{ij}	Interpolation matrix

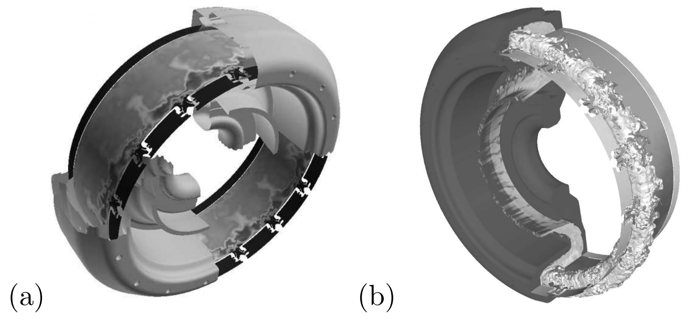


Figure 10.13: (a) 93M Tetra LES by Boileau et al. [12] (b) 336M Tetra LES by Wolf et al. [120]

It has been seen in chapter 2 that simulating configurations such as an aeronautical burner with LES requires important computational power. Similar simulations have already been accomplished with much bigger meshes. For example Boileau et al. [12] and Wolf et al. [120] have applied LES to simulate complex combustion in a full Turbomeca Ardiden burner using respectively 93 and 336 million tetrahedral meshes, Fig 10.13. At the time this work has been carried out, such computational power is only achievable using massively parallel computers. These specific computer architectures imply many HPC specific problems which must be dealt with in order to take advantage of the huge computational power available. In this thesis the objective is to establish a code coupling methodology for large LES simulations using massively parallel computers. Even though the LES configuration considered in this work is more modest in size and geometrical extent, the problem remains still very challenging.

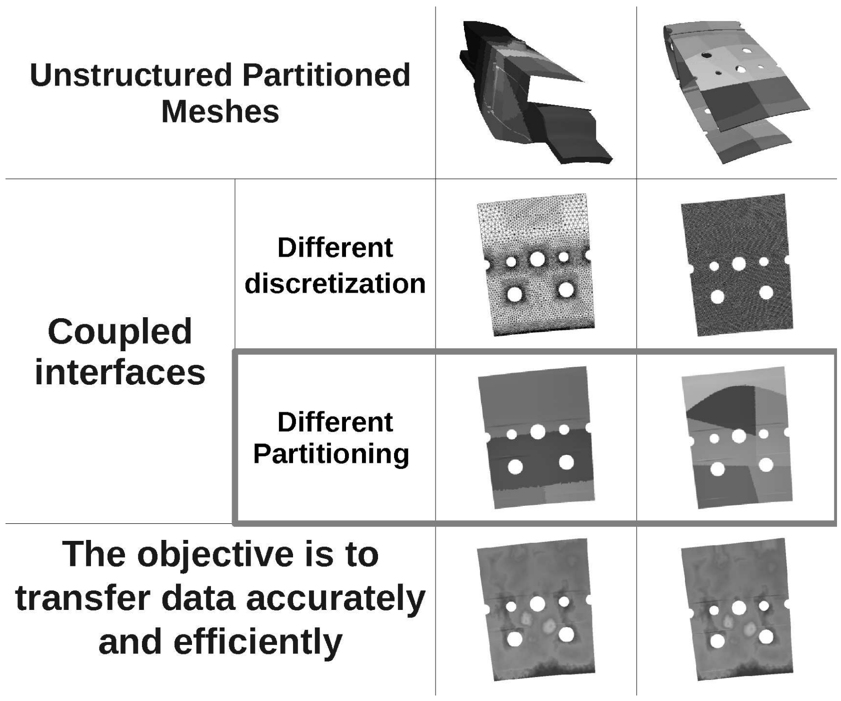

Unstructured Partitioned Meshes
Coupled interfaces
Different discretization
Different Partitioning
The objective is to transfer data accurately and efficiently

Figure 10.14: Summary of the technical issues for data coupling

Simulating this configuration using typical massively parallel machines implies handling complex partitioned geometries distributed over huge networks of processors. Typically if using the BlueGene/P massively parallel super computer architecture, important constraints on communication patterns and memory foot print for the algorithms have to be considered. This is of particular importance for parallel coupled problems: to exchange data between parallel solvers, communication routes have to be established. These routes may be designed using different patterns: one can choose to gather the interface data on one or a few processors and then scatter this information to the other solver. Or one can choose to communicate directly between processors of the two solvers. The comparison of these two choices is carried out in this part and the results show that direct communication is clearly the most scalable pattern. However directly connecting processors from a solver to an other requires computing communication routes based on the coupling interfaces. This requires defining the coupling interfaces: it is important to identify the geometrical surfaces which are to be connected in the coupled problem. Dealing with complex geometry a very specific problem may appear due to different discretizations on curved surfaces. This problem may lead to improper interface connection and hence inaccuracies in the coupled computation. To solve this problem the problematic cases have been identified and a simple methodology has been proposed. Having identified geometrically the interfaces, the connection step can be performed. Connecting two geometrical interfaces requires geometrical search algorithms. On large meshes the interface mesh can become considerable, it is hence important to use efficient search algorithms. But such algorithms are generally based on additional data structures (notably trees, see chapter 10) which may have a considerable memory foot print. Gathering the entire interface meshes on a single processor in order to use such algorithms may lead to exceeding the local available memory and eventually to a software failure. To avoid the memory limitation at this step, it is important to design a parallel geometrical search method capable of directly handling the distributed geometrical interfaces provided by the available mesh partitioning, Fig 10.14. A method based on distributed hash tables and geometrical hashing is proposed in this thesis. This method is first validated on a basic test case and then used on a real industrial geometry.

Chapter 11

Issues specific to HPC

11.1 Introduction

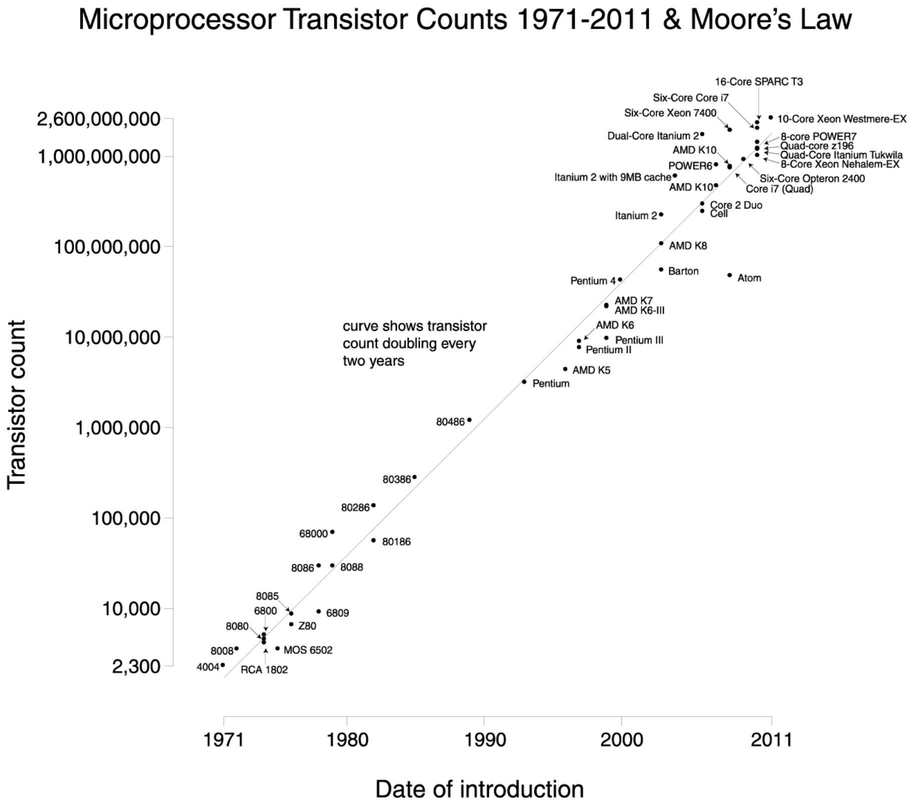


Figure 11.1: Moore's law

As shown in chapter 1, LES is a promising path for greater simulation accuracy. However the drawback of LES is that it is computationally expensive. This is why high performance computing (HPC) is essential to LES. The HPC world evolves very rapidly, Fig 11.1, notably due to the cost improvements made on consumer products hardware. The scientific computational community which traditionally used very expensive high end processors, now begins to use more low cost processors while still obtaining a performance gain [1]. Finally because of the thermal dissipation constraints, CPU frequencies have reached a threshold since about 2005, redirecting modern HPC towards massively

[1]This has started with the transfer from vector processors to scalar processors and is continuing with the progressive usage of the GPGPU

parallel solutions. The goal of this part is to be able to run coupled codes on modern massively parallel architectures.

11.2 Massively parallel architectures

Presenting parallel machines could lead to a very exhaustive discussion. Instead only three main classes of parallel machines will be considered here:

The first class is built by connecting a set of computers using a high speed network such as infiniband. Generally these clusters of computers are composed of high-end processors connected to large memory banks. Because they use powerful processors and large memory banks even poorly parallelized codes can perform well on such machines. These machines are generally the easiest machines on which a simulation code can be ported. However because of the use of powerful processors with large memory banks (which are rarely entirely used, at least for a solver like AVBP), they are expensive and consume a lot of power.

Figure 11.2: An IBM BlueGene super computer

A solution proposed by IBM to this cost and power consumption is the BlueGene architecture, Fig 11.2. Thousands of low-end processors with small memory banks are connected using a specially designed high performance network. The software environment is maintained extremely lightweight: a primitive operating system is provided giving access to most of the key functionalities needed for basic code execution. However many standard functions commonly present on UNIX/Linux systems are not implemented. Porting codes to this architecture is not a simple task. What is more because this architecture is based on rather slow cores (850Mhz for BlueGene/P) and since the memory accessible per core is low (512Mb on BlueGene/P when all cores are used), algorithms centralizing computation or memory load have to be rethought. The advantage is that these machines are less expensive and provide a much a higher GFlop/Watt ratio than standard computational clusters. BlueGene supercomputers can count several hundreds of thousands of cores and have lead the TOP-500 super computer ranking during several years.

Finally a very recent type of processor, namely the General Purpose Graphics Processing Unit (GPGPU), is introducing a new type of computing paradigm [60]. Porting codes to GPGPU is very difficult and specially for solvers using hybrid unstructured grids such as AVBP. Use of GPGPUs is however beyond the scope of this thesis.

11.3 Issues specific to Massively parallel

Scalability or code performance response to an increase of processors used is therefore the key for modern software development in HPC environments. A key for scalability is to reduce the quantity of sequential code. This is explained through Amdahl's law[3]:

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}} \tag{11.1}$$

Here P is the proportion of a program that can be parallelized, $1 - P$ the proportion of the code which is sequential, $S(N)$ is the maximum speed-up obtained using N processors. Hence the maximal speed-up using an infinite number of processors is given by the limit $\lim_{N\to\infty} S(N) = \frac{1}{1-P}$ which depends solely on the sequential proportion of the code. Therefore designing scalable methods implies reducing the sequential proportion of the code.

Note also that Amdahl's law[3] only considers execution time, but in practice an increase of the computational power (more processors) is generally linked to a larger problem to solve. If the sequential portions of the code contain algorithms which process amounts of data scaling with the problem size, then a hard limit is reached when the memory required for those algorithms reaches the available memory. This constraint is very architecture dependent. On clusters equipped with large memory banks per processor this is not an issue. On architectures such as BlueGene this rapidly becomes the most important constraint: if there is not enough memory for the sequential algorithms the code can not continue. Therefore data distribution is also a key aspect of scalability.

A final scalability aspect to take into account is communication. Sequential code generally implies many-to-one and one-to many communication patterns. These patterns generally imply important communication stress, notably long message queuing and possible packet collisions.

Since the fundamental task in code coupling is to transfer and transform data between two separate solvers, efforts should be made to parallelize the coupling portion of the code, avoid data centralization, and therefore communication stress. Most methods implemented yet are based on a client server model: the entire distributed interface is gathered on to a service processor. The service processor hence performs the remapping or interpolation and then scatters this interface information on to the destination solver's interface.

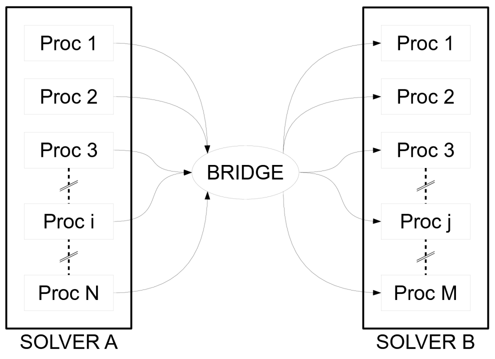


Figure 11.3: Centralized Communication scheme (CCS)

The following problem illustrates these scalability issues. Interface data has to be transferred from a parallel solver A to a parallel solver B. Solvers A and B have partitioned non matching interfaces (the most general case), therefore a transformation must be applied to transfer this data. This operation can be performed using two different schemes: the Centralized Communication Scheme (CCS), Fig. 11.3, and the Direct Communication Scheme (DCS), Fig. 11.4.

CCS implies using at least one or many service processors. In many cases only one service processor is used, therefore only one is considered here. CCS can be broken into three main steps:

1. The data is gathered from solver A to the service processor (many-to-one communication pattern).

2. The data is processed on the service processor.
3. The data is scattered from the service processor to solver B (one to many communication).

Hence this scheme focuses the stress on the service processor:

- receiving data from all the processors can lead to network collisions and therefore degraded message passing performance.
- enough memory to gather the entire data on the service processor is mandatory, and in some cases may not be possible (BlueGene).
- the service processor has to process all the data in a sequential way.

Also due to the two levels of communication the latency expected by this method is the sum of the latency of the gather and scatter operations and the execution time of the data processing. For large set of data this scheme may thus lead too high latencies.

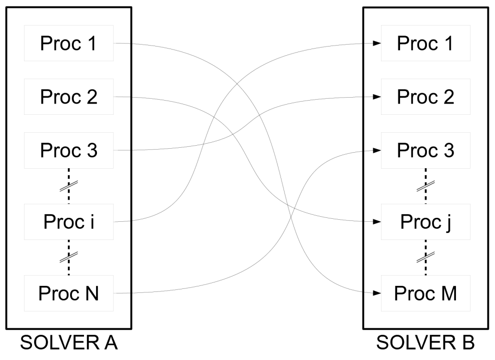


Figure 11.4: Direct Communication scheme(DCS)

On the contrary the DCS scheme does not rely on a service processor. It requires that all the processors know *a priori* the communication routes between solver A and solver B processors, and if the data has to be transformed, the operation must be done in a partitioned way, either on the sending processor or on the receiving processor. In DCS only one level of communication is needed, leading to lower latencies than in CCS. Also the quantity of data processed by each processor remains small because the communication routes are based on the adjacency between processors of solver A and of solver B at the interface. Therefore this scheme implies less memory, communication and CPU stress.

A toy application demonstrating the difference between these two schemes is built. This application does not implement the full transformation stage. For CCS, the service processor can be viewed as a network router transferring messages from solver A processors to solver B processors (the routing is known a priori). For DCS the messages are sent directly from processors of A to processors of B. The same start and end points for each message are used with CCS and DCS. The toy application is timed for each scheme with messages of size 100Kb and different processor counts: from 64 to 4096 processors (the processors are evenly distributed among the solvers), Fig. 11.5. This test has been performed on an SGI-ALTIX ICE super computer. Looking at Fig. 11.5 which shows the two scheme responses as a function of processors involved in coupling, it is clear that the transfer time on the CCS scales with the number of processors used, whereas the DCS maintains almost constant timings regardless of the processor count.

The conclusion is obvious, in order to remain scalable the communication scheme must stay direct (DCS). The result is not surprising since all scalable codes use this direct communication schemes

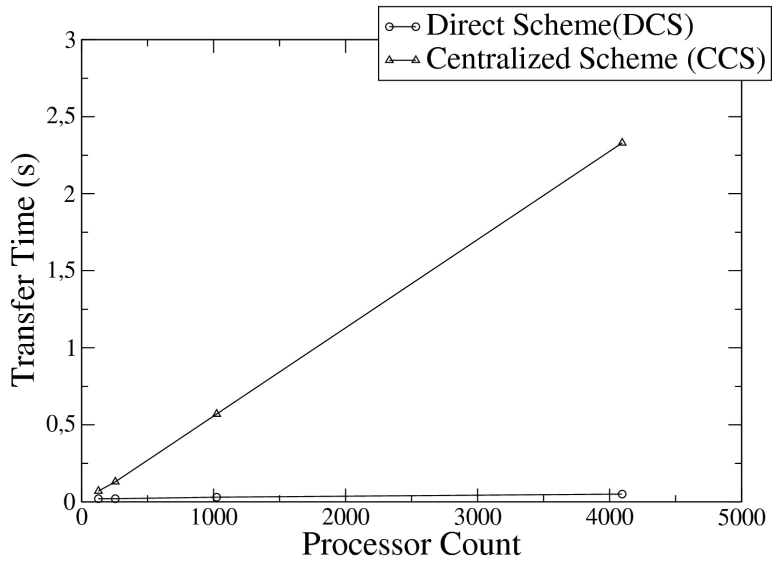


Figure 11.5: Comparison transfer time between CCS and DCS

for their internal communications. Therefore the inter-solver communications should also follow this model. Choosing this model however implies identifying a priori the communication routes between the processors involved in coupling. The interpolation methods must also be adapted to operate in this distributed model.

Chapter 12

Computational view of code-coupling for unstructured meshes

From the computer science point of view code-coupling is merely an inter program communication mechanism. However because unstructured meshes are used in parallel environments, additional complexity is added: first the interfaces between the solvers have to be identified on the different meshes. Once the meshes are partitioned, the solvers must be connected according to the partitioned interfaces. Then the transforms to apply to the data in order to transfer the data between the partitioned interfaces must be computed. Finally, during the execution of the solvers data must be transferred and transformed according to the connection and transformations established in the previous points.

In this work the coupling mechanism is broken into 3 phases:

- the first phase is the setup phase, in this phase the interface to couple are identified,
- the initialization phase, the communication routes and transformations are established,
- the runtime phase, the actual inter-solver communications are performed.

Each of these points are presented in this chapter.

12.1 The setup phase

As presented in chapter 5, in the aeronautical chamber, several different complex geometrical surfaces have to be coupled, Fig 12.1. Some methods have to be developed to handle the specific difficulties of complex geometry. Although the following methods will seem to be details, they are mandatory for correct handling of such problems.

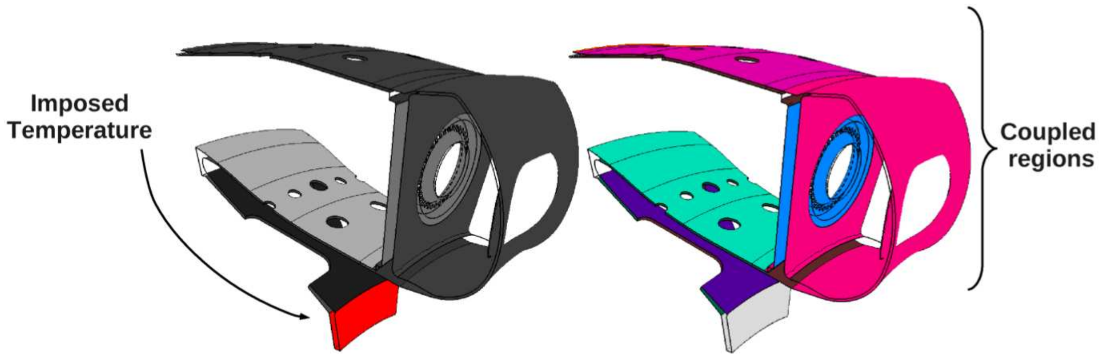


Figure 12.1: Regions which should be coupled between the solid and the fluid solvers in the aeronautical burner configuration

When dealing with industrial aeronautical burners, the solid domain is essentially composed of thin metallic walls which will be coupled on both sides. Locally strong curvature and surface discretization imposed by meshing can lead to improper interpolation. These are emphasized by the different discretizations in the fluid and the solid meshes. In extreme cases, the outside of a mesh can be projected on the inside of the other and vice versa, Fig. 12.2. An example of such situation on a real case is shown on Fig. 12.3.

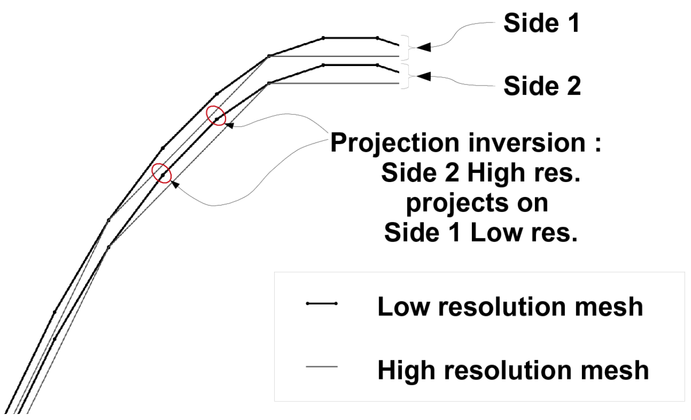

Side 1
Side 2
Projection inversion : Side 2 High res. projects on Side 1 Low res.
Low resolution mesh
High resolution mesh

Figure 12.2: Interpolation inversions

To handle such situations properly the coupled nodes are not treated in a global pool but in separate *coupling regions*. These coupling regions define a subset of the entire geometry allowing the restriction of the geometrical searches and interpolations to a subset of the global geometry. A coupling interface between solver A and B is defined by the couple of coupling region meshes in solvers A and B. In this application the coupling regions are defined by joining individual boundary conditions of the uncoupled configuration, Fig 12.4. This method requires that the different mesh surfaces are divided in a compatible way, but this should always be possible as long as the surface meshes involved in coupling discretize the same region in space.

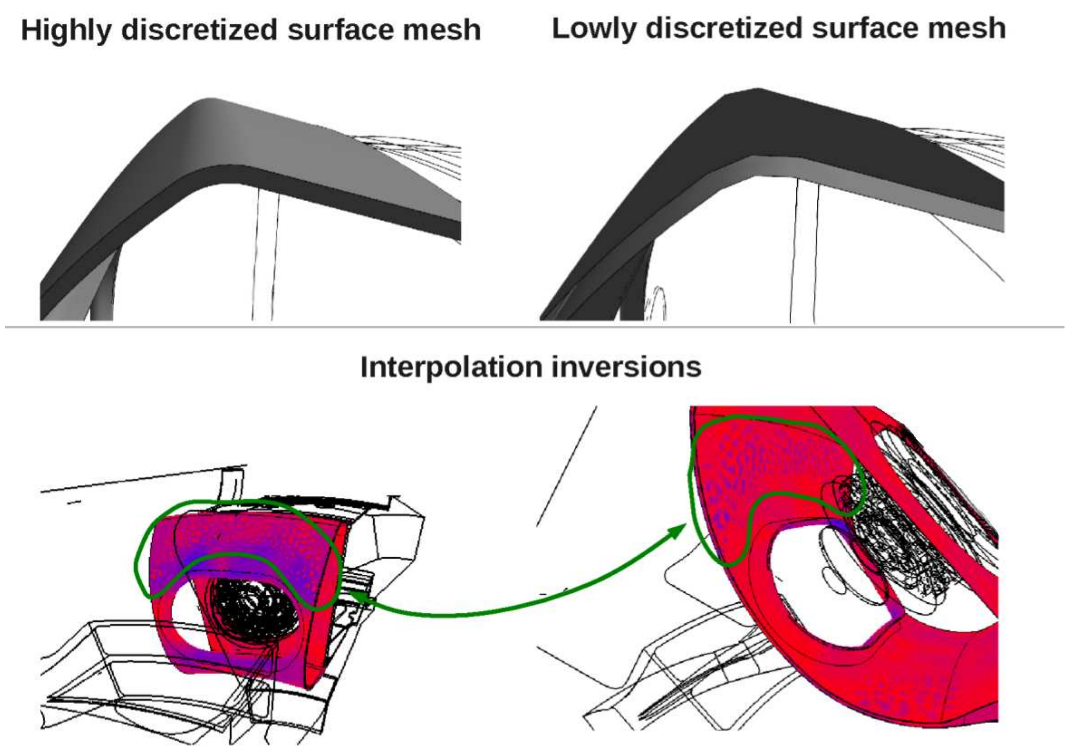

Highly discretized surface mesh
Lowly discretized surface mesh
Interpolation inversions

Figure 12.3: Interpolation inversion example

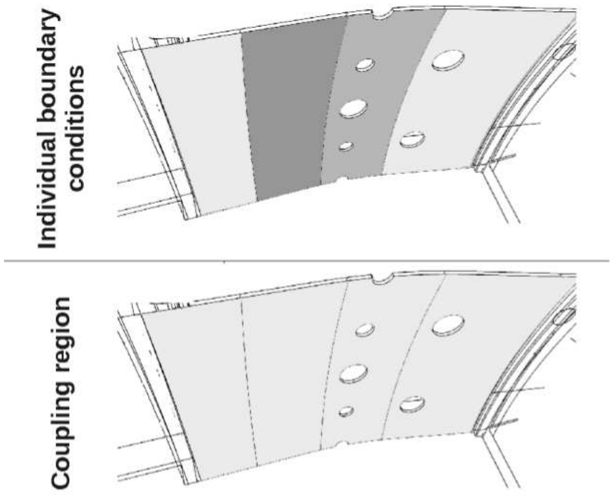


Figure 12.4: Example of the definition of a coupling region defined by the union of boundary condition regions

The identification of the different surfaces which define a coupling region is a complex task: in the aeronautical burner configuration case considered the fluid mesh is divided into 70 boundary surfaces and the solid mesh is divided into 50. In order to ease this work phase a graphical tool has been developed. It allows to identify visually the surfaces of each mesh and hence obtain the lists of boundary conditions within each mesh defining the coupling regions. Not only does this tool improve the user efficiency but it also adds analysis and verification of projections to check if the coupling region meshes are compatible. This tool is presented in A.3.

12.2 The initialization phase

The initialization phase uses the identified coupling regions and the mesh partitioning to calculate communication routes and linear transforms.

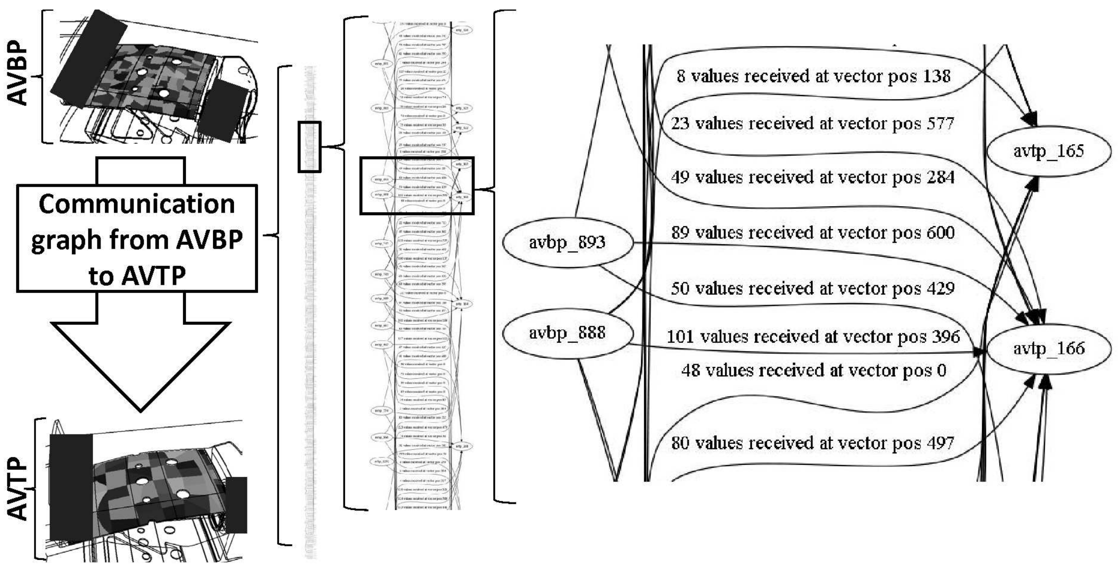

AVBP
Communication graph from AVBP to AVTP
AVTP
8 values received at vector pos 138
23 values received at vector pos 577
49 values received at vector pos 284
89 values received at vector pos 600
50 values received at vector pos 429
101 values received at vector pos 396
48 values received at vector pos 0
80 values received at vector pos 497
avbp_893
avbp_888
avtp_165
avtp_166

Figure 12.5: Communication graph example

In the context of parallel solvers, the global mesh is cut into smaller sub meshes called partitions. This partitioning mechanism is applied to each mesh independently, depending on the processor count assigned to each solver. Hence the coupling regions identified within the setup phase are also cut into smaller regions. In order to exchange data between the solvers these partitioned coupling regions have to be connected. Considering that neither the interface meshes, nor the coupling regions partitions coincide, each partitioned coupling region can be connected to several other coupling region partitions. The communication routes can be represented for each solver using two graphs: the first being the emission graph, the second the reception graph. Both graphs do share the same sources and destinations but they each store different data. The emission graph has to indicate to the emitter the selection of data it must gather, and to whom it should send the data. Because we are dealing with unstructured solvers, the data selections are necessarily irregular, hence they are implemented as node lists. The order of the nodes within these lists is also important because this ordering has to be shared with the receiver when it will decode the message which will be transferred between G^S and G^d. The receiver graph indicates to the receiver from who it should wait data, how much data and where it should place this data within its local reception space. An example of an emission graph is shown on Fig. 12.5. Methods to efficiently calculate these connections are described in 13.1.

After having established the communication routes, the transforms allowing to interpolate the data from mesh A to mesh B and vice versa have to be determined. It is important to understand that each vertex may have several possible sources (process ranks) for interpolation. Selecting or combining these sources is dependent on the interpolation method used. Determining these interpolation coefficients is a complex task which is described in 13.2.

12.3 The runtime phase

The runtime phase corresponds to the actual execution of the solvers. In this phase the coupling procedures should essentially focus on reducing the time spent in the coupling routines. The time spent in the coupling routines can be separated into two different parts:

- the time used by the coupling procedures for data processing (interpolation),
- the time spent during communication.

The time used by the coupling procedures for data processing depends on the implementation of the code, the type of interpolation, the machine type... For linear interpolation with precomputed coefficients this time remains extremely low. However the time spent during communication is more difficult to maintain low. In-fact this time is composed of two contributions: a wait time corresponding to the synchronization time between the two communicating processors and the actual communication time. Although it is possible to use communication schemes allowing to do some processing during the communication wait phase[1], it is generally not applicable to physics simulation (it would mean advancing a processor even if its input boundary conditions have not arrived). However reducing synchronization times is fundamental to be able to correctly take advantage of massively parallel machines and these synchronization times are highly dependent on the *load balancing*.

Load balancing in parallel solvers relying on space partitioning is based on the assumption that each processor applies the same operations every where in space. Hence load balancing is obtained by ensuring that the amount of geometry (elements, nodes...) is evenly distributed among the processors. In a coupled application each solver's processors apply different operations. Hence the amount of time for each solver may vary for each iteration. This difference in processing time results in potential wait periods at each synchronization between the solvers (communication). By recording for each process the instant when it starts and stops communicating it is possible to represent these activity and idle periods on Gant diagrams[2].

	AVBP	AVTP
N_{iter}	30	30
N_{proc}	128	128
Δt	1.5s	0.07s

Table 12.1: Ill balanced example parameters

An example of an ill balanced application is presented in Fig. 12.6. This example corresponds to a AVBP/AVTP coupled simulation on BlueGene[3]. AVBP is noted solver A and AVTP is noted solver B. In this example each solver is assigned 128 processors: $N^A_{procs} = N^B_{procs} = 128$ [4]. In this test case each solver executes $N^A_{iter} = N^B_{iter} = 30$ iterations between each communication and the iteration duration for AVBP and AVTP are respectively $\Delta t^A = 1.5s$, $\Delta t^B = 0.07s$. All the parameters defined here are summarized in Table 12.1. Even though all the processors in a solver do not communicate (all processors are not on the coupling interface), these synchronization waits end up blocking all the processors in a solver because of the internal communications within that solver.

The machine load L can be defined the ratio between the active time and the total time ($T_{Active} + T_{Wait}$):

$$L = \frac{T_{Active}}{T_{Active} + T_{Wait}} \tag{12.1}$$

[1]This is exploited in this application to overlap communication and interpolation when possible see section 13.3.

[2]This requires a common clock between the processors. The MPI specification states that MPI_Wtime provides this service.

[3]The test case used for this example is the same test case as the one used for the validation of the distributed geometrical searches in section13.4.

[4]On BlueGene the processor count per solver must be a multiple of a hardware defined value which depends of the machine.

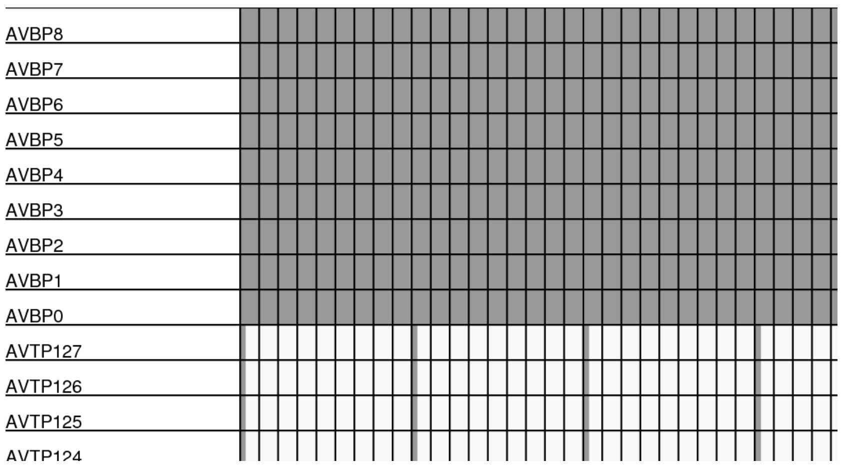


Figure 12.6: A portion of the Gant diagram of an ill balanced coupled application. The white areas represent processors waiting, the gray areas represent active processors. Each cell corresponds to $5s$

The times are counted as CPU time which means that the number of processors is considered. The total active is the sum of the active time for each solver:

$$T_{Active} = N^A_{iter} \Delta t^A N^A_{procs} + N^B_{iter} \Delta t^B N^B_{procs} \tag{12.2}$$

To express the wait time it is assumed that the solver having the less activity time waits the solver having the longest activity time (the solver having the longest activity time does not wait). The wait time for solver A can be modeled by:

$$T^A_{Wait} = max\left(0, N^B_{iter}\Delta t^B - N^A_{iter}\Delta t^A\right) N^A_{procs} \tag{12.3}$$

This expression computes a non zero wait time for solver A only if (and only if) solver A's activity time is shorter than solver B's activity time. Similarly for solver B one gets:

$$T^B_{Wait} = max\left(0, N^A_{iter}\Delta t^A - N^B_{iter}\Delta t^B\right) N^B_{procs} \tag{12.4}$$

The total wait time is the sum of the wait time for each solver (only one of the wait times can be non zero):

$$T_{Wait} = T^A_{Wait} + T^B_{Wait} \tag{12.5}$$

Applying this model to the current example we can calculate a load of $L = 52\%$. Meaning that due to an inappropriate load balance only 52% of the machine is efficiently used. Graphically the load L represents the ratio of the gray area over the total area on the Gant diagram.

	AVBP	AVTP
N_{iter}	5	15
N_{proc}	108	12
Δt	2.4s	0.6s

Table 12.2: Well balanced example parameters

This means that improving load balancing does not mean reducing the wait time for each processor in a blind way. The amount of processors for each solver should also be considered and in some cases sightly increasing the wait time for a solver may reduce the global wait time and hence improve the load factor. Graphically improving the machine load means reducing the white area on the Gant diagram.

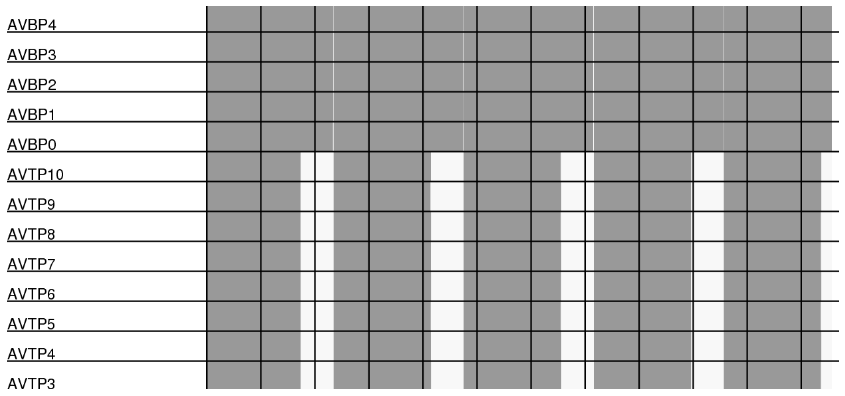


Figure 12.7: A portion of the Gant diagram of a well balanced coupled application. The white areas represent processors waiting, the gray areas represent active processors. Each cell corresponds to $5s$

To illustrate this an example of a well balanced application is introduced. Its parameters are summarized in Table 12.2 and a portion of its Gant diagram is shown on Fig. 12.7. This example corresponds to a coupled AVBP/AVTP application[5] executed on a CERFACS internal cluster HP-C7000 (AMD MagnyCours 2.2Ghz) (the processor split is chosen to match the hardware architecture, on this machine there is 12 cores per node). Using the above formula the load is 97.6%. To visualize the difference between the well and ill balanced applications the entire Gant diagrams (all the processors are included) are presented in Fig. 12.8.

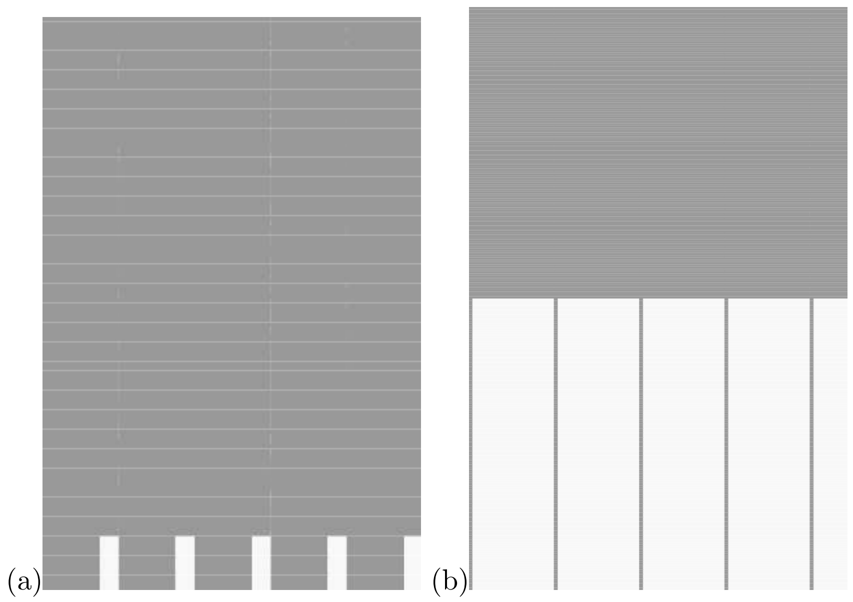


Figure 12.8: Gant diagram of: (a) the well balanced application, (b) the ill balanced application

This discussion has provided a simple method to understand and model the load balancing problem, provided that basic timings are known for each solver. This method should be easily extensible to more solvers. However no general solution has been proposed to solve the load balancing problem. For this application it is possible to use the iteration ratio within the limits shown in part II to modify the load balance. This is not always possible, notably if the coupled solvers are synchronized in physical time. In such cases a solution is to modify the processor split, however this is not possible on all architectures. Finally an interesting path would be to dynamically modify the work load (maybe by on the fly repartitionning). Although this seems to be a very complex task, it has already been developed in the YALES solver. But extending this feature to coupled applications may be even more

[5]The test case used is the target application presented in part V.

difficult.

Chapter 13

Algorithms and Methods implemented

The algorithms and methods implemented within this thesis dealing with parallel communications, transforms and distributed geometrical searches are presented in this chapter. These problems are tightly linked because it is the geometrical searches that provide the communication routes which are after used for the parallel communications and transforms. The geometrical search is done in two steps. First a coarse step which calculates for each processor involved in coupling a list of candidates which may share data with it. The second step is the actual calculation of the communication routes. Each possible candidate analyzes the data which may be exchanged and computes a response indicating the portion of the interface it actually shares. Finally the methods used to actually perform the data transfer and transformation are applied and are further detailed here.

13.1 The geometrical search, first step: the coarse routing step

Geometrical searches may be done using a variety of different methods: a first method would be to simply gather all the geometry on a single processor and use the search algorithms presented in chapter 10. While this method remains simple it has an essential drawback, it is clearly not scalable due to the data centralization.

Another simple method is to calculate each process's local geometry bounding box. The bounding boxes are then exchanged with all the other processes. A step that can be performed optimally using collective communication methods. Finally each processor calculates the list of processes with which it is susceptible to exchange data. This technique is rather straightforward however its drawback is that the bounding boxes, specially if they are axis aligned which is the case in this algorithm, give a very coarse representation of the geometry. The bounding box volumes may be far greater than the actual geometry, hence resulting in long lists of possible candidates for the calculation of the geometrical routes.

A more complex method has been developed, implemented and used within this thesis. This method maps objects or values to geometrical location in space using a distributed data structure. In the case of geometrical searching for routing calculations, the method maps lists of processors to geometrical locations. Since this mapping task only needs to be done once, the implementation done in this thesis is a static version of this method. However the data structures and algorithms a clearly adapted to dynamic mapping. So the method could be extended to geometrically map other problems such as particles for Lagrangian problems or moving meshes.

13.1.1 Routing using a distributed hash table

The routing method discussed in this section introduces a structure which allows to map data to space in a distributed way. Originally distributed object location has been developed to provide scalable and decentralized internet search services [113], with famous applications being peer-to-peer file transfer protocols. These methods have been described in detail by Rowstron et al. [98]. However these methods are generally used to locate objects using simple keys such as character strings (names, addresses, telephone numbers) not geometrical coordinates. In this thesis a method capable of locating objects from their coordinates on a distributed server is presented. This method relies on coupling two methods, namely the spatial hashing mechanism [115] and the distributed hash table [77]. These methods are described in the following paragraphs.

Hashing mechanisms

A hash table [51, 59, 87] (also referred as hash map) is a data structure implementing an associative array, Fig 13.1, between identifying values called keys and their associated values. The association couples $(key, value)$ are sorted and stored into a data table T.

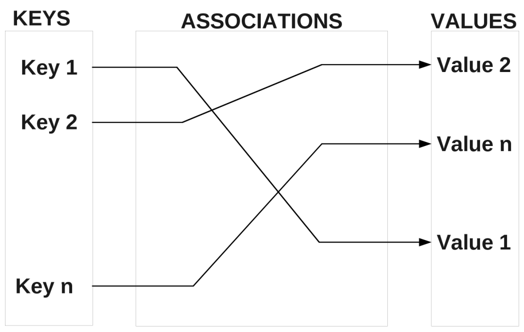


Figure 13.1: Associative array

The sorting mechanism uses a function, named hash function noted here h, to associate each identifying object k to an integer i named hash code, hence $h(k) \mapsto i$. This function is chosen in order to map as evenly as possible the input values over the output range, in other words every hash code has almost the same probability of being generated. Many hash functions are capable of treating variable size keys, they generally generate machine size integers, hence on most machines a hash function associates a 32-bit integer to a variable length sequence of bytes (which can be an array, a string, an object, ...). In this thesis the hash function used is a function called one-at-a-time published by Jenkins [51]. The hash code i is then used to generate an index in a data table T to store k .

The data table T is composed of $N_{hash\ size}$ variable size sets of elements noted $e_1, e_2, \ldots, e_n$ for each index $j \in 0 \ldots N_{hash\ size} - 1$. The sets of elements are called hash buckets (also hash slots or hash directories), $T_j = e_1, \ldots, e_n$ is the hash bucket j and it contains n elements: $e_1, \ldots, e_n$. A common implementation of these data tables is to use linked lists to allow variable size sets of elements. Generally the hash code can be any integer within the range $0..2^{Machine\ Word\ Size} - 1$ which is huge, it can not be directly used as an index for T. In most implementations the index to T is generated taking the remainder of the integer division of i by $N_{hash\ size}$, hence $j = i\ modulo\ N_{hash\ size}$ (where $a\ modulo\ q$ is the positive remainder r of $a = pq + r$).

When the value associated to the key k is requested, instead of comparing k to all the keys of each association, k is compared only to the keys which have the same hash code. Hence the search space is reduced to the hash bucket associated to k through the hashing mechanism. Since the hash function is chosen to be as uniform as possible, the associations are evenly distributed within the buckets. Therefore the look-up operation within a hash table containing N associations is performed

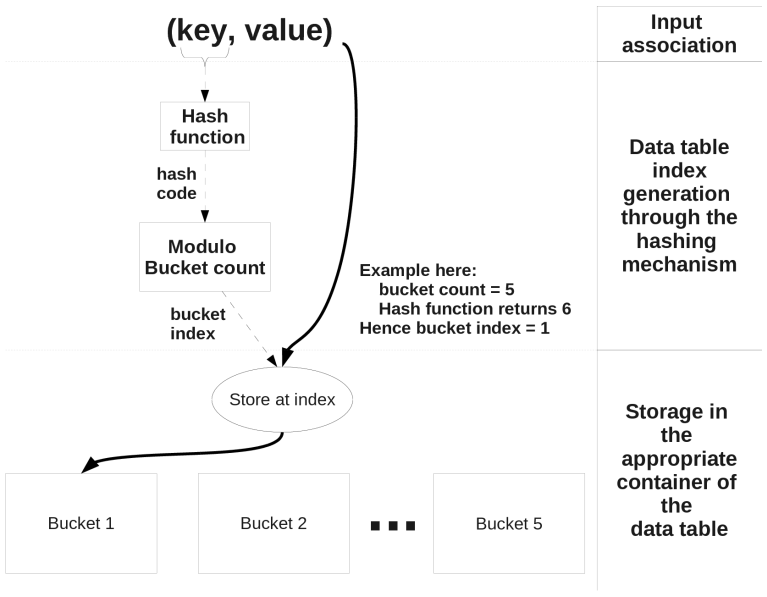

(key, value)
Input association
Hash function
hash code
Modulo Bucket count
bucket index
Example here: bucket count = 5 Hash function returns 6 Hence bucket index = 1
Data table index generation through the hashing mechanism
Store at index
Bucket 1
Bucket 2
Bucket 5
Storage in the appropriate container of the data table

Figure 13.2: Adding an association to a hash table

in $N/N_{hash\ size}$ tests in average, by contrast linear searching would require N tests. Hence choosing high $N_{hash\ size}$ can greatly speed up association look-ups. What is more, in many implementations the hash buckets are implemented using linked lists. Therefore a high $N_{hash\ size}$ only requires allocating $N_{hash\ size}$ list heads which is cheap (linked list heads are generally implemented using computer pointers).

In the special case where the hash table usage can be cut into two phases: at first the hash table is only used to add associations. Then only look-ups are performed to the hash table. The hash buckets can be implemented by simple arrays which can be sorted efficiently (using standard quick sort [48] or heap sort [19], both in average of $O(nlog_2(n))$ operations) allowing the look-ups to be performed using binary searches (complexity of $O(log_2(n))$).

Distributed Hash tables

A Distributed hash table [77] is a special hash table for which the hash buckets have been distributed over a network (Fig. 13.3). Such data structures allow to divide the memory load of the hash table over a cluster of computers. The load balancing of this method is ensured by the hash function's uniformity property.

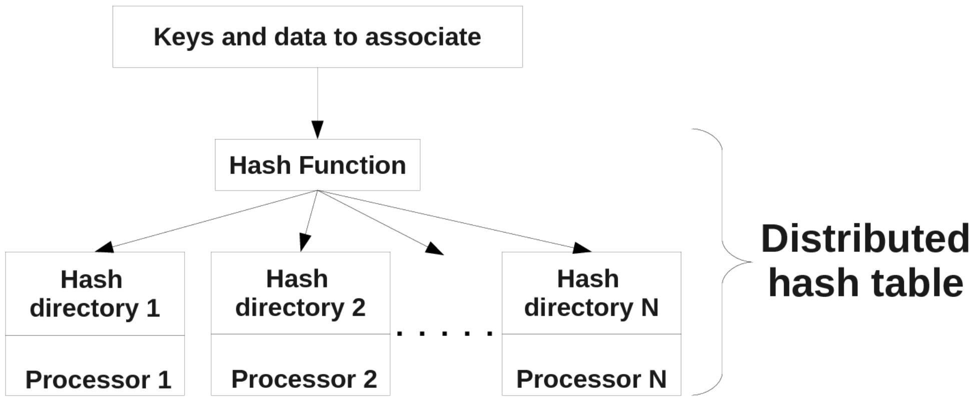


Figure 13.3: Distributed hashing

Spatial hashing

Spatial hashing, also called geometric hashing [121], is a mechanism relying on hash tables to associate spatial coordinates (x, y, z) to an object O. The idea is to generate from the spatial coordinates a deterministic identifying value which can then be used as a key k to store the association (k, O) in a hash table. The method demonstrated by Teschner [115] is to project the (x, y, z) coordinates onto a uniform grid, Fig 13.4. The grid coordinates are represented by triplets of integers (p, q, r), $\Delta x, \Delta y, \Delta z$ are the grid steps in each direction. The projection p from $(x, y, z) \mapsto (p, q, r)$ is defined by:

$$p((x, y, z)) \mapsto \begin{cases} p = E\left[\frac{x}{\Delta x}\right] \\ q = E\left[\frac{y}{\Delta y}\right] \\ r = E\left[\frac{z}{\Delta z}\right] \end{cases} \tag{13.1}$$

where $E[u]$ represents the integer part of u.

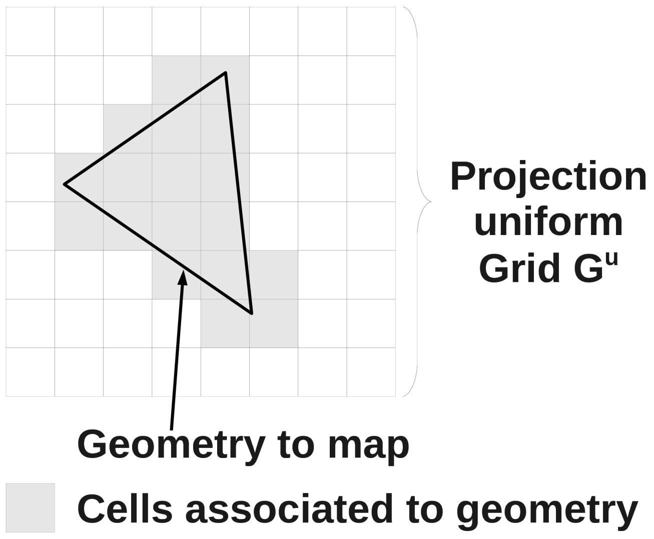


Figure 13.4: Geometry projection to cell list

Using Spatial hashing with distributed hash tables to compute communication routes

The calculation of the communication routes between the solvers A and B is divided into two phases: at first the solver A acts as a server and B as a client, then the roles are inverted. The server processes are noted P_i^S, the client processes are noted P_i^C. The server solver A maps its interface geometry into the DHT which is localized on the server solver processors. Hence the server solver processors have a dual role since they both act as clients and servers of the DHT, this is typical of peer-to-peer networking.

The DHT is initialized on the server processes with a bucket count, i.e. the count of hash buckets which are to be distributed over the server processors. A server process can only handle 1 hash bucket, therefore the hash index generated can be used to identify a process. In the implementation it is possible to choose the amount of processors handling a hash bucket, in practice to obtain the maximal data distribution all servers processors are used. To simplify the discussion it will be considered that all the server processes manage a hash bucket.

The common way to implement peer-to-peer protocols [68] is to use concurrent threads, with at least one thread for the server task. However in this case the code is targeted to run on hardware which may not be thread compatible and with only basic MPI support.

Therefore the communication patterns with the DHT are broken down into two phases:

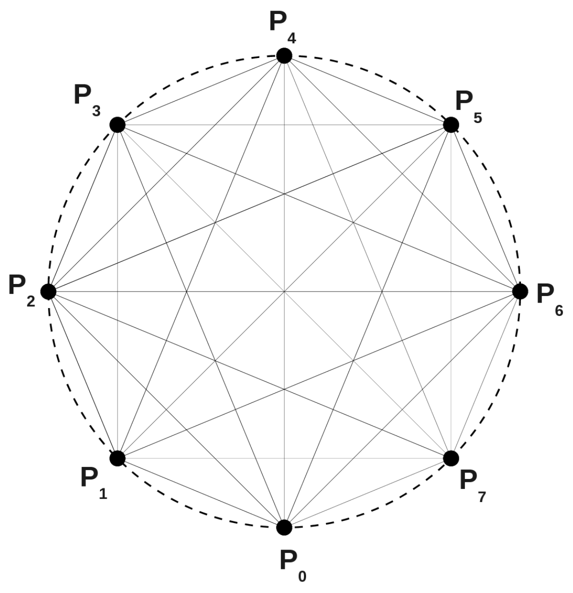


Figure 13.5: Peer to peer communication step 1

1 In the first phase all the peers send to their peers the amount of data they are going to send, Fig 13.5. When no data needs to be exchanged the communication size sent is zero.

2 Each peer allocates the memory for the incoming messages and initiates the emission and reception of the non empty messages, Fig 13.5. Each peer then waits for the messaging process to end.

The communications rely on all-to-all communication[1] for the first phase and on non blocking communication for the second. This is not performed by only one all-to-all communication because MPI specifies that the all-to-all primitive operates on fixed size messages. Therefore to create a unstructured form of all-to-all primitive the MPI all-to-all is used to transfer only one integer, i.e. the size of the message[2] .

To map the server's geometry each process of the server P_i^S performs the following actions:

$S1$ P_i^S's interface geometry is projected onto the uniform grid G^u yielding a list of cell coordinates $Cs_n^i = (r_n^i, q_n^i, p_n^i)$,

$S2$ P_i^S then uses the DHT through the unstructured all-to-all communication protocol to mark the association between Cs_n^i and the processor rank i.

At the end of this phase the server's geometry is mapped in the DHT. Now the server listens for incoming messages from the client server. The unstructured all-to-all communication protocol is still used but in this case no messages are received or sent between the clients and the servers.

Concurrently the clients processes perform the following actions:

[1]This type of communication can be greatly optimized by the MPI library even on unstructured communicators using torus communication algorithms.

[2]It may seem possible to implement this protocol using a simple MPI_AlltoAllv call but the problem is that MPI_AlltoAllv requires knowing *a priori* the amount of data it has to receive. Therefore the first MPI_AlltoAll transferring the message size is necessary. A solution using an MPI_AlltoAll to transfer the message size followed by an MPI_AlltoAllv may also be considered instead of using direct non-blocking point-to-point communications to transfer the messages. This last solution has not yet been implemented nor tested.

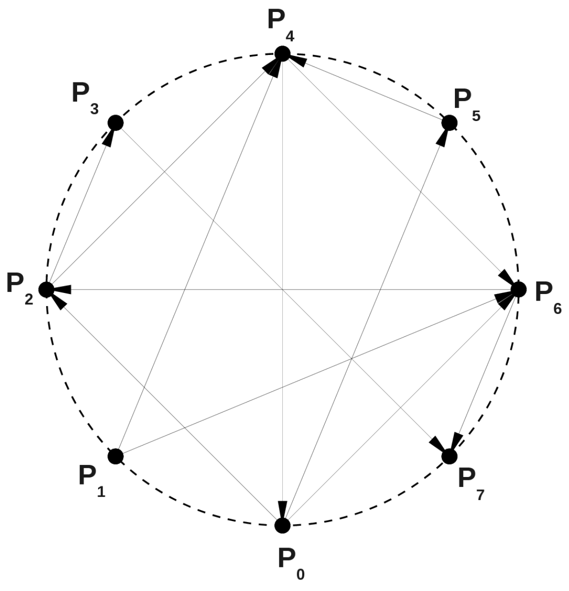


Figure 13.6: Peer to peer communication step 2

$C1$ Each process P_j^C of the client projects its interface geometry onto the uniform grid G^u yielding another cell list $Cc_n^j = (r_n^j, q_n^j, p_n^j)$.

$C2$ Each process P_j^C interrogates the DHT to look-up the process ranks associated to the cell list $Cc_n^j = (r_n^j, q_n^j, p_n^j)$. This phase requires the server to be in listen state, i.e. it needs phase $S2$ to end. The DHT replies a list of couples of cells and process ranks $((r_n^j, q_n^j, p_n^j), Rank_n)$

At this stage the client processors have a list of possible candidates with which they may communicate during the runtime phase.

Each possible candidate is also associated to a list of cells therefore to a certain portion of geometry. Hence more precise information on which process ranks may share which geometrical parts is available. Also the size of the cells of the uniform grid G^u drives the precision of this method and can therefore be adapted.

13.2 The geometrical search, second step: building the communication graph and the partitioned interpolation matrices

After the coarse routing step, each client process has a list of candidate server processes and for each candidate a set of cells associated to it. The client's interface grid G^c is split for each candidate process S_i using the cell set $Cells_{S_i}$ associated to it. Hence a set of grids G^{S_i} are calculated by intersecting the client's interface grid and the candidate S_i cell set: $G^{S_i} = G^c \cap Cells_{S_i}$. The grids G^{S_i} are then sent to the candidate processes S_i using a the unstructured all-to-all protocol described earlier.

Consequently the server processes receive a set of grids, one for each client process which may be related to it. Using the methods and algorithms presented in chapters B.3.3 and 10 interpolation matrices are computed. An additional *quality of interpolation* scalar $Q \geq 0$ is also calculated for each vertex of the destination (client's) grid. The value is interpolation type dependent:

- For nearest neighbor interpolation, this scalar is defined by the distance between the source and target vertices,

- for linear interpolation, the scalar is defined by the distance between the orthogonal projection of the target vertex onto the source element,
- for linear conservative interpolation, the scalar is defined by the ratio of intersected area from the element on element projection.

This value is used to combine or discriminate the sources for each vertex of the client's grid: each vertex of the client may be sent to several server processes to evaluate interpolation coefficients (several processors may share the same cells of the uniform grid G^u, the coarser G^u is the more this happens). How the client actually deals with these cases is interpolation type dependent.

Hence each client receives for each G^{S_i} a response consisting in an interpolation matrix T_{ij} and a quality of interpolation scalar for each vertex (or row of the matrix T_{ij}) Q_i. Therefore each vertex of the client's grid G^c obtains a set of interpolation coefficients assessed by a scalar Q. Only the nearest neighbor and linear interpolation methods have been implemented yet. For these two interpolation methods, the lower Q is the closer the source data is from the destination point, therefore the interpolation coefficients of the source having the lowest Q are selected. In the linear interpolation implementation interpolation coefficients are always defined even if no element containing the projection of the destination vertex is found. In such a case the implementation falls back to nearest neighbor interpolation.

Client's vertex	Lists of (source_rank,source_vertex_index,coefficient)
0	(43, 76, 1.0)
1	(44, 21, 1.0)
...	...

(a)

Client's vertex	Lists of (source_rank,source_vertex_index,coefficient)
1	(56, 1045, 0.2), (56, 1047, 0.5), (56, 1047, 0.3)
2	(56, 2021, 0.1), (56, 2000, 0.7), (56, 2036, 0.2)
...	...

(b)

Client's vertex	Lists of (source_rank,source_vertex_index,coefficient)
1	(56, 2021, 0.05), (56, 2000, 0.3), (56, 2036, 0.25), (56, 2027, 0.1), (56, 2040, 0.05), (56, 2037, 0.25)
2	(58, 145, 0.2), (57, 89, 0.4), (56, 2021, 0.2), (58, 149, 0.2),
...	...

(c)

Figure 13.7: Interpolation data sources after data source selection: (a) for the nearest neighbor case, (b) for the linear interpolation case, (c) for the linear conservative interpolation case

For conservative interpolation, the process would be more complex because some elements may be partially projected onto the interface geometry of several processors in which case sums of coefficients should be considered. This may be more complex than imagined here, it may require more testing and proofing and has not yet been implemented.

Therefore each vertex of the client's interface has now a set of interpolation coefficients, each coefficient can be associated to a vertex of the interface of one of the server processors.

This is represented on Fig 13.7:

- for the nearest neighbor interpolation the data for each vertex can only come from the same source, only one vertex is associated to the destination vertex.
- for the linear interpolation the data for each vertex can only come from the same source, however in the normal interpolation case with triangles 3 coefficients are given.
- for the linear conservative the data may come from several sources.

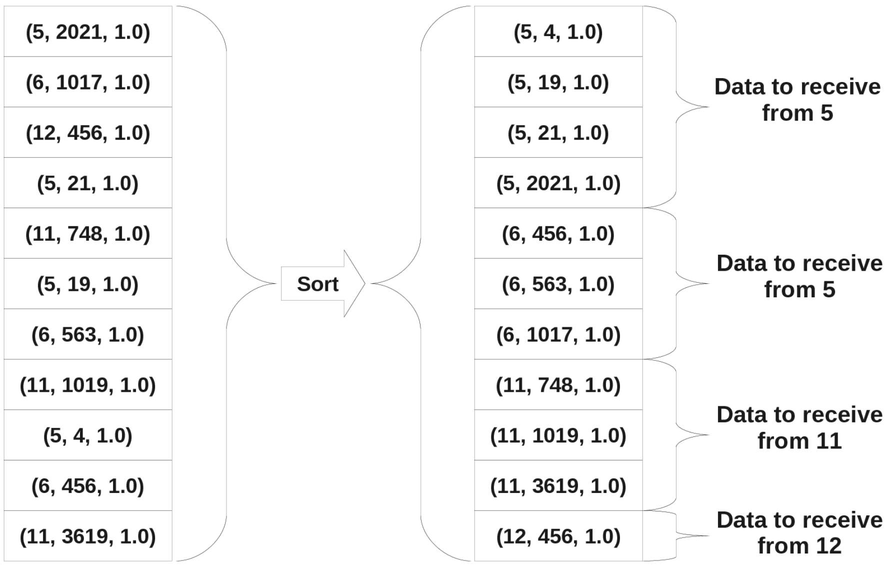


Figure 13.8: Construction of the reception vector by sorting the data sources

To build the data structures used for actual communication this information must become communication friendly. Hence the triplets $(source\_rank, source\_vertex\_index, coefficient)$ are sorted using the $source\_rank$ as primary key and the source vertex index as secondary key [3], Fig 13.8.

By analyzing the sorted triplets it is possible to determine the amount of unique vertex data to receive from each processor R_{S_i}. It is important to keep in mind that the sorted list may contain several times the same location $(source\_rank, source\_vertex\_index)$ (if several vertices of the client depend on this same source vertex), in such a case only one location should be considered.

Once this has been determined the reception vector can be built:

- the reception vector's size is the sum of the amount of data to receive from each individual source: $\sum_{S_i} R_{S_i}$.
- the offset of S_i's data in the reception vector is built by summing successively R_{S_i}.

Therefore each row in the reception vector corresponds to a unique vertex location $(source\_rank, source\_vertex\_index)$, hence it is possible to generate a list for each source S_i of vertex indices using the $source\_vertex\_index$ value. Also the emission list for $source\_rank$ can be built using the $source\_vertex\_index$ value for each unique location. This list is then sent to each

[3] The list of $(source\_rank, source\_vertex\_index, coefficient)$ is sorted using the $source\_rank$ if two triplets have the same $source\_rank$ then they are sorted using $source\_vertex\_index$

source S_i therefore the source knows which data should be sent to each receiver, of course the order is important.

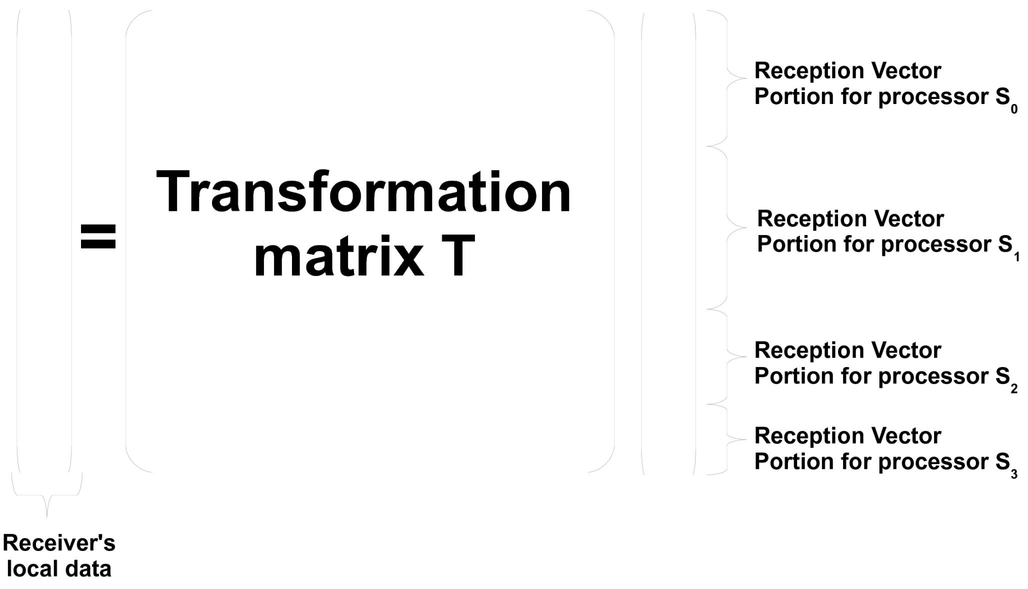

=
Transformation matrix T
Reception Vector Portion for processor S_0
Reception Vector Portion for processor S_1
Reception Vector Portion for processor S_2
Reception Vector Portion for processor S_3
Receiver's local data

Figure 13.9: Reception vector and matrix

The final transformation matrix can be built using the interpolation matrix, the interpolation coefficients for each destination grid vertex considering the index remapping issued from the reception vector construction (sorting, and double locations eliminations). To maintain efficiency the remapping process uses hash tables. Figure 13.9 shows the data structures obtained.

For efficient storage and faster matrix vector operation, the interpolation matrix is stored in a form similar to the compressed sparse row format (CSR). However instead of storing the row pointer the row width is stored, the matrix is stored in 3 arrays described in Table 13.1.

Array	Dimensions	Description
Row_Width_i	$0..row\_count$	the number of non zero values in row i
Col_Index_i	$0..(\sum_{i=1}^{row\_count} Row\_Width_i - 1)$	the column index of each non zero value scanned from left to right and top to bottom
$Value_i$	$0..(\sum_{i=1}^{row\_count} Row\_Width_i - 1)$	each non zero value scanned from left to right and top to bottom

Table 13.1: Storage of the interpolation matrix

13.3 Direct Communication using Interpolation overlap

Generally each interface processor of a solver has several counterparts, and in a physical simulation one must transfer several variables, all the communications being implemented using non blocking schemes. The communication/interpolation process allows to overlap communications and interpolations if possible, Fig 13.10. At first each processor of the solver initiates all its sends and all its receives. Then it waits for all its pending communications to end. When data messages are received, it checks if it has received an entire physical variable from its counterparts (all the partitions of the field of a specific variable). Once at least an entire field is received, the linear transform is applied

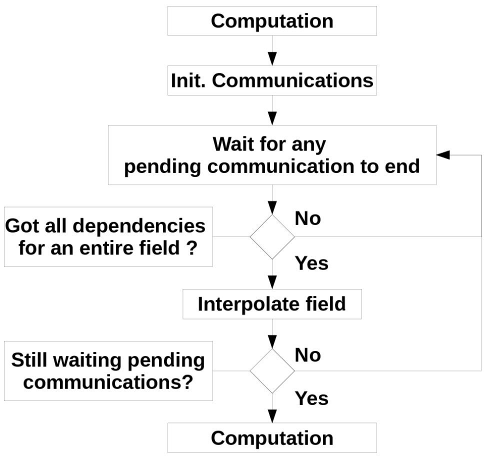

Computation
Init. Communications
Wait for any pending communication to end
Got all dependencies for an entire field ?
No
Yes
Interpolate field
Still waiting pending communications?
No
Yes
Computation

Figure 13.10: Communication algorithm with overlapped interpolation

to that field and the output is directly written inside the solvers buffers (which have been registered to the coupling layer before). All this transformation is executed while data messages can still be received.

```
$k \leftarrow 0$ for $i = 0$ to $row\_count - 1$ do
    $w \leftarrow Row\_Width[i]$;
    $s \leftarrow 0$;
    for $j = 0$ to $w - 1$ do
        $s \leftarrow s + Value_k Reception\_Vector[Col\_Index[k]]$
    end
    $Interpolated\_Data[i] \leftarrow s$
end
```
Algorithm 8: Interpolation computation

13.4 Tests of the proposed method

The presented methods have been implemented using object oriented modular C++ code. The code has been heavily instrumented with self checks, notably assertions [97, 42] have been used to test all the key hypothesis, pre-conditions and post-conditions of the class methods. Each key algorithmic feature has been developed using a specific class, allowing for basic testing via unitary tests. Of course presenting in detail these technical tests process is of little interest for this thesis. Rather only fundamental key results are presented in this section. The first test evaluates the scalability of the DHT implementation. Indeed the DHT implementation is the key feature allowing the communication route computations in a distributed way. Then a complete test including the initialization phase and the runtime phase on a complex geometrical configuration is presented.

Scalability of the route computation algorithm

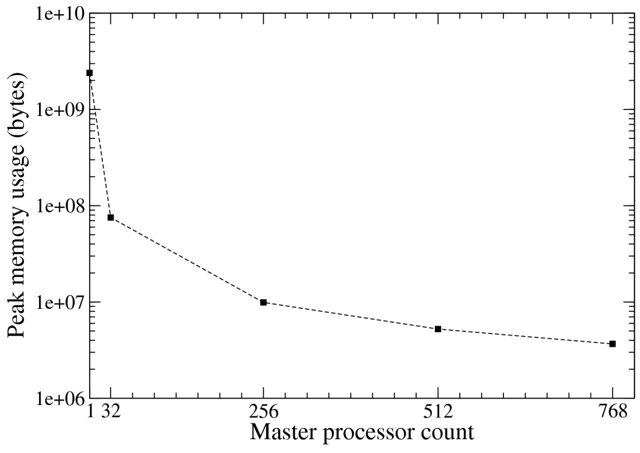


Figure 13.11: Peak memory usage on server solver

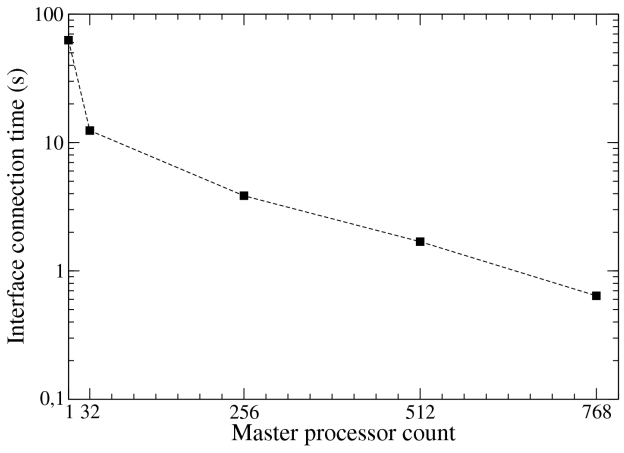


Figure 13.12: Interface connection time

In this first test, a key functionality is checked: the distributed hash table mechanism. The test is carried out in the following way:

- Two pseudo solvers are started, one will act as a server the other one as a client. The solvers do not have an equal processor count.
- A global set of nodes is distributed throughout each solver's processors.
- The server solver maps it's nodes using the distributed hash table.
- Each processor of the client solver asks the server through the distributed hash table the location(s) of its local nodes (several locations are possible).

The algorithm has been first developed and tested on small amounts of nodes. Then the test presented has been carried out using a 1024 core partition on a SGI ALTIX ICE super computer (JADE): the 1024 cores are partitioned into a server solver with 768 computing cores and a client with 256 computing cores. The server solvers manages 100 million nodes and maps them using its DHT. The client solver interrogates the server solver's DHT to obtain the location of the 100 million nodes. This test is performed for different DHT master processor counts.

As explained before, scalability must be analyzed from a general point of view, that is not only considering *CPU* stress, but also *memory* stress. This is why the results presented in this discussion include algorithm timings and memory footprint. These values are obtained by instrumenting the code, that is adding timers at strategic points and instrumenting the dynamic allocator.

The local peak memory consumption used by the method for different master processor count, Fig. 13.11, shows that the main goal has been achieved: by increasing the number of master processors, the memory load can be distributed over the solver's processors. An added benefit seen on Fig. 13.12 is a global application speed that is also increasing with the number of master processors. This is explained by the reduction of the quantity of data that each master processor has to process.

Further testing: application on an industrial geometry

The entire interface processor adjacency computation and interpolation process has been tested on a complex geometrical configuration. The geometrical configuration is a simpler version of the aeronautical burner combustion chamber used for different calculations (Projet ANR CIS 2007- SIMTUR), Fig. 13.14. This is simply a preliminary test used to validate the computational methods, it has not been converged. The fluid mesh is composed of a 9.2 million tetrahedra, the solid mesh of 6.7 million tetrahedra. The test consists in coupling the flame tube liners, involving 170 thousand nodes on the fluid side and 226 thousand nodes on the solid side. Since the tests on the DHT have shown that the most effective configuration is when all processors are master processors, the tests have been executed in pure *peer-to-peer* mode. These tests have also been carried out on a SGI ALTIX ICE super computer (JADE). The results have been summarized in Table 13.2.

Processors		Memory peak		Connection wall clock time
AVBP	AVTP	AVBP	AVTP	
128	32	12Mb	26Mb	15.5s
256	64	9Mb	17Mb	11s
512	128	7Mb	15Mb	9s

Table 13.2: Full connection process test results

The primary objective which is maintaining a reasonable memory consumption has been clearly obtained. Note also that the connection timings remain relatively low and do decrease with processor count. These values do not decrease linearly with processor count: due to partitioning the processors handling the boundaries do not scale linearly with processor count.

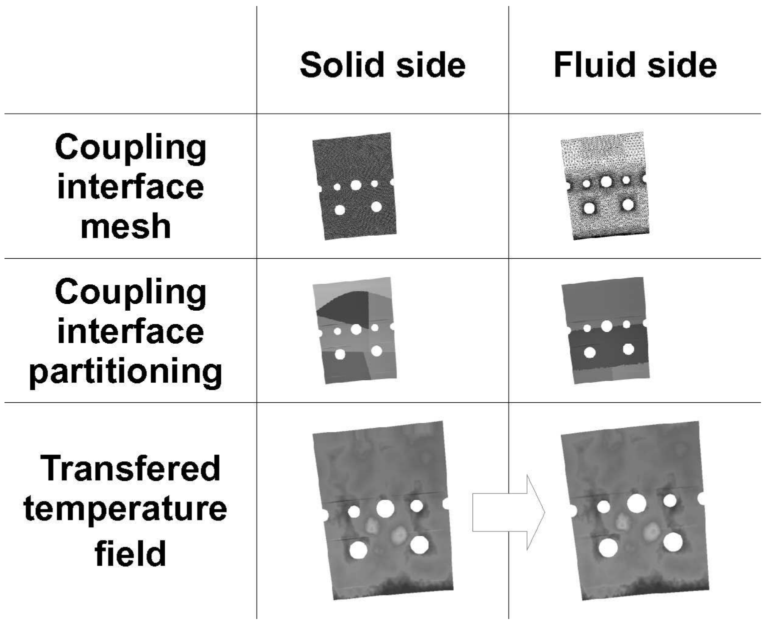


Figure 13.13: Illustration of the transfer process on the flame tube face of the external combustion liner wall

Runtime tests have also been carried out, to check the implementation of the communication and interpolation methods and to assess the efficiency of the transfer method. The extensive tests of the interpolation methods implementation have already been presented in part III. Figure 13.13 shows a data transfer on a flame tube liner wall using linear interpolation. This result shows qualitatively that the communication and interpolation routines are correctly implemented.

As for the transfer efficiency, the transfer process has been timed on the fluid and solid sides. Artificial synchronization between the solvers has been added to ensure that the synchronization time due to load balancing problems is not measured. Hence the actual transfer time, that is to say, the communication time and the interpolation time, measured is of the order of $10^{-4}s$. This is clearly consistent with the low latency characteristic of the direct communication scheme shown in section 11.3. What is more the iteration time for each code being in these case of the order of $1s$, the transfer timings can be neglected.

Figure 13.14: An instantaneous view of the coupled preliminary test application

mesh connection methods must also be stated here, notably the methodology developed in the PUNDIT program [110].

Conclusion

In this part the issues inherent to HPC have been explained. The solution as explained by Ahmdahl's law[3] is to reduce the amount of sequential code. For code coupling this means implementing parallel routines for data exchange and interpolation. Also new difficulties rise as massively parallel computer architectures tend to change: new machines tend to use less powerful processors with small memory banks but in greater numbers. This implies that algorithms should not only be thought to run as fast as possible but also within a small memory footprint which is a clear difficulty for sequential parts of the code which have to treat the entire geometry. The geometrical search algorithms used to determine the interpolation coefficients may suffer from these difficulties. A solution allowing to perform these geometrical searches in a purely distributed way has been proposed, explained and tested.

Also problems specific to coupling complex geometry have been shown and a simple solution consisting in dividing the coupling interface mesh into sub-surfaces has been proposed. While this solution allows better control of the geometrical projections, it requires human intervention. A tool capable of simplifying this task is presented in A.3.

Now that all the methods required to solve the technical problems of conjugate heat transfer relying on LES have been explained the next step is to consider the full application.

Part V

Application to an aeronautical burner

Table of Contents

Nomenclature

a	ratio between the number of iterations within the solid and the fluid
s	Curvilinear coordinate used as abscissa or the profiles
t	time (s)
T_j^i	the temperature at node j of the stored temperature field i
x'	Flame tube axial coordinate
$Y+$	boundary layer distance evaluated by the wall model

In this final part a coupled application is built relying on the coupling methodology, interpolation methods and computational methods developed in parts II, III and IV. The configuration is the aeronautical burner which is presented in part I. First of all, the setup of the coupled application is presented. Then this application is used to perform scalability tests of the coupling methodology, hence supplying a complete validation of the efficiency of the work presented in part IV. Then results are discussed and analyzed, notably by comparisons between uncoupled and coupled results.

Chapter 14

Coupled application

Due to confidential material in this chapter, portions of the original text of this chapter have been stripped and/or modified. The non confidential information has been left, essentially the information relative to the computational aspect of code coupling, e.g. the number of vertices/elements per coupling region, the scalability of the algorithms, etc. Of course this information is higly dependent on the context but the main purpose of the original chapter was to show the complexity of the coupled application developped and its scalability.

14.1 Coupled application setup

The coupled application is built by associating a fluid mesh and a solid mesh. The fluid surface mesh is divided into 70 sub surfaces and the solid surface is divided into 50 sub surfaces. The coupling surfaces are built by joining sub surfaces in each configuration's sub-surface. This is a complex task because the corresponding coupling surfaces in each solver must match in order to transfer correctly the data fields. A solution to this problem has been proposed in this thesis: a graphical tool has been developed capable of displaying the different meshes, their sub surfaces and of building coupling regions interactively. This tool is presented in appendix A.3.

The vertices count, element count and processors used for each coupling region are reported in Tables 14.1 and 14.2. It is interesting to note that the solid coupling regions contain more elements than the fluid coupling regions. As explained in part I this is due to the thin walls in the solid domain and that for industrial reasons (at least 4 elements should be in each wall). It is clear that this situation could be better dealt with using elements more adapted to high anisotropy such as hexahedrons. But keeping in mind that the thermal solver is at least an order of magnitude faster than the fluid solver and that the goal of this application is also to test these computational methods on complex geometrical problems, a high vertex count is acceptable as long as it does not penalize severely the application. This is also interesting for flux conservation: the linear interpolation method was used for this computation (the linear conservative method is not yet been implemented for parallel computations), interpolating from a coarse to a fine mesh is therefore preferable. The flux conservation has been measured (by on the fly inline diagnostics) for the different region groups, Table 14.3, the average flux conservation error being 0.12%, meaning that the flux is correctly preserved from the fluid to the solid computation. Another interesting feature is that each processor can be used for several coupling regions and that the ratio of processors involved in the coupling process is clearly different between the solvers: 33% of AVBP processors handle at least a coupling region while 98% of AVTP processors do.

Region Group	Region	Vertex count	Triangle count	Processors concerned (*)
Snout	St1	30512	52638	61
	St2	27913	48832	64
	St3	109761	207319	64
	St4	116859	221156	61
Internal support	Is1	13347	24065	29
	Is2	2612	3958	16
	Is3	16393	29416	32
External support	Es1	10591	18759	21
	Es2	2279	3484	12
	Es3	7814	13903	12
Flame tube	Ft1	118296	222344	65
	Ft2	118290	222884	64
	Ft3	78645	146519	53
	Ft4	80275	150066	52
Total		733587	1365343	250

Table 14.1: Solid region information. (*) A processor can manage several coupling regions.)

Group	Region	Vertex count	Triangle count	Processors concerned (*)
Snout	St1	19607	30305	157
	St2	9817	15096	74
	St3	8140	12473	84
	St4	8321	12913	77
Internal support	Is1	1193	1677	28
	Is2	267	236	16
	Is3	3455	5749	25
External support	Es1	1451	1901	35
	Es2	255	237	16
	Es3	716	882	22
Flame tube	Ft1	32726	54686	163
	Ft2	31245	52013	155
	Ft3	20326	33474	113
	Ft4	22042	36613	124
Total		159561	258255	588

Table 14.2: Fluid region information. (*) A processor can manage several coupling regions.

Region group	flux conservation error
Snout	0.13%
Internal support	0.05%
External support	0.13%
Flame tube	0.16%

Table 14.3: Flux conservation for the different region groups.

Computation details

Apart from the coupling regions, the boundary conditions in both solvers remained unchanged. The fluid field was initialized using the computation presented in chapter 4. The solid field was initialized using the same start point than the non coupled simulation presented in chapter 5. Off course it would have been a better choice to start it from the converged uncoupled solution, convergence would have certainly been faster.

As for the variables transferred between the solvers, at first the solid solver transferred its border temperature, and the fluid solver transferred its convective temperature, heat coefficient and heat flux. During the first tests different coupling methods have been tried out: using the fluid convective temperature and heat coefficient or using directly the fluid heat flux. Due to the numerical construction of the heat coefficient in AVBP wall models, it seemed safer numerically to use the fluid heat flux. Indeed the heat coefficient, H_c, is computed by dividing the heat flux, ϕ, by a temperature difference between the fluid convective temperature, T_c, and a reference temperature, T_{ref}, which is set to the border temperature (which is the solid temperature at the last coupling iteration), Eq (14.1).

$$H_c = \frac{\phi}{T_c - T_{ref}} \tag{14.1}$$

The fluid convective temperature is computed by averaging the temperatures of the cells connected to the first off-wall node, Fig. 14.1. This procedure can generate serious numerical problems when the convective temperature and the reference temperature are very close, H_c can become extremely large which can destabilize the conduction solver. This problem is specific to LES, in RANS such problems are less likely to happen since such situations clearly happen in transient phases and are very unlikely to appear for a stationary converged solution. Therefore the heat flux ϕ coupling was preferred to

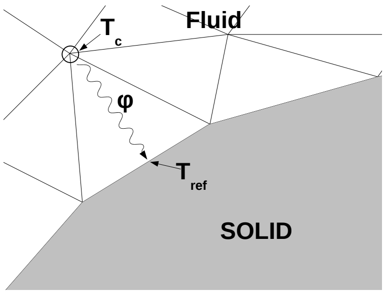


Figure 14.1: Wall heat flux

the H_c, T_c formulation. Hence the three variable transfers were only done during the preliminary and scalability tests, during the rest of the simulation only the heat flux ϕ was transferred from the fluid solver to the solid solver.

14.2 Application scalability assessment

The scalability tests have been carried out on an IBM BlueGene/P machine[1] allowing to perform runs with thousands of cores. This type of machine has been designed for massively parallel applications, it allows access to thousands of relatively small cores: each core runs at 850Mhz and has access to less than 512Mb of ram when all cores on a compute node are used. This architecture is presented in more details in chapter 11. These machines are adapted to extremely highly scalable kernels making them difficult to use for complex versatile solvers. Hence very few examples of code coupling have already been attempted on such machines.

Another difficulty on the BlueGene/P architecture is that even if MPMD is possible, load balancing is made very difficult because each process core count must be a multiple of a hardware defined constant (it corresponds to the number of compute nodes per IO node which is a specific characteristic of the machine). On the machine used (Babel) the value is 256. Hence load balancing for this application is almost impossible: the thermal solver runs much faster than the LES solver and we have to assign at least 256 cores to it.

The objective of these tests is to assess the impact of the coupling procedure on the aggregate application's performance. As said earlier, on this machine proper load balancing is not achievable: the thermal solver has too many processors compared to the LES solver. The global application's performance is hence driven by the LES solver, that is to say increasing or decreasing the LES solver's performance changes the coupled application's performance. Hence only the LES scalability curve is relevant in this test. The effect of the coupling procedures on the LES solver's performance is measured by modifying the LES iteration count between two coupling iterations.

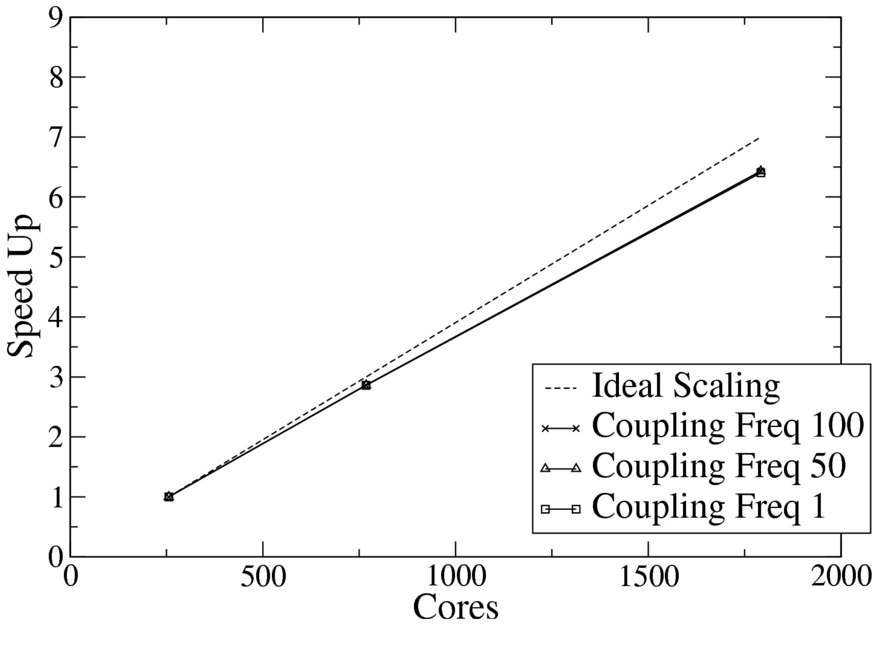


Figure 14.2: LES solver scalability curve for different coupling frequencies. Coupling Freq. 50 means that 50 LES iterations are executed between two coupling updates. Coupling Freq. 1 means that a couling update is done at each LES iteration.

The scalability curves, Fig. 14.2 and 14.3, and the timings in Table 14.4, show that the inter-solver communication has no noticeable impact on the LES solver's performance. The non ideal scalability of AVBP for this case can be explained by the use of a relatively small mesh only 15M Cells for 2048

[1]The machine used is Babel, IDRIS BlueGene/P system.

cores and the use of a less optimal partitioning algorithm: RIB [109], according to Gourdain [43] using METIS [56] should provide better scalability.

To understand the amount of data transferred between the solvers it is important to state that at each coupling iteration a complete information exchange, with interpolation, in both directions for every processor on the coupling interface is executed: 3 variables are transferred from the LES to the conduction solver (convective temperature, heat coefficient and heat flux), one from the conduction solver to the LES computation (border temperature). Using 2048 cores, the longest transfer took approximately $2.96ms$ which represents less than 0.5% of a LES iteration, 660ms, for that configuration on that machine. Also the actual average transfer time measured was $0.17ms$ and 86% of the transfers are finished in less than $0.4ms$. These statistics do not include the processors which are not involved in coupling. Hence 260 thousand triangles are interpolated onto 1.3 million triangles and vice versa three times in less than $3ms$.

Considering that in a typical application it is not required to synchronize the solvers at every time step the actual price of the coupling procedure is even lower, e.g. on this case coupling every 10 LES iterations means that the coupling procedures account for 0.05% of the time spent in the simulation. These results show that the methodologies built throughout this thesis work and that multiphysics is possible using the MPMD paradigm efficiently on massively parallel machines.

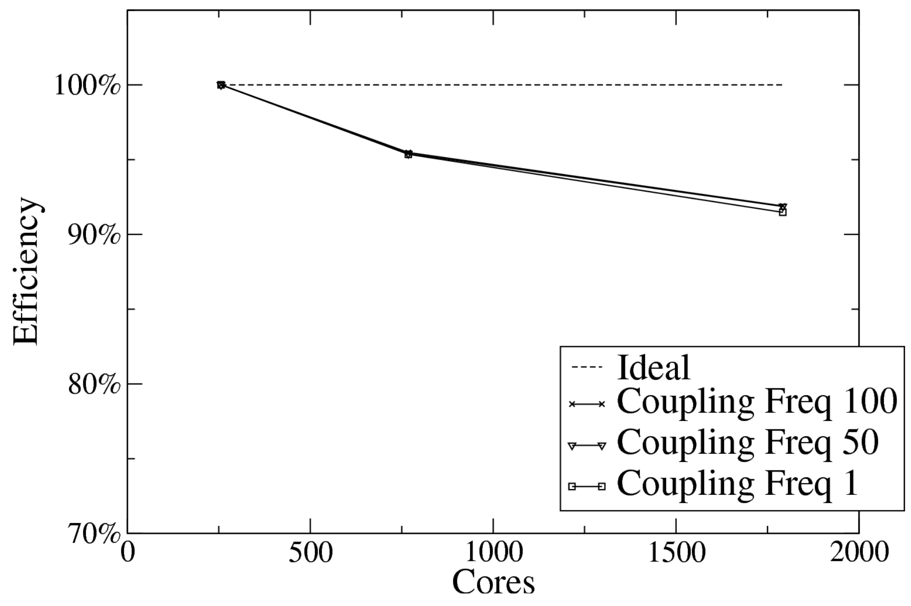


Figure 14.3: LES solver efficiency for different coupling frequencies

cores			performance/ideal performance			
Total	AVBP	AVTP	Ideal	100	50	1
512	256	256	1	1	1	1
1024	768	256	3	2.864	2.862	2.860
2048	1792	256	7	6.433	6.431	6.403

Table 14.4: Scalability results

Chapter 15

Results

Due to confidential material this chapter has been stripped from the public release of this manuscript.

Conclusion

In this final part the coupled application has been presented. Then results considering scalability, convergence, heat transfer have been presented. And for each domain two simulations have been compared: with and without coupling. The scalability results have demonstrated the efficiency of the methods developed during this thesis. As for convergence, although the simulation was not completely converged, the coupled simulation has been executed until the variation between two consecutive solid mean temperature fields was inferior to 0.04%. Comparing the coupled temperature field to the uncoupled temperature field the benefits of code multiphysics become clear: using the coupling procedure it is possible to see the position of the actual hot spots and hence deduce the positions of high thermal stress. Also some non intuitive but nonetheless very interesting features can be seen such as the difference of temperature between the internal and external liners in the aeronautical burner case. Finally looking at the fluid simulation the differences are less striking, still subtle differences in the output temperature can be seen which can be very interesting for turbine engineers. Indeed according to Sehitoglu, H. [105] a difference of 56°C of the blade operating temperature reduces the life expectancy by a factor of 1.6. Even though this simulation can not be used to compared to the actual engine due to the lack of models for the multiperforated plates, it still illustrates the potential of conjugate heat transfer coupling for industrial applications.

General Conclusion

The objective of this thesis was to investigate the issues for multiphysical code coupling on industrial configurations considering the HPC context. More particularly one of the solvers used is a combustion LES solver, the other is a thermal solver, hence bringing difficulties such as the impossibility to run the two simulations in a time synchronized manner due to very different characteristic times. Also due to combustion unsteady and nonlinear nature when solved using LES, steady state coupling algorithms based on optimization methods, which can be used with RANS, do not seem adapted. It is true that this direction has not been clearly investigated throughout this work and may be an alternative to the tight coupling methodology proposed in this thesis. But in that case probably the only way to cope with LES unsteadiness would be to average over long time periods which may be more expensive than the tight coupling methodology. This is especially true considering the large computational requirements of LES compared to RANS. However this tight coupling methodology requires the ability to exchange data fields between the LES and the thermal solver at very high frequency, meaning that the data transfers between the two solvers have to be *as fast as possible.* Hence the data transfers must be fast, but since this work is essentially to investigate and demonstrate methods for future computations, one must also consider that the data transfer solutions should be *scalable.* When scalability is considered traditionally only speed is considered but as super computers evolve, more and more cutting edge machines appear with great quantities of cores disposing of very limited memory banks. This is why it is important to consider this limitation notably during the initialization phases. A distributed geometrical search and interpolation coefficient computation method has thus been devised. This method uses modern computing paradigms such as peer-to-peer computing, geometry hashing and recursive binary space partitioning algorithms to achieve this goal. The bases of non matching grid interpolation have also been treated, starting from the original interpolation theories coming from signal sampling theory. But also a slight alternative to the traditional signal sampling method has been investigated: conservative interpolation. It has been shown that using slightly different method it is possible to eliminate a great deal of the aliasing artifacts inherent to standard sample based interpolation methods. Unfortunately due to lack of time this method has not been tested within the full scale target application.

This works is thus at the cross roads of various different disciplines. Indeed not only computational fluid dynamics and thermal conduction were required to tackle these problems but also computational science, numerical methods and signal theory play an important role in this thesis.

The resulting target application has shown promising results notably regarding scalability. Unfortunately proper modeling for the thermal flux of the multiperforated plates used in the real engine is the missing feature of the target application. Accurate wall thermal flux modeling is a necessary point to take advantage of the increased predictability potential provided by multiphysics. Nevertheless some interesting features can still be seen when the non coupled computations have been compared to the coupled computations.

As for the actual industrial spin-offs, even though this work is not directly industry ready, a simple tool based on the interpolation methods presented in this work has been implemented and is actually used within Snecma Villaroche teams to transfer data from different solvers efficiently and integrated into larger computations chains. Also the code developed for the sequential geometrical searches and interpolation methods have been packaged into a library which has been used by different PhD students at CERFACS, notably E. Collado [23], T. Pedot [84] and J. Richard [96]. Many concepts and ideas developed during this work has also been transferred by F. Duchaine into the CERFACS official coupler Open-Palm, notably the coupling region concept. Of course the source code developed during this thesis, the interpolation code, the coupling library, the graphical interface may be reused or extended for future works.

Part VI

Appendix

Table of Contents

Appendix A

Software developed during this thesis

In this chapter the main software which have been developed in this thesis are briefly presented.

A.1 A basic solution Interpolator

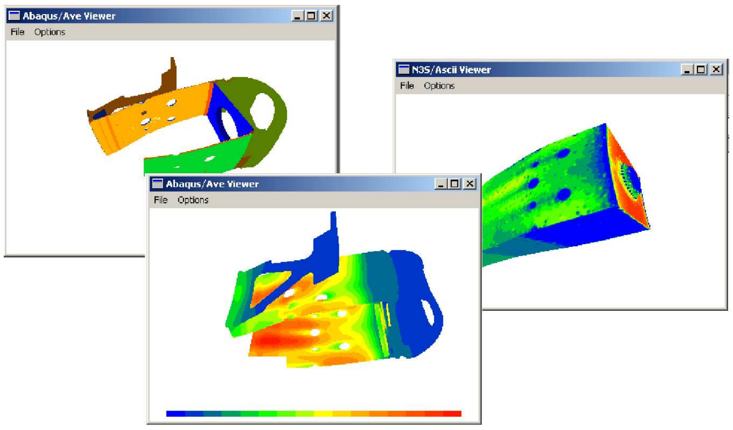


Figure A.1: Visualization tools associated to the basic interpolator

This thesis started by a two month internship at Snecma Villaroche. The objective was to provide an efficient and flexible tool to couple a RANS solver N3S, Abaqus used as a thermal solver and a radiative solver ASTRE in an industrial context. Snecma already had tools to transfer interface data from the different solvers but the transfers relied on a complex pool of tools which were not necessarily optimized. In order to simplify the coupling process a unique tool was developed using the Kd-Tree search algorithm for interpolation. The tool was developed in object oriented C++ using a concept of input and output modules allowing the tool to be extended easily to the different file formats used by each code. Also because custom treatment needed to be applied to the interpolated data a concept of external user functions was devised. During the tests and development of this tool it became clear that viewing the exact geometries involved during the interpolation process was a key issue. Therefore small viewers relying on OpenGL were developed allowing to view the input meshes, output meshes and the data interpolated, Fig A.1. Also a clear view of the connectivities was then available, Fig A.2. An other key aspect understood during this work was that industrial geometries can be extremely complex and some times geometrical searches do not provide the result expected. In those cases it is important to be able to guide the interpolation process by subdividing the interfaces into sub sets of geometrical elements to avoid those ill effects. Finally this work was an excellent testing ground for the Kd-Tree implementation and allowed to measure its actual benefits (on some geometries the interpolation time was reduced from several hours to a few seconds). This set of tools represents 25000 lines of code for the interpolation routines, the input-output readers and writers and visualisation tools. The development of this tool has continued and it has been integrated into the actual thermal conception chain.

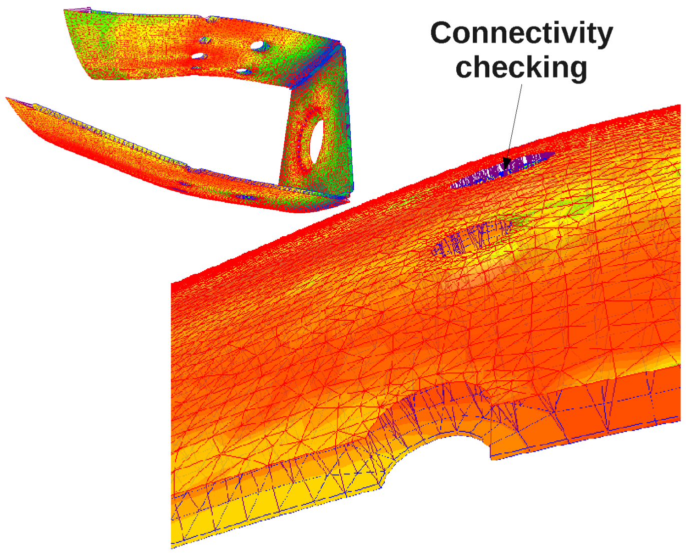


Figure A.2: Basic visual connectivity checker associated to the interpolator

A.2 The coupling library

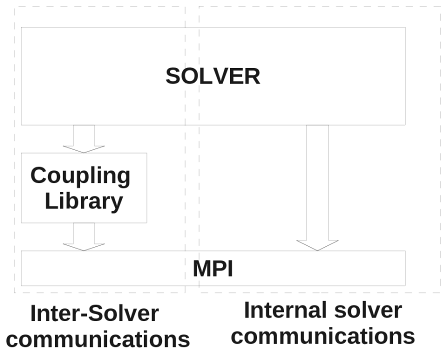


Figure A.3: Interface of the multiphysics setup tool

The coupling library developed during this thesis has been written essentially in object oriented C++ (58000 lines of code) and is interfaced to the solvers via 4600 FORTRAN lines of code. The C/FORTRAN interfaces use the ISO-C-FORTRAN 2003 bindings to ensure clean portability. The library is linked with solver executable, the program main is relocated inside the library, the solver main becoming a subroutine. The library is responsible of initializing the MPI environment, creating different communicators and modifying each solver's environment. It allows the solver to run as if they were independent while providing simple functions to exchange data. It is placed as an abstraction layer between the solver and MPI for inter-process communications, Fig A.3.

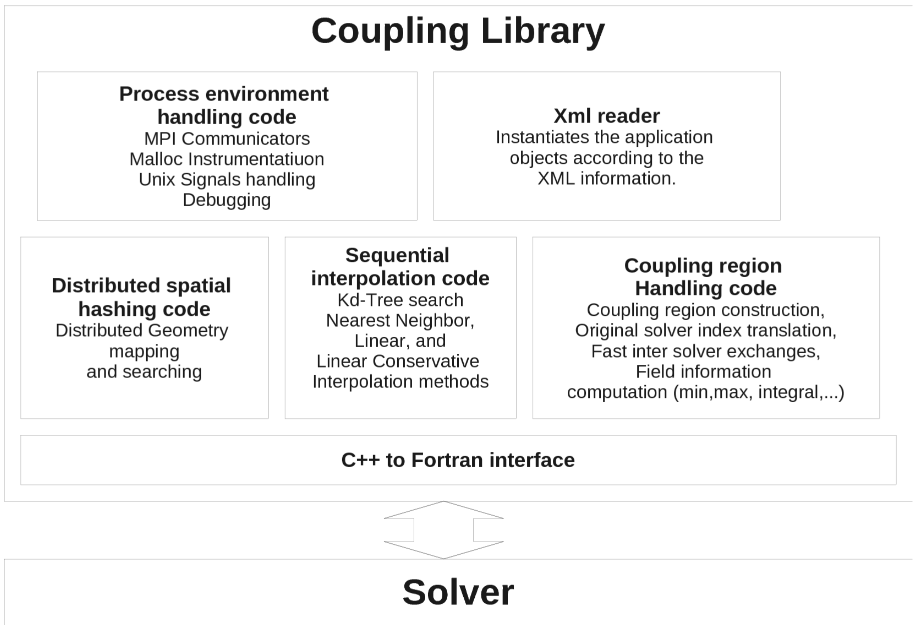

Coupling Library
Process environment handling code
MPI Communicators
Malloc Instrumentatiuon
Unix Signals handling
Debugging
Xml reader
Instantiates the application objects according to the XML information.
Distributed spatial hashing code
Distributed Geometry mapping and searching
Sequential interpolation code
Kd-Tree search
Nearest Neighbor, Linear, and Linear Conservative Interpolation methods
Coupling region Handling code
Coupling region construction, Original solver index translation, Fast inter solver exchanges, Field information computation (min,max, integral,...)
C++ to Fortran interface
Solver

Figure A.4: The main elements of the coupling library code

The coupling library contains code required for interpolation, sequential and distributed geometrical searches but also for process management and code debugging (UNIX signal handlers), Fig A.4. Additional instrumentation has been also included such as the possibility to export the transferred fields to ensight compatible files or compute statistics on the transferred values (min, max, average, surface integral). The entire configuration of the coupled application is contained in a single XML[17] file, see listing A.1. The XML parser used is the light-weight open-source portable library TinyXML[116]. This file details the environment parameters, specific solver parameters related to coupling and the coupling definitions.

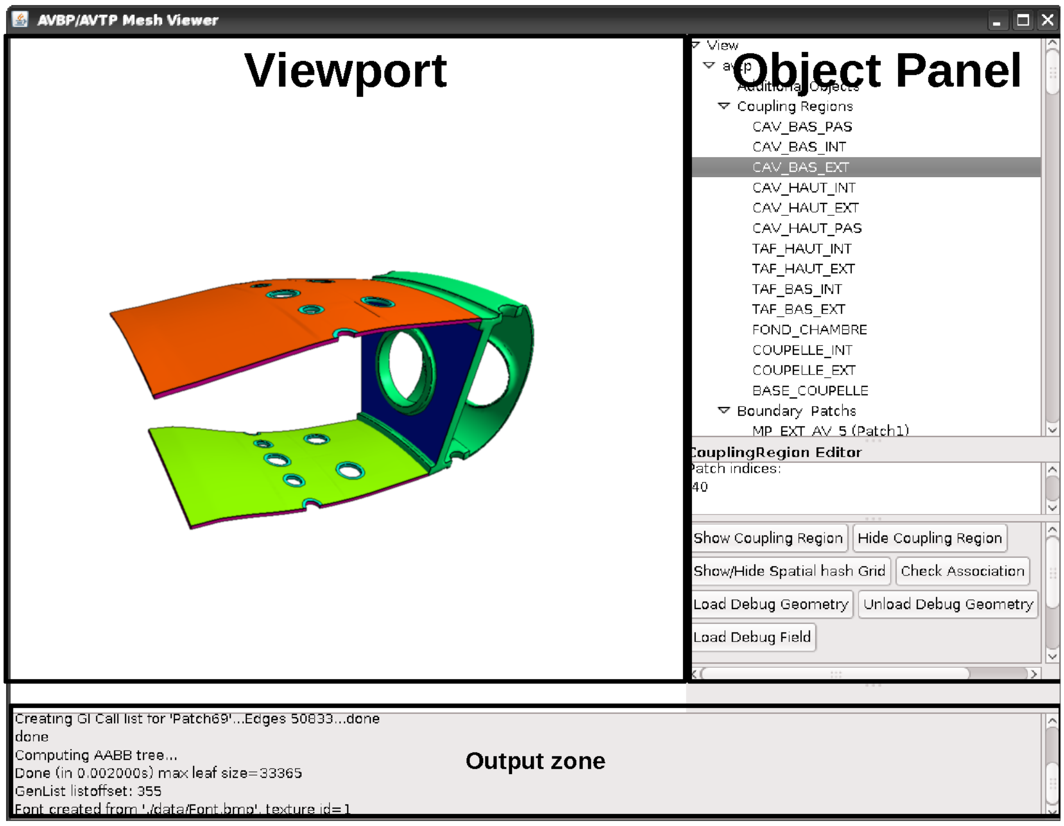
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<multiphysics_problem name="sample problem" solvercount="2">

  <solver cpl_freq="10" dht_node_count="35" force_barrier_sync="1"
  name="avbp" print_info="1" show_mem_stats="1" print_htmlinfo="0"
  ignore_root_process="1" use_connectivity_cache="1" write_connectivity_cache="1"
  stderr="./logs_avbp/ERR_%07i.log" stdout="./logs_avbp/OUT_%07i.log"
  group_outputs="1" group_dir_base="./logs_%s/grp_%04i/"
  write_emission_graph="0" write_reception_graph="0"
  workdir="./fluid_prepart" worker_count="35" cpl_dump_data="0" cpl_dump_temporal="0"
  cpl_check_data="0" cpl_max_iter="10" cpl_xfer_temperature="1"
  cpl_xfer_hconv_tconv="1" cpl_xfer_flux="1" cpl_use_qresolv="0" write_dbg_geom="0"
  arguments="dummy -server kpl_distinct_server -nclient 35">
    <cpl_region cpl_reg_id="5" extra_layer_size="1" interp_type="1" name="CAV_BAS_PAS"
        patch_names="CAVITE_BAS_PASSAGE" patchs="34" remote_solver="1" resX="0.03" resY
        ="0.03" resZ="0.03"/>
(...)
    <cpl_region cpl_reg_id="14" extra_layer_size="1" interp_type="1" name="TAF_BAS_EXT"
         patch_names="MP_INT_AM_5,MP_INT_AM_6,MP_INT_AM_7,MP_INT_AM_8,MP_INT_AM_9,
        MP_INT_AM_10" patchs="58,59,60,61,62,63" remote_solver="1" resX="0.03" resY="
        0.03" resZ="0.03"/>
    <cpl_region cpl_reg_id="16" extra_layer_size="1" interp_type="1" name="FOND_CHAMBRE
        " patch_names="FOND_CHAMBRE_INT,SWIRLER_OUTPUT_1,SWIRLER_OUTPUT_2,
        SWIRLER_OUTPUT_3" patchs="12,27,28,29" remote_solver="1" resX="0.03" resY="0.03
        " resZ="0.03"/>
    <cpl_region cpl_reg_id="1" extra_layer_size="1" interp_type="1" name="COUPELLE_INT"
         patch_names="COUPELLE_INT_1,COUPELLE_INT_2" patchs="36,38" remote_solver="1"
        resX="0.03" resY="0.03" resZ="0.03"/>
    <cpl_region cpl_reg_id="2" extra_layer_size="1" interp_type="1" name="COUPELLE_EXT"
         patch_names="COUPELLE_EXT_1,COUPELLE_EXT_2" patchs="37,39" remote_solver="1"
        resX="0.03" resY="0.03" resZ="0.03"/>
    <cpl_region cpl_reg_id="3" extra_layer_size="1" interp_type="1" name="BASE_COUPELLE
        " patch_names="FOND_CHAMBRE_EXT,TUBE_WALLS_TOP,TUBE_WALLS_BOTTOM,OUT_COUPELLE"
        patchs="13,14,15,66" remote_solver="1" resX="0.03" resY="0.03" resZ="0.03"/>
  </solver>
  <solver cpl_freq="10" dht_node_count="11" force_barrier_sync="1"
  name="avtp" printf_info="1" show_mem_stats="1" print_htmlinfo="0"(...)>
    <cpl_region cpl_reg_id="5" extra_layer_size="1" interp_type="1" name="CAV_BAS_PAS"
        patch_names="BRIDE_BAS_PASSAGE" patchs="41" remote_solver="0" resX="0.03" resY=
        "0.03" resZ="0.03"/>
(...)
    <cpl_region cpl_reg_id="3" extra_layer_size="1" interp_type="1" name="BASE_COUPELLE
        " patch_names="FOND_CHAMBRE" patchs="32" remote_solver="0" resX="0.03" resY="
        0.03" resZ="0.03"/>
  </solver>

</multiphysics_problem>
```

Listing A.1: Example of an XML configuration file for the coupled application

A.3 A graphical user interface for 3D unstructured mesh coupling



```
Creating Gl Call list for 'Patch69'...Edges 50833...done
done
Computing AABB tree...
Done (in 0.002000s) max leaf size=33365
GenList listoffset: 355
Font created from './data/Font.bmp', texture id=1
```

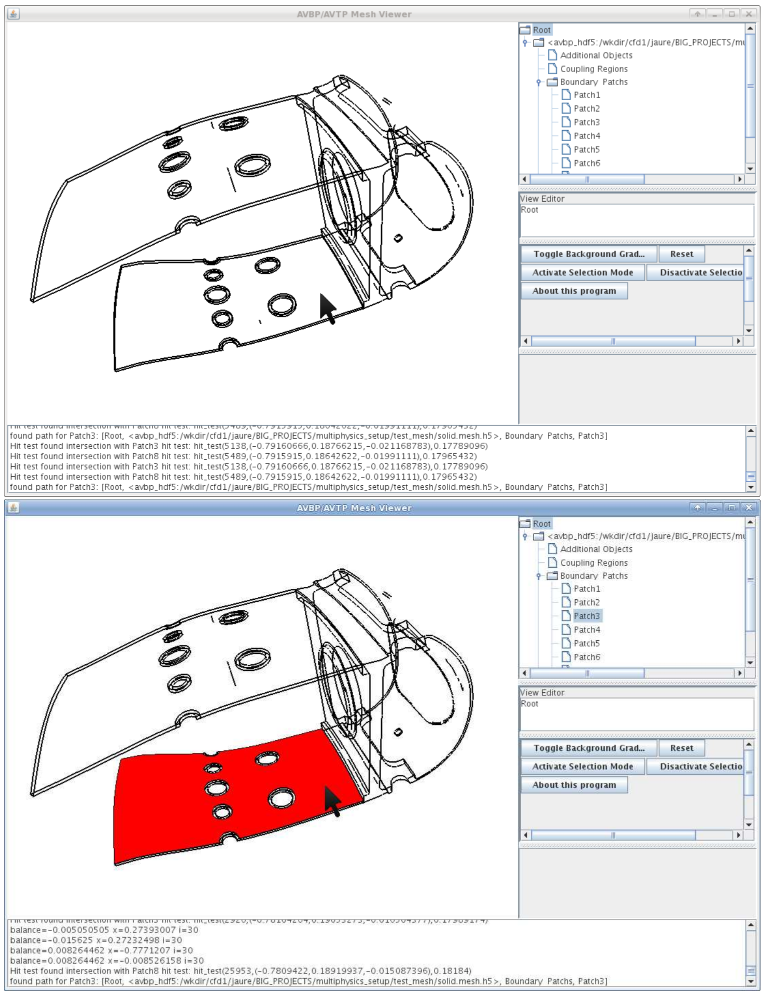
Figure A.5: Interface of the multiphysics setup tool

As has been shown throughout this thesis the setup of complex coupled applications is a difficult task. It requires a simultaneous visualization of the geometries and to some of their specific features. For the AVBP/AVTP case considering the choice to build coupling regions from the existing boundary surfaces, simple access to those features was mandatory. A solution to this problem has been proposed in this thesis: a graphical tool has been developed capable of displaying the different meshes and their sub surfaces. The tool can greatly simplify the process of coupling region creation but also provides some other useful features. Indeed the tool is also capable of giving a visual aid to control whether coupling regions in each mesh match.

The tool was built using the Java programming language [34] (20000 lines of code) and uses the industry standard rendering library OpenGL [108]. Although parts of its code had already been developed in the visualization tools presented in section A.1, this tool by its structure allows far more advanced features. Its interface is composed of 3 main parts, Fig A.5:

- the viewport, this is where all the geometry is displayed,
- the object panel, each object which can be edited is displayed within a tree, the selected object properties and actions are displayed on the sub-panel which is beneath the object tree.
- the output panel, this panel is located underneath the viewport and displays messages from the application's core: progress messages, error messages, results computed, etc...

Interactivity has been added by full support of the mouse for rotating, zooming and translation. Also it is possible to select objects using either the object tree or their visual representation in the viewport, Fig A.6. The implementation of this feature is actually an other application of AABB trees



```
found path for Patch3: [Root, <avbp_hdf5:/wkdir/cfd1/jaure/BIG_PROJECTS/multiphysics_setup/test_mesh/solid.mesh.h5>, Boundary Patchs, Patch3]
Hit test found intersection with Patch3 hit test: hit_test(5138,(-0.79160666,0.18766215,-0.021168783),0.17789096)
Hit test found intersection with Patch8 hit test: hit_test(5489,(-0.7915915,0.18642622,-0.01991111),0.17965432)
Hit test found intersection with Patch3 hit test: hit_test(5138,(-0.79160666,0.18766215,-0.021168783),0.17789096)
Hit test found intersection with Patch8 hit test: hit_test(5489,(-0.7915915,0.18642622,-0.01991111),0.17965432)
found path for Patch3: [Root, <avbp_hdf5:/wkdir/cfd1/jaure/BIG_PROJECTS/multiphysics_setup/test_mesh/solid.mesh.h5>, Boundary Patchs, Patch3]
```

```
balance=-0.005050505 x=0.27393007 i=30
balance=-0.015625 x=0.27232498 i=30
balance=0.008264462 x=-0.7771207 i=30
balance=0.008264462 x=-0.008526158 i=30
Hit test found intersection with Patch8 hit test: hit_test(25953,(-0.7809422,0.18919937,-0.015087396),0.18184)
found path for Patch3: [Root, <avbp_hdf5:/wkdir/cfd1/jaure/BIG_PROJECTS/multiphysics_setup/test_mesh/solid.mesh.h5>, Boundary Patchs, Patch3]
```

Figure A.6: The viewport is allows to select objects such as boundary surfaces by simply clicking on them

presented within part III. Indeed when a user clicks on the screen, a line is actually computed from an imaginary viewpoint (corresponding to the user's eye in 3D rendering theory [73]) to the point clicked by the user, Fig A.7. This hit ray is then tested with the geometry to find possible intersection points. The process is actually greatly accelerated by building an AABB tree (see chapter 10) which allows to eliminate unnecessary tests by testing if the hit ray intersects the tree's successive boundary boxes. This process is actually the basic process which is implemented in Ray-Tracers [40]. This interactivity comes out to be quite handy to select and handle the different boundary regions.

The coupling boundary regions are built by selecting boundary patches from two lists, the corresponding coupling regions are displayed interactively in the viewport, Fig A.9.

Also an other useful feature is that the tool can help in controlling the interpolation quality. The tool projects using geometry of a coupling region onto the other and then vice-versa. The distance between the original point and its projected point are recorded and the 50 worst projections (which are the most apart) are displayed by small segments. The projection method only uses nearest neighbor projection so it can show that some projections are bad because the two meshes are discretized very differently but the actual linear interpolation would be of good quality. However it is clearly capable of finding small gaps between surfaces or incorrect coupling region constructions. The XML files used for the coupling region definition are created and edited by this tool.

Other features such as computing fluxes through holes, Fig A.10, have also been coded and much more. This tool has actually been a great aid to perform some very specific tasks which are not

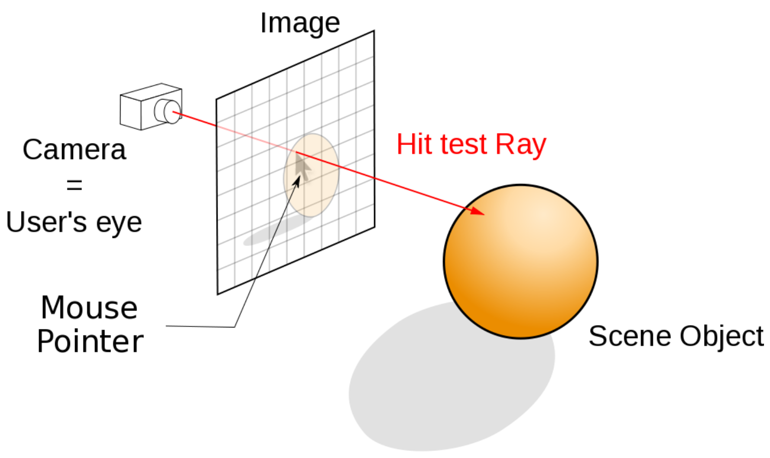
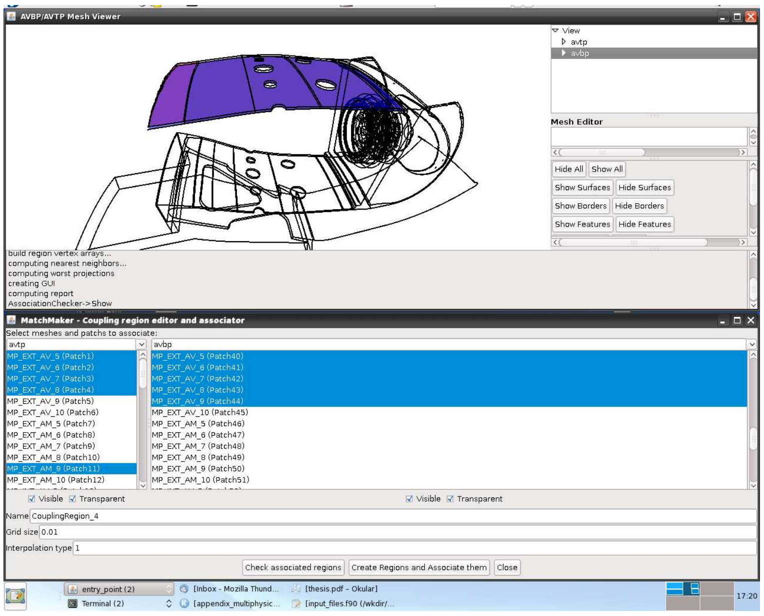


Figure A.7: Process to do a hit test from a user click: user clicks on a 2D image the search is performed in a 3D space using a hit test ray



```
build region vertex arrays...
computing nearest neighbors...
computing worst projections
creating GUI
computing report
AssociationChecker->Show
```

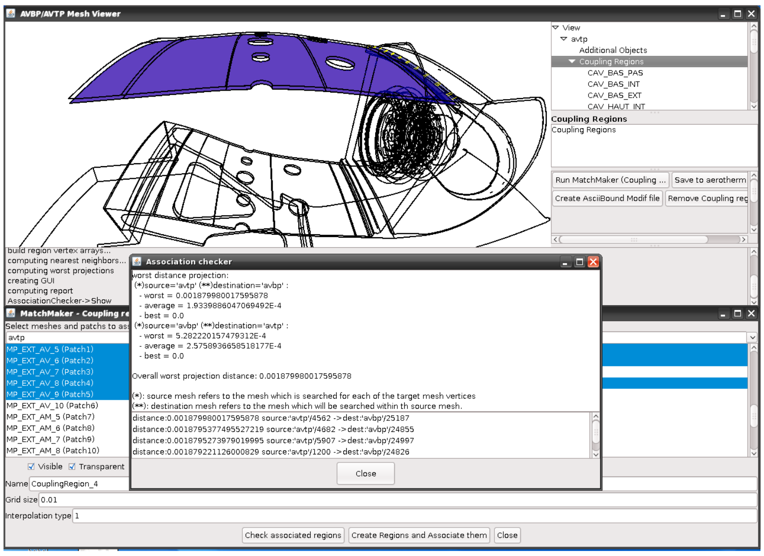
Select meshes and patchs to associate:

avtp	avbp
MP_EXT_AV_5 (Patch1)	MP_EXT_AV_5 (Patch40)
MP_EXT_AV_6 (Patch2)	MP_EXT_AV_6 (Patch41)
MP_EXT_AV_7 (Patch3)	MP_EXT_AV_7 (Patch42)
MP_EXT_AV_8 (Patch4)	MP_EXT_AV_8 (Patch43)
MP_EXT_AV_9 (Patch5)	MP_EXT_AV_9 (Patch44)
MP_EXT_AV_10 (Patch6)	MP_EXT_AV_10 (Patch45)
MP_EXT_AM_5 (Patch7)	MP_EXT_AM_5 (Patch46)
MP_EXT_AM_6 (Patch8)	MP_EXT_AM_6 (Patch47)
MP_EXT_AM_7 (Patch9)	MP_EXT_AM_7 (Patch48)
MP_EXT_AM_8 (Patch10)	MP_EXT_AM_8 (Patch49)
MP_EXT_AM_9 (Patch11)	MP_EXT_AM_9 (Patch50)
MP_EXT_AM_10 (Patch12)	MP_EXT_AM_10 (Patch51)

Name CouplingRegion_4
Grid size 0.01
Interpolation type 1

Figure A.8: Construction of a coupling region by simply selecting the boundary surfaces of each mesh

necessarily available in commercial products. The interpolation problems showed in part IV have actually been understood using this tool. A great deal of the images present in this thesis have been rendered using this tool.



```
build region vertex arrays...
computing nearest neighbors...
computing worst projections
creating GUI
computing report
AssociationChecker->Show
```

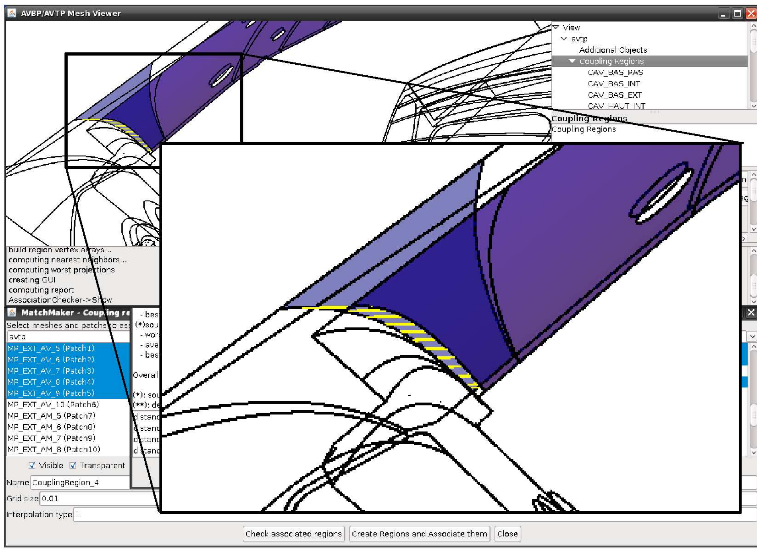
Name CouplingRegion_4
Grid size 0.01
Interpolation type 1

```
worst distance projection:
 (*)source='avtp' (**)destination='avbp' :
  - worst = 0.001879980017595878
  - average = 1.9339886047069492E-4
  - best = 0.0
 (*)source='avbp' (**)destination='avtp' :
  - worst = 5.282220157479312E-4
  - average = 2.5758936658518177E-4
  - best = 0.0

Overall worst projection distance: 0.001879980017595878

(*): source mesh refers to the mesh which is searched for each of the target mesh vertices
(**): destination mesh refers to the mesh which will be searched within th source mesh.
```

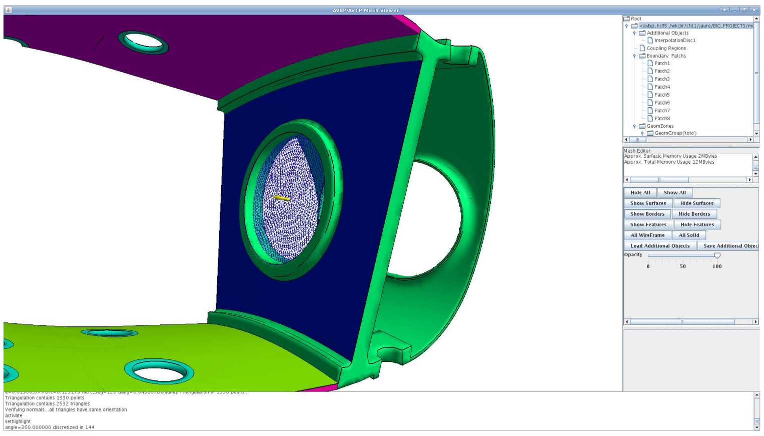
```
distance:0.001879980017595878 source:'avtp'/4562 ->dest:'avbp'/25187
distance:0.0018795377495527219 source:'avtp'/4682 ->dest:'avbp'/24855
distance:0.0018795273979019995 source:'avtp'/5907 ->dest:'avbp'/24997
distance:0.001879221126000829 source:'avtp'/1200 ->dest:'avbp'/24826
```



```
build region vertex arrays...
computing nearest neighbors...
computing worst projections
creating GUI
computing report
AssociationChecker->Show
```

Name CouplingRegion_4
Grid size 0.01
Interpolation type 1

Figure A.9: Using the association tool checker can help locate mistakes or non matching surfaces. This example shows a mistake in the construction of a coupling region: when two coupling regions match properly the distance between the source and destination points are extremely small. The tool displays in yellow the 50 longest paths between source and destination points between the two coupling regions. On this example a surface located on the casing side of the wall of the flame tube is associated with a surface on the other side. The long yellow lines help to find the location of the problem.



```
Approx. Surfacic Memory Usage 2MBytes
Approx. Total Memory Usage 12MBytes
```

```
Triangulation contains 1330 points
Triangulation contains 2532 triangles
Verifying normals...all triangles have same orientation
activate
sethighlight
angle=360.000000 discretized in 144
```

Figure A.10: Using this tool it is possible to measure fluxes passing through holes by meshing extra surfaces in those holes and interpolating solutions onto them. The construction of the surfaces is done visually by clicking on the geometry.

Appendix B

Geometric formulas and algorithms for interpolation

B.1 Barycentric coordinates for linear interpolation

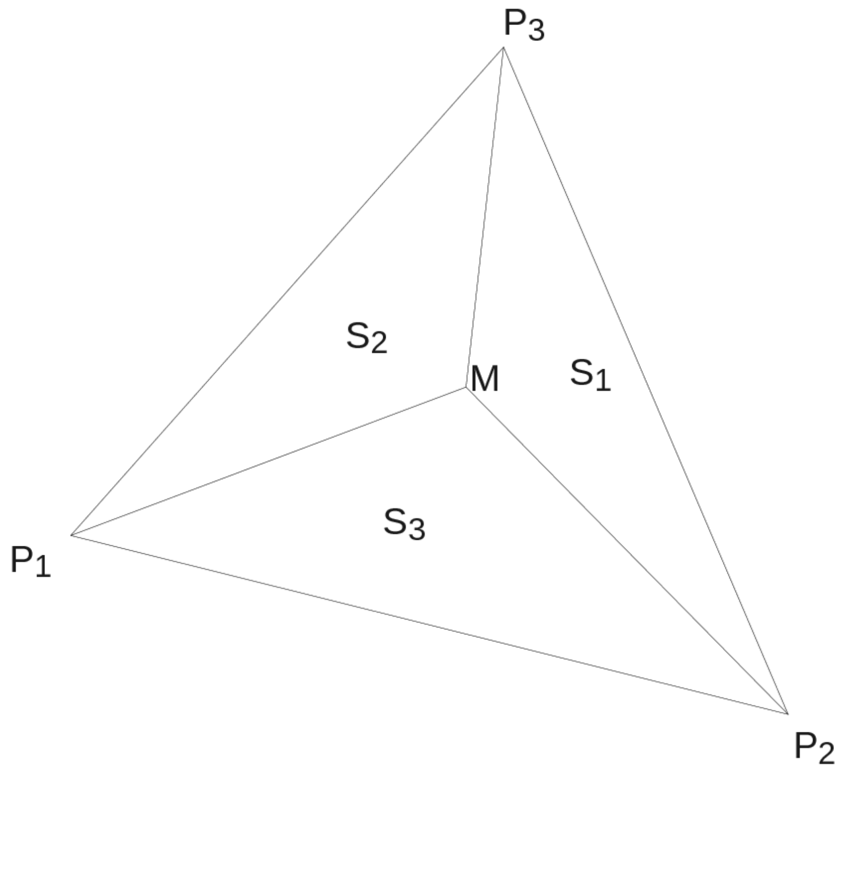


Figure B.1: Barycentric coordinates for a triangle $E = (P_1P_2P_3)$

Let the space dimensions be N. For a simplex E, i.e. a polyhedron composed of $N+1$ vertices (segments in 1D, triangles in 2D, tetrahedra in 3D), $N+1$ barycentric coordinates λ_i can be defined.

These barycentric coordinates allow to interpolate linearly f for any point inside E. Let λ_i be the barycentric coordinates of M inside E. Let P_i be E's vertices. E can be split into $N+1$ new simplexes S_i using M. λ_i can be defined by :

$$\lambda_i = \frac{Volume(S_i)}{Volume(E)} \tag{B.1}$$

Here $Volume$ is in a generic sens, for a segment it is its length, for a triangle its area, for a tetrahedron its volume.

Properties

- For M inside E, $0 \leq \lambda_i \leq 1$.
- For any M, $\sum \lambda_i = 1$

Calculating barycentric coordinates is fairly simple, the equations used to define them for segments, triangles and tetrahedrons are included in the following part of this section.

B.1.1 Barycentric coordinates for a segment in 1D

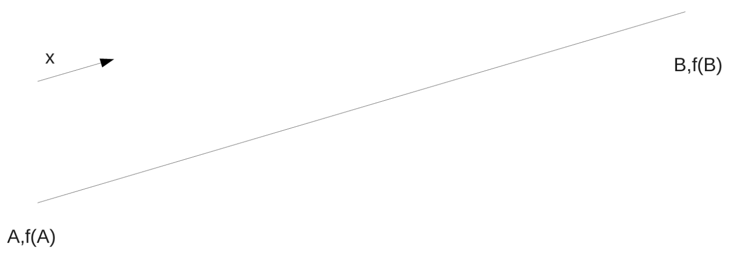


Figure B.2: A P1 segment

Let f be a function varying linearly on (AB), $f(x_A)$ and $f(x_B)$ are known, and the coordinates x_A,x_B.

The gradient of f is $\nabla f = \frac{f(x_B)-f(x_A)}{x_B-x_A}$. We can write:

$$f(x) = \nabla f \cdot (x - x_A) + f(x_A) \tag{B.2}$$

Lets now define the barycentric coordinates λ_i such as $f(x_M) = f(x_A) \cdot \lambda_1(x_M) + f(x_B) \cdot \lambda_2(x_M)$

$$\lambda_1(x) = 1 - \frac{x - x_A}{x_B - x_A} \tag{B.3}$$

$$\lambda_2(x) = \frac{x - x_A}{x_B - x_A} \tag{B.4}$$

B.1.2 Barycentric coordinates for a triangle in 2D

Let f be a function varying linearly on (ABC), $f(A)$,$f(B)$ and $f(C)$ are known, and the coordinates of A,B,C.

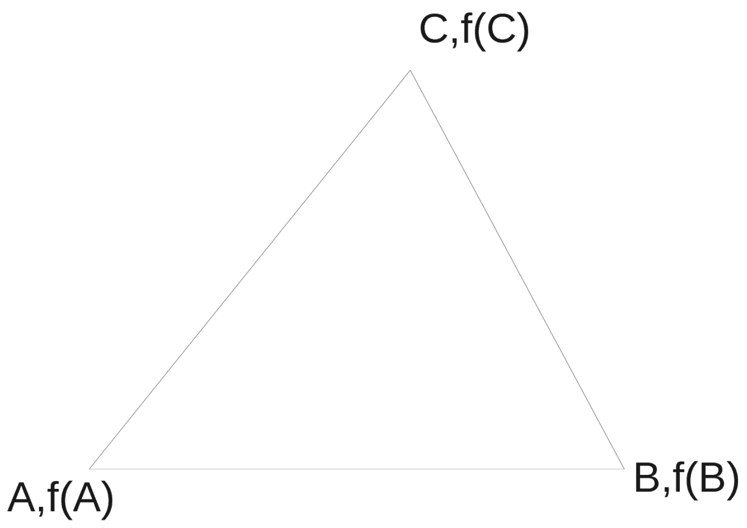


Figure B.3: A P1 triangle

We can thus write:

$$f(x,y) = a \cdot x + b \cdot y + c \tag{B.5}$$
$$\nabla f = a\vec{x} + b\vec{y} \tag{B.6}$$
$$\tag{B.7}$$

and for a point M:

$$f(M) = \nabla f \cdot \vec{AM} + f(A) \tag{B.8}$$

Thus for B and C:

$$\begin{cases} f(B) - f(A) = \nabla f \cdot \vec{AB} \\ f(C) - f(A) = \nabla f \cdot \vec{AC} \end{cases} \tag{B.9}$$

Introducing the following notations,

$$\begin{cases} P = f(B) - f(A) \\ Q = f(C) - f(A) \end{cases} \tag{B.10}$$

$$\begin{cases} m_{11} = x_B - x_A, m_{12} = y_B - y_A \\ m_{21} = x_C - x_A, m_{22} = y_C - y_A \end{cases} \tag{B.11}$$

Eq. B.28 becomes:

$$\begin{cases} P = a \cdot m_{11} + b \cdot m_{12} \\ Q = a \cdot m_{21} + b \cdot m_{22} \end{cases} \tag{B.12}$$

Which can be solved:

$$\begin{cases} a = \frac{P \cdot m_{22} - Q \cdot m_{12}}{det} \\ b = \frac{Q \cdot m_{11} - P \cdot m_{21}}{det} \end{cases} \tag{B.13}$$
$$\tag{B.14}$$

with $det = m_{22} \cdot m_{11} - m_{12} \cdot m_{21}$

Now lets introduce the barycentric coordinates λ_i, they are affine functions defined by

$$\begin{cases} \lambda_1(A) = 1, \lambda_1(B) = 0, \lambda_1(C) = 0 \\ \lambda_2(A) = 0, \lambda_2(B) = 1, \lambda_2(C) = 0 \\ \lambda_3(A) = 0, \lambda_3(B) = 0, \lambda_3(C) = 1 \end{cases} \tag{B.15}$$

Using Eq.B.14 and the 3 point definition of each λ_i we obtain

$$\lambda_1(x,y) = \frac{-m_{22} + m_{12}}{det} \cdot (x - x_A) + \frac{-m_{11} + m_{21}}{det} \cdot (y - y_A) + 1 \tag{B.16}$$

(B.17)

$$\lambda_2(x,y) = \frac{m_{22}}{det} \cdot (x - x_A) + \frac{-m_{21}}{det} \cdot (y - y_A) + 0 \tag{B.18}$$

(B.19)

$$\lambda_3(x,y) = \frac{-m_{12}}{det} \cdot (x - x_A) + \frac{m_{11}}{det} \cdot (y - y_A) + 0 \tag{B.20}$$

(B.21)

And can check that for any (x, y),

$$\lambda_1(x,y) + \lambda_2(x,y) + \lambda_3(x,y) = 1 \tag{B.22}$$

These barycentric coordinates are the interpolation weights for a linear interpolation:

$$f(M) = f(A) \cdot \lambda_1(M) + f(B) \cdot \lambda_2(M) + f(C) \cdot \lambda_3(M) \tag{B.23}$$

B.1.3 Barycentric coordinates for a tetrahedron in 3D

Let f be a function varying linearly on $(ABCD)$, $f(A)$,$f(B)$,$f(C)$ and $f(D)$ are known, and the coordinates of A,B,C,D.

We can thus write:

$$f(x,y) = a \cdot x + b \cdot y + c \cdot z + d \tag{B.24}$$

$$\nabla f = a\vec{x} + b\vec{y} + c\vec{z} \tag{B.25}$$

(B.26)

and for a point M:

$$f(M) = \nabla f \cdot \vec{AM} + f(A) \tag{B.27}$$

Thus for B and C:

$$\begin{cases} f(B) - f(A) = \nabla f \cdot \vec{AB} \\ f(C) - f(A) = \nabla f \cdot \vec{AC} \\ f(D) - f(A) = \nabla f \cdot \vec{AD} \end{cases} \tag{B.28}$$

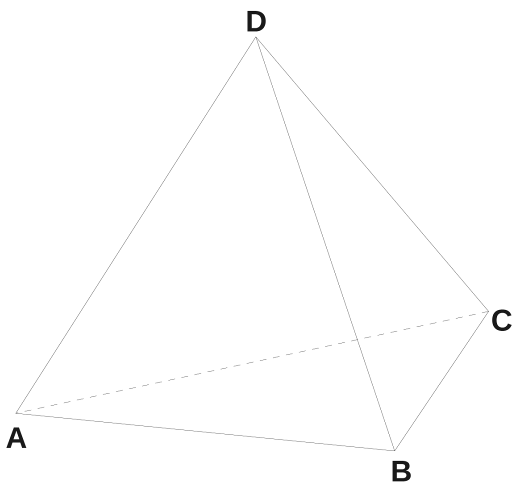


Figure B.4: A Tetrahedron

Introducing the following notations,

$$\begin{cases} P = f(B) - f(A) \\ Q = f(C) - f(A) \\ R = f(D) - f(A) \end{cases} \tag{B.29}$$

$$\begin{cases} m_{11} = x_B - x_A, m_{12} = y_B - y_A, m_{13} = z_B - z_A \\ m_{21} = x_C - x_A, m_{22} = y_C - y_A, m_{23} = z_C - z_A \\ m_{31} = x_D - x_A, m_{32} = y_D - y_A, m_{33} = z_D - z_A \end{cases} \tag{B.30}$$

Eq. B.28 becomes:

$$\begin{cases} P = a \cdot m_{11} + b \cdot m_{12} + c \cdot m_{13} \\ Q = a \cdot m_{21} + b \cdot m_{22} + c \cdot m_{23} \\ R = a \cdot m_{31} + b \cdot m_{32} + c \cdot m_{33} \end{cases} \tag{B.31}$$

Which can be solved:

$$\begin{cases} a = \frac{P(m_{33} \cdot m_{22} - m_{32} \cdot m_{23}) - Q(m_{33} \cdot m_{12} - m_{32} \cdot m_{13}) + R(m_{23} \cdot m_{12} - m_{22} \cdot m_{13})}{det} \\ b = \frac{-P(m_{33} \cdot m_{21} - m_{31} \cdot m_{23}) + Q(m_{33} \cdot m_{11} - m_{31} \cdot m_{13}) - R(m_{23} \cdot m_{11} - m_{21} \cdot m_{13})}{det} \\ c = \frac{P(m_{32} \cdot m_{21} - m_{31} \cdot m_{22}) - Q(m_{32} \cdot m_{11} - m_{31} \cdot m_{12}) + R(m_{22} \cdot m_{11} - m_{21} \cdot m_{12})}{det} \end{cases} \tag{B.32}$$

$$\tag{B.33}$$

with

$$det = m_{11}(m_{33} \cdot m_{22} - m_{32} \cdot m_{23}) - m_{21}(m_{33} \cdot m_{12} - m_{32} \cdot m_{13}) - m_{31}(m_{23} \cdot m_{12} - m_{22} \cdot m_{13}) \tag{B.34}$$

Now lets introduce the barycentric coordinates :

$$\begin{cases} \lambda_1(A) = 1, \lambda_1(B) = 0, \lambda_1(C) = 0, \lambda_1(D) = 0 \\ \lambda_2(A) = 0, \lambda_2(B) = 1, \lambda_2(C) = 0, \lambda_2(D) = 0 \\ \lambda_3(A) = 0, \lambda_3(B) = 0, \lambda_3(C) = 1, \lambda_3(D) = 0 \\ \lambda_4(A) = 0, \lambda_4(B) = 0, \lambda_4(C) = 0, \lambda_4(D) = 1 \end{cases} \tag{B.35}$$

By combining Eq. B.35 and Eq. B.33 gives (the equation is written as a vector dot product for concision)

$$\begin{aligned}\lambda_1(x,y,z) &= 1 + \\ &\begin{pmatrix} x-x_A \\ y-y_A \\ z-z_A \end{pmatrix} \cdot \begin{pmatrix} \frac{-(m_{33}\cdot m_{22}-m_{32}\cdot m_{23})+(m_{33}\cdot m_{12}-m_{32}\cdot m_{13})-(m_{23}\cdot m_{12}-m_{22}\cdot m_{13})}{det} \\ \frac{(m_{33}\cdot m_{21}-m_{31}\cdot m_{23})-(m_{33}\cdot m_{11}-m_{31}\cdot m_{13})+(m_{23}\cdot m_{11}-m_{21}\cdot m_{13})}{det} \\ \frac{-(m_{32}\cdot m_{21}-m_{31}\cdot m_{22})+(m_{32}\cdot m_{11}-m_{31}\cdot m_{12})-(m_{22}\cdot m_{11}-m_{21}\cdot m_{12})}{det} \end{pmatrix}\end{aligned}$$

$$\lambda_2(x,y,z) = \begin{pmatrix} x-x_A \\ y-y_A \\ z-z_A \end{pmatrix} \cdot \begin{pmatrix} \frac{(m_{33}\cdot m_{22}-m_{32}\cdot m_{23})}{det} \\ \frac{-(m_{33}\cdot m_{21}-m_{31}\cdot m_{23})}{det} \\ \frac{(m_{32}\cdot m_{21}-m_{31}\cdot m_{22})}{det} \end{pmatrix}$$

$$\lambda_3(x,y,z) = \begin{pmatrix} x-x_A \\ y-y_A \\ z-z_A \end{pmatrix} \cdot \begin{pmatrix} \frac{-(m_{33}\cdot m_{12}-m_{32}\cdot m_{13})}{det} \\ \frac{(m_{33}\cdot m_{11}-m_{31}\cdot m_{13})}{det} \\ \frac{-(m_{32}\cdot m_{11}-m_{31}\cdot m_{12})}{det} \end{pmatrix} \tag{B.36}$$

$$\lambda_4(x,y,z) = \begin{pmatrix} x-x_A \\ y-y_A \\ z-z_A \end{pmatrix} \cdot \begin{pmatrix} \frac{(m_{23}\cdot m_{12}-m_{22}\cdot m_{13})}{det} \\ \frac{-(m_{23}\cdot m_{11}-m_{21}\cdot m_{13})}{det} \\ \frac{(m_{22}\cdot m_{11}-m_{21}\cdot m_{12})}{det} \end{pmatrix} \tag{B.37}$$

Using λ_i, we can interpolate linearly f over $(ABCD)$:

$$f(M) = f(A)\cdot\lambda_1 + f(B)\cdot\lambda_2 + f(C)\cdot\lambda_3 + f(D)\cdot\lambda_4 \tag{B.38}$$

B.2 Calculating the integral of a linear function over P1 elements

B.2.1 Calculating the integral of a linear function on a triangle

To calculate the integral of an affine function over a triangle, we will use a transformation Φ defined by:

$$\Phi(u,v) = (x,y) \tag{B.39}$$

and

$$\begin{cases} \Phi(0,0) = (x_A, y_A) \\ \Phi(0,1) = (x_B, y_B) \\ \Phi(1,0) = (x_C, y_C) \end{cases} \tag{B.40}$$

$$\Phi(u,v) = \begin{cases} x = x_A + u\cdot(x_C - x_A) + v\cdot(x_B - x_A) \\ y = y_A + u\cdot(y_C - y_A) + v\cdot(y_B - y_A) \end{cases} \tag{B.41}$$

The Jacobian of Φ is thus

$$J_\Phi = \begin{bmatrix} x_C - x_A & x_B - x_A \\ y_C - y_A & y_B - y_A \end{bmatrix} \tag{B.42}$$

And

$$|J_\Phi| = |(x_C - x_A)(y_B - y_A) - (y_C - y_A)(x_B - x_A)| \tag{B.43}$$

Now lets calculate the integral of f over (ABC):

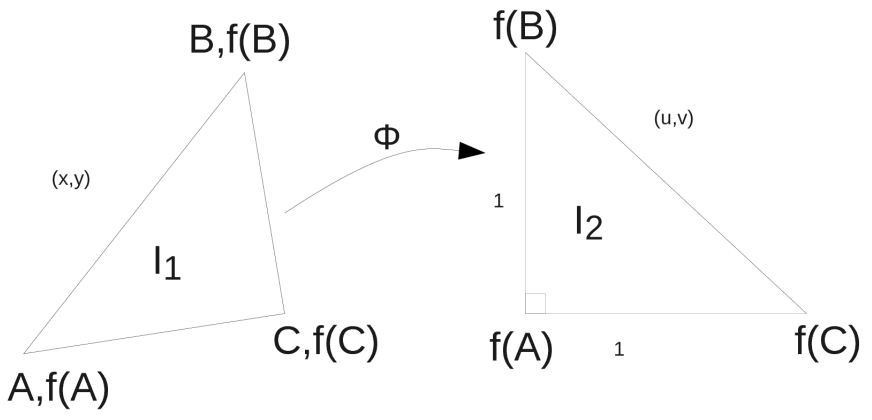


Figure B.5: Calculating an integral on a triangle

$$I = \iint_{(ABC)} f \cdot dx \cdot dy = |J_\Phi| \iint_{T_2} (f \circ \Phi) \cdot du \cdot dv \tag{B.44}$$
$$= |J_\Phi| \int_{v=0}^{1} \left(\int_{u=0}^{1-v} f(\Phi(u,v)) \cdot du \right) dv$$

And injecting the definition of Φ from Eq. B.41:

$$\begin{aligned} f(\Phi(u,v)) &= f_A \\ &+ u\left[a(x_C - x_A) + b(y_C - y_A)\right] \\ &+ v\left[a(x_B - x_A) + b(y_B - y_A)\right] \end{aligned}$$

$$f(\Phi(u,v)) = f_A + u\left[f_C - f_A\right] + v\left[f_B - f_A\right] \tag{B.45}$$

The following notations are introduced:

$$\begin{cases} P = f_C - f_A \\ Q = f_B - f_A \\ R = f_A \end{cases} \tag{B.46}$$

$$\begin{aligned} \int_{u=0}^{1-v} f(\Phi(u,v)) \cdot du &= \left[\frac{u^2}{2} P + u(R + vQ) \right]_0^{1-v} \\ &= \frac{(1-v)^2}{2} P + (1-v)(R + vQ) \\ &= v^2 \left(\frac{1}{2} P - Q \right) + v\,(Q - P - R) + R + \frac{P}{2} \end{aligned} \tag{B.47}$$

Injecting Eq. B.48 in Eq. B.45 yields:

$$\begin{aligned}
I &= |J_\Phi| \left[\frac{v^3}{3} \left(\frac{1}{2}P - Q \right) + \frac{v^2}{2} (Q - P - R) + v(R + \frac{P}{2}) \right]_0^1 \\
&= |J_\Phi| \left(\frac{1}{3} \left(\frac{1}{2}P - Q \right) + \frac{1}{2} (Q - P - R) + R + \frac{P}{2} \right) \\
&= |J_\Phi| \left(\frac{1}{6}P + \frac{1}{6}Q + \frac{1}{2}R \right)
\end{aligned}$$

Finally using Eq. B.46 we obtain:

$$I = \frac{1}{6}|J_\Phi| \left(f_A + f_B + f_C \right) \tag{B.48}$$

B.2.2 Calculating the integral of a linear function on a tetrahedron

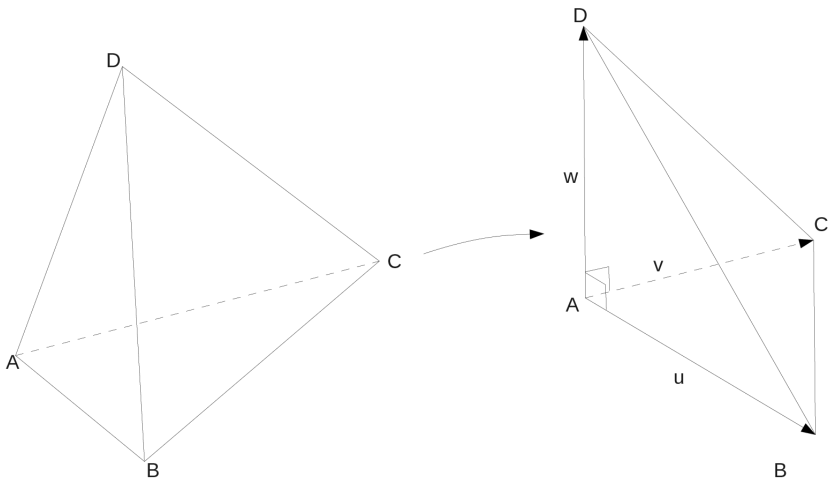


Figure B.6: Calculating an integral on a tetrahedron

To calculate the integral of an affine function over a tetrahedron, we will use a transformation Φ defined by:

$$\Phi(u, v, w) = (x, y, z) \tag{B.49}$$

and

$$\begin{cases} \Phi(0,0,0) = (x_A, y_A, z_A) \\ \Phi(1,0,0) = (x_B, y_B, z_B) \\ \Phi(0,1,0) = (x_C, y_C, z_C) \\ \Phi(0,0,1) = (x_D, y_D, z_D) \end{cases} \tag{B.50}$$

$$\Phi(u, v, w) = \begin{cases} x = x_A + u \cdot (x_B - x_A) + v \cdot (x_C - x_A) + w \cdot (x_D - x_A) \\ y = y_A + u \cdot (y_B - y_A) + v \cdot (y_C - y_A) + w \cdot (y_D - y_A) \\ z = z_A + u \cdot (z_B - z_A) + v \cdot (z_C - z_A) + w \cdot (z_D - z_A) \end{cases} \tag{B.51}$$

The Jacobian of Φ is thus

$$J_\Phi = \begin{bmatrix} x_B - x_A & x_C - x_A & x_D - x_A \\ y_B - y_A & y_C - y_A & y_D - y_A \\ z_B - z_A & z_C - z_A & z_D - z_A \end{bmatrix} \tag{B.52}$$

And

$$\begin{aligned} |J_\Phi| = |&(x_B - x_A)(y_C - y_A)(z_D - z_A) \\ &+ (x_C - x_A)(y_D - y_A)(z_B - z_A) \\ &+ (x_D - x_A)(y_B - y_A)(z_C - z_A) \\ &- (x_D - x_A)(y_C - y_A)(z_B - z_A) \\ &- (y_D - y_A)(z_C - z_A)(x_B - x_A) \\ &- (z_D - z_A)(x_C - x_A)(y_B - y_A)| \end{aligned} \tag{B.53}$$

Now lets calculate the integral of f over $(ABCD)$:

$$I = \iiint_{(ABCD)} f \cdot dx \cdot dy \cdot dz = |J_\Phi| \iiint_{T_2} f \circ \Phi \cdot du \cdot dv \cdot dw \tag{B.54}$$

$$I = |J_\Phi| \int_{w=0}^{1} \left(\int_{v=0}^{1-w} \left(\int_{u=0}^{1-w-v} f(\Phi(u,v)) \cdot du \right) dv \right) dw \tag{B.55}$$

With

$$\begin{aligned} f(\Phi(u,v,w)) =& u \cdot [a \cdot (x_B - x_A) + b \cdot (y_B - y_A) + c \cdot (z_B - z_A)] + \\ & v \cdot [a \cdot (x_C - x_A) + b \cdot (y_C - y_A) + c \cdot (z_C - z_A)] + \\ & w \cdot [a \cdot (x_D - x_A) + b \cdot (y_D - y_A) + c \cdot (z_D - z_A)] + f_A \end{aligned} \tag{B.56}$$

By calculating successively the integrals and injecting the expressions of f_B, f_C, f_D:

$$I = \frac{1}{24}|J_\Phi|(f_A + f_B + f_C + f_D) \tag{B.57}$$

This result can be checked, for $f : x \rightarrow 1$, the integral I should give the volume of tetrahedron:

$$Vol_{ABCD} = \frac{1}{6}\left|\left(\vec{AB} \wedge \vec{AC}\right) \cdot \vec{AD}\right| \tag{B.58}$$

$$Vol_{ABCD} = \frac{1}{6}\left|\left(\begin{bmatrix} x_B - x_A \\ y_B - y_A \\ z_B - z_A \end{bmatrix} \wedge \begin{bmatrix} x_C - x_A \\ y_C - y_A \\ z_C - z_A \end{bmatrix}\right) \cdot \begin{bmatrix} x_D - x_A \\ y_D - y_A \\ z_D - z_A \end{bmatrix}\right| \tag{B.59}$$

$$Vol_{ABCD} = \frac{1}{6}|J_\Phi| \tag{B.60}$$

And if $f_A = f_B = f_C = f_D = 1$:

$$I = \frac{1}{24}|J_\Phi|(1+1+1+1) = \frac{1}{6}|J_\Phi| = Vol_{ABCD} \tag{B.61}$$

B.3 Intersection calculation

In this section a few algorithms usefull for geometrical intersection calculations are presented. These algorithms are used in chapter B.3.3 for surface to surface projections. Finally an extension to volumetric conservative interpolation method by element on element projection is discussed.

B.3.1 Intersection procedure for two segments

This section details the intersection procedure used to compute the intersection of two segments in space. While this problem seems trivial, proper implementation of the intersection of two segments in a 2D space should consider all possible intersection cases:

- No intersection,
- a single intersection point
- an intersection segment.

A generic procedure capable of handling these different cases is presented here. The input of the procedure is composed of:

- the two segments A and B, A is defined by $A_1 = (A_{1,x}, A_{1,y}$ and $A_2 = (A_{2,x}, A_{2,y}$, likewise B is defined by B_1 and B_2.
- a tolerance value ϵ

This procedure does not accept degenerate segments, i.e. $||A_1A_2|| > \epsilon$ and $||B_1B_2|| > \epsilon$.

The output is a set of points:

- no intersection results in an empty set,
- a single point of intersection results in a point set containing only one point.
- an intersection segment results in a point set containing 2 points, the points which define the intersection segment.

The first operation to do is to determine whether the segment vectors are co-linear, this can be done by checking if $\|det(A_1\vec{A}_2, B_1\vec{B}_2)\| \leq \epsilon$.

If the segment vectors $A_1\vec{A}_2$ and $B_1\vec{B}_2$ are co-linear, the two segments are parallel, they have either no intersection or they may have an intersection segment. To determine if the two segments may have an intersection a simple test is to check if $A_1\vec{A}_2$ and $A_1\vec{B}_1$ are co-linear (this can also be done using a determinant calculation). If $A_1\vec{A}_2$ and $A_1\vec{B}_1$ are co-linear, a simple procedure to determine the intersection segment is to project A_1, A_2, B_1 and B_2 on a vector of the line supporting the two segments (in this example $O = A_1$ and $u = normalize(A_1\vec{A}_2)$ where $normalize : \vec{u} \mapsto \frac{\vec{u}}{||u||}$). Then by comparing the projected coordinates of the 4 points it is relatively simple to deduce the two points defining the intersection segment. If no intersection is found return an empty set.

```
if $\|det(\vec{A_1A_2}, \vec{B_1B_2})\| \leq \epsilon$ then
    Direction vectors are parallel, either the segment do not intersect or their intersection is a segment
    calculate if $\vec{A_1A_2}$ and $\vec{A_1B_1}$ are parallel if $\|det(\vec{A_1A_2}, \vec{A_1B_1})\| > \epsilon$ then
        $\vec{A_1A_2}$ and $\vec{A_1B_1}$ are not parallel, no intersection possible
        return $\emptyset$ ;
    end
    $\vec{A_1A_2}$ and $\vec{A_1B_1}$ are parallel, calculate segment of intersection
    $\vec{u} \leftarrow normalize(\vec{A_1A_2})$ ;
    $O \leftarrow A_1$ ;
    $u^A_{min} \leftarrow min((A_1 - O) \cdot \vec{u}, (A_2 - O) \cdot \vec{u})$ ;
    $u^A_{max} \leftarrow max((A_1 - O) \cdot \vec{u}, (A_2 - O) \cdot \vec{u})$ ;
    $u^B_{min} \leftarrow min((B_1 - O) \cdot \vec{u}, (B_2 - O) \cdot \vec{u})$ ;
    $u^B_{max} \leftarrow max((B_1 - O) \cdot \vec{u}, (B_2 - O) \cdot \vec{u})$ ;
    $R \leftarrow \emptyset$ ;
    The point check within R should consider the tolerance $\epsilon$  This will return at most 2 points: it can return no points if no intersection found, if the segments are placed exactly one next to the other, then only one point is returned  if the segments intersect, then the two intersection points are returned  if $u^A_{min} \in [u^B_{min}, u^B_{max}]$ and $(O + u^A_{min}\vec{u}) \notin R$ then
        $R \leftarrow R \cup (O + u^A_{min}\vec{u})$ ;
    end
    if $u^A_{max} \in [u^B_{min}, u^B_{max}]$ and $(O + u^A_{max}\vec{u}) \notin R$ then
        $R \leftarrow R \cup (O + u^A_{max}\vec{u})$ ;
    end
    if $u^B_{min} \in [u^A_{min}, u^A_{max}]$ and $(O + u^B_{min}\vec{u}) \notin R$ then
        $R \leftarrow R \cup (O + u^B_{min}\vec{u})$ ;
    end
    if $u^B_{max} \in [u^A_{min}, u^A_{max}]$ and $(O + u^B_{max}\vec{u}) \notin R$ then
        $R \leftarrow R \cup (O + u^B_{max}\vec{u})$ ;
    end
    return $R$
end
```

Algorithm 9: A general calculation procedure of the intersection of two segments, case with $\vec{A_1A_2}$ and $\vec{B_1B_2}$ are co-linear

If the segment vectors $\vec{A_1A_2}$ and $\vec{B_1B_2}$ are not co-linear the intersection of the lines containing the segments A a and B is calculated. A simple way to do this is to write for the intersection point $M = (M_x, M_y)$:

$$\begin{cases} M \in (A_1A_2) \Leftrightarrow det(\vec{A_1M}, \vec{A_1A_2}) = 0 \\ M \in (B_1B_2) \Leftrightarrow det(\vec{B_1M}, \vec{A_1B_2}) = 0 \end{cases} \tag{B.62}$$

Therefore the following equation system can be written

$$\begin{cases} dy_A M_x - dx_A M_y = A_{1,x}dy_A - A_{1,y}dx_A = P \\ dy_B M_x - dx_B M_y = B_{1,x}dy_B - B_{1,y}dx_B = Q \end{cases} \tag{B.63}$$

with $dx_A = A_{2,x} - A_{1,x}$, $dy_A = A_{2,y} - A_{1,y}$ and likewise for dx_B and dy_B. The solution to this system is

$$\begin{cases} M_x = \frac{Qdx_a - Pdx_b}{-dy_a dx_b + dx_a dy_b} \\ M_y = \frac{Qdy_a - Pdy_b}{-dy_a dx_b + dx_a dy_b} \end{cases} \tag{B.64}$$

Then by checking the position of M with the points A_1 and A_2 it is simple to deduce if M is on the segment A. The same method is used to determine if M is on segment B.

If M is on both segments then return M, otherwise return an empty set.

```
if $\|det(\vec{A_1A_2}, \vec{B_1B_2})\| > \epsilon$ then
    In this case the support lines intersect, therefore there can be no intersection or a single
    intersection point ;
    Solve system of equations to find intersection point $M = (M_x, M_y)$ ;
    $dx_A \leftarrow A_{2,x} - A_{1,x}$ ;
    $dy_A \leftarrow A_{2,y} - A_{1,y}$ ;
    $dx_B \leftarrow B_{2,x} - B_{1,x}$ ;
    $dy_B \leftarrow B_{2,y} - B_{1,y}$ ;
    $P \leftarrow A_{1,x}dy_A - A_{1,y}dx_A$ ;
    $Q \leftarrow B_{1,x}dy_B - B_{1,y}dx_B$ ;
    $idet \leftarrow 1/det(\vec{A_1A_2}, \vec{B_1B_2})$;
    $M_x = (Qdx_A - Pdx_B)idet$ ;
    $M_y = (Qdy_A - Pdy_B)idet$ ;
    Check if $M$ in segment $A$ ;
    if $\vec{A_1M} \cdot \vec{A_1A_2} < -\epsilon$ or $\vec{A_2M} \cdot \vec{A_1A_2} > \epsilon$ then
        $M$ is outside of segment $A$ ;
        return $\emptyset$
    end
    Check if $M$ in segment $A$;
    if $\vec{B_1M} \cdot \vec{B_1B_2} < -\epsilon$ or $\vec{B_2M} \cdot \vec{B_1B_2} > \epsilon$ then
        $M$ is outside of segment $B$ ;
        return $\emptyset$
    end
    $M$ is inside segments $A$ and $B$ ;
    return $M$
end
```

Algorithm 10: A general calculation procedure of the intersection of two segments, case with $\vec{A_1A_2}$ and $\vec{B_1B_2}$ are not co-linear

B.3.2 Algorithm: The jarvis March - Convex Hull in 2D

This is a brief presentation of a simple convex hull algorithm called Jarvis March Fig. B.7, published by R. A. Jarvis in 1973 [50]. It's purpose is to find the convex hull of a set of points, in other words, finding the minimal convex polygon englobing the set of points. This algorithm is used to compute the intersection polygons after the computation of the intersection points. Lets consider a set of N points $p_0, p_1, \dots, p_N$ in a 2D space E.

1. First find a point that will be on the convex hull, for example the left most point (with the minimal x) this point is noted H_0.

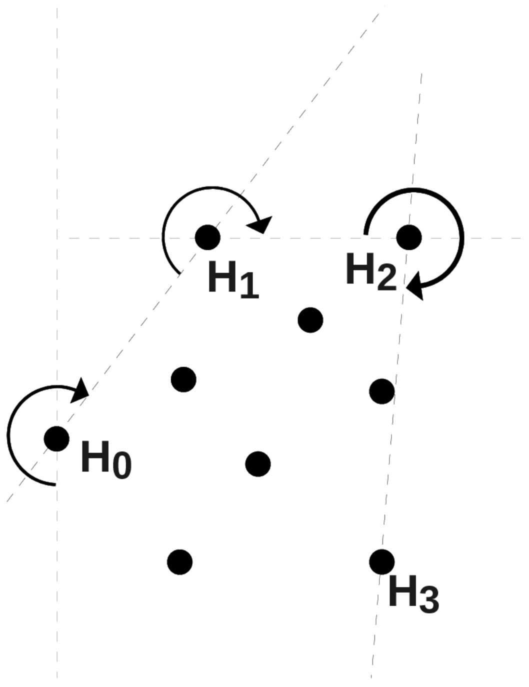


Figure B.7: The Jarvis March algorithm

2. At H_i calculate the polar coordinates of the other points relative to H_i and vector $\overrightarrow{H_{i-1}H_i}$ and select point that gives the minimal angle (left most point). Add this point to the Hull as H_{i+1}.

3. continue (2) until the selected point is H_0 which means the loop has been calculated, the convex hull is defined by $H_0, \ldots, H_M$.

This algorithm's complexity is $O(nh)$ where n is the number of points in the point set , h the number of points on the convex hull. More efficient algorithms exist to solve this problem, notably Graham scan, however in this problem the points sets are very small so using this algorithm is still acceptable.

This algorithm can not be extended to 3D, however other algorithms exist to solve this problem in 3 or more dimensions, notably incremental hull algorithm.

```
$pointOnHull \leftarrow leftmostpointinS$ ;
$i \leftarrow 0$ ;
repeat
    $P_i \leftarrow pointOnHull$;
    $endpoint \leftarrow S_0$;
    for $j \leftarrow 1$ to $Sizeof(S) - 1$ do
        if $S_j$ on left of line from $P_i$ to endpoint or$endpoint = pointOnHull$ then
            $endpoint \leftarrow S_j$;
        end
    end
    $pointOnHull \leftarrow endpoint$;
    $i \leftarrow i + 1$
until $endpoint = P_0$;
```

Algorithm 11: Sort intersection point set to obtain an intersection polygon - Convex Hull Algorithm(Jarvis march)

B.3.3 Possible method for volumetric conservative interpolation

This section describes an extension of conservative interpolation to volumes. To extend the conservative interpolation method to volumes it is necessary to compute the intersection of two tetrahedrons, then compute the integral of a data field on that intersection.

To terahedron/terahedron intersection procedure can be extended from the triangle/triangle intersection procedure described in chapter . Though the problem is slightly more complex it can be computed in the same way:

1 First compute the set of intersection polyhedron vertices.

2 Build the intersection polyhedron from the intersection polyhedron vertices.

The computation of the intersection polyhedron vertices can be carried out in the same way as for the triangle case: first all the vertices of each tetrahedron are tested to see if they are inside the other tetrahedron. The vertices which are found to be inside the other polyhedron are added to the intersection polyhedron point set. Then the intersection points coming from the tetrahedrons faces need to be computed. Since the faces are triangular this can be done by using the triangle/triangle intersection computation procedure (in chapter).

The intersection polyhedron vertices are now calculated, but like in the triangle/triangle intersection, the intersection polyhedron still needs to be constructed. Knowing that the intersection polyhedron must be convex (the intersection of two convex polyhedrons is convex), the intersection polyhedron can be constructed using convex hull algorithms. The Jarvis march (Gift Wrapping) algorithm described in appendix B.3.2 is only valid for 2D space, for 3D space different algorithms have to be considered such as Incremental Hull [94] or QuickHull [7, 94, 6]. In a demonstration program written using the code presented in appendix A.2, the intersection polyhedron of two arbitrary tetrahedrons is computed, Fig B.8. The algorithm used for this demonstration is Incremental Hull [94]. Though this algorithm is not very efficient (complexity $O(n^2)$), it is a very straightforward method to build a convex hull in arbitrary space and the tetrahedron intersection problem yields very small intersection point sets. If efficiency is a concern QuickHull (complexity $O(nlog(n))$) could be used instead.

The intersection polyhedron can be broken into a set of tetrahedrons (in the same way that a convex polygon can be broken into a triangle fan). Then using the formula presented in appendix B.2.2 on each tetrahedron of the intersection polyhedron decomposition it is possible to obtain an approximation of the integral on the intersection polyhedron. This integral is exact for a mesh of P1 elements. Using similar methods as described in chapter it is also possible to build a sparse matrix containing all the coefficients to compute those integrals. Finally this method can be used for hybrid mesh projection by breaking the mesh elements into tetrahedrons.

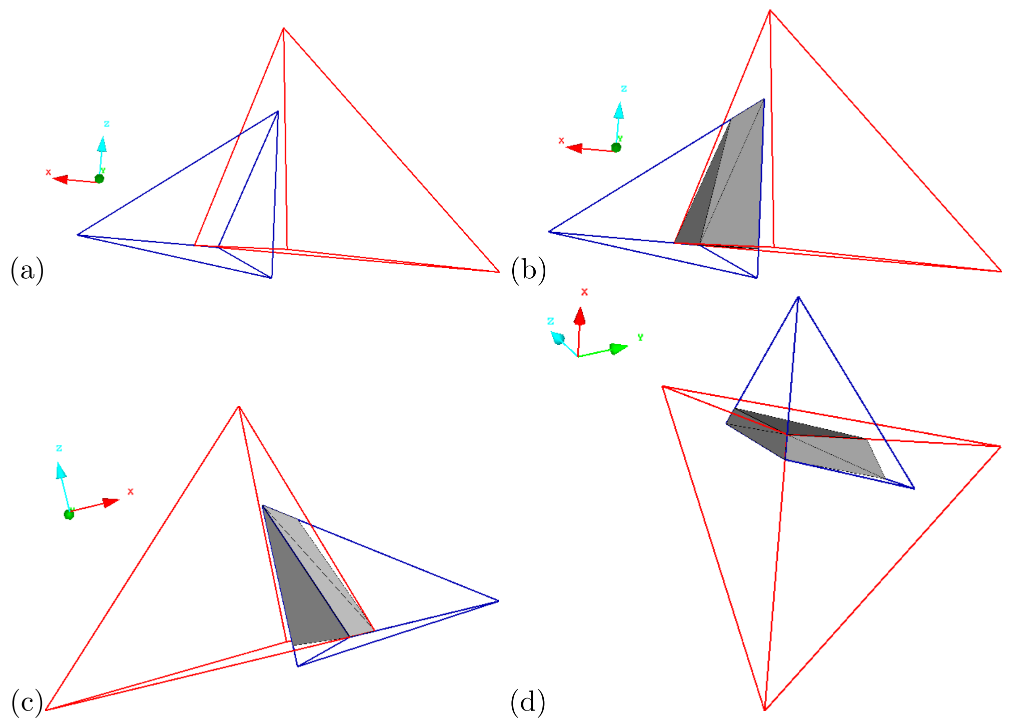


Figure B.8: Demonstration of the intersection of two tetrahedra. Two tetrahedra are presented in (a), their intersection polyhedron is computed by first computing the intersection points and then using incremental hull. Different views of the tetrahedrons and their intersection polyhedron are showed in (b),(c) and (d).

Bibliography

[1] Federal Aviation Administration. FAA Aerospace Forecast Fiscal Years 2012-2032 . Technical report, U.S. Department of Transportation Federal Aviation Administration Aviation Policy and Plans, 2012.

[2] J. Amaya, E. Collado, B. Cuenot, and T. Poinsot. Coupling LES, radiation and structure in gas turbine simulations. In NASA Ames/Stanford Univ. Center for Turbulence Research, editor, *Proc. of the Summer Program*, volume in press, 2010.

[3] G.M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM.

[4] A. Piacentini and. PALM: A Dynamic Parallel Coupler. In José Palma, A. Sousa, Jack Dongarra, and Vicente Hernández, editors, *High Performance Computing for Computational Science — VECPAR 2002*, volume 2565 of *Lecture Notes in Computer Science*, pages 355–367. Springer Berlin / Heidelberg, 2003. 10.1007/3-540-36569-9_32.

[5] ANSYS. *ANSYS FLUENT 12.1 User's Guide*, 2010.

[6] D. Avis, D. Bremner, and R. Seidel. How good are convex hull algorithms? *Computational Geometry*, 7(5–6):265–301, 1997. 11th ACM Symposium on Computational Geometry.

[7] C. Bradford Barber, D.P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE*, 22(4):469–483, 1996.

[8] T. Baritaud, T. Poinsot, and M. Baum. *Direct Numerical Simulation for Turbulent Reacting Flows.* Centre de recherche sur la combustion turbulent. Éditions Technip, 1996.

[9] Y. Bazilevs, M.C. Hsu, I. Akkerman, S. Wright, K. Takizawa, B. Henicke, T. Spielman, and T. E. Tezduyar. 3D simulation of wind turbine rotors at full scale. Part I: Geometry modeling and aerodynamics . *INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS*, July 2010.

[10] J. Blinn. *Jim Blinn's Corner: Dirty Pixels.* Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Morgan Kaufmann, 1998.

[11] C. Bogey and C. Bailly. Large eddy simulations of round free jets using explicit filtering with-/without dynamic Smagorinsky model. *International Journal of Heat and Fluid Flow*, 27(4):603–610, 2006. Special Issue of The Fourth International Symposium on Turbulence and Shear Flow Phenomena - 2005 - Special Issue of The Fourth International Symposium on Turbulence and Shear Flow Phenomena - 2005.

[12] M. Boileau, G. Staffelbach, B. Cuenot, T. Poinsot, and C. Bérat. LES of an ignition sequence in a gas turbine engine. *Combust. Flame*, 154(1-2):2–22, 2008.

[13] G. Boudier, L. Y. M. Gicquel, T. Poinsot, D. Bissières, and C. Bérat. Comparison of LES, RANS and Experiments in an Aeronautical Gas Turbine Combustion Chamber. *Proc. Combust. Inst.*, 31:3075–3082, 2007.

[14] G. Boudier, L. Y. M. Gicquel, T. Poinsot, D. Bissières, and C. Bérat. Effect of mesh resolution on Large Eddy Simulation of reacting flows in complex geometry combustors. *Combust. Flame*, 155(1-2):196–214, 2008.

[15] H. Boughanem and A. Trouvé. Validation du code de simulation directe NTMIX3D pour le calcul des écoulements turbulents réactifs. Technical Report 42907, Institut Français du Pétrole, 1996.

[16] B. A. Boville and P. R. Gent. The NCAR Climate System Model, version one. *J. Climate*, 11:1115–1130, 1998.

[17] T. Bray, J. Paoli, and C.M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. *W3C standards*, 2000.

[18] K.Y. Bülah and R.H. Scanlan. Resonance, tacoma narrows bridge failure, and undergraduate physics textbooks. *Am, J. Phys*, 59:2, 1991.

[19] S. Carlsson. Average-case results on heapsort. *BIT Numerical Mathematics*, 27(1):2–17, 1987.

[20] S. Chemin. *Etude des Interactions Thermiques Fluide-Structure par un Couplage de Codes de Calcul.* PhD thesis, Université de Reims Champagne Ardenne, 2006.

[21] G. Chesshire and W. Henshaw. A scheme for conservative interpolation on overlapping grids. *SIAM Journal on Scientific Computing*, 15(4):819–845, 1994.

[22] O. Colin, F. Ducros, D. Veynante, and T. Poinsot. A thickened flame model for large eddy simulations of turbulent premixed combustion. *Phys. Fluids*, 12(7):1843–1863, 2000.

[23] E. Collado-Morata, N. Gourdain, F. Duchaine, and L.Y.M. Gicquel. Effects of free-stream turbulence on high pressure turbine blade heat transfer predicted by structured and unstructured les. *International Journal of Heat and Mass Transfer*, (0):–, 2012.

[24] Computational Dynamics Limited. *STAR-CD Version 3.15, User Guide and Methodology Manuals*, 2001.

[25] COMSOL AB, Stockholm: COMSOL AB. *Multiphysics User's Guide*, 2005.

[26] COMSOL Conference. *Large Scale Simulation on Clusters using COMSOL 4.2* , October 2011.

[27] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C Stein. *Introduction to Algorithms*, pages 947–957. MIT Press and McGraw-Hill, Second Edition, 2001.

[28] S Corrsin. On the Spectrum of Isotropic Temperature Fluctuations in an Isotropic Turbulence. *Journal of Applied Physics*, 22(4):469–473, April 1951.

[29] J. W. Deardorff. A numerical study of three-dimensional turbulent channel flow at large Reynolds numbers. *Journal of Fluid Mechanics*, 41:453–480, 1970.

[30] Reaction Design. *THE CHEMKIN THERMODYNAMIC DATABASE*, 2000.

[31] E. R. Van Driest. Turbulent Boundary Layer in Compressible Fluids. *J. Aeronaut. Sci.*, 18(3):145–160, 216, 1951.

[32] F. Duchaine, A. Corpron, L. Pons, V. Moureau, F. Nicoud, and T. Poinsot. Development and assessment of a coupled strategy for conjugate heat transfer with Large Eddy Simulation. Application to a cooled turbine blade. *International Journal of Heat and Fluid Flow*, 30(6):Pages 1129–1141, 2009.

[33] U.S. Departement Of Energy. The Opportunities and Challenges of Exascale Computing. pages 12–13, 2010.

[34] J. Farrell. *Java Programming*. SAM 2010 Compatible Products Series. Course Technology, 2011.

[35] P.E. Farrell, M.D. Piggott, C.C. Pain, G.J. Gorman, and C.R. Wilson. Conservative interpolation between unstructured meshes via supermesh construction. *Computer Methods in Applied Mechanics and Engineering*, 198:2632–2642, 2009.

[36] D.A. Field. A generic delaunay triangulation algorithm for finite element meshes. *Advances in Engineering Software and Workstations*, 13(5â€“6):263–272, 1991.

[37] A. Fouilloux and A. Piacentini. The PALM Project: MPMD Paradigm for an Oceanic Data Assimilation Software. *Lecture Notes In Computer Science*, pages 1423 – 1430, 1999.

[38] H. Fuchs, Z.M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. *SIGGRAPH Comput. Graph.*, 14(3):124–133, July 1980.

[39] M.B. Giles. Stability analysis of numerical interface conditions in fluid-structure thermal analysis. *INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS*, 1997.

[40] A.S. Glassner. *An Introduction to Ray Tracing*. Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Academic Press, 1989.

[41] S.K. Godunov and V.S. Ryabenkii. The Theory of Difference Schemes. An Introduction. *North Holland, Amsterdam*, 1964.

[42] O. Goloubeva, M. Rebaudengo, M. Sonza Reorda, and M. Violante. Soft-error detection using control flow assertions. In *Defect and Fault Tolerance in VLSI Systems, 2003. Proceedings. 18th IEEE International Symposium on*, pages 581–588, nov. 2003.

[43] N Gourdain, L Gicquel, M Montagnac, O Vermorel, M Gazaix, G Staffelbach, M Garcia, JF Boussuge, and T Poinsot. High performance parallel computing of flows in complex geometries: I. Methods. *Comput. Sci. Disc.*, 2:015003, 2009.

[44] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI, Portable Parallel Programming with the Message Passing Interface*. The MIT Press, 2nd edition edition, 1999.

[45] F. Haugen. *Discrete-time signals and systems*. TechTeach, February 2005.

[46] Karlsson Hibbitt and Sorensen. *ABAQUS/standard: user's manual, version 5.7*. Number vol. 3 in ABAQUS/standard: User's Manual, Version 5.7. Hibbitt, Karlsson & Sorensen, 1997.

[47] C. Hirsch. *Numerical Computation of Internal and External Flows*. John Wiley, New York, 1988.

[48] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962.

[49] S.S. Hossain, S. Hossainy, Y. Bazilevs, V.M. Calo, and T.J.R. Hughes. Mathematical Modeling of Coupled Drug and Drug-encapsulated Nanoparticle Transport in Patient-Specific Coronary Artery Walls. Technical report, Institute for Computational Engineering and Sciences (ICES) The University of Texas at Austin, USA, 2011.

[50] R.A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2(1):18–21, 1973.

[51] R. Jenkins. Algorithm Alley. *Dr Dobbs Journal*, 22(9):107–110, 1997.

[52] W.P Jones and B.E Launder. The prediction of laminarization with a two-equation model of turbulence. *International Journal of Heat and Mass Transfer*, 15(2):301–314, 1972.

[53] W. Joppich and M. Kürschner. MpCCI—a tool for the simulation of coupled applications. *Concurrency and Computation: Practice and Experience*, 18:183–192, feb 2006.

[54] E.I. Jury. *Theory and Application of the Z-Transform Method.* Krieger Pub Co, 1973.

[55] B. A. Kader. Temperature and Concentration Profiles in Fully Turbulent Boundary Layers. *Int. J. Heat and Mass Transfer*, 24(9):1541–1544, 1981.

[56] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. Technical Report 98-019, University of Minnesota, Department of Computer Science/Army HPC Research Center, 1998.

[57] R.J. Kee, J.F. Grcar, M.D. Smooke, and J.A. Miller. FORTRAN program for modeling steady laminar one-dimensional premixed flames. *Sandia National Laboratories Report*, SAND 85-82(SAND85-8240):114 p, 1985.

[58] G. Kelsall and C. Troger. Prediction and control of combustion instabilities in industrial gas turbines. *Applied Thermal Engineering*, 24(11 - 12):1571–1582, August 2004.

[59] D. Knuth. *The Art of Computer Programming, volume 3, Sorting and Searching*, pages 506–542. Addison-Wesley, 1973.

[60] J. Krüger and R. Westermann. Linear algebra operators for GPU implementation of numerical algorithms. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[61] P.Y. Lagrée. Equation de la chaleur en instationnaire, March 2010.

[62] N. Lamarque. *Schémas numériques et conditions limites pour la simulation aux grandes échelles de la combustion diphasique dans les foyers d'hélicoptère.* Phd thesis, INP Toulouse, 2007.

[63] G. Lartigue, U. Meier, and C. Bérat. Experimental and numerical investigation of self-excited combustion oscillations in a scaled gas turbine combustor . *Applied Thermal Engineering*, 2004.

[64] B. E. Launder and B. I. Sharma. Application of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc. *Letters Heat Mass Transfer*, 1:131–137, December 1974.

[65] P. D. Lax and B. Wendroff. Difference schemes for hyperbolic equations with high order of accuracy. *Commun. Pure Appl. Math.*, 17:381–398, 1964.

[66] R. Löhner. *Applied Computational Fluid Dynamics Techniques: An Introduction Based on Finite Element Methods.* John Wiley & Sons, 2008.

[67] C.C. Long, Y. Bazilves, M.-C. Hsu, J. Feinstein, and A. Marsden. Fluid Structure Interaction Simulations of the Fontan Procedure using Variable Wall Properties. *International Journal for Numerical Methods in Biomedical Engineering*, 2012.

[68] A.W.S. Loo. *Peer-To-Peer Computing.* Computer Communications and Networks. Springer, 2007.

[69] L.G. Margolin and Mikhail Shashkov. Second-order sign-preserving conservative interpolation (remapping) on general grids. *Journal of Computational Physics*, 184(1):266–298, 2003.

[70] R. J. II Marks. *Introduction to Shannon Sampling and Interpolation Theory.* Springer-Verlag, 1991.

[71] R.J. Marks. *Advanced topics in Shannon sampling and interpolation theory.* Springer texts in electrical engineering. Springer-Verlag, 1993.

[72] P. Moin, K. D. Squires, W. Cabot, and S. Lee. A dynamic subgrid-scale model for compressible turbulence and scalar transport. *Phys. Fluids*, A 3(11):2746–2757, 1991.

[73] T. Möller, E. Haines, and N. Hoffman. *Real-Time Rendering.* Ak Peters Series. A.K. Peters, 2008.

[74] E. Morata Collado, N. Gourdain, F. Duchaine, and L.Y.M. Gicquel. Effect of free-stream turbulence on high pressure turbine blade heat transfer using structured and unstructured LES. submitted to IJHMT.

[75] V. Moureau. YALES2 home page on www.coria-cfd.fr.

[76] V. Moureau, P. Domingo, and L. Vervisch. Design of a massively parallel CFD code for complex geometries. *Comptes Rendus Mécanique*, 339(2–3):141–148, 2011. High Performance Computing.

[77] M. Naor and U. Wieder. A Simple Fault Tolerant Distributed Hash Table. In M. Kaashoek and Ion Stoica, editors, *Peer-to-Peer Systems II*, volume 2735 of *Lecture Notes in Computer Science*, pages 88–97. Springer Berlin / Heidelberg, 2003.

[78] F. Nicoud and F. Ducros. Subgrid-Scale Stress Modelling Based on the Square of the Velocity Gradient Tensor. *Flow, Turbulence and Combustion*, 62(3):183–200, 1999. 10.1023/A:1009995426001.

[79] F. Nicoud, F. Ducros, and T. Schønfeld. Towards Direct and Large Eddy Simulations of Compressible Flows in Complex Geometries. In *Notes in Numerical Fluid Mechanics*, pages 157–171. 1998.

[80] J. Nolen. Partial differential equations and diffusion processes. Technical report, Stanford University. Department of Mathematics, 2009.

[81] G. Oppattaiyamath, N. Reddy, S. Jammy, and V. Kulkarni. Conjugate Heat Transfer Analysis for Hypersonic Flow over Finite Thickness Flat Plate. *Journal of Aerospace Engineering*, November 2011.

[82] S.A. Orszag. Analytical theories of turbulence. *Journal of Fluid Mechanics*, 41(02):363–386, 1970.

[83] T. Passot and A. Pouquet. Numerical simulation of compressible homogeneous flows in the turbulent regime. *J. Fluid Mech.*, 181:441–466, 1987.

[84] T. Pedot. *Prediction of Coke/Deposit in Industrial Furnace Burner (Fluid Mechanics, Combustion, Heat).* PhD thesis, Institut National Polytechnique de Toulouse, 2011.

[85] G. Petrone, C. de Nicola, D. Quagliarella, J. Witteveen, and G. Laccarino. Analysis and Optimization of Wind Turbine Noise under Uncertainty. Technical report, Fourth International Meeting on Wind Turbine Noise, 2011.

[86] A. Piacentini, T. Moreland, A. Thévenin, and F. Duchaine. Open-PALM: an Open Source Dynamic Parallel Coupler. *Proceedings of Coupled Problems 2011*, pages 183–192, 2011.

[87] J. Pieprzyk and B. Sadeghiyan. *Design of hashing algorithms.* Lecture notes in computer science. Springer-Verlag, 1993.

[88] U. Piomelli, W. H. Cabot, P. Moin, and S. Lee. Subgrid-scale backscatter in turbulent and transitional flows. *Phys. FluidsA*, 3(7):1766–1771, July 1991.

[89] T. Poinsot and D. Veynante. *Theoretical and Numerical Combustion.* R.T. Edwards, 2001.

[90] S. B. Pope. *Turbulent flows*, chapter 10. Cambridge University Press, 2000.

[91] S. B. Pope. *Turbulent flows*, chapter 13. Cambridge University Press, 2000.

[92] S. B. Pope. *Turbulent flows*, chapter 6. The scales of motion, pages 182–189. Cambridge University Press, 2000.

[93] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction.* Texts and Monographs in Computer Science. Springer-Verlag, 1985.

[94] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction.* Texts and Monographs in Computer Science. Springer-Verlag, 1985.

[95] R. Redler, S. Valcke, and H. Ritzdorf. OASIS4 – a coupling software for next generation earth system modelling. *Geoscientific Model Development*, 3(1):87–104, 2010.

[96] J. Richard, F. Nicoud, et al. Towards the effect of the fluid ftructure coupling on the aeroacoustic instabilities of solid rocket motors, 2011.

[97] D.S. Rosenblum. A practical approach to programming with assertions. *Software Engineering, IEEE Transactions on*, 21(1):19–31, jan 1995.

[98] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In Rachid Guerraoui, editor, *Middleware 2001*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer Berlin / Heidelberg, 2001.

[99] P. Sagaut. *Large eddy simulation for incompressible flows: an introduction.* Scientific computation. Springer, 2006.

[100] S. Sampath. *Sampling Theory And Methods.* Alpha Science International, 2005.

[101] F. Sarghini, U. Piomelli, and E. Balaras. Scale-similar models for large-eddy simulations. *Phys. FluidsA*, 11(6):1596–1607, 1999.

[102] J.W. Sawyer. *Sawyer's Gas Turbine Engineering Handbook: Application.* Sawyer's Gas Turbine Engineering Handbook. Gas Turbine Publications, 1972.

[103] T. Schmitt. *Simulation des grandes échelles de la combustion turbulente en régime supercritique.* PhD thesis, Université de Toulouse - Ecole doctorale MEGeP, CERFACS - CFD Team, Toulouse, June 2009.

[104] T. Schönfeld and M. Rudgyard. Steady and unsteady flow simulations using the hybrid flow solver AVBP. *AIAA*, 37(11):1378–1385, 1999.

[105] H. Sehitoglu. *Thermomechanical Fatigue Behavior of Materials*, page 233. Number n° 1186 in ASTM special technical publication. ASTM, 1993.

[106] L. Selle, G. Lartigue, T. Poinsot, R. Koch, K.-U. Schildmacher, W. Krebs, B. Prade, P. Kaufmann, and D. Veynante. Compressible Large-Eddy Simulation of turbulent combustion in complex geometry on unstructured meshes. *Combust. Flame*, 137(4):489–505, 2004.

[107] M. Shevtsov, A. Soupikov, and A. Kapustin. Highly Parallel Fast Kd-tree Construction for Interactive Ray Tracing of Dynamic Scenes. *EUROGRAPHICS 2007*, 26, Number 3, 2007.

[108] D. Shreiner, B. Licea-Kane, and G. Sellers. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 4.1.* OpenGL Series. Addison Wesley Professional, 2012.

[109] H.D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2(Issue 2-3):135–148, 1991.

[110] J. Sitaraman, M. Floros, A. M. Wissink, and M. Potsdam. Parallel Unsteady Overset Mesh Methodology for a Multi-Solver Paraidgm with Adaptive Cartesian Grids. *26th AIAA Applied Aerodynamics Conference*, 2008.

[111] J. Smagorinsky. General Circulation Experiments with the Primitive Equations. *Monthly Weather Review*, 91:99, 1963.

[112] W. Richard Stevens and Gary R. Wright. *TCP/IP Illustrated: the protocols*, volume 1. Addison-Wesley Professional, 1994.

[113] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, August 2001.

[114] H. Tennekes and J.L. Lumley. *A First Course in Turbulence.* Mit Press, 1972.

[115] M. Teschner and B. Heidelberger. Optimized Spatial Hashing for Collision Detection of Deformable Objects. *Proc. VMV, Munich, Germany*, 2003.

[116] L. Thomason. Tinyxml website, 2000.

[117] Th. v. Karman. Mechanical similitude and turbulence. *National Advisory Committee for Aeronautics*, 1930.

[118] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4), 1997.

[119] Wickert. *Lectures Notes - Introduction to Signals and Systems*, chapter 4 - Sampling and Aliasing. University of Colorado Colorado Springs, 2010.

[120] P. Wolf, G. Staffelbach, A. Roux, L. Gicquel, T. Poinsot, and V. Moureau. Massively parallel LES of azimuthal thermo-acoustic instabilities in annular gas turbines. *C. R. Acad. Sci.Mécanique*, 337(6-7):385–394, 2009.

[121] H.J. Wolfson and I. Rigoutsos. Geometric Hashing: An Overview. *IEEE Comput. Sci. Eng.*, 4(4):10–21, October 1997.

[122] V. Yakhot, S. A. Orszag, S. Thangam, T. B. Gatski, and C. G. Speziale. Development of turbulence models for shear flows by a double expansion technique. *Physics of Fluids A: Fluid Dynamics*, 4(7):1510–1520, 1992.

[123] Z. Yosibash and E. Priel. Simulating the coupled active and passive mechanical response of the artery wall by high order finite elements. Technical report, Computational Mechanics Laboratory, Mechanical Engineering Department,Ben-Gurion University of the Negev, Beer-Sheva, Israel, 2011.

[124] K. Zhou, Q. Hou, R. Wang, and B. Guo. Real-time KD-tree construction on graphics hardware. *ACM Trans. Graph.*, 27(5):126:1–126:11, December 2008.

[125] U. Zurcher, J.C. Badoux, and M. Mussard. The World's First Industrial Gas Turbine Set at Neuchâtel (1939). An International Historic Mechanical Engineering Significance of Landmark, September 1988.