Fast, flexible Surface to Surface thermal radiative transfer for sintering applications

Rémi Gerard

▶ **To cite this version:**

Rémi Gerard. Fast, flexible Surface to Surface thermal radiative transfer for sintering applications. Mechanics of materials [physics.class-ph]. Université Paris sciences et lettres, 2022. English. NNT : 2022UPSLM057 . tel-04083277

HAL Id: tel-04083277
https://pastel.hal.science/tel-04083277

Submitted on 27 Apr 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à MINES Paris

Fast, flexible Surface to Surface thermal radiative transfer for sintering applications
Transfert thermique par rayonnement rapide et flexible pour des applications au frittage

Soutenue par
Rémi GERARD
Le 4 juillet 2022

École doctorale n°364
Sciences Fondamentales et Appliquées

Spécialité
Mécanique numérique et matériaux

Composition du jury :

Assaad ZOUGHAIB Professeur, Mines Paris	*Président*
Joan BAIGES Professeur, Universitat Politècnica de Catalunya	*Rapporteur*
Charbel HABCHI Professeur, Notre Dame University	*Rapporteur*
Mireille BOSSY Directrice de recherche, INRIA Sophia Antipolis	*Examinatrice*
Aurélien LARCHER Docteur, Mines Paris	*Co-encadrant de thèse*
Elie HACHEM Professeur, Mines Paris	*Directeur de thèse*

MINES ParisTech | PSL

To my wife Lauriane and my family

Acknowledgements

I would like to thank, first and foremost, my two supervisors Elie Hachem and Aurélien Larcher. Their help, their patience and their competence were fundamental in this work. Their different but complementary insights about many areas of the thesis have guided me throughout these three years of hard work. I thank them too for the countless inspiring discussions I have been able to share with them, both about specific Ph.D. matters and general musings about research and science. I call myself lucky to have worked under their combined tuition.

I have also had the great opportunity to work in collaboration with SCC through Lucas Sardo and Giulia Lissoni. They have relentlessly tried and upgraded our work to an industrial scale, and provided invaluable industrial simulations. It is through their work that the results of this thesis have been validated.

This thesis project is an original idea from Thalès Alenia Space, who funded it during its whole duration. Fine Ceramics Technologies also contributed a great deal through and through. Working with Thomas, Nisrine, Olivier, Daniel, Uwe and of course Stéphanie was very enriching, and the relationship we have built between our team at CFL and them all was productive, driven but also very cordial. I thank them all here for the technical discussions, the tours they have given us in their manufacturing halls in Cannes and Rauenstein, and the matters of industrial application they ever stood for which is also an important tenet of research done at Mines Paris.

Cemef is a laboratory with a peculiar atmosphere. Between the sea, the mountains and the forest, working there was a pleasant experience, but most of all because of my fellow doctorate students. Not being alone in one's work is always good, but living for three years with that bolsterous team of research mates was a neverending storm of laughter, friendship and sometimes even sport. Thank you, *les nazes.* You know who you are.

Finally, my thanks go to my family. I have lost count of how many Paris-Antibes trains I have taken to see them, but each and everyone of them was worth it - in addition to the fact they made the trip from their side a few times too. Their support was never faltering, during the good days and the bad ones. And of course, I will conclude with a huge thanks to my dear Lauriane who was my girlfriend when I started this Ph.D., and my wife when I finished it. Without her support and

patience, this work would have not seen completion.

Contents

Contents iii

List of Figures vii

List of Tables xi

1 Introduction 1
- 1.1 Résumé . . . 2
- 1.2 General context . . . 3
- 1.3 Fundamental equations . . . 5
  - 1.3.1 The equation of conservation of energy . . . 5
  - 1.3.2 Flux density as a function of radiance . . . 6
  - 1.3.3 Laws of black-body radiation . . . 9
  - 1.3.4 The radiative transfer equation . . . 13
- 1.4 Radiative transfer state of the art . . . 15
  - 1.4.1 The Discrete-Ordinate Method . . . 15
  - 1.4.2 The DTRM . . . 16
  - 1.4.3 The P_1 method . . . 17
  - 1.4.4 The Monte-Carlo method . . . 19
  - 1.4.5 The Surface-to-Surface method . . . 21
  - 1.4.6 Overview of the different methods . . . 22
- 1.5 Summary . . . 22
  - 1.5.1 Objective of the thesis . . . 23
  - 1.5.2 Thesis outline . . . 24

Bibliography . . . 25

2 Immersed Surface-to-Surface modelling 33
- 2.1 Résumé . . . 34
- 2.2 Numerical framework . . . 35
  - 2.2.1 Interface description . . . 36
- 2.3 S2S radiation . . . 37

2.3.1 Overview and related work . . . 37
2.3.2 S2S hypotheses . . . 38
2.3.3 Facet list construction . . . 45
2.3.4 Flux computation . . . 46
2.3.5 Coupling with thermal transfer . . . 58
2.4 Numerical results . . . 59
2.4.1 Simple radiation . . . 59
2.4.2 Conduction–radiation . . . 64
2.5 Summary . . . 79
Bibliography . . . 81

3 Kd-tree acceleration structure **85**
3.1 Résumé . . . 86
3.2 Naive obstruction computation . . . 87
3.2.1 Local half-space culling . . . 88
3.3 Binary tree structures . . . 88
3.3.1 Metanode kd-trees . . . 89
3.3.2 Bounding volume kd-trees . . . 126
3.4 Parallel computation . . . 141
3.4.1 Radiosity preprocessing . . . 141
3.4.2 Thermal transfer . . . 147
3.5 Summary . . . 150
Bibliography . . . 150

4 Industrial applications **155**
4.1 Résumé . . . 156
4.2 Sintering oven . . . 157
4.3 Simulation meshing and setup . . . 161
4.3.1 Ferrule setup . . . 161
4.3.2 Three loads setup . . . 163
4.3.3 Hat setup . . . 166
4.4 Results and discussion . . . 168
4.4.1 Steady state ferrule results . . . 168
4.4.2 Ferrule temperature evolution . . . 170
4.4.3 Steady state three loads results . . . 175
4.4.4 Three loads temperature evolution . . . 178
4.4.5 Steady state hat results . . . 182
4.4.6 Hat temperature evolution . . . 184
4.4.7 Discussions . . . 189
4.5 Summary . . . 191
Bibliography . . . 192

5 Conclusions and Perspectives **193**
5.1 Conclusions 194
5.2 Perspectives 195

List of Figures

1.1 Sintering oven . . . 3
1.2 Memory footprint for simulation of some physical processes . . . 4
1.3 Ceramic T-squares . . . 4
1.4 Description of radiance . . . 7
1.5 Description of flux density . . . 7
1.6 Description of irradiance and radiant exitance . . . 9
1.7 Black-body spectral emissive power . . . 11
1.8 Spectral emissivity of chromium . . . 12
1.9 Quadrature S_6 . . . 15
1.10 Comparison between DOM, FVM, and DTRM . . . 16
1.11 Comparison between DTRM and FVM . . . 17
1.12 Comparison between P3 and FVM . . . 19
1.13 Monte-Carlo method complexity curve . . . 20

2.1 Spectral directional emissivity of SiC . . . 40
2.2 Spectral normal emissivity of SiC . . . 42
2.3 Vacuum total hemispherical emissivity of SiC . . . 43
2.4 2d intersections of elements by the solid interface. . . . 45
2.5 Relative position of elementary surfaces dS_i and dS_j. . . . 48
2.6 Crossed strings method . . . 49
2.7 Same view factor given by the projection of different shapes . . . 50
2.8 Spherical or cubic mesh . . . 51
2.9 Tetrahedra-based obstruction check . . . 55
2.10 Pure radiation test case in concentric circles . . . 60
2.11 Pure radiation test case in homothetic squares. . . . 61
2.12 Pure radiation test case in cube within cube. . . . 63
2.13 Conduction–radiation test case . . . 65
2.14 Mesh used for the infinitely long rods . . . 66
2.15 Thermal propagation and radiative flux in the rods. . . . 66
2.16 Electrical analogy of thermal transfer through the rods. . . . 67
2.17 Progression of the temperature inside the rod . . . 71

2.18 Immersion of a 2D crown in the domain. . . . 73
2.19 Thermal propagation and radiative flux density in the rods . . . 74
2.20 Progression of the temperature inside the rod . . . 76
2.21 Immersion of two spheres in the spherical domain . . . 77
2.22 Progression of the temperature through the concentric spheres . . . 79

3.1 Leaf-based 2D kd-tree . . . 90
3.2 Leaf-and-inner based 2D kd-tree . . . 91
3.3 Axis-aligned bounding boxes . . . 93
3.4 Metanode structure . . . 96
3.5 kd-tree generation time . . . 100
3.6 Bunny, dragon and Buddha . . . 102
3.7 Standard KNN search . . . 103
3.8 Pathing of the unwind algorithm . . . 109
3.9 Inner axis-aligned bounding boxes . . . 115
3.10 Sequential ray tracing through kd-tree . . . 117
3.11 Various kd-tree ray tracing algorithms . . . 122
3.12 BV kd-tree . . . 127
3.13 BV kd-tree generation, steps 1 and 2 . . . 133
3.14 BV kd-tree generation, step 4 . . . 134
3.15 BV kd-tree generation, steps 6 and 7 . . . 135
3.16 BV kd-tree generation time . . . 136
3.17 Double ray tracing for obstruction . . . 140
3.18 False positives in obstruction computation . . . 141
3.19 Flux density of the hat . . . 142
3.20 Computation time for S2S steps for the hat. . . . 143
3.21 Flux density of the ferrule . . . 145
3.22 Computation time for S2S steps for the ferule. . . . 146
3.23 Computation time for thermal transfer in the hat . . . 148
3.24 Computation time for thermal transfer in the ferrule . . . 149

4.1 Industrial simulation geometries . . . 158
4.2 Sintering oven . . . 159
4.3 FCT oven schematics . . . 160
4.4 Ferrule and cubic domain mesh. . . . 162
4.5 Three parts and cylindric domain mesh. . . . 165
4.6 Hat and parallelepipedic domain mesh. . . . 167
4.7 Flux and temperature on the surface of the ferrule . . . 169
4.8 Temperature inside the ferrule . . . 171
4.9 Numerical thermal probes on the ferrule . . . 172
4.10 Ferrule probes 1 to 3 and 13 to 15 . . . 173

4.11 Ferrule probes 4 to 6 and 16 to 18 . . . 173
4.12 Ferrule probes 7 to 12 . . . 174
4.13 Temperature for the three parts . . . 176
4.14 Flux density for the three parts . . . 177
4.15 Numerical thermal probes on the 3 parts . . . 179
4.16 Ferrule probes 1 to 3 and 13 to 15 . . . 180
4.17 Parallelepiped probes . . . 181
4.18 Cylinder probes . . . 182
4.19 Surface flux density and temperature for the hat . . . 183
4.20 Bulk temperature for the hat . . . 184
4.21 Physical and numerical temperature probes placed in the hat. . . . 185
4.22 Hat probe 5. . . . 185
4.23 Hat probe 1. . . . 186
4.24 Hat probe 14. . . . 188
4.25 Hat probe 6. . . . 189

List of Tables

1.1 Radiation modellings comparison 22

2.1 Concentric circles pure radiation case 61
2.2 Square within square pure radiation case 62
2.3 cube within cube pure radiation case 64
2.4 Long rods conduction-radiation case 70
2.5 Concentric crown conduction–radiation case 75
2.6 Concentric spheres conduction–radiation case 78

3.1 Stanford images tree generation time 101
3.2 Construction choices for metanode kd-tree and BV kd-tree. 128

4.1 Ferrule physical details 162
4.2 Ferrule simulation details 163
4.3 Three parts physical details 164
4.4 Three parts physical details (continued) 164
4.5 Ferrule simulation details 164
4.6 Hat physical details 167
4.7 Ferrule simulation details 168

List of Algorithms

1 Naive algorithm for obstruction computing 87
2 Local half-space culling algorithm 89
3 Locate point into tree algorithm 92
4 Naive recursive kd tree generation algorithm 94
5 Metanode construction algorithm 95
6 kd-tree sequential build algorithm 99
7 Standard Nearest-Neighbour searching algorithm 107
8 Semi-stack-based Nearest-Neighbour searching algorithm 108
9 Unwinding the inner tree of the metanode 110
10 Walking down the subtrees 114
11 Recursive travel down procedure using inner AABBs 116
12 Sequential algorithm for kd-tree traversal 119
13 Branchless locate in metanode algorithm 123
14 Metanode-adapted kd-backtrack 124
15 BV kd-tree generation 131
16 BV kd-tree traversal 137

Chapter 1

Introduction

Contents

1.1 Résumé 2
1.2 General context 3
1.3 Fundamental equations 5
1.3.1 The equation of conservation of energy 5
1.3.2 Flux density as a function of radiance 6
1.3.3 Laws of black-body radiation 9
1.3.4 The radiative transfer equation 13
1.4 Radiative transfer state of the art 15
1.4.1 The Discrete-Ordinate Method 15
1.4.2 The DTRM 16
1.4.3 The P_1 method 17
1.4.4 The Monte-Carlo method 19
1.4.5 The Surface-to-Surface method 21
1.4.6 Overview of the different methods 22
1.5 Summary 22
1.5.1 Objective of the thesis 23
1.5.2 Thesis outline 24
Bibliography 25

1.1 Résumé

Cette introduction présente le problème de cette thèse : la modélisation du transfert thermique par rayonnement dans le cadre du frittage de la céramique. Tout d'abord, les enjeux spécifiques du rayonnement par rapport aux autres formes de transfert thermique sont abordés : forte dépendance directionnelle et coût de calcul supérieur. Les équations qui décrivent le transfert radiatif sont introduites. Elles permettent de définir les grandeurs pertinentes de cette étude.
La résolution numérique de ces équations pose des difficultés particulières qui ont amené au développement de nombreuses méthodes de modélisation du transfert radiatif. Un aperçu de l'état de l'art de ces méthodes est présenté, en exposant les cas d'applications idéaux de chacune. Dans le cas propre du frittage de céramique, opération qui prend place sous vide, la méthode qui nous a paru la plus adéquate est la méthode dite "surface to surface" (S2S). Nous développons brièvement son concept fondateur.
La démarche globale de la thèse est alors présentée. Le partenaire industriel, Thalès Alenia Space, recherche un modèle de frittage de la céramique permettant de prédire les propriétés mécaniques locales du matériau en fonction des conditions de frittage. Cette thèse représente la première moitié de cette démarche, où l'on vise à prédire le chargement thermique du matériau durant le frittage. Pour ce faire, nous nous proposons d'utiliser la méthode S2S en combinaison avec une approche de maillage immergé pour définir la céramique dans le four. De plus, nous faisons usage d'arbres binaires pour ordonner les éléments rayonnants et accélérer la résolution numérique du problème. La conjonction de ces trois éléments est nouvelle et constitue le principal apport de cette thèse.
Le plan sera le suivant. Le premier chapitre détaillera la formulation de maillage immergé choisie, ainsi que le fonctionnement précis de la méthode S2S et les choix d'hypothèses adoptées. Une batterie de cas tests mis en regard de solutions analytiques est effectuée afin de valider la précision du modèle retenu. Le chapitre suivant expliquera les arbres binaires choisis pour accélérer la résolution du modèle S2S. Deux sortes d'arbres kd sont présentés, et leurs mérites respectifs sont discutés. Une brève bibliographie pour chaque sorte d'arbre est présentée. Les différents choix d'implémentation effectuée sont expliqués et justifiés. Enfin, la performance en calcul parallèle est aussi étudiée. Le dernier chapitre se concentre sur trois cas d'application sur des pièces industrielles réelles. Des cycles de frittage simple sont appliqués sur chaque pièce, et on étudie la cartographie thermique rendue par le modèle dans chaque cas. Les champs de température et de densité de flux thermique sont étudiés et commentés. Pour le dernier cas, des données expérimentales de frittage sont à disposition. On compare notre modèle à ces données pour le valider. Enfin, une conclusion reprend les différentes avancées de ce travail et évoque des pistes de recherche supplémentaire.

1.2 General context

Radiation is an important part of thermal transfer. It plays a dominant role over conduction and convection in a number of industrial applications that involve locally or globally high temperatures, such as in furnace vitrification [1–3], charcoal combustion chambers [4–6], steel ingot casting [7–9], thermal shielding [10–12], additive manufacturing [13–15] or photovoltaic central light collection [16–18]. Radiation is also dominating thermal transfer in ceramic sintering ovens, like those used by Thalès Alenia Space (TAS) for satellite parts (Figure 1.1).

Figure 1.1: Vacuum induction-based sintering oven ([19]).

Nowadays, study and optimization of thermal transfer in any of these industrial processes is almost always carried out with the help of a numerical model for simulation purposes. The amount of computational power at hand is a limit to the accuracy of the models the industrial user can deploy, as the most powerful models are often also the most demanding in computer resources. In this context, thermal radiation is the most computationally expensive thermal transfer mode to properly model. Figure 1.2 indicates the amount of RAM required by the Comsol solver to solve different physical equations pertaining to different physical processes, the number of degrees of freedom being the same across the simulations. One can see how thermal radiation modelling memory cost increases much faster than any other modelling.

This is due to several reasons. The Radiative Transfer Equation (RTE) that describes how radiation is exchanged among objects is highly nonlinear, since it includes the surface temperature of the objets to the power of four. By contrast, conduction or convection thermal fluxes are linearly dependant on the temperature

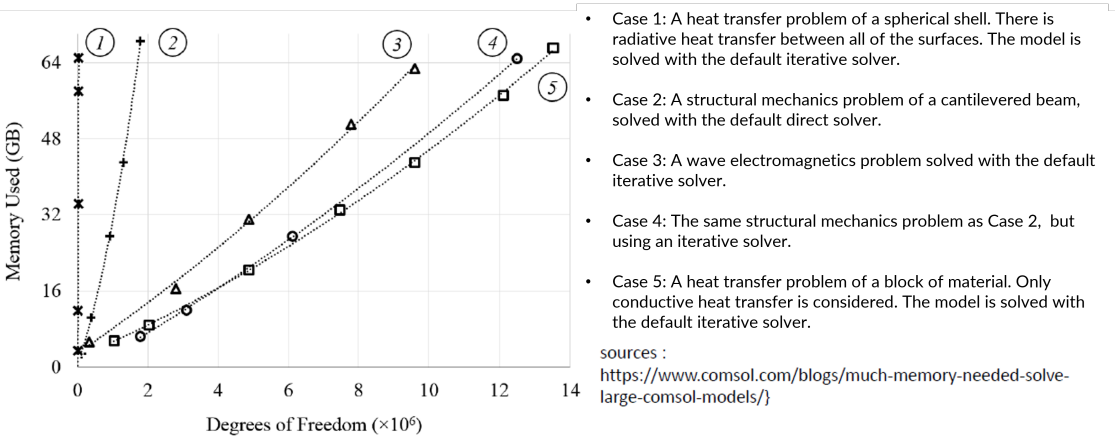


Figure 1.2: Memory footprint to solve different physical equations for a given number of degrees of freedom by Comsol ([20]).

of the objects. Another major hurdle is the high dimensionality of the RTE. Position in space and relative orientation between radiating objects matter a lot in the expression of radiative transfer between them, which means the RTE includes two angular dimensions in addition to three spatial dimensions. This also includes radiation upon itself for radiating items with nonconvex and convoluted shapes, like Since radiation is also dependant on wavelength, this means yet another dimension to the equation. Finally, the matter of radiation becomes even more complex when some of the participating objects can't be assimilated to a collection of zero-thickness radiating surfaces and possess a significant volumic behaviour towards radiation. This includes a soot-rich atmosphere in a furnace ([21]), a semitransparent glass container ([22]), or a plankton-rich ocean surface pierced by a sun ray ([23]).

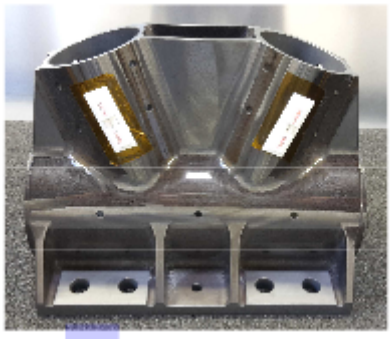


Figure 1.3: Ceramic T-squares from Thalès Alenia Space ([24]).

Radiation upon simple-shaped, standard items like steel ingot has been detailed in the litterature ([25]), but as soon as we wish to model radiation for objects with

more convoluted shapes, like the ceramic parts routinely used by TAS (see Figure 1.3), we have to deal with all these aforementioned issues. This problem has urged the scientific community to develop a number of radiation models over the years. While some of these methods, such as the P1 model [4, 26, 27] or the Rosseland approach [10, 28, 29] average out the contribution of radiation as a boundary condition, some others, like the DOM [30–32], the DTRM [32–34] or the Monte-Carlo ray tracing [35–37] instead account for the various trajectories of the heating rays. A powerful and adaptable modelling, the Surface-to-Surface (S2S) or radiosity method, has been recently gaining a lot of traction, providing a good compromise between computational cost and accuracy [20, 38–43]. This method allows to account for the directionality of radiation and strong local variations when the radiating surfaces are diffusive and approximated as grey bodies.

1.3 Fundamental equations

1.3.1 The equation of conservation of energy

The sintering ovens used by TAS contain either vacuum (for silicon carbide ceramics) or N_2 gas (for silicon nitride ceramics). The walls contain electrical coils that heat them by induction. From then, the walls, forming an enclosure, radiate heat towards the ceramic parts that have been laid inside. An order of magnitude for oven temperature is 1600 °C to 2200 °C. Ceramics is at first approximation a very opaque material, meaning radiation does not penetrate it, and thermal radiation received by the ceramics amounts to a thermal boundary condition. This will be added to the conduction thermal transfer the ceramics receives from the support on which it lays. In the case of ovens operating under an N_2 atmosphere, convection will contribute to the thermal boundary condition too. From then on, heat will propagate in the bulk of the ceramics through conduction. The ceramics is considered immobile as a first approximation during the sintering. Equation of energy conservation in the ceramics reads ([44]):

$$\int_V \rho c_p \frac{\partial T}{\partial t} dV = \int_V \dot{Q}_{generated} dV - \int_S (\vec{Q}_{conduction} + \vec{Q}_{convection} + \vec{Q}_{radiation}) \cdot \vec{n} dS \quad (1.1)$$

- $\dot{Q}_{generated}$ is null for the ceramics. For the oven however it amounts to the volumic power brought in by the electrical spires over time.
- $\vec{Q}_{conduction}$ amounts to the flux density the ceramics receives from the support on which it lays. Locally, it reads $Q_{conduction} = \frac{T_{support} - T}{R_c}$, with R_c being the thermal contact resistance. However, the support is heated by the oven almost

the same way the ceramics is, and the temperature difference between them is small. In addition, graphite felt is often used as the support or the crucible for the ceramic ([45–47]), and is a good thermal insulator (its conductivity is of the order of $2\,\mathrm{W\,K^{-1}\,m^{-1}}$). As a result, as a first approximation, this flux is small before the radiative flux the ceramics receives.

- $\vec{Q}_{convection}$ amounts to the convective flux density between the ceramics and the oven atmosphere, when there is one. It reads $Q_{convection} = h(T_{air} - T)$, h being the heat transfer coefficient. It relies on the volumic mass of the considered atmosphere. To further simplify our problem, we will first consider the sintering taking place under a vacuum ($10^{-5}\,\mathrm{mbar}$). For this reason, this flux can be neglected too.

- $\vec{Q}_{radiation}$ amounts to the radiative flux density arriving at an elementary surface of the ceramics from all its environment (and including from itself, if the ceramics is of a concave shape). Throughout this thesis, flux density will be considered, unless otherwise specified, to be an *emitted* flux by convention. This means negative values for flux density are ascribed to surfaces that are gaining energy through radiative transfer, while positives values describe surfaces that are instead losing energy through radiative transfer.

  With our chosen set of hypotheses, radiative flux density is the dominating term on the right of the equation. This assumption is actually found fairly often in studies of thermal transfer in ceramic sintering: Li and Chen [45], Li et al. [46, 47] also disregard conduction and convection to only keep radiation. Thus, correctly estimating $\vec{Q}_{radiation}$ at each point of the ceramics surface is of paramount importance to properly solve the energy conservation equation. Our aim is then to evaluate this quantity.

1.3.2 Flux density as a function of radiance

The exact physical meaning of the flux density, expressed in $\mathrm{W\,m^{-2}}$, is not so trivial. In order to better explain it, an important parameter to consider is the *spectral radiance* of thermal radiation $I_\lambda(x, t, \vec{u})$, expressed in $\mathrm{W\,m^{-3}\,sr^{-1}}$. It corresponds to the amount of power brought by radiation at a point x centered on an elementary surface dS at time t, contained in an elementary cone of solid angle $d\Omega$, according to a direction $\vec{u}$ and following a wavelength λ. Integrating spectral radiance over all directions yields *spectral flux density* $\vec{Q}_{radiation,\lambda}$, also called *spectral emissive power*. In the domain of thermal transfer, when no significant wavelength-dependant phenomena are considered, an often used value is the *radiance* $I(x, t, \vec{u})$, expressed in $\mathrm{W\,m^{-2}\,sr^{-1}}$, which is equal to the spectral radiance but integrated over all wavelengths. Figure 1.4 illustrates the concept of radiance. The term radiance is used

to describe both ingoing and outgoing radiation, relatively to the normal of the considered elementary surface.

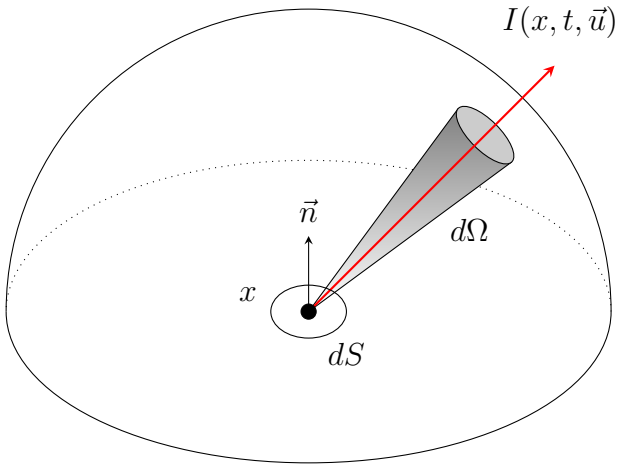


Figure 1.4: Radiance leaving the elementary surface dS. Spectral radiance would be equally illustrated.

Then, flux density arriving at or leaving through surface dS is simply the integration of radiance from all directions $\vec{u}$ of the unit sphere. Figure 1.5 illustrates flux density through a surface dS once radiance has been integrated over all directions.

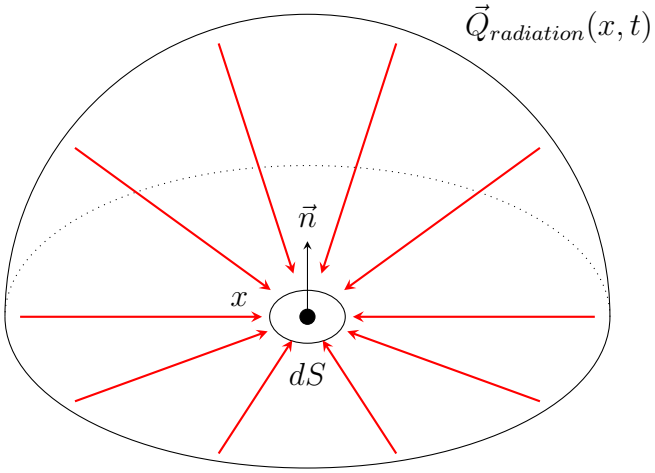


Figure 1.5: Illustration of flux density arriving at, or leaving, surface dS. Spectral flux density would be represented the same way.

In the most general case, this relationship is described by the following equation:

$$\vec{Q}_{radiation}(x,t) = \int_{4\pi} I(x,t,\vec{u})\vec{u}d\Omega \tag{1.2}$$

Note that the elementary surface dS belongs to a plane that defines two half-spaces, one being *above* and another one *below* (relatively to its normal). The elementary surface usually sits at the surface of some radiating object, which means the two half-spaces are composed of two different materials (like ceramics and air, or metal and vacuum, etc.). In a lot of cases, the normal of the elementary surface is defined so that the above half-space contains air or vacuum (a transmitting medium), while the below half-space contains a solid that is considered non-transmitting. Then, radiance in the below half-space is null as no radiation can take place, and only radiance in the above half-space is relevant. The corresponding flux density is then defined by integrating radiance over the directions describing only the above unit hemisphere, instead of the total unit sphere. Equation 1.2 then becomes:

$$\vec{Q}_{radiation}(x,t) = \int_{2\pi_{above}} I(x,t,\vec{u})\vec{u}d\Omega + \int_{2\pi_{below}} I(x,t)\vec{u}d\Omega \tag{1.3}$$

$$\vec{Q}_{radiation}(x,t) = \int_{2\pi_{above}} I(x,t,\vec{u})\vec{u}d\Omega \tag{1.4}$$

$$\tag{1.5}$$

However, one notes that the final value of interest for the ceramic is not exactly the vector field $\vec{Q}_{radiation}$. We can see in Equation 1.1 that what we actually want to compute is the flux of $\vec{Q}_{radiation}$ through the complete ceramic surface S, noted $W_{radiation}$ in W:

$$W_{radiation} = \int_S \vec{Q}_{radiation} \cdot \vec{n}dS \tag{1.6}$$

By expanding $\vec{Q}_{radiation}$ as per Equation 1.5 we obtain:

$$W_{radiation} = \int_S (\int_{2\pi} I(x,t,\vec{u})\vec{u} \cdot \vec{n}d\Omega)dS \tag{1.7}$$

This allows us to introduce the *scalar flux density* $Q_{radiation}$ as defined per

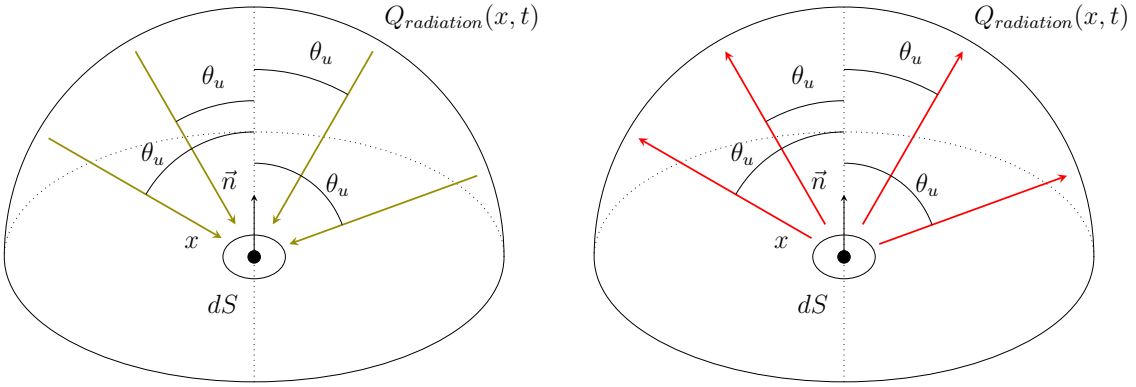
$$Q_{radiation}(x,t) = \int_{2\pi} I(x,t,\vec{u})\vec{u} \cdot \vec{n}d\Omega \tag{1.8}$$

$$Q_{radiation}(x,t) = \int_{2\pi} I(x,t,\vec{u})\cos(\theta_u)d\Omega \tag{1.9}$$

$$W_{radiation} = \int_S Q_{radiation}dS \tag{1.10}$$

The scalar flux density $Q_{radiation}$ is the amount of power in $\mathrm{W\,m^{-2}}$ actually received by an elementary surface dS from all directions, when taking into account

that rays of radiance coming from almost tangent directions contribue less than those that come from the normal direction. It is integrated over all wavelengths; spectral scalar flux density $Q_{radiation,\lambda}$ expresses the same quantity for a specific wavelength. Angle θ_u is the angle between the normal $\vec{n}$ of the surface and the direction $\vec{u}$ of that particular ray of radiance. We say *radiant exitance* to design an emitted flux and *irradiance* to design an received flux. Figure 1.6 illustrates the two interpretations (ingoing and outgoing) of scalar flux density through the same surface dS. Henceforth, whenever we will refer to flux density, that will actually be scalar flux density, since that is the value of interest for us.



(a) Irradiance received by the elementary surface dS at point x.

(b) Radiant exitance emitted by the elementary surface dS at point x.

Figure 1.6: Illustration of flux density arriving at, or leaving, surface dS. Spectral flux density would be represented the same way. Angle of incidence θ_u between each direction and the normal of dS is represented.

We needed this development because many methods for computing radiative transfer as we will later present them either deal with directly computing the scalar interpretation of flux density $Q_{radiation}$, or computing the radiance I and then integrating it over all directions. For the latter case, the notion of radiance, and the ways to compute it, are based upon the principle of black-body radiation. The closely related notion of grey body will also be a foundation of our approach in Chapter 2. For these reasons we will thus make a short description of what constitutes black-body radiation.

1.3.3 Laws of black-body radiation

A theoretical black-body, as was established by Kirchhoff [48], is a body that absorbs 100% of all incident electromagnetic radiation on its surface regardless of wavelength or direction. When compared to another body at the same temperature and surface,

the black-body will emit more radiative power for any possible wavelength. No material behaves completely as a theoretical black-body, but it is possible to build devices that approach the radiative pattern of black-bodies ([49, 50]). Since a black-body defines an absolute maximum of radiative power emitted, it serves a useful purpose as a calibrating reference for many radiation applications.

Planck's law As black-bodies behave as isotropical emitters, their spectral radiance does not depend on direction. It only depends on the considered wavelength and the temperature of the black-body. Planck ([51]) has determined the exact expression of the spectral radiance I^b_λ of the black-body at temperature T and for wavelength λ: it reads, for any direction $\vec{u}$:

$$I^b_\lambda(T) = \frac{2hc_0^2\lambda^{-5}}{exp(\frac{hc_0}{k\lambda T}) - 1} \tag{1.11}$$

This equation uses c_0 the speed of light in vacuum, h the Planck constant and k the Boltzmann constant. Since this radiance is constant with regard to direction, obtaining the spectral emissive power $Q^b_{radiation,\lambda}$ of the black-body through Equation 1.9 is very simple, and integrating the cosinus over the unit hemisphere gives:

$$Q^b_{radiation,\lambda}(T) = \pi I^b_\lambda(T) = \frac{2\pi hc_0^2\lambda^{-5}}{exp(\frac{hc_0}{k\lambda T}) - 1} \tag{1.12}$$

The spectral emissive power graph for various temperatures is displayed in Figure 1.7. For a given temperature, any material's spectral emissive power curve will fit below the curve of the black-body of same temperature.

Wien's law One notes that there is a maximum of spectral emissive power for a specific wavelength. Derivating Equation 1.12 gives us Wien's law: for a black-body, the product of the given temperature T and the wavelength for which spectral emissive power is maximum λ_m is constant. As we can see in Figure 1.7, the value of λ_m increases towards the higher wavelengths as the temperature decreases.

$$\lambda_m T = 2898\,\mu\text{m}\,\text{K} \tag{1.13}$$

In particular, 95% of the total spectral emissive power is output in the wavelength band comprised between $0.5\lambda_m$ and $5\lambda_m$. This remains a useful order of magnitude for materials that behave not too differently from grey bodies. For example, a black-body at 1600 K concentrates almost all its radiative power between 0.9 µm to 9 µm.

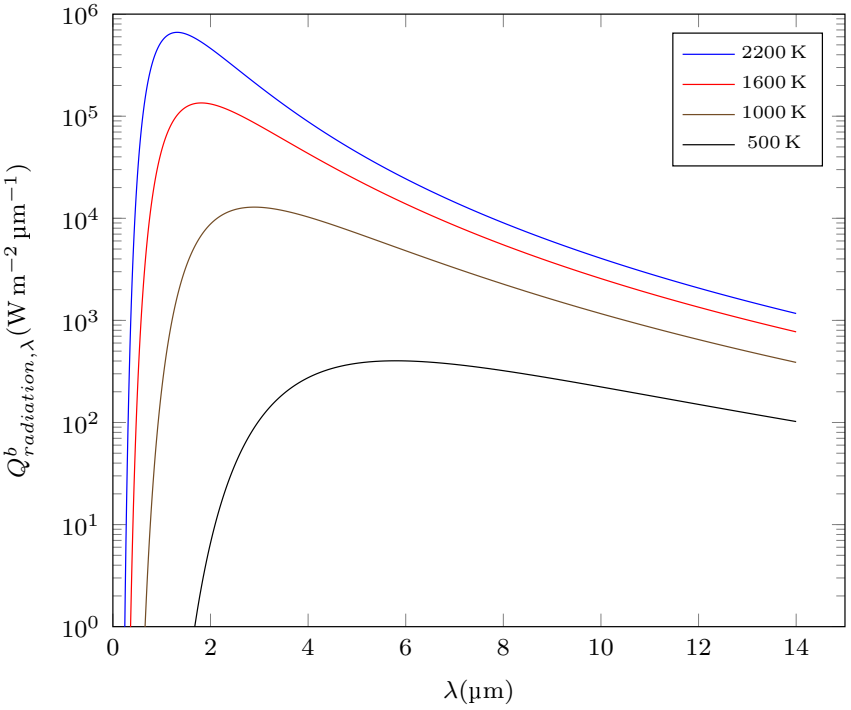


Figure 1.7: Black-body spectral emissive power for various surface temperatures.

Stefan's law Finally, integrating spectral emissive power over all wavelengths gives us the radiant exitance $Q^b_{radiation}$ of a black-body of elementary surface dS at temperature T: the amount of radiative power it emits by unit of surface in $W \cdot m^{-2}$ is, according to Stefan's law:

$$Q^b_{radiation}(T) = \sigma T^4 \tag{1.14}$$

$$\sigma = \frac{2\pi^5 k^4}{15 c_0^2 h^3} = 5.67\,\mathrm{W\,m^{-2}\,K^{-4}} \tag{1.15}$$

where σ is the Stefan constant. This latter law is the most general and encompassing one: it gives no information about either wavelength or direction of emission. Its formulation is more convenient than Planck's law, since it only depends of one value which is the body's temperature. If a given material can be shown to radiate approximately like a black-body, this means one can use, without too much inaccuracy, the very simple Stefan's law to describe its radiation.

Emissivity In order to compare a given material's radiative behaviour to that of the black-body, a convenient concept is *emissivity*. This adimensional value between 0 and 1 is the ratio between the material's considered radiative value (ra-

diance, spectral radiance, exitance, etc.) and that of the black-body at the same temperature T. Different emissivities can thus be defined:

$$\epsilon_{\lambda,\vec{u}} = \frac{I_\lambda(\vec{u},T)}{I^b_\lambda(T)} \tag{1.16}$$

$$\epsilon^{\cap}_{\lambda} = \frac{Q_{radiation,\lambda}(T)}{Q^b_{radiation,\lambda}(T)} \tag{1.17}$$

$$\epsilon = \frac{Q_{radiation}(T)}{Q^b_{radiation}(T)} \Leftrightarrow Q_{radiation}(T) = \sigma\epsilon T^4 \tag{1.18}$$

The *directional monochromatic emissivity* or *spectral directional emissivity* $\epsilon_{\lambda,\vec{u}}$ is the most general definition. Its measurement is not easy, and a number of publications focus entirely on determining this emissivity for a new material or with a new method ([52–55]). Some radiation models (reflection on the Moon's craters by Vogler et al. [56], colored coating with radiative cooling by Xi et al. [57]) need to take into account a wavelength-dependant, direction-dependant emissivity. Figure 1.8, for example, displays the spectral directional emissivity of high purity polished chromium at 823 K and 873 K. We can see how emissivity varies a lot according to wavelength, and is relatively constant in direction up to about 80° of incidence. This behaviour is typical of metals. We also see that a temperature difference of only 50 K is enough to affect emissivity in a significative way. In addition to this, corroded or oxydized chromium would present a different emissivity profile. Normal spectral emissivity $\epsilon_{\lambda,\vec{n}}$ is the special case where the direction $\vec{u}$ of interest is taken equal to the normal $\vec{n}$ of the material, since that is the direction for which spectral directional emissivity is easiest to measure.

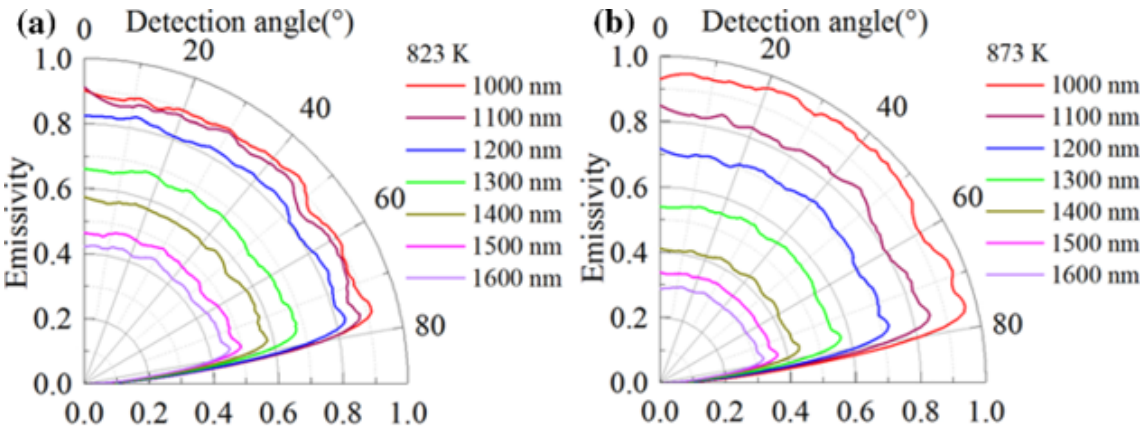


Figure 1.8: Spectral directional emissivity of high-purity polished chromium at different temperatures. Detection angle is from the normal of the surface ([52]).

The *spectral hemispherical emissivity* $\epsilon_\lambda^\cap$ stands for the ratio of the spectral radiant exitance of the given material on the spectral radiant exitance of the black-body of same temperature, for a given wavelength. This value gives a good compromise between describing accurately the radiation behaviour of the material and keeping the formulation relatively concise. Typical values of spectral hemispherical emissivity will be shown in Chapter 2.

The last value is simply called *emissivity*, *total hemispherical emissivity*, or sometimes *emittance*. In Equation 1.18, emissivity depends neither from direction nor from wavelength. This situation is called the *grey-body hypothesis* and can be taken when a given material's spectral directional emissivity is not too far from being constant within the wavelength and direction domains where the bulk of radiative transfer is going to happen. We can then write:

$$\epsilon_{\lambda,\vec{u}} = \epsilon \qquad \forall \lambda, \vec{u} \tag{1.19}$$

It greatly simplifies radiative transfer description, as in this case, the given material's radiative transfer is simply described by Stefan's law multiplied by its now-constant emissivity. This is shown in the second part of Equation 1.18. Tables of grey-body emissivity for a lot of materials at any given temperatures are very useful ([58, 59]), as they give a first approximation of the radiative behaviour of said material. Some applications where radiation plays only a small or moderate part in thermal transfer are content to use the grey-body hypothesis to considerably simplify radiation modeling, even for materials having an emissivity profile actually quite dependant on wavelength or direction ([60, 61]).

However, when the grey-body hypothesis is not justified, it is required to properly describe the real radiance of the considered material, which is quite different in the most general case from the radiance of the black-body. We will now detail the relevant radiance equation.

1.3.4 The radiative transfer equation

The *radiative transfer equation* (RTE) describes how, for a given wavelength λ, the spectral radiance $I_\lambda(x, \vec{u}, t)$ varies according to position x, time t and direction $\vec{u}$. It is a volumic equation: it models a photon moving in the bulk of a given homogeneous or heterogeneous material at a celerity c.

$$\frac{1}{c}\frac{\partial I_\lambda}{\partial t} + \vec{u} \cdot \vec{\nabla} I_\lambda = \epsilon_{\lambda,\vec{u}} I_\lambda^b(T) - (\kappa_{\lambda,\vec{u}} + \sigma_{\lambda,\vec{u}})I + \frac{\sigma}{4\pi}\int_{4\pi} \Phi(\vec{u}', \vec{u}) I_\lambda(x, \vec{u}', t) d\Omega' \tag{1.20}$$

- $\frac{1}{c}\frac{\partial I_\lambda}{\partial t} + \vec{u} \cdot \vec{\nabla} I_\lambda$ amounts to the variation of the number of photons arriving at x over time in the Eulerian view of movement. Light speed c in the current

medium intervenes in the formula. While in a non-vacuum medium of refractive index n this value is defined as $c = \frac{c_0}{n}$ with c_0 the speed of light in vacuum, it remains of the same order of magnitude as c_0 in most cases given that typical values of refractive indices are between 1 and 4. Hence, $\frac{1}{c}\frac{\partial I}{\partial t}$ is most of the time very small compared to $\vec{u}.\nabla I$. Characteristic time of radiation is of the order of $\frac{\lambda}{c}$, and thus about equal to 10^{-14} s for visible light in a vacuum. On the other hand, a typical diffusion time in ceramics is 10^{-2} s. Thus, we only keep term $\vec{u}.\nabla I$ in the RTE.

- $\epsilon_{\lambda,u} I^b_\lambda$ amounts to the thermal emission of radiation at location x. I^b_λ is the black-body radiance at wavelength λ and temperature T in all directions equally, including direction $\vec{u}$. $\epsilon_{\lambda,u}$ is the spectral directional emissivity of the material at position x for that wavelength and direction.

- $-(\kappa_{\lambda,\vec{u}} + \sigma_{\lambda,\vec{u}})I$ is the proportion of photons arriving at x that are either absorbed by the material or reflected in any direction other than $\vec{u}$. $\kappa_{\lambda,\vec{u}}$ is the absorption coefficient, $\sigma_{\lambda,\vec{u}}$ is the scattering coefficient. They are expressed in m^{-1} and stand for the average distance a given photon can travel in the considered material before being absorbed or scattered towards another direction, respectively. These values also depend in the most general case on direction and wavelength. $\beta = (\kappa + \sigma)$ is called the extinction coefficient of the medium and defines the global resistance of the medium to the transmission of radiation. When radiation is studied as a purely surface phenomenon with no transmission, or at the interface of a materiel which is very thick compared to its mean free path before absorption, we also use the absorptivity α of that material. This nondimensional value stands for the ratio of incident radiation that is not reflected upon encountering the surface over the total amount of incident radiation.

- $\frac{\sigma}{4\pi}\int_{4\pi}\Phi(\vec{u}', \vec{u})I_\lambda(x, \vec{u}', t)d\Omega'$ amounts to the entirety of photons coming from all directions of space $\vec{u}'$ and that have been scattered by the material towards direction $\vec{u}$. Φ is the phase function, that describes how scattering is dependant on both incident and scattered ray directions. Φ also describes how radiation is reflected when arriving at a two-dimensional medium discontinuity (like striking a solid surface after having traveled through air). If Φ is constant, then reflection is isotropic: light is reflected in all directions equally, with no special dependance on the incoming direction. Otherwise, it is anisotropic, or specular: light is reflected towards a single specific direction that can be predicted with Fresnel's law. Most materials exhibit a mix of the two behaviours. For example, a mirror is a very strongly specular surface, while most wooden (non polished) surfaces present isotropic reflection.

We now have introduced the concepts of flux density and radiance, as well as their governing equations. Let us briefly review some of the established methods used to compute these values and from there on the radiative transfer part of the equation of energy conservation.

1.4 Radiative transfer state of the art

1.4.1 The Discrete-Ordinate Method

The *discrete-ordinate method* (DOM) ([62]) tries to access radiance by solving the RTE. To this end, it discretizes the angular space around point x in finite volumes. For each such volume, we integrate over all solid angles around the volume the incoming scattering term through a quadrature rule. This means we attribute a weight function to a finite number of points in the domain and add up the contribution of each to the scattering. The integration domain is here the unit sphere, and the finite number of points over which we integrate the scattering means we evaluate the ETR in a finite number of incoming ray directions. Figure 1.9 shows how an octant (an eighth of unit sphere) can be sampled by a Gauss quadrature. Tables provide the weight function to be applied to each point of the quadrature depending on its direction. The heat ray trajectory is also split into finite volumes. Starting from the emitting volume, the RTE is solved in each volume along the ray trajectory. Each volume takes as ingoing boundary conditions the flux that has been computed for the previous volume of the same ray and the fluxes from the adjacent volumes belonging to adjacent rays.

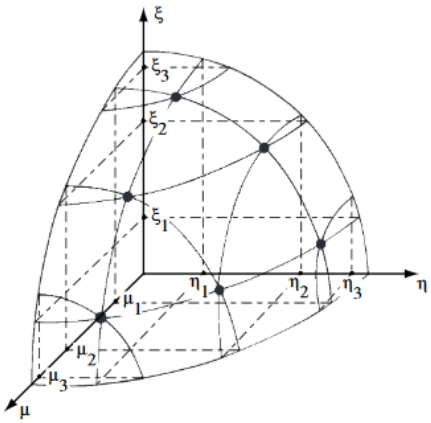


Figure 1.9: Six points symmetric quadrature (quadrature S_6) ([63]).

Once the RTE has been solved all along the ray trajectory, the process starts

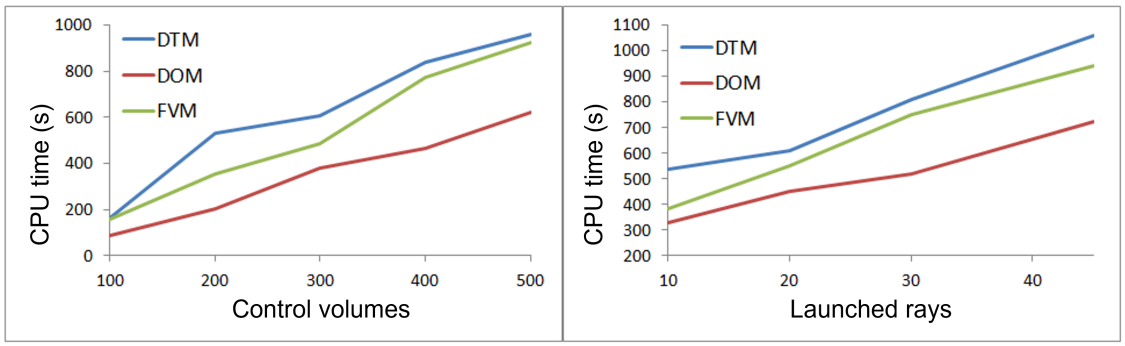


Figure 1.10: Comparison of computation time between DOM, FVM and DTRM depending on mesh refinement and quadrature (from [65]).

over from the beginning of the ray, until the computed flux has converged. The accuracy and speed of this method rely on how fine the volume mesh is and how many points have been chosen for the quadrature.

A related formulation is the finite volumes method (FVM). The scattering integral is numerically computed piecewise on each solid angle domain between two discrete directions ([64]).
These methods are able to take into account all physical aspects of radiation: specular reflection, wavelength-dependant effects, anisotropic lighting, participating medium...and their accuracy can be fine-tuned by adjusting the mesh and/or the quadrature. These assets enable them to properly acccount for important flux or intensity gradients. Mishra et al. [65] detail a comparison between the DOM, the FVM and the DTRM method to be described below. The DOM is the superior method among the three in the case of a strongly directional intensity and a participating medium. The main results of the comparison are displayed in Figure 1.10.

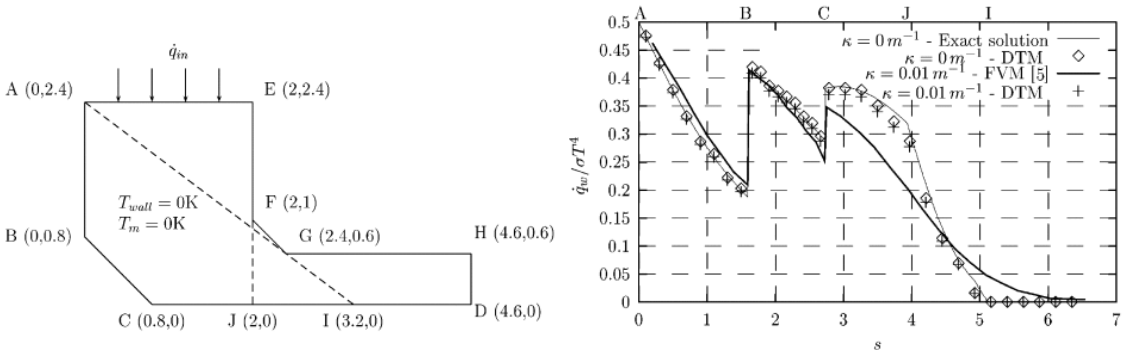
However, when the medium has a high extinction coefficient β, these methods have trouble converging. In addition, the FVM can suffer from numerical diffusion when the medium is weakly or non diffusive, which artificially flattens important thermal gradients.

1.4.2 The DTRM

The *discrete transfer radiation method* (DTRM) or *discrete transfer method* (DTM) has been brought forth by Lockwood and Shah [66]. It does not rely on a spatial mesh. Instead, the RTE is integrated over a finite number of rays. Each ray is split into small segments, and intensity is computed at the end of each segment with intensity at the beginning of the segment as input, and taking into account exponential

extinction by the medium all along. The RTE is thus resolved at each time step by a Galerkin method. Contrary to the DOM, we do not gather the contribution of ingoing reflected rays by extracting data from the neighbouring rays. Instead of this, this contribution is replaced by a diffused intensity, averaged out from all rays in the system. This makes the computation simpler since each ray computation is not coupled with its neighbour's anymore.

This method is simpler to implement than the DOM, and its accuracy can be fine-tuned by adjusting the number of rays launched. However, the fact that the rays constitute a discrete space sample leads to an inaccuracy called the ray effect. Far from the emission zone, some surfaces can see no ray reaching them. Feldheim and Lyabert [67] have compared the DTRM and the FVM and shown that the ray effect could be smoothed out by launching enough rays; in that case the DTRM provided better results than the FVM when the latter suffers from numerical diffusion. Figure 1.11 is taken from their study and shows the thermal flux computed along a wall facing a radiative thermal source. The wall is divided into segments and its various segments do not all face the source the same way; the DTRM can take these discontinuities into account better than the FVM.



(a) Wall geometry *(b) Flux received by the wall*

Figure 1.11: Adimensional thermal flux received along the ABCJID wall from AE wall in a weakly or non-diffusing medium ([67]).

1.4.3 The P_1 method

The goal of the P_1 *method* is to bypass the computation of the radiance entirely and to directly compute the integrated radiative flux at point x. It relies on a number of mathematical operations to apply to the intensity. The complete process is rather involved; it can be found explained in detail by Modest [68].

Since the most troublesome part of the RTE is the integral term, the P1 method will make it disappear. Spectral radiance and phase function are decomposed into spherical harmonic series. This separates the variables for the spectral radiance: for each term of the series, instead of having one function of position and direction, we have a product of one vector function of position and another one of direction. Similarly the phase function is represented as the product of vector functions of the ingoing direction only, or the outgoing direction only. This makes a lot of scalar products appear, of which we can cross out a lot since the spherical harmonics are orthogonal for this scalar product. By integrating over all directions for all terms of the series separately, the end result is a series of linear partial differential equations depending only on position and not on direction anymore. At this stage, we keep the Nth first such harmonics and we solve the system. Keeping only the first one gives its name to the P_1 method. This gives us the spectral radiance. By repeating this process to the relation between radiance and flux density (in Equation 1.2) to bypass that integral too, one can access the spectral flux density as a function of spectral radiance, itself a function of position only, with no integral terms. It is possible to achieve the same set of equations by using physical considerations of integrating the spectral radiance several times over all solid angles ([69]).

The purpose of these transformations is to turn an integro-differential equation into a system of linear partial differential equations, which is more easily handled. From there, a number of methods can be used from its resolution. The P_1 method is very fast, since it bypasses the RTE entirely to directly give the flux density. Yang and Modest ([70]) have compared the DOM and the P_3 method (thus keeping the first three harmonics). Figure 1.12 is from their work: they compute the flux passing through the middle horizontal line of a 2D square when the bottom side is set to a given temperature, and all others to 0 K. They then make the extinction coefficient β of the medium vary, and compare the obtained fluxes to the analytical solution. They conclude that the P_3 is more accurate than the DOM when β is low, and that the computation time is completely indifferent to the variations of β. On the other hand the FVM method initially takes as much as the P_3 to converge when β is low, but then twice as much when β becomes high.

When the extinction coefficient becomes high, the system can be further simplified into the so-called Rosseland approximation ([71]). In that case, thermal radiative flux is reformulated into a conduction flux, with its own equivalent conductivity $\lambda_{Rosseland}$. On the other hand, one can complexify the formulation even further for a better accuracy and range of application, like the SP_N method ([72]). However, in that case the mathematics involved rapidly become very burdensome, for only moderate gains. For this reason, going up to P_3 is a good compromise.

A weakness of this method is high gradients of temperatures or radiances, as well

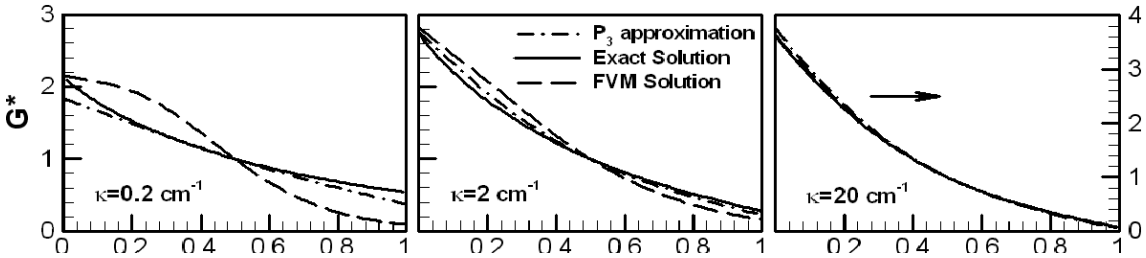


Figure 1.12: Adimensional thermal flux across the $x = 0.5$ plane, according to various absorption coefficients ([70]).

as weakly participating media. This is because the way a proper radiance integration has been avoided amounts to averaging out the spatial contribution of radiance akin to a diffusion. This means high gradients will be modelled with a higher degree of error, while weakly participating media are precisely those where the heat rays are not diffused very much.

1.4.4 The Monte-Carlo method

As all *Monte-Carlo methods* do, this technique tries to replicate the physics happening in the environment in a local and random way, and to repeat this random simulation over and over again. Statistics tells that when the number of random repetitions is high enough, the average of the randomly generated results becomes close to the real physical value. In the case of thermal radiation, the random process consists of launching a high number of heat rays, each of them corresponding to an elementary surface of the problem emitting thermal radiation towards a specific direction ([73]). When this number is high enough, the sum of all such exchanged radiant energy converges towards the theoretical exchange process.

First, one meshes the items that will participate in radiation, as well as the space separating them. For each mesh element, we have an elementary facet (or volume) contributing to radiation. We then compute a number of random space coordinates both to generate some point on the element that will be used as the ray's beginning and some direction towards which the ray will be oriented. The quantity of energy ascribed to each ray is ponderated by the number of rays leaving that element and the surface/volume of the element. This way, local energy conservation is conserved. Each ray is launched, and the element from which it starts registers a loss of energy. The ray path across the volumic mesh is followed, until it hits a solid surface. It then has a probability of being either reflected or absorbed. If it traverses a participating medium, or a partially transparent solid, a probabilistic function determines the distance after which that ray will end up absorbed, or dispersed. When the ray finally is absorbed somewhere, the element where that takes place registers a gain

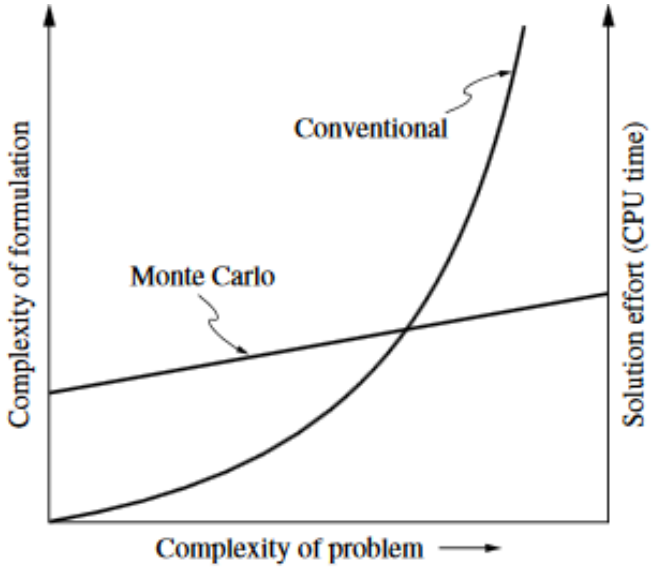


Figure 1.13: Paradigm of the Monte-Carlo method in how more complex problems are handled ([74]).

of energy. When all rays have been launched, a total energy bilan is established and radiation has effectively been processed.

This model can take into account all radiation phenomena: participating medium, transparent solids, specular reflection, wavelength-dependant effects...Since it can be applied to all situations, it is sometimes taken as a reference when another method wants to evaluate the results it generates. The Monte-Carlo method is also very scalable. From a small, simple system to a large system with a set of convoluted radiating objects and properties, the mathematical formalism remains exactly the same. This means the computing power required by the Monte-Carlo method as the problem grows more complex tends to increase slowlier than that of other methods mentioned in this brief review, as pointed out by Modest ([74]). Figure 1.13 is taken from his work. It is also easy to set up on parallel computation. Similarly to the P_1 method, Monte-Carlo bypasses completely the problem of generating the radiance through the RTE to directly access the flux density. Instead of doing this through a series of mathematical transformations, it accomplishes this by "brute-forcing" the physics of the flux density through the sheer number of simulations it makes.

The Monte-Carlo method can be enhanced through a number of ways that leave its core principle simple. The *quasi-Monte-Carlo method* of Kersch et al. ([75]) does not generate ray directions in a truly random way, but instead uses certain point-generating low-discrepancy sequences, like the Halton sequence. These sequences produce a point spread from which rays can be launched and that maps the unit

sphere in a more uniform way than truly random sequences. This upgrade accelerates the convergence of the error towards zero. The *backward Monte-Carlo method* of Gratyi et al. [76] launches the rays not from where radiation is emitted, but where it is collected. In the case where the radiation target is very small (i.e. a telescope) and the radiation source very large in comparison (the atmosphere of a planet), an extremely high number of rays would need to be launched from the source, so that a statistically significant number of them would hit the target.

A troublesome aspect of the Monte-Carlo method is how computationally expensive it is, even for simple cases. For easy geometries, the other existing methods produce just as good results but in much faster computation times, whereas the Monte-Carlo method will need to launch millions of rays even in these situations.

1.4.5 The Surface-to-Surface method

The *Surface-to-Surface method* (S2S) ([20], [38], [77]) is also called radiosity method, or zonal method when the medium is participating. The S2S method aims at extracting the flux density directly, not from the RTE but from a simpler set of equations which implicitely take the directionality of the RTE into account. When medium is nonparticipating and the materials behvave as grey-bodies, for each radiating elementary surface (extracted through meshing), a sum of total radiant exitance is established. This means we add up the flux density produced by the surface itself through black-body radiation, and the flux density that comes from elsewhere but has been reflected off the surface. We substract the flux density the surface absorbs itself. The surfaces are considered nonspecular, so that they emit radiation equally in all directions. One can then establish a matrix of relationships between the net produced flux of all surfaces, since they all interact with one another. Inverting a matrix is less involved than numerically tackling the complicated RTE. We thus obtain the flux density for each elementary surface directly.

The S2S method is at its best when medium is reasonably non-participating and radiating objects can be considered non-specular and opaque, as it turns the problem of radiation into a much simpler problem of linear system resolution. The size of the chosen surface mesh allows us to fine-tune the accuracy we want from this method. When specular reflection has to be considered, and when volumic elements start to interfere with radiation (either from a participating medium or a partially transparent object where radiation penetrates in its bulk), the so-called zonal method can still be applied but the simplicity of the S2S formalism is lost. Besides, objects where surface radiative properties can have strong local variations will require a very fine mesh for the method to capture this, which can further slow down its application.

1.4.6 Overview of the different methods

The various radiation modellings we have brought up all have a specific range of applications where they perform best. Table 1.1 sums up their assets and their drawbacks.

Method	Strong point	Limitation	Typical use case
DOM	All physical phenomena can be accounted for	Slow when the extinction of the medium is high	Solar radiation through a forest canopy ([78])
DTRM	Same as DOM, simple to implement	Ray effect	Curved trajectory of radiation in variable index of refraction medium ([22])
P_1	Gives an all-purpose and simple result	Inaccurate where gradients are high	Charcoal combustion chamber ([21])
Monte-Carlo	Can efficiently handle the most complex cases	Slow and cumbersome for simple cases	Propagation of a laser beam in the ocean through a school of plankton ([23])
S2S	Efficient for non-participating medium	Specular reflexion is poorly handled	Sun-heated fluid in thermal power plant ([79])

Table 1.1: Summary of the specifics of each presented radiation modelling method

In the specific case of TAS, sintering takes place either in a vacuum, or under an atmosphere of nitrogen N_2. In the first case, the medium is obviously nonparticipating. This is the strong point of the S2S method. Moreover, the exacting geometry of the ceramic parts to be sintered requires to be accurately captured. A method that proceeds to a spatial averaging of the contribution of radiation, like the P_1 method, would fail to render this. Finally, ceramic (as well as the graphite inner walls of the oven) is a reasonably grey-body, opaque, anisotropic material, which fits the application range of the S2S method. This is why we have estimated this method would fit our modelling needs.

1.5 Summary

We have now introduced the topic of this thesis. The problem of a proper modelling of radiative thermal transfer in industrial processes has been developed. The

main problems one encounters with radiation are its strongly nonlinear equation, its dependency on very local and hardly accessible material properties, its dependance both on local and global geometry and its behaviour as both a surface and a volumic phenomenon. We have detailed how radiative flux density derives from radiance, and how radiance is described by a relatively complex Radiative Transfert Equation. A brief overview of a number of radiative thermal transfer methods has then been presented. Some of them try to numerically solve the RTE through a number of spatial division schemes (the DOM, the FVM, the DTRM). Some others prefer to directly compute the radiative flux density by considering direct radiative surface exchange (the Monte-Carlo method, the S2S method). Yet other methods apply important transformations to the RTE to turn it into a simpler equation and solve it from there (the P_1 method, the Rosseland approximation). The S2S method has appeared to us as the most appropriate choice to model radiative transfer in the sintering ovens for producing TAS ceramic parts.

1.5.1 Objective of the thesis

TAS and its partners currently use a trial-and-error method to design and sinter their ceramics items, which, though effective, is a slow and costly way of proceeding. Switching to a more agile, comprehensive method and gaining some insight and predictive power about the exact physical processes at play during sintering would thus be a great boon. The aim is to establish an end-to-end approximate behaviour law starting from the thermal environment during sintering and ending with an heuristics for ceramic density, porousness, cracks location, etc. This thesis deals with the first half of this project: developping a better understanding of total thermal transfer occuring to the ceramics being sintered. The second half, beyond the scope of this work, would be to start from a predicted thermal field obtained on the ceramics thanks to the solver being developed here, and to establish from there a microstructural evolution model during sintering. Thus, our aim during these three years was to achieve the following output: a reasonably accurate, fast and adaptable simulation of the thermal transfer occuring in and around the ceramics being sintered, with a focus on radiative thermal tranfer. In order to reach this aim, this thesis offers the novel approach of bringing together several existing, but separated, techniques into a robust and adaptable solver: S2S method, immersed meshes and kd-tree-accelerated ray tracing. Some authors have already implemented S2S radiation in the context of an immersed boundary method (Favre et al. [80, 81], Schmid et al. [82]) while others have investigated kd-tree ray tracing associated with radiosity (Bindick et al. [40]). However, to the best of our knowledge, no litterature can be found on the combination of all three methods.

1.5.2 Thesis outline

This thesis is composed of the following chapters.

Chapter 2 explains the immersed mesh formulation used for radiating facet detection, introducing the level-set function. It then details the specific hypotheses of the S2S modelling in greater length than what was done in this introduction. The notions of view factor and radiosity are presented, as well as the design choices we have made to compute them - including view factor obstruction. Radiation is then coupled as a source term into a convection–diffusion equation for a more complete thermal transfer solving. Finally, the solver is evaluated against steady-state 2D and 3D simple test cases featuring both pure radiation and convection–radiation situations. Analytical solutions for the expected radiative fluxes and temperatures are provided and compared against the computed results.

Chapter 3 introduces the problem of unacceptable computation time in the obstruction process of the S2S solver. In order to solve that issue, a kd-tree data structure is introduced and a ray tracing algorithm is developed so that the obstructing radiating facets are detected faster. A nearest-neighbour algorithm is also developed. A second, more performant variant of kd-tree and accompanying ray tracing algorithm is then produced. That second kd-tree's scalability is tested as we try the solver in parallel computation on an increasing number of cores. Both obstruction computation and ongoing radiosity computation scalability are evaluated.

Chapter 4 assesses the performance of the solver on three larger test cases, depicting both radiative cooling, and radiative heating as in a sintering oven. The test cases come from industrial partners and feature simple loads close to real industrial items. We assess the physical correctness of the results, and the effects of position inside the oven are seen. The solver is compared to a test case for which experimental data is available. Results both in steady-state and in transient state thermal transfer are commented on. The simulations highlight some strong points of the proposed solver as well as some paths for future enhancement.

Bibliography

[1] F.-T. Lentes, N. Siedow, Three-dimensional radiative heat transfert in glass cooling processes, Glass Science and Technology : Glastechnische Berichte 335 (1999) 188–196. 3

[2] G. Thömmes, R. Pinnau, M. Seaïd, T. Götz, A. Klar, Numerical methods and optimal control for glass cooling processes, Transport Theory and Statistical Physics 31 (2002) 513–529.

[3] D. Clever, J. Lang, Optimal control of radiative heat transfer in glass cooling with restrictions on the temperature gradient, Optimal Control: Applications and Methods 33 (2012) 157–175. 3

[4] G. Krishnamoorthy, A computationally efficient P_1 radiation model for modern combustion systems utilizing pre-conditioned conjugate gradient methods, Applied Thermal Engineering (june 2017) 197–206. 3, 5

[5] L. Wu, X. Xu, W. Zhou, Y. Su, X. Li, Heat transfer analysis of blast furnace stave, International Journal of Heat and Mass Transfer 51 (jun. 2008) 2824–2833.

[6] V. Cuervo-Piñera, D. Cifrián-Riesgro, P.-D. Nguyen, V. Battaglia, M. Fantuzzi, A. D. Rocca, M. Ageno, A. Rensgard, C. Wang, J. Niska, T. Ekman, C. Rein, W. Adler, Blast furnace gas-based combustion systems in steel reheating furnaces, Energy Procedia 120 (august 2017) 357–364. 3

[7] B. Thomas, I. Samarasekera, J. Brimacombe, Mathematical model of the thermal processing of steel ingots : Part I. Heat flow model, Metallurgical and Material Transactions B (mar. 1987) 119–130. 3

[8] J. Wei, H. Zhang, L. Zheng, C. Wang, B. Zhao, Modeling and improvement of silicon ingot directional solidification for industrial production systems, Solar Energy Materials and Solar Cells 93 (september 2009) 1531–1539.

[9] Z. Malinowski, T. Telejko, B. Hadała, Influence of heat transfer boundary conditions on the temperature field of the continous casting ingot, Archives of Metallurgy and Materials 57 (march 2012). 3

[10] S. L. Foll, Modélisation du couplage conduction/rayonnement dans les systèmes de protection thermique soumis à de très hauts niveaux de températures, Ph.D. thesis, Institut national des sciences appliquées de Lyon (2014). 3, 5

[11] L. A. Dombrovsky, S. Dembele, J.-X. Wen, A simplified model for the shielding of fire thermal radiation by water mists, International Journal of Heat and Mass Transfer 96 (may 2016) 199–209.

[12] A. V. Nenarokomov, L. A. Dombrovsky, I. V. Krainova, O. M. Alifanov, S. A. Budnik, Identification of radiative heat transfer parameters in multilayer thermal insulation of spacecraft, International Journal of Numerical Methods for Heat and Fluid 27 (march 2017). 3

[13] S. A. Khairallah, A. T. Anderson, A. Rubenchik, W. E. King, Laser powder-bed fusion additive manufacturing: Physics of complex melt flow and formation mechanisms of pores, spattern and denuation zones, Acta Materiala 108 (april 2016) 36–45. 3

[14] Z. Yan, W. Liu, Z. Tang, X. Liu, N. Zhang, M. Li, H. Zhang, Review on thermal analysis in laser-based additive manufacturing, Optics and Laser Technology 106 (october 2018) 427–441.

[15] P. Michaleris, Modeling metal deposition in heat transfer analyses of additive manufacturing processes, Finite Elements in Analysis and Design 86 (september 2014) 51–60. 3

[16] K. Reddy, N. S. Khumar, Combined laminar natural convection and surface radiation heat transfer in a modified cavity receiver of solar parabolic dish, International Journal of Thermal Sciences 109 (jan. 2008) 1647–1657. 3

[17] J. Polo, J. Ballestrín, E. Carra, Sensitivity study for modelling atmospheric attenuation of solar radiation with radiative transfer models and the impact in solar tower plant production, Solar Energy 134 (september 2016) 219–227.

[18] A. Razagui, K. Abdeladim, S. Semaoui, A. H. Arab, S. Boulahchiche, Modeling the forecasted power of a photovoltaic generator using numerical weather prediction and radiative transfer models coupled with a behavioral electrical model, energy Reports 6 (february 2020) 57–62. 3

[19] F.-A. GmbH, Vakuum-induktions-sinteranlage. 3

[20] B. Gaume, Réduction d'un problème d'auto-rayonnement par modes de branche : Application aux échanges thermiques dans un domaine multi-enceintes, Ph.D. thesis, Université Evry-Val d'Essonne (2016). 4, 5, 21

[21] G. Krishnamoorthy, A computationally efficient P1 radiation model for modern combustion systems utilizing pre-conditioned conjugate gradient methods, Applied Thermal Engineering (2017) 197–206. 4, 22

[22] S. H. Sarvari, A new approach to solve the radiative transfer equation in plane-parallel semitransparent media with variable refractive index based on the discrete transfer method, International Communications in Heat and Mass Transfer (2016) 54–59. 4, 22

[23] P. Chen, et al., Semi-analytic Monte Carlo radiative transfer model of laser propagation in inhomogeneous sea water within subsurface plankton layer, Optics and Laser Technology (2019) 1–5. 4, 22

[24] G. Bosc, L. Cornillon, M. Rancurel, J.-M. Pellissier, S. Béhar-Lafenêtre, Verification of ceramic structures: qualification and proof testing. 4

[25] B. Thomas, I. Samarasekera, J. Brimacombe, Mathematical Model of the Thermal Processing of Steel Ingots: Part 1, Heat Flow Model, Metallurgical and Material Transactions B (1987) 119–130. 4

[26] P. Cheng, Two-dimensional radiating gas flow by a moment method, American Institute of Aeronautics and Astronautics (1964) 1662–1664. 5

[27] R. McClarren, C. Hauck, Robust and accurate filtered spherical harmonics expansions for radiative transfer, Journal of Computational Physics (aug. 2010) 5597–5614. 5

[28] M. Malek, N. Izem, M. S. Mohamed, M. Seaid, M. Wakrim, Numerical solution of Rosseland model for transient thermal radiation in non-grey optically thick media using enriched basis functions, Mathematics and Computers in Simulation 180 (february 2021) 258–275. 5

[29] K. Prasad, K. Vajravelu, P. Datti, B. Raju, MHD flow and heat transfer in a power-law liquid film at a porous surface in the presence of thermal radiation, Journal of Applied Fluid Mechanics 6 (2013) 385–395. 5

[30] W. Fiveland, Three-dimensional radiative heat-transfert solutions by the discrete-ordinates method, Journal of Thermophysics and Heat Transfer (oct. 1988) 309–316. 5

[31] P. Ricca, R. Furfaro, Analytical discrete ordinate method for radiative transfer in dense vegetation canopies, Journal of Quantitative Spectroscopy and Radiative Transfer (mar. 2013) 60–69.

[32] S. M. et al, Development and comparison of the DTM, the DOM and the FVM formulations for the short-pulse laser transport through a participating medium, International Journal of Heat and Mass Transfer (jan. 2006) 1820–1832. 5

[33] F. Lockwood, N. Shah, A new radiation solution method for incorporation in general combustion prediction procedures, in: Eighteenth International Symposium on Combustion, 1981, pp. 1405–1414.

[34] S. H. Sarvari, A new approach to solve the radiative transfer equation in plane-parallel semitransparent media with variable refractive index basde on the discrete transfer method, International Communications in Heat and Mass Transfer (nov. 2016) 54–59. 5

[35] J. Farmer, J. Howell, Comparison of Monte Carlo strategies for radiative transfer in participating media, Advances in Heat Transfer (jun. 1998) 333–429. 5

[36] A. Kersch, W. Morokoff, A. Schuster, Radiative heat transfer with quasi-Monte Carlo methods, Transport Theory and Statistical Physics (1994) 1001–1021.

[37] S. Gratyi, D. Levin, A. Walker, Rassvet : Backward Monte Carlo radiative transfer in spherical-shell planetary atmospheres, Icarus (mar. 2010) 188–196. 5

[38] Q. Schmid, Numerical radiative transfer using an immerse volume method, Ph.D. thesis, Ecole Nationale Superieure des Mines de Paris (2016). 5, 21

[39] A. F. et al, Modeling of the surface-to-surface radiation exchange using a Monte Carlo method, Journal of Physics : Conference Series (2016).

[40] S. Bindick, M. Stiebler, M. Krafczyk, Fast kd-tree based hierarchical radiosity for radiative heat transport problems, Internation Journal for Numerical Methods in Engineering 86 (september 2014) 1082–1100. 23

[41] C. Sevart, T. Bergman, Evolutionay design method for a conducting solid cooled by combined free convection and radiation, Journal of Heat Transfer (january 2021).

[42] T. G. Revanna, S. S. Katte, Effects of axial conduction and internal radiation on thermodynamic optimization of a tubular space radiator, Thermal Science and Engineering Progress 22 (may 2021).

[43] K. De, H. Wu, G. Huang, X. Xu, Y. Liu, Condensation-free radiant cooling with double-skin infrared-transparent membranes, Building and Environment 193 (april 2021). 5

[44] D. Marchio, P. Reboux, Introduction aux transferts thermiques, Presses des Mines, 2011. 5

[45] D. Li, Z. Shen, Sintering by intense thermal radiation (sitr): A study of temperature distribution by simulation and experiments, Journal of the European Ceramic Society 35 (12) (2015) 3303–3309, 8th International Symposium on Nitrides in Conjunction with the 7th International Workshop on Spinel Nitrides and Related Materials and the Annual Meeting of the Marie Curie ITN 7th Framework Programme FUNEA. 6

[46] D. Li, Y. Liu, Y. Zhong, L. Liu, E. Adolfsson, Z. Shen, Dense and strong zro2 ceramics fully densified in ¡15min, Advances in Applied Ceramics 118 (1-2) (2019) 23–29. 6

[47] D. Li, E. G. de Moraes, P. Guo, J. Zou, J. Zhang, P. Colombo, Z. Shen, Rapid sintering of silicon nitride foams decorated with one-dimensional nanostructures by intense thermal radiation, Science and Technology of Advanced Materials 15 (4) (2014) 045003, pMID: 27877704. 6

[48] G. Kirchhoff, Ueber das verhältniss zwischen dem emissionsvermögen und dem absorptionsvermögen der körper für wärme und licht, Annalen der Physik 185 (2) (1860) 275–301. 9

[49] K. C. Lapworth, T. J. Quinn, L. A. Allnutt, A black-body source of radiation covering a wavelength range from the ultraviolet to the infrared, Journal of Physics E: Scientific Instruments 3 (2) (1970) 116–120. 10

[50] K. Mizuno, J. Ishii, H. Kishida, Y. Hayamizu, S. Yasuda, D. N. Futaba, M. Yumura, K. Hata, A black body absorber from vertically aligned single-walled carbon nanotubes, Proceedings of the National Academy of Sciences 106 (15) (2009) 6044–6047. 10

[51] M. Planck, Ueber das gesetz der energieverteilung im normalspectrum, Annalen der Physik 4 (1901) 553. 10

[52] K. Yu, R. Tong, K. Zhang, An apparatus for the directional spectral emissivity measurement in the near infrared band., International Journal of Thermophysics 42 (29 march 2021) 80. 12

[53] M. Sun, I. K. Lloyd, Emissivity of silicon carbide composites as a function of temperature and microstructure, MRS Proceedings 410 (1995) 411.

[54] G. Yanming, S. Pang, Z. Luo, Y. Shuai, H. Tan, H. Qi, Measurement of directional spectral emissivity at high temperatures, International Journal of Thermophysics 40 (11 2018).

[55] L. Campo, R. Pérez-Saéz, X. Esquisabel, I. Ferna´ndez, M. Tello, New experimental device for infrared spectral directional emissivity measurements in a controlled environment, Review of Scientific Instruments 77 (11 2006). 12

[56] K. J. Vogler, P. E. Johnson, R. W. Shorthill, Modeling the non-grey-body thermal emission from the full moon, Icarus 92 (1) (1991) 80–93. 12

[57] W. Xi, Y. Liu, W. Zhao, R. Hu, X. Luo, Colored radiative cooling: How to balance color display and radiative cooling performance, International Journal of Thermal Sciences 170 (2021) 107172. doi:https://doi.org/10.1016/j.ijthermalsci.2021.107172. 12

[58] Transmetra, Table of emissivity of various surfaces, https://www.transmetra.ch/images/transmetra_pdf/publikationen_literatur/pyrometrie-thermografie/emissivity_table.pdf (april 1999). 13

[59] J. M. Jones, P. E. Mason, A. Williams, A compilation of data on the radiant emissivity of some materials at high temperatures, Journal of the Energy Institute (2019). 13

[60] M. Grote, E. Pohl, P. H. Saptogino, D. Diarra, Impact of solid body emissivity on radiative heat transfer efficiency in furnaces – a numerical study, Energy Procedia 120 (2017) 628–634, iNFUB - 11th European Conference on Industrial Furnaces and Boilers, INFUB-11. doi:https://doi.org/10.1016/j.egypro.2017.07.205. 13

[61] J. Staggs, H. Phylaktou, The effects of emissivity on the performance of steel in furnace tests, Fire Safety Journal 43 (1) (2008) 1–10. 13

[62] W. Fiveland, Three-dimensional radiative heat-transfert solutions by the discrete-ordinates method, Journal of Thermophysics and Heat Transfer (1988) 309–316. 15

[63] N. Martin, Développement de la méthode SN à schémas diamants d'ordres élevés en géométrie 3D cartésienne, Ph.D. thesis, Ecole Polytechnique de Montréal (2008). 15

[64] L. Trovalet, Résolution numérique des transferts par rayonnement et conduction au sein d'un milieu semi-transparent pour une géométrie 3d de forme complexe, Ph.D. thesis, Université Henri Poincaré - Nancy Université (2011). 16

[65] S. Mishra, et al., Development and comparison of the DTM, the DOM and the FVM formulations for the short-pulse laser transport through a participating medium, International Journal of Heat and Mass Transfer (2006) 1820–1832. 16

[66] F. Lockwood, N. Shah, A new radiation solution method for incorporation in general combustion prediction procedures, In Eighteenth International Symposium on Combustion. The Combustino Institute (1981) 1405–1414. 16

[67] V. Feldheim, P. Lyabert, Solution of radiative heat transfer problems with the discrete transfer method applied to triangular meshes, Journal of Computational and Applied Mathematics (2004) 179–190. 17

[68] M. Modest, Radiative Heat Transfer, Academic Press, 2013. 17

[69] P. Cheng, Two-dimensional radiating gas flow by a moment method, American Institute of Aeronautics and Astronautics (1964) 1662–1664. 18

[70] J. Yang, M. Modest, High-order PN approximations for radiative transfer in arbitrary geometries, Proceedings of Eurotherm78 - Computational Thermal Radiation in Participating Media II (april 2006). 18, 19

[71] Le Foll, Stéphane, Modélisation du couplage conduction/rayonnement dans les systèmes de protection thermique soumis à de très hautes températures, Ph.D. thesis, Institut National des Sciences Appliquées de Lyon (2014). 18

[72] R. G. McClarren, C. D. Hauck, Robust and accurate filtered spherical harmonics expansions for radiative transfer, Journal of Computational Physics (2010) 5597–5614. 18

[73] J. Farmer, J. Howell, Comparison of Monte Carlo strategies for radiative transfer in participating media, Advances in Heat Transfer (1998) 333–429. 19

[74] M. Modest, Radiative Heat Transfer, Academic Press, 2013. 20

[75] A. Kersch, W. Morokoff, A. Schuster, Radiative heat transfer with quasi-Monte Carlo methods, Transport Theory and Statistical Physics (1994) 1001–1021. 20

[76] S. Gratyi, D. Levin, A. Walker, Rassvet: Backward Monte Carlo radiative transfer in spherical-shell planetary atmospheres, Icarus (2010) 188–196. 21

[77] A. Frank, et al., Modeling of the surface-to-surface radiation exchange using a Monte Carlo method, Journal of Physics: Conference Series (2016). 21

[78] P. Ricca, R. Furfaro, Analytical discrete ordinate method for radiative transfer in dense vegetation canopies, Journal of Quantitative Spectroscopy and Radiative Transfer (2013) 60–69. 22

[79] K. Reddy, N. S. Kumar, Combined laminar natural convection and surface radiation heat transfer in a modified cavity receiver of solar parabolic dish, International Journal of Thermal Sciences (2008) 1647–1657. 22

[80] F. F. et al, Numerical simulations of conjugate convection combined with surface thermal radiation using an immersed-boundary method, in: Journal of Physics: Conference series, Vol. 745, 2016. 23

[81] F. Favre, O. Antepara, C. Oliet, O. Lehmkuhl, C. Perez-Segarra, An immersed boundary method to conjugate heat transfer problems in complex geometries. application to an automotive antenna, Applied Thermal Engineering 148 (2019) 907–928. 23

[82] Q. Schmid, C. Mensah, E. Hachem, Y. Mesri, A versatile immersed surface-to-surface method for 2d radiative exchange: implementation and validation, International Journal of Heat and Mass Transfer (2018). 23

Chapter 2

Immersed Surface-to-Surface modelling

Contents

2.1 Résumé 34
2.2 Numerical framework 35
2.2.1 Interface description 36
2.3 S2S radiation 37
2.3.1 Overview and related work 37
2.3.2 S2S hypotheses 38
2.3.3 Facet list construction 45
2.3.4 Flux computation 46
2.3.5 Coupling with thermal transfer 58
2.4 Numerical results 59
2.4.1 Simple radiation 59
2.4.2 Conduction–radiation 64
2.5 Summary 79
Bibliography 81

2.1 Résumé

Dans ce chapitre, tout d'abord nous exposons l'approche immergée employée pour définir le maillage utilisé pour construire l'objet rayonnants. Au lieu de passer par un maillage ajusté défini autour de l'objet rayonnant, celui-ci est plongé dans un maillage volumique. Un fonction "level-set" définit la distance signée de chaque point du maillage à la frontière de l'objet, et définit ainsi implicitement son contour. Les hypothèses spécifiques de la méthode Surface to Surface sont détaillées avec soin. Les hypothèses de rayonnement non participant, de rayonnement purement en surface, de corps gris, de rayonnement isotrope et indépendant vis-à-vis de la longueur d'onde et enfin d'émissivité constante sont étudiées. On montre qu'elles sont raisonnables dans le cas du frittage de carbure de silicium et nitrure de silicium. La liste des facettes rayonnantes est extraite du contour implicite des objets. Pour mieux ajuster la géométrie des facettes à la géométrie réelle de l'objet par-delà les imprécisions de discrétisation, des noeuds virtuels sont construits à l'intersection de l'interface et des éléments du maillage, et des facettes virtuelles en conséquence.
La manière dont le flux rayonnant est construit à partir des facettes est détaillé, en rappelant les concepts de facteur de vue et de radiosité. Le problème particulier de l'obstruction est évoqué : il faut déterminer quelles facettes barrent le passage du rayonnement aux autres. Nous établissons une brève bibliographie de cette question et retenons l'approche adaptative de Walton. Le flux est ensuite converti en grandeur volumique et intégré à la résolution de l'équation de la chaleur dans le cadre d'un modèle éléments finis . Nous résolvons le système linéaire constituant le rayonnement à l'aidre d'une méthode GMRES au premier pas de temps. Ensuite, à tous les pas de temps suivants, le rayonnement est calculé à partir des températures du pas de temps précédent, puis l'équation de la chaleur est résolue avec ce nouveau terme de rayonnement en terme source.
Des tests numériques simples sont effectués dans le cas de géométries pour lesquelles il est possible d'obtenir des solutions analytiques. Les tests purement radiatifs et conducto-radiatifs sont menés, en 2D et 3D, en régime permanent, en maillage immergé et en maillage ajusté. Nous considérons le cas de cercles concentriques et de conduits long à une dimension en 2D, et de sphères concentriques en 3D. Le développement exact des solutions analytiques est explicité à l'aide d'une analogie électrique pour établir les résistances thermiques dans le système. Dans tous les cas, les solutions numériques sont très proches des solutions analytiques déterminées. Cela nous permet de valider la précision de notre approche S2S.

The radiosity method we have decided to implement considers radiative interaction between purely surfacic elements. To this end, it requires a list of radiating surfaces in the system. That list can be extracted from the volumic mesh of the ceramics in a classic coupled mesh approach. Alternatively, it can directly be extracted from the mesh of the enviromnent by the use of an immersed mesh method. After an explanation of how we have implemented this second approach, this chapter will detail, step by step, how our S2S method works. The working hypotheses will be considered carefully. Once we generate the flux density, we enter it into Equation 1.1 that we solve with a finite P_1 elements method. This allows us to produce very simple radiation test cases that we compare to analytical results to verify the accuracy of the proposed solver.

2.2 Numerical framework

One of the end goals of this thesis is to produce a cartography of the temperature inside the ceramic parts. To this end, we solve the heat equation and we need a volumic mesh to propagate conduction in the bulk of the elements. Now, we need a surfacic mesh in addition to that for the radiating surfaces. Besides, we want to be able to define our ceramic parts inside the volumic mesh of the (more or less constantly defined from one simulation to another) sintering oven. We could use a coupled mesh approach with a different volumic mesh for the ceramic parts themselves, and solve the heat equation in each mesh separately. Extracting the surface elements from that body-fitted volumic mesh of the ceramics would provide us with the required radiating surfaces. However, if possible, we would like the proposed solver to rapidly adapt if we want to change the position of the ceramic items in the oven. There is also the fact that during sintering, some ceramics occurs a significant shrinkage (up to about 20% volume). While this shrinkage will be neglected at first for the sake of simplicity, we would like to be able to organically account for it at some later stage of development. That shrinkage changes the geometry of the ceramics over time, which means its radiating surfaces constantly evolve. With a body-fitted mesh approach, we would have to redefine the volumic mesh defining the ceramic, and the ceramic-shaped hole in the volumic mesh defining the oven, on a regular basis during simulation. We could then redefine the list of radiating surfaces every once in a while. While entirely possible, this solutions appears rather unwieldy to us. A solution that allows us to do this more efficiently is the immersed mesh approach. Ceramic parts will only be defined by a surfacic mesh immersed into the global volumic mesh. This gives us, at the same time, an implicit definition of the radiating facets through the surfacic mesh we needed.

This approach towards S2S radiation has been investigated by Schmid et al [1].

2.2.1 Interface description

The immersed approaches, also called monolithic approaches, impose the use of an appropriate constitutive equation describing both the fluid and the solid domains. In our case, the fluid domain is either vacuum or oven atmosphere and the solid domain the ceramics part and the inner components of the oven. Their use can be seen for example in [2]. This offers a great flexibility to deal with different shapes or to change easily the physical properties for each immersed structure. Therefore, we start by computing the signed distance function (level set) of a given geometry to each node of the mesh. Using the zero isovalue of this function, we can easily identify the fluid–solid interface. Consequently, different parts are immersed in a larger domain of different constitutive laws. This part will only briefly explain the immersed approach process, but more details can be found in [3].

At any point $\mathbf{x}$ of the computational domain Ω, the level-set function α corresponds to the signed distance function d from the fluid-solid interface Γ_{im}. In turn, Γ_{im} is given by the zero isovalue of the function α:

$$\begin{aligned} \alpha(\mathbf{x}) &= \pm d(\mathbf{x}, \Gamma_{\text{im}}), \mathbf{x} \in \Omega, \\ \Gamma_{\text{im}} &= \{\mathbf{x}, \alpha(\mathbf{x}) = 0\}. \end{aligned} \tag{2.1}$$

In this chapter, the following sign convention is used: $\alpha \geq 0$ inside the solid domain defined by the interface Γ_{im} and $\alpha \leq 0$ outside this domain.

As explained, the signed distance function is used to localize the interface of the immersed structure but it is also used to initialize the desirable properties on both sides of the latter. Indeed, for the elements crossed by the level-set functions, fluid–solid mixtures are used to determine the element effective properties. A Heaviside function $H(\alpha)$ is then defined as follow:

$$H(\alpha) = \begin{cases} 1 & \text{if } \alpha > 0 \\ 0 & \text{if } \alpha < 0 \end{cases} \tag{2.2}$$

In the numerical approximation of the problem we will consider a partition of the computational domain Ω with a finite element mesh made by a collection of element $\{K\}$.

The Heaviside function can be smoothed to obtain a better continuity at the interface [4] using the following expression:

$$H_\varepsilon(\alpha) = \begin{cases} 1 & \text{if } \alpha > \varepsilon \\ \frac{1}{2}\left(1 + \frac{\alpha}{\varepsilon} + \frac{1}{\pi}\sin\left(\frac{\pi\alpha}{\varepsilon}\right)\right) & \text{if } |\alpha| \leq \varepsilon \\ 0 & \text{if } \alpha < -\varepsilon \end{cases} \tag{2.3}$$

where ε is a small parameter such that $\varepsilon = O(h_{\text{im}})$, known as the interface thickness, and h_{im} is the mesh size in the normal direction to the interface. In the vicinity of the interface, it can be computed using the following expression:

$$h_{\text{im}} = \max_{j,l \in K} \nabla\alpha \cdot \mathbf{x}^{jl}, \tag{2.4}$$

where $\mathbf{x}^{jl} = \mathbf{x}^l - \mathbf{x}^j$ and K is the mesh element under consideration. According to the chosen approximations, the Heaviside function is then approximated using linear interpolations ($P1$) between fluid and solid properties or a piecewise constant interpolation ($P0$).

Finally, we combine this approach with an anisotropic mesh adaptation algorithm to ensure an accurate capturing of the discontinuities at the fluid–solid interface. However, the level-set function intersects the mesh element arbitrarily. It is possible then to overtake the discontinuity appearing at the interface by using mesh adaptation and regularization. The regularization parameter can be seen as the thickness or the resolution of the interface. It is shown that using local adaptivity, stretched elements at the interface are obtained and one can then choose a priori the resolution of the thickness to be very small, which leads to very sharp interfaces, favorable for simulating fluid–structure conduction problems and devising exact radiating surfaces.

2.3 S2S radiation

The radiosity method has been recorded in thermal transfer handbooks since 1965 [5]. It has then been picked up in the 80s by computer graphic scientists, in order to describe how light radiates from surfaces to surfaces for correctly rendering illuminated images [6–8]. Since light is a form of radiation, the method is indeed is useful to both domains of research.

2.3.1 Overview and related work

The S2S method is applied in a domain where all the surfaces involved in radiation are split into a discrete set of n elementary small surfaces. The core principle of the S2S method then relies on the following fundamental operation: evaluating how the radiation leaving each of the n surfaces is distributed to hit all the other surfaces. This is called the *view factor matrix* computation. This process can be adjusted to the problem according to a number of specifications: the n surfaces taken for radiation computation can be directly extracted from the surface elements of the mesh. But this set can be refined if a greater accuracy is required; or facets can also be gathered into clusters if, on the contrary, speed of execution is deemed more

important than accuracy [6]. These choices can be made at a local level, and indeed some parts of the surface mesh can be refined where most of the radiation occurs while some others can be clustered in areas where radiation plays a less dominant role. Furthermore, this operation is only dependent on the geometry of the system: it can be precomputed, and as long as the geometry does not change during simulation, the computed values can be used as-is for all time steps. This adaptability and simplicity constitutes the most interesting point of the radiosity method.

Assembling the view factor matrix is key to the speed and accuracy of the S2S method, and direct numerical integration of the double surface integral (we will see it in Equation 2.9) that defines a given view factor is costly. Many S2S variants differ by the way they process that important step. Hottel and Sarofim [9] developped the so-called "crossed-string method" which allows for an analytical computation of 2D view factors. Walton [10] transforms the double suface integral computation of view factors into a simpler single linear integral, including the complicated case of partially obstructed view factors. He proceeds to numerical integration via Gauss quadrature from that stage. Cohen and Greenberg [11] devised a method called the hemi-cube technique, in which they installed a square grid surrounding a given facet and then projected each other facet onto it. This both allowed for an efficient computation of obstruction and turned the view factor computation from that between two arbitrarily-shaped facets into that between one arbitrarily-shaped facet and one square surface parallel to the facet, simplifying the computation. Franck et al. [12] have instead used a Monte-Carlo method, massively launching rays in random directions from each radiating surface, and the fraction of rays hitting the opposite surface out of the total number of launched rays gives a good approximation of the view factors between them. This means they have embedded a Monte-Carlo method to handle that one specific step of the S2S method.

2.3.2 S2S hypotheses

We will now recall the assumptions under which the S2S model is considered valid. We will in particular study the status of α-SiC, or 6H-SiC, as one of the silicon carbide used for ceramics materials by TAS' supplier.

2.3.2.1 Surface-only radiation

Surface-to-surface radiation, as its name indicates, handles radiation as a purely surface phenomenon. This means that upon striking a surface, radiation is considered to be utterly absorbed into the bulk of the radiating elements, or reflected off of it. There is no transmission. Conservation of energy then means the sum of absorbed

and reflected radiation is equal to the amount of incident radiation. It immediately follows that the ratio r of radiation that is reflected off of the given surface (integrating over all reflected directions), as a property of the surface material, is defined by

$$r_{\vec{u},\lambda} = 1 - \alpha_{\vec{u},\lambda} \tag{2.5}$$

r being the reflectivity of the material and α its absorptivity as was defined in Chapter 1 subsubsection 1.3.4. Barring any further hypotheses, these values both depend on the local material, direction, wavelength and temperature. This hypothesis also means radiation is emitted from a layer of matter that is located within a characteristic length of penetration from the surface that is small with regard to the dimension of the system. Such radiating elements are said to be *opaque*. In practise, most of the production of radiation occurs within 20 nm of the surface [13]. We have also seen in Chapter 1 that Wien's law indicates 95% of the total amount of radiant energy emitted by the black-body stands in a wavelength band of $0.5\ \lambda_m - 5\ \lambda_m$. Though we do not always deal with black-bodies or grey-bodies, this gives us a broad estimate of the usual wavelength domain for energy emission. For most industrial applications, ambiant temperature can vary between 300 K and 2500 K, which amounts to a wavelength band of 0.6 µm to 50 µm (from visible to far infrared wavelengths). One can infer from Ordal et al.([14]) that for most metals, complete absorption of visible and infrared radiation occurs within 0.1 µm of the surface. For other opaque (in the visible range) materials such as silicon nitride ceramics [15] or zirconium–yttrium ceramics [16] skin thickness for visible and near infrared wavelengths is of the order of 100 µm . This is a negligible thickness for industrial products for which typical detail size is that of 1 cm. Most liquids and gases are not opaque to radiation however, but the S2S model only considers thermal radiation between solid surfaces.

2.3.2.2 Nonparticipating medium

This brings the next hypothesis: the medium between the solid, radiating surfaces is considered transparent to radiation. Obviously this is true when the medium is vacuum, but this also holds true for diatomic gases who do not possess a polar moment (N_2, O_2, etc) [17]. That remark means that considering the S2S model even in the case of silicon nitride ceramic sintering under an N_2 atmosphere is still a valid approach. This also means that for example dry air (notwithstanding the trace amounts of gas other than N_2 and O_2) is actually transparent to radiation. The medium does not absorb, nor reflects any radiation, but fully transmits it. It is possible to extend the S2S method to the case of non-transparent media, in which case it is called the zonal method [18].

2.3.2.3 Grey-body radiation

One of the most important hypotheses we make is that silicon carbide ceramics behave reasonably like grey-bodies as far as radiation is concerned. This is critical, since the S2S modelling is only applicable in the case of grey-body radiators. We have seen in Chapter 1 that the grey-body hypothesis relies on the assumption that the spectral directional emissivity of the considered material is actually constant for all directions and wavelengths as shown in Equation 1.19.

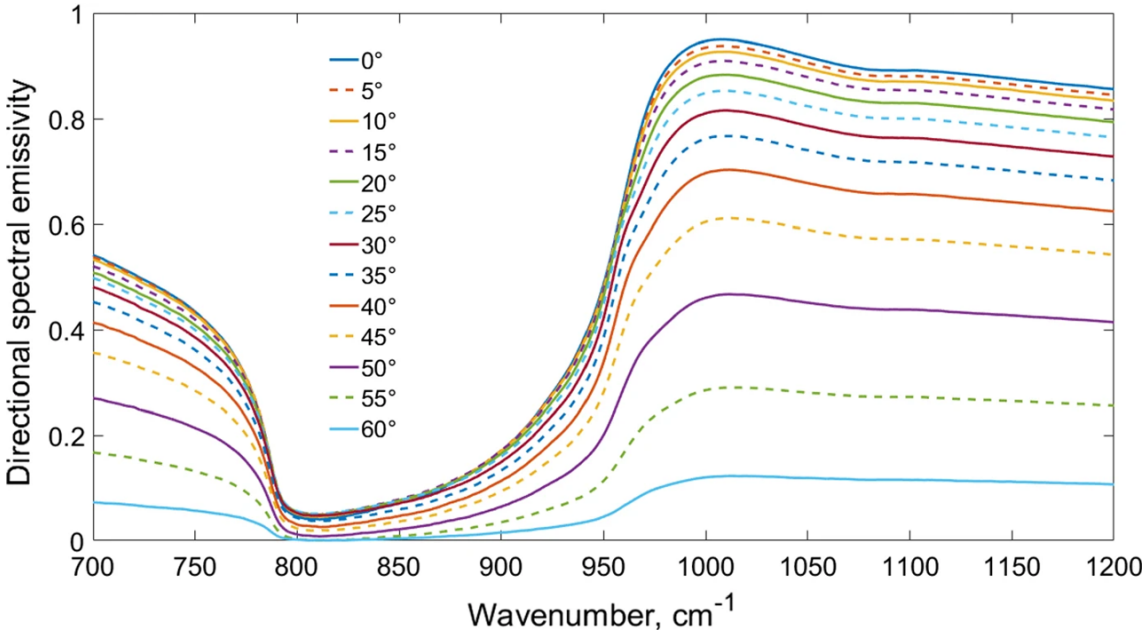


Figure 2.1: Spectral directional emissivity of SiC at various incidence angles for wavenumbers 700 to 1200 cm^{-1} (equivalent to wavelengths 8 to 14 µm from right to left) at 800°K([19]).

Direction independency Figure 2.1 is taken from Guo et al. ([19]) and displays the spectral directional emissivity $\epsilon_{\lambda,\vec{u}}$ of SiC at 800 K, between 8 and 14 µm for angles of incidence (from the normal direction) from 0 to 60°. We can see that $\epsilon_{\lambda,\vec{u}}$ varies a lot according to wavelength, but also to direction. However, the values remain within 0.5 from each other for angles of incidence between 0 and 30°. We have seen in Equation 1.9 that the flux density received by the surface is the integral of the radiance from all directions but multiplied by the cosinus of the angle to the normal. This means that the directions coming near the normal weigh more heavily in the integral than the directions close to the tangent. As a result, we would not change too much the local behaviour of radiation if we assigned to each direction the same

value of spectral directional emissivity as the value taken for the normal direction. Moreover, according to Jones et al. ([20]), an aggregation of experimental data for non-metallic radiating surfaces has shown that the ratio of hemispherical emissivity on normal emissivity is usually comprised between 0.95 and 1. This is supported by Watanabe et al. ([21]) who assess that ceramics materials can usually approximate their spectral hemispherical emissivity with their spectral normal emissivity. This indicates that, for a given wavelength, the aforementioned approximation would lead to an integrated hemispherical emissivity of almost the same value than that of the real hemispherical emissivity.

In addition to this, ceramics can be considered isotropic reflectors at first approximation ([22]), so that they are both isotropic emitters and reflectors. This satisfies the "diffuse surfaces" requirement of the S2S model.

Wavelength independency Figure 2.1 shows us that SiC emissivity is, in fact, dependant on wavelength. However, this statement can be moderated by the fact the wavelength band displayed on that figure is quite narrow, from 8 to 14 µm. Figure 2.2 is taken from [23] and indicates the normal spectral emissivity of SiC for a wide array of temperatures and over a larger wavelength band. For all temperatures, we see that emissivity is not far from being constant between 2 and 9 µm, at about $\epsilon = 0.83$. A grey-body at 800 K would, according to Wien's law as seen in Chapter 1 subsubsection 1.3.3, radiate about 80% of its total emissive power in that wavelength range. This means that a grey-body at that temperature and an emissivity $\epsilon = 0.83$ would radiate 80% of its total emissive power in a way very similar to a wafer of SiC would, according to Figure 2.2. The 8 µm to 14 µm band plays a relatively small role in the total emissive power of a grey-body at 800 K, meaning we can dismiss this band without altering the validity of the comparison too much. This is why the strong variations of emissivity for real SiC in that area do not prevent us from making a reasonable assumption in indicating that the relevant part of SiC emissivity does not depend too much on wavelength. This assumption combined with the diffuse radiation assumption detailed in the previous paragraph allows us to make the assumption that SiC radiation can be reasonably modelled as a grey-body radiation. From now on in this work, all radiating materials we will consider will be taken as grey-body emitters.

2.3.2.4 Constant emissivity

The above hypotheses are enough to allow for the S2S model. However, we will make one more assumption that, while not strictly needed, simplifies our modelling considerably. The S2S model only requires emissivity to be independant of wavelength and direction. However, that does not mean in and of itself that emissivity is now a constant value. It still depends on other local parameters: temperature

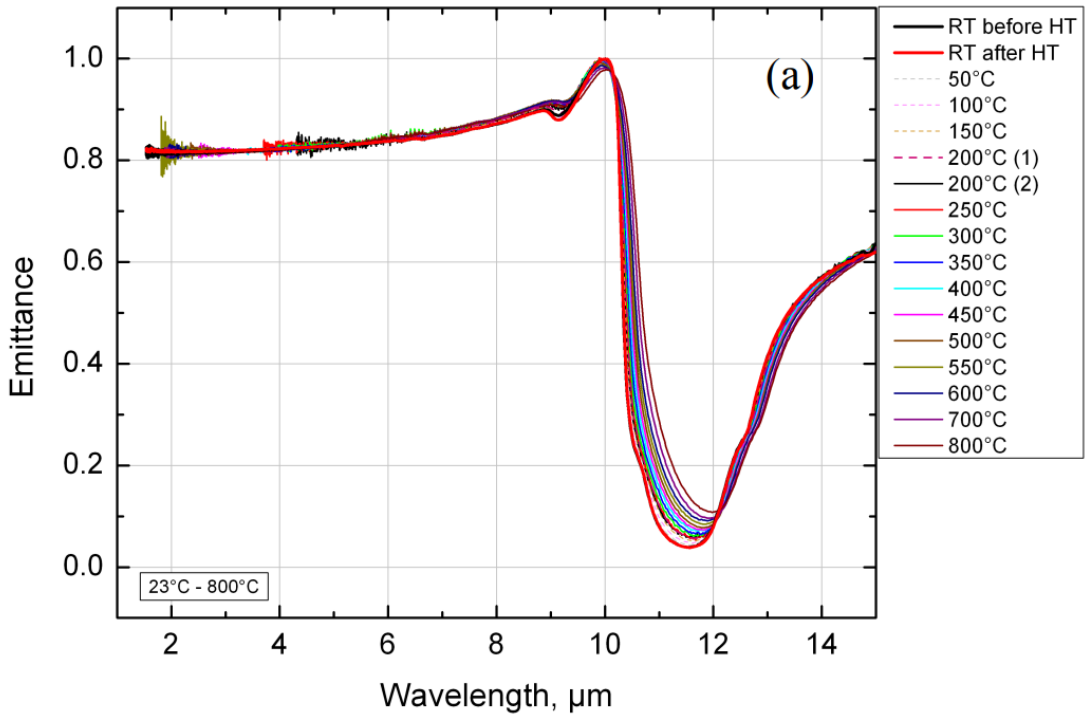


Figure 2.2: Spectral normal emissivity of SiC for wavelengths 2 µm *to* 14 µm *from room temperature to* 800 K *([23]). "RT before/after HT" means "room temperature before/after heating".*

([19, 20, 23, 24]), but also temperature hysteresis ([23]) residual surface stress ([25]), porosity ([26]), roughness ([20], air or nitrogen pressure ([24])...The order of magnitude of emissivity change these factors can impose is comprised between 0.05 and 0.2, except for the temperature as is shown in Figure 2.3.

All these factors mean that emissivity measurements are very dependant on the specific SiC samples used, and that from one source to the other values for emissivity might vary quite considerably. In addition, no article mentions the exact status of its SiC sample with regard to *all* of the aforementioned criteria. For example, Balat-Pichelin and Bousquet ([24]) detail how six different authors give varying values for normal total emissivity of SiC, and the values at around 1700 K to 1900 K move from 0.83 to 0.96.

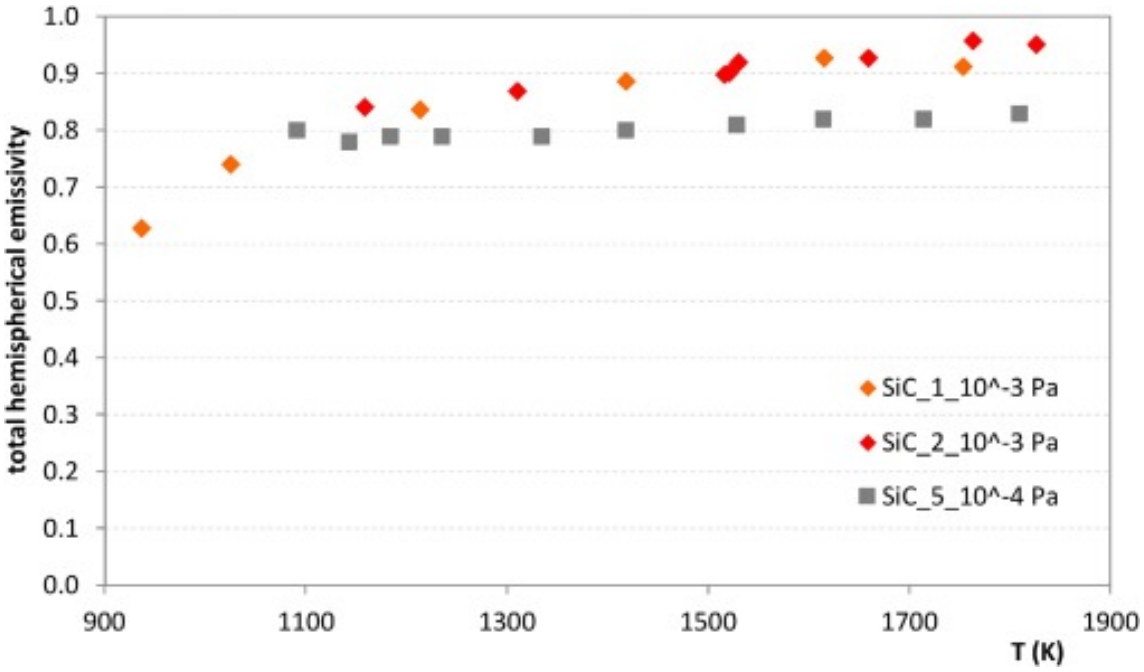


Figure 2.3: Total hemispherical emissivity of SiC under a vacuum from 0.27 mPa *(SiC-5) to* 1.5 mPa *(SiC-1 and SiC-2) and from* 900 K *to* 1900 K *([24])*

Figure 2.3 comes from their publication. We can see how different magnitudes of vacuum actually play a non-negligible role on the emissivity. In addition, temperature plays a greater role on the emissivity of lower-vacuum SiC. In the case of the sintering process we are trying to model, it takes place under an atmosphere of about 1 mPa. Though such a pressure does not appear in Figure 2.3, we can conservatively infer it would present a dependancy to temperature on the same order of magnitude than the two cases at 1.5 mPa. This would mean it can vary from about 0.6 to about 0.95 as temperature increases from 900 K to 1900 K.

The conclusion for us is that, for a given SiC ceramic part, mapping out the exact field of total hemispherical emissivity on its complete surface would require us to compute, in real-time as the simulation goes on, the exact surface condition of the ceramic with regard to half a dozen parameters. But even if we were able to create an advanced model that could perfectly foretell the density, the pressure, the roughness, etc. we still would not be able to bring forth a perfectly accurate emissivity value, since the exact relationship between all these values and emissivity is, at most, a correlation under specific circumstances. Besides, emissivity values given by different papers vary between each other by an amount which is of the same order of magnitude than the amount due to porosity or surface stress variation. In the end, there is no point in devising an emissivity model trying to be accurate up to more than being rounded to the nearest 0.1.

For this reason, we make a great simplification step and consider emissivity to be absolutely constant in all regards during our simulations. This is a first approach that will be refined as the solver is further enhanced. Since the impact of temperature on emissivity is not only quite high, as shown in Figure 2.3, but also better-documented than most other factors enumerated before, we expect the next step in emissivity refinement will be to move from constant to temperature-dependant emissivity.

Consequence on absorptivity and reflectivity An immediate consequence of this hypothesis is Kirchhoff's law expanded to grey bodies: when a given body at temperature T is at thermal equilibrium and subject to incoming radiation from a given direction $\vec{\Omega}$ and at a given wavelength λ, it absorbs a portion α of that radiation. It must then emit back the same amount of radiation in the same direction and wavelength so that energy is conserved. In the most general case, Kirchhoff's law thus concludes that directional monochromatic absorptivity $\alpha(\vec{\Omega}, \lambda, T)$ of that body must be equal to to its directional monochromatic emissivity $\epsilon(\vec{\Omega}, \lambda, T)$:

$$\alpha(\lambda, \vec{u}, T) = \epsilon(\lambda, \vec{u}, T) \tag{2.6}$$

This law is true without the need for the grey body hypothesis. But if we consider all radiative properties to be, in fact, constant, it amounts to a much more general relation between emissivity ϵ, absorptivity α and reflectivity r:

$$\alpha = \epsilon \tag{2.7}$$

$$r = 1 - \epsilon \tag{2.8}$$

Henceforth, we will directly make use of ϵ, and $1-\epsilon$ respectively, whenever emissivity and absorptivity, or reflectivity respectively, are needed. We will not reference α or r anymore.

Then, to compute immersed radiative transfer, we need more than an implicit definition of the interface Γ_1. The first step of the S2S algorithm is to reconstruct a list of explicit facets out of the immersed mesh to determine where radiation effectively takes place.

2.3.3 Facet list construction

S2S methods used in the context of body-fitted meshes usually have a direct access to the list of facets (that is, lines in 2D and triangles in 3D) that contributes to radiation. Indeed, these facets are the border elements of the mesh of the domain and of the mesh of the solid items. When used in combination with our immersed mesh, all participating facets are not border elements: the facets on the solid interface Γ_1 actually belong to bulk elements, so there is no easy access to the list of elements containing the participating facets. Moreover, in immersed meshes, participating facets do not always (and in fact, rarely do) coincide with existing element facets.

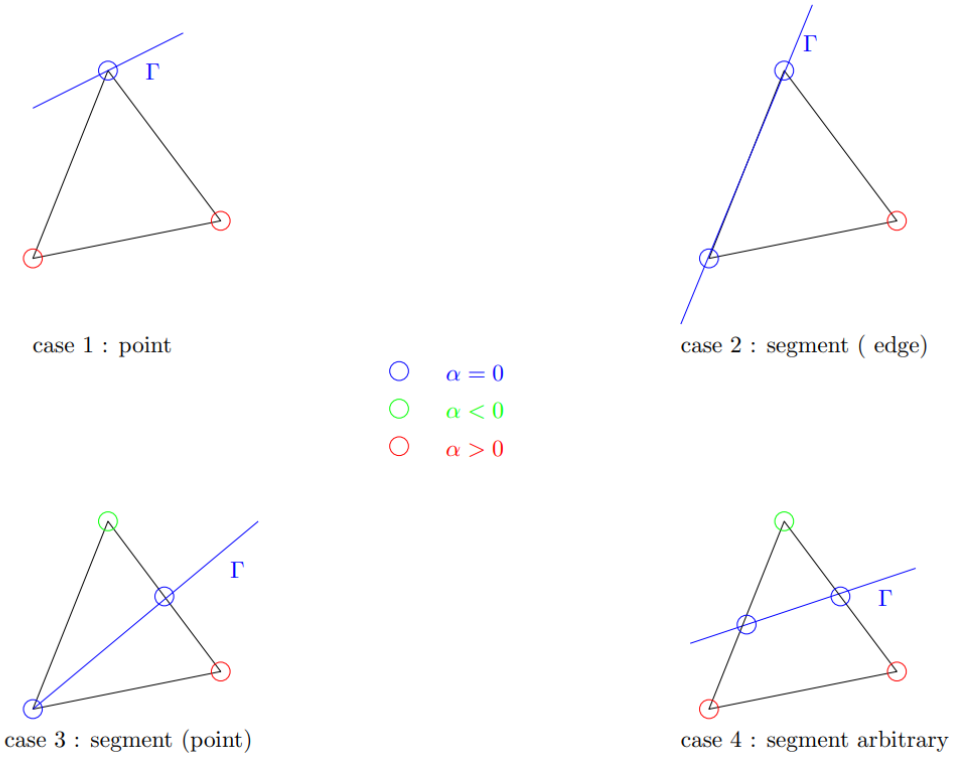


Figure 2.4: 2d intersections of elements by the solid interface.

The first step is thus to determine all the elements that are intersected by the

solid interface Γ_1. Figure 2.4 shows the multiple cases that can be encountered in 2D. For each element of the mesh, the solver evaluates the level-set at each vertex of the elements. Several cases then arise:

1. The level-set takes a 0 value at a single vertex of the element, and takes either all-negative or all-positive values at all the other vertices (case 1 in Figure 2.4). In such a case the solver does not add anything to the list of particpating facets.

2. The level-set takes a 0 value at exactly two vertices of the element (case 2 in Figure 2.4). This means one of the edges of the element is indeed on the interface Γ_1, and this edge is added to the list of participating facets.

3. The level-set takes a 0 value at one vertex, and two values of different signs at the other two (case 3 in Figure 2.4). In that case the solver creates a virtual vertex at the position of the intersection between the solid interface and the edge of the element between the two vertices at which the level-set takes a non-zero value. It then creates a virtual facet anchored between the 0-value vertex and the virtual vertex, and adds it to the list.

4. The level-set takes three non-zero values, not all of them being of the same sign (case 4 in Figure 2.4). This means the solid interface crosses the element right through the middle. Similarly to what happens in case 3, the solver creates two virtual vertices and then a virtual facet between them. This facet is then added to the list.

5. The level-set function takes all-negative or all-positive values at all vertices. This means the solid interface does not cross the element at all.

The outer shell of the domain Ω is also defined by a level-set function, and the facets of the border of the domain that contribute to radiation are also computed by interface detection. We could have simply retrieved the flat elements composing the border of the domain, but detecting them by level-set as well ensures maximum generability of the method. This way it is easy to modify the dimensions of the outer radiating object (i.e., a room's walls) on the fly without having to remesh the domain.

2.3.4 Flux computation

Now that we have the list of all radiating facets of the system at hand, let us introduce the physical meaning of the S2S method. It relies on energy conservation. That is, in any system comprising radiation, all radiation that is emitted somewhere will be absorbed elsewhere. There is no loss of energy. In an enclosed system such as a kiln, this can easily be understood. In an open system such as a solar collector,

the sky itself can be segmented into radiating facets that have a certain power of absorption and emission, so that we fall back to an enclosed radiating system. In such a system, a given facet F_i emits radiation from black-body emission. In turn, it partially re-emits and partially absorbs radiation from black-body emission of a facet F_j, plus radiation emitted by a facet F_k, reflected by F_j towards itself. Approaches that handle radiation ray by ray, like Monte-Carlo or the DTRM, have to carefully consider how reflection is handled, possibly multiple times in a row for a given ray until absorption occurs. This is a complicated and time-consuming matter. The radiosity method bypasses this problem by computing, at each time step, the *total* amount of radiation leaving each facet - implicitely including the multiple reflections each facet receives and emits back. This approach is, of course, made possible by the assumption that reflection proceeds in an isotropic way. Under this hypothesis, if facet F_i of emissivity ϵ_i receives a net amount J of thermal radiation from facet F_j and an amount K from facet F_k, then it will isotropically reflect back in the half-space before it an amount of radiation equal to $(1 - \epsilon_i)(J + K)$, regardless of the specific positions of F_j and F_k. This situation is equivalent to that of facet F_i receiving $J + K$ radiation at once from facet F_j only and nothing from facet F_k. The end result is thus that the S2S method assesses thermal radiation as a *purely scalar value*, that can be linearly composed with itself regardless of its source. In other words, the S2S method deals with the scalar flux density $Q_{radiation}$ and the scalar flux $W_{radiation}$ as they were introduced in Chapter 1.

2.3.4.1 View factor computation

The directional nature of the flux density, however, is not simply blanked out by the S2S method. It is taken into account through a quantity called the *view factor* or *form factor*.

Definition and general properties The view factor is defined for a pair of facets: F_{ij} is the view factor pertaining to the facet pair (F_i, F_j). The view factor matrix mentioned at the beginning of this chapter is set up to account for the complete radiating system. The intuitive definition of the view factor F_{ij} is that F_{ij} is equal to the proportion of the total radiation emitted by facet F_i that will strike facet F_j. Its mathematical definition is as follows. Let there be two facets F_i and F_j of respective surfaces S_i and S_j, normals $\vec{n_i}$ and $\vec{n_j}$, separated by a vector $\vec{r_{ij}}$. These normals are such that the angles between $\vec{n_i}$ and $\vec{r_{ij}}$ and $\vec{n_j}$ and $\vec{r_{ij}}$ are, respectively, Θ_i and Θ_j. This setup is displayed with elementary 2D surfaces in Figure 2.5. Then:

$$F_{ij} = \frac{1}{S_i} \int_{S_i} \int_{S_j} \frac{\cos(\Theta_i).\cos(\Theta_j)}{r_{ij}^2} \, \mathrm{d}\Theta_j \, \mathrm{d}\Theta_i \tag{2.9}$$

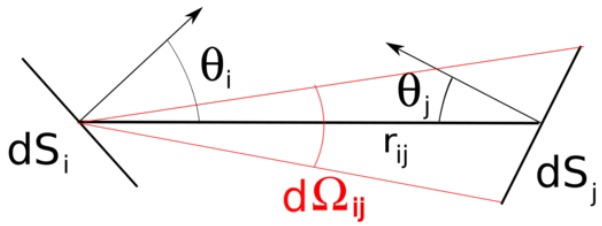


Figure 2.5: Relative position of elementary surfaces dS_i and dS_j.

With such a definition, that means if we also define ϕ_i the radiative flux density emitted by facet F_i and S_i its surface, we can then compute Φ_{ij} the radiative flux transmitted between F_i and F_j:

$$\Phi_{ij} = S_i \phi_i F_{ij} \tag{2.10}$$

The view factor is an adimensional number depends on geometrical parameters only, and needs to be calculated only once at the first time step of the simulation (except if at some point any radiating item moves inside the system). Filling the view factor matrix is, however, an arduous process. The view factor is quite a computationally expensive term to calculate, because of its double integral nature. Moreover, the radiosity matrix is rarely sparse, so a system involving N facets will have to calculate $O(N^2)$ terms. As a result, finding various ways to accelerate the computation of the view factors is in and of itself a dense topic of research [27–30]. First, three properties of the view factor allow for a more efficient computation. Given a closed system containing N facets, for any given facets F_i and F_j of respective surfaces S_i and S_j:

$$F_{ii} = 0 \qquad \text{no self-radiation} \tag{2.11}$$

$$S_i F_{ij} = S_j F_{ji} \qquad \text{reciprocity of radiation} \tag{2.12}$$

$$\sum_{j=1}^{N} Fij = 1 \qquad \text{conservation of energy} \tag{2.13}$$

These three relations are proven as follows. Since a facet is a flat plane for a 3D object and a straight segment for a 2D object, it is not strictly concave. As a result, no part of a facet can "see" any other part of the facet. Hence Equation 2.11 is derived: a facet can never emit radiation onto itself. Then, if we return to Equation 2.10, we notice that the amount of radiation transmitted between F_i and F_j can also be written as a function of the flux density received by facet F_j, ϕ_j:

$$\Phi_{ij} = S_j \phi_j F_{ji} \tag{2.14}$$

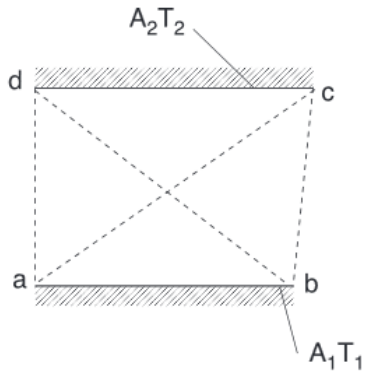


Figure 2.6: Crossed strings method to determine the view factor between two surfaces in 2D.

Since that holds true regardless of the values taken by ϕ_i and ϕ_j, if we consider the case of a system composed of only these two facets (like two infinitely extended walls facing each other), then we would have

$$\phi_i = \phi_j \leftrightarrow S_i F_{ij} = S_j F_{ji} \tag{2.15}$$

We thus derive Equation 2.12. Finally, Equation 2.13 comes from the fact that in a closed environment, the complete half-space standing before facet F_i is defined by radiating facets with no hole through which radiation could escape. This means all radiation leaving F_i is spread out between all the other facets of the system.

The first advantage of these formulae is that thanks to Equation 2.12, one only has to compute half of all the view factors and deduce the other half through reciprocity of radiation. Moreover, for any one give facet F_i, we only consider the facets F_j such that $\vec{n_i}$ points towards F_j and $\vec{n_j}$ points towards F_i. For all other cases, this means at least one of the facets is "turning its back" to the other, and as such no radiation should be considered and the view factor is set to 0. After this, several ways exist to compute the view factors in an efficient way.

Crossed-strings method For 2D geometries, we have already mentioned the crossed-string method [9], that provides an analytical answer to the problem. Given two surfaces (even if of curved shape) A_1 and A_2, of temperature T_1 and T_2, formed of the lines (ab) and (cd), then:

$$A_1 F_{12} = A_2 F_{21} = \frac{ad + bc}{2(ac + bd)} \tag{2.16}$$

Figure 2.6 explains the process. It works even if there is a third facet F_k partially obstructing the view between the two others: the "ropes" can loop around the edges

of F_k to still connect the edges of F_i and F_j. This allows quick computations of 2D view factors. Since this method is very fast and gives the exact result for the view factor, there is no reason to ever use any other method as far as 2D computations are concerned. Unfortunately, it cannot be extended to 3D geometries.

Hemicube method The *hemicube method* from Cohen and Greenberg [31] originates from image rendering, but like other view factor-related methods it also has its uses in thermal transfer (Gaume [32]). It relies on the following geometric property: for a given point looking at a given solid angle cone, different surfaces can be projected onto the viewer the same way, like the Moon eclipsing the Sun despite their different distance from Earth. Figure 2.7. shows the practical application of this property to view factors.

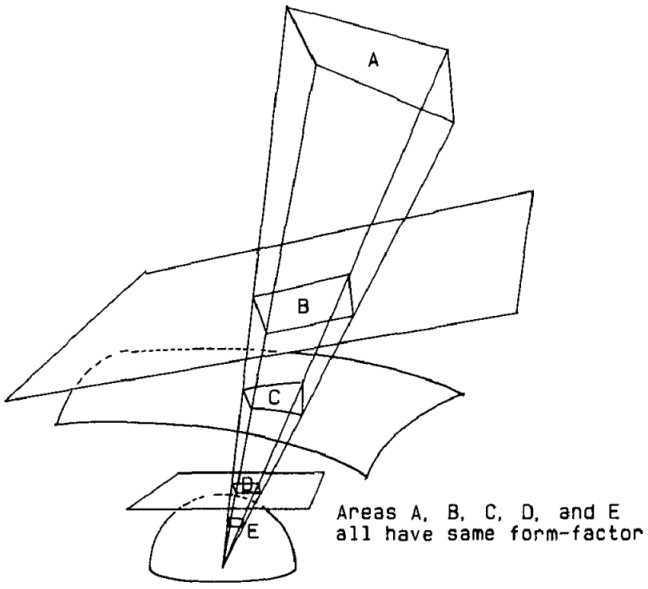


Figure 2.7: Different shapes, when projected into the same surface, render the same projected area ([31]).

This allows us to swap the computation of the view factor for an arbitrarily complex surface to that of the view factor for an analytically resolvable surface. All there is to do is to find the proper simple surface to place between the emitting point and the receiving shape. In practise, we want to establish the view factor from S_i to S_j. We establish a hemicube-shaped mesh around surface S_i, composed of square elements. Starting from a given elementary surface dS_i from S_i, we draw the cone-like projection extending to the borders of S_j. This projection will intersect the hemicube through several elementary squares. Then, the view factor $dF_{i\to j}$ is equal to the sum of the view factor between dS_i and each of the intersected elementary

squares. Since they are either parallel or normal to the plane containing dS_i and they are of a very simple shape, numerical integration of each of these elementary view factors is very fast. Once this is done, $dF_{i\to j}$ is integrated over all of surface S_i to finally access $F_{i\to j}$. Figure 2.8 sums up the idea behing this method. A hemisphere can be used instead of a hemicube.

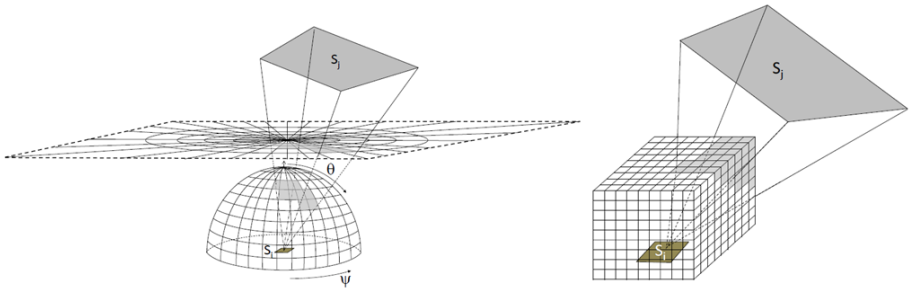


Figure 2.8: Hemicube method using either a cubic or a spherical mesh ([32]).

The hemicube method computes view factors one line of the matrix at a time. For a given facet S_i, all other facets are projected onto its hemicube. If a given hemicube mesh element is covered by two or more projections originating from different facets, a distance check is made to see which one is closer from S_i. That projection is put above the other.

The hemicube method has the drawback of discretizing the facet projections into a set of square mesh elements. If the hemicube is not refined enough, aliasing will appear and view factors will either be over or underestimated. More involved versions of the hemicube method have been devised to combat these side effects [33].

Monte-Carlo method This method seen in Frank et al. [34], Daunt et al. [35] or in the commercial software FloTHERM consists in using Monte-Carlo ray tracing, not for computing the thermal exchange itself as seen in Chapter 1, but for computing the view factors only. Similarly to the fact that this ray tracing could allow us to estimate repartition of the flux from facet to facet without having to solve the complicated RTE, here it allows us to estimate view factors without having to solve Equation 2.9. A set number of points is randomly determined on each facet, and a ray is launched from each point in a random direction. When the ray encounters a facet, it has a random chance to be either absorbed or bounced back. This depends on the emissivity of said facet. We keep track of which facet each ray eventually gets absorbed into. View factor F_{ij} already being an adimensional ratio, it is very simply approximated by counting the number of rays leaving F_i that have hit F_j divided by the total number of rays leaving F_i. Since rays are treated as finite-length

segments that are terminated as soon as they are absorbed somewhere, a given ray can never be absorbed by more than a single facet. This naturally solves the problem of obstruction: if a ray ends up being absorbed somewhere, this means there is no possible facet that was standing on its path that it got through. Enhancements such as the quasi-Monte-Carlo methods are also implemented by Frank et al. The strengths of the Monte-Carlo method remain the same in that context: it bypasses through a very simple and scalable process a view factor computation that can be very costly, and it naturally takes obstruction into account.

However, its weaknesses do remain the same too. Unless an extremely high number of rays is launched, it is difficult to satisfy simultaneously Equations 2.12 and 2.13. Due to the randomness of the ray tracing, even for two facets F_i and F_j of same shape and dimension, when rays are launched from F_i and received by F_j, it often amounts to F_{ij} being slightly different than F_{ji}. An *ad hoc* correction can be applied to ensure reciprocity, but then it will unbalance energy conservation. Taylor et al. [36] suggest one could compute only half the view factors and then use the reciprocity law to deduce the other ones, but then that would amount to spreading the random error that is present in the computation of the first half of the view factors. Another method proposed by van Leersum [37] is to determine all view factors through Monte-Carlo ray tracing and then iteratively modify them one after another so that they progressively satisfy Equations 2.12 and 2.13 with a decreasing amount of error. But Daunt et al. point out this means changing the view factors away from their real theoretical value in an uncontrolled way. They have developed a technique using constrained maximum likelihood estimation to maximize the probability that the view factors in the matrix will respect reciprocity and energy conservation, will all be positive, and that symmetry of view factors is ensured for symmetric facets. Their method is quite involved and entails solving a constrained nonlinear programming problem. But then, we are starting to lose the ease of use and simplicity of the Monte-Carlo method.

Adaptive integration The idea of this method is directly solve the view factor equation, but by applying mathematical transformations to make it simpler. View factors are computed one by one, instead of by batch as is the case for the hemicube method. Walton [38] has dubbed this process *adaptive integration*. When surfaces S_i and S_j are small compared to the distance separating them, there is little benefit in taking into account with a great degree of accuracy all the small details of one surface, given they will be almost invisible to the other surface. A naive numerical integration of the view factor equation, even with a coarse quadrature, will yield a result close to that of an analytical solution. On the other hand, when the two surfaces are very close to each other, accurately capturing the shape of each matters a lot. A direct numerical integration would require a very fine meshing of the sur-

faces, for a prohibitive computation time. In such cases, one can turn to Schröder and Hanrahan [39], who have established analytical formulas that return the exact view factor between two polygons of any shape, though they include dilogarithm functions, that are themselves costly to numerically evaluate. It is 200 times faster to evaluate their formula to compute the view factor between two squares sharing an edge than for the view factor between two separate squares of arbitrary relative position. This indicates the purely analytical method is not always the most judicious choice. Between these two extremes, one can apply a degree of transformations upon the view factor equation before numerically solving the new expression. Stokes' theorem turns the double surface integral of Equation 2.9 into a double linear integral. We then numerically integrate only over the edges of the facet, which is less expensive. We split the contours of the facets into n_i and n_j,and have the following:

$$F_{ij} = \frac{1}{4\pi S_i} \sum_{k=1}^{n_i} \sum_{l=1}^{n_j} \int_{e_k^i} \int_{e_l^j} \ln(d(e_k^i, e_l^j)) u_k v_l \, \mathrm{d}u_k \, \mathrm{d}v_l \tag{2.17}$$

where e_k^i is the k-th segment making up the discretized contour of F_i, $d(e_k^i, e_l^j)$ is the Euclidean distance between the two segments being integrated and u_k and v_l are the integration variables. A 3-points Gauss quadrature is then used to compute the value of the logarithm function. As one can see, a problem arises when two facets sharing an edge are analyzed, as the logarithm will be evaluated at 0. A different formula is used in this case [40]. For two elementary surfaces sharing an edge dl:

$$\mathrm{d}^2 F_{ij} = \frac{\mathrm{d}l^2}{2}(3 - \ln(\mathrm{d}l^2)) \tag{2.18}$$

One can also refer to tables of differential view factors from a differential area to a surface of a certain simple shape (from for example Howell [41]) and integrate these differential view factors over the first surface. This turns Equation 2.9 into a simple surface integral. Using such transformations, Walton has reported how the double surface integral in Equation 2.9 can be turned into any of a double linear integral, single surface integral, single linear integral, and up to the direct analytical formulation of Schröder and Hanrahan. Softwares such as View3D have a store of available such combined analytical-numerical approaches and for each surface pair they have to evaluate, they choose the most appropriate one to compute the view factor at hand.

A drawback of this method is that it does not naturally account for obstruction, as opposed to the hemicube method for example. For a given surface pair we have no *a priori* knowledge of a potential third surface standing in the middle of the two and blocking off thermal transfer. The naive way to proceed is to evaluate all the remaining surfaces of the system, but the computational cost is prohibitive.

Between the hemicube method, the Monte-Carlo method and the adaptive integration method, all seemed equally relevant to the topic of ceramic sintering than the others. We decided to select adaptive integration as it was the lightest one to implement. Thus we had a working version of the solver as fast as possible. We decided, as a first version, to implement the double linear transformation as shown on Equation 2.17 for all view factors to be computed, and to later implement the real "adaptive" part of the process that would adjust the transformation used to each considered view factor. That meant we had to deal with a dedicated obstruction handling step, which will now be detailed.

2.3.4.2 Obstruction handling

The last parameter to consider when computing the view factors between two facets is the possibility that a third facet is located between these two and blocking off, totally or partially, emitted radiation from one to the other. Obstruction handling is very important, because if properly carried out, it allows to assign 0 to a number of view factors without having to compute them. This also makes the view factor matrix more sparse. Finally, is it crucial for conservation of energy as a whole that radiation be not computed where none should be received. Obstruction handling is a costly operation, because at first approximation, for each of the N^2 view factors F_{ij} in a system of N facets, there are $O(N)$ possibly obstructing facets to be checked. That means view factor computation as a whole takes now $O(N^3)$ operations. As a result, many workarounds are deployed to try and diminish the required number of operations, and some papers focus specifically on this point [30]. Two of the simplest filters for obstruction that we have implemented are as as follows:

1. For closed systems of a convex shape, the facets that make up the border of the system cannot be in the way of any possible couple of other facets. For nonconvex systems, one can still compute, for any given border facet B_i, the plane in which B_i belongs, and then if all other facets belong in only one of the two half-spaces defined by this plane (that is, whether all other facets are "above" or "below" B_i). Such facets can be removed from the list of facet checked for obstruction.

2. Facet F_k must lie in the portion of space between the planes containing F_i and F_j.

The biggest simplification made with obstruction, however, is the following: obstruction is considered to be a boolean value. Either facet F_k completely obstructs the pair (F_i, F_j), and we have $F_{ij} = 0$; or it does not obstruct at all, and we proceed to check for F_{k+1}. Theoretically, one should account for partial obstructions,

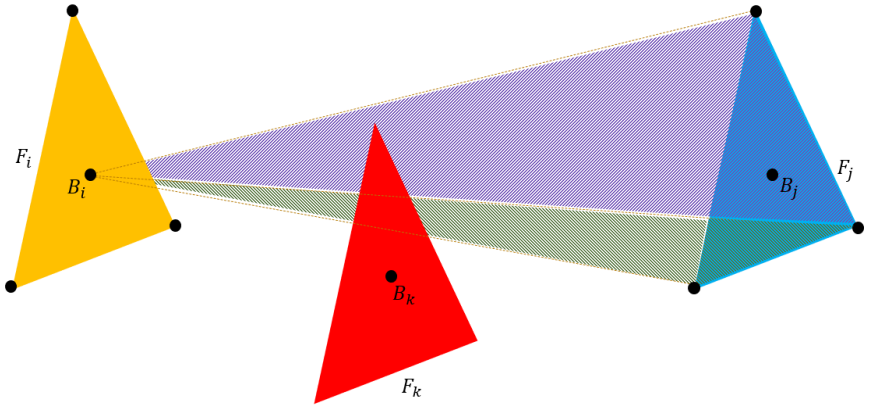


Figure 2.9: Obstruction check for F_k between F_i and F_j. B_k is not within the projected tetrahedra and thus F_k does not obstruct F_{ij}.

and devise a corrected view factor accordingly. However, such an operation is extremely computationally costly for a modest benefit in accuracy if the facets are small enough. When the mesh is refined enough, the resulting facets tend to be very small before the objects immersed in the mesh, so that in practice all obstructions are all-or-nothing.

Obstruction is defined by the following: when considering facets F_i, F_j and F_k, the barycentres B_i, B_j and B_k are computed. Then, vectors from B_i to each of the vertices of F_j are computed. This defines a triangle in 2D, its edges being the two such drawn vectors and facet B_j itself; or a tetrahedra in 3D, its edges being the three such drawn vectors and the edges of facet B_j itself. Then, we check whether barycentre B_k is inside this triangle or tetrahedra. If it is not, we then construct the vectors from B_j to the vertices of F_i, define again a triangle or tetrahedra, and check again whether B_k is within it or not. If the answer is still no, we then consider facet F_k to be too offset from the path of radiation from B_i to B_j to obstruct it. F_k is considered non obstructing, and we move on to check for F_{k+1}. Figure 2.9 sums up this process.

2.3.4.3 Radiosity

A useful intermediate value is introduced, which is called *radiosity*, noted J. It stands for the sum of all surface radiation density (in $\mathrm{W\,m^{-2}}$) leaving the facet, regardless of radiation that is received by it. The radiosity for a given facet is related to its radiative flux density, the determination of which is the end goal of the S2S method. Hence we strive to determine the radiosity vector $\mathbf{J}$. This means that facet F_j receives from F_i (of surface S_i) a total amount of radiation equal to $S_i F_{ij} J_i$. In return, F_i receives from F_j an amount of radiation equal to $S_j F_{ji} J_j$. Thanks to

the rule of reciprocity (Equation 2.12) we can rewrite this as F_i receiving $S_i F_{ij} J_j$. Radiosity J_i of facet F_i with temperature T_i and emissivity ϵ_i is thus defined to be equal to the sum of the flux density produced by the facet by black-body emission and the reflected part of all radiosity received from other facets. We recall that reflectivity is equal to $1 - \epsilon_i$ given our hypotheses.

$$S_i J_i = S_i \epsilon_i \sigma T_i^4 + (1 - \epsilon_i) \sum_j S_j F_{ji} J_j \tag{2.19}$$

Equation 2.19 shows that the vector of radiosities can be deduced from a matrix inversion with some reformuling:

$$J_i - (1 - \epsilon_i) \sum_j F_{ij} J_j = \epsilon_i \sigma T_i^4 \Leftrightarrow \mathbf{MJ} = \mathbf{T} \tag{2.20}$$

where matrix $\mathbf{M}$ and vector $\mathbf{T}$ are defined as follows:

$$M_{ij} = \begin{cases} 1 & \text{if } i = j \\ -(1 - \epsilon_i) F_{ij} & \text{if } i \neq j \end{cases} \tag{2.21}$$

$$T_i = \sigma \epsilon_i T_i^4 \tag{2.22}$$

As a result, we simply need to invert matrix $\mathbf{M}$. It is not very sparse: common orders of magnitude for the ratio of non-zero values are between 40% and 60%. That means a direct inversion would be quite computationally costly. However, it also is a symmetric positive definite, diagonally-dominant matrix. There are a number of schemes one can use to invert that matrix. Leblond et al. [42] explain that while Gauss-Siedel (GS) and Jacobi methods will both converge, GS systematically faster. However, GS takes a greater number of iterations to converge when the system presents a high emissivity. This is precisely the situation we are in: ceramic and graphite emissivity is of the order of 0.8 or higher. The conjugate gradient (CG) method is relatively insensitive to system emissivity, so it seems more suited to our problem. Baranoski et al. [43] have compared GS, CG as well as two other methods: the progressive refinement method and the Chebyshev method for radiosity resolution. They conclude the Chebyshev method is the fastest of the four in most cases, except for very high average emissivity and matrix density, where CG performs best.

We have decided to use the KSP online package for linear system resolution; it offers, among others, the GMRES method, which is very close to the CG method (Favre et al. [44] made the same choice). CG convergence is only guaranteed with symmetric positive definite matrices, which is the case of the radiosity matrix $\mathbf{M}$ in theory. However, it might be that due to numerical imprecisions in view factor computation, our matrix $\mathbf{M}$ is actually not really symmetric. GMRES converges

even for nonsymmetric matrices, and for this reason that choice seemed safer even if maybe slower than CG in the best cases. This is why we have chosen the GMRES solver for our case. It is interesting to keep in mind we could try the Chebyshev solver, also provided by KSP, in case we try to model a low-emissivity test case.

The GMRES resolution method turns computational time from $O(N^2)$ to $O(N)$ operations for a matrix of order N. Once the matrix is inverted, we obtain the radiosities vector. A final formula allows us to extract the net radiant exitance from the radiosity:

$$Q_i = \frac{\epsilon_i}{1-\epsilon_i}(\sigma T_i^4 - J_i) \tag{2.23}$$

A last check is made at this stage. Due to energy conservation, we should have $\sum_i S_i\phi_i = 0$. It happens often enough that the sum of the fluxes is actually a non-negligible quantity, due to inaccuracies in view factor computation or in matrix inversion. To ensure no energy is either created or lost, we define a vector of secondary fluxes through a very simple correction:

$$Q'_i = Q_i - \frac{1}{S_i N}\sum_j S_j Q_j \tag{2.24}$$

Energy conservation is then guaranteed for that flux density vector. Flux density is then complete and the S2S solving is over. We obtain the radiative flux received by each facet W_i:

$$W_i = S_i Q'_i \tag{2.25}$$

2.3.4.4 Behaviour over time

During the very first time step of the computation, all of the aformentioned operations have to be executed. From that point on, as long as the radiating items do not move, there is no need to perform again view factor and obstruction computation. This speeds up considerably the computation of flux density for each subsequent time step, for two reasons.

1. The radiosity matrix $\mathbf{M}$ appearing on Equation 2.21 has not changed. There is only the blackbody emission vector $\mathbf{T}$ to update. For this we update the temperature field, as it has been output by resolution of thermal transfer at the previous time step, and from that we recompute $\mathbf{T}$.

2. Using reasonably short time steps, we make the assumption the temperature field $\mathbf{T_{n+1}}$ is only slightly different from that of the previous time step $\mathbf{T_n}$. We thus assume the solution vector $\mathbf{J_{n+1}}$ is not too different from the solution vector $\mathbf{J_n}$ computed at the previous time step. Thus, we set $\mathbf{J_n}$ as the initial guess for the resolution of the GMRES method. This accelerates convergence considerably.

As soon as an item has moved in the environment, then the whole process needs to be restarted. Our use of an immersed mesh for facet determination is a first step into handling that restarting in an efficient way in a future development of the solver.

2.3.5 Coupling with thermal transfer

Now that we have computed the list of the radiative flux for all the facets in the system, let us see how we can plug this data in the general resolution of thermal transfer. We use a convection–diffusion (CD) equation, which is a volumic equation that solves thermal transport for a given P1 element.

$$\rho c_p \frac{\partial T}{\partial t} + \rho c_p \vec{v} \cdot \vec{\nabla} T - \lambda \vec{\nabla} \cdot (\vec{\nabla} T) = Q \tag{2.26}$$

where:

- $\lambda \vec{\nabla} \cdot (\vec{\nabla} T)$ stands for the diffusive term. λ is the conductivity in the given element.

- $\rho c_p \vec{v} \cdot \vec{\nabla} T$ stands for the convective term. v is the speed of the fluid (if any) at the boundaries of the element. ρ and c_p are the volumic mass and the specific heat capacity.

- Q stands for the source term. It accounts for any input (or output) of energy, of which amount has already been calculated earlier in the time step or is constant with regard to the temperature (like energy produced by radioactivity).

All values in the above equation are volumic in nature. That explains why radiation does not have a term directly associated to it: in the context of the S2S modelisation, it is an inherently surface phenomenon. In order to account for the energy gained or lost by the element through radiation, we compute beforehand how much net absolute energy is brought by radiation to the edges of the cell. We then divide that absolute amount by the volume of the element, so that we obtain an equivalent volumic radiative terme Q_r. That term is then added to Equation 2.26 as a source term. The underlying physical hypothesis is that we essentially consider radiation to be stationary when compared to convection or diffusion. Indeed, at each new time step, we take as input the field of temperatures calculated at the end of the previous time step, we first solve the radiation with the S2S method using that temperature, and we then use the radiation field as an input for Equation 2.26 after the aforementioned conversion. This is justified, however, by the fact that the charateristic time of radiation is $1/c$, much smaller than the characteristic times of convection or conduction.

2.4 Numerical results

We present here the results of the simulations produced by the proposed solver coded as explains beforehand. We conduct comparisons with analytical test cases. We will use immersed mesh definition for the radiating items when applicable, though body-fitted mesh definition is also an option. We first recall the set of assumptions under which the S2S modelling takes place in this chapter:

1. All solid are fully opaque to radiation.
2. Emissivity is a constant for a given surface.
3. Radiating surfaces are diffuse.
4. The medium between radiating surfaces is fully transparent to radiation.

2.4.1 Simple radiation

The first step is to verify the proposed solver performance in cases of purely radiative thermal transfer. To this end, we consider several simple geometries, both in 2D and in 3D, for which analytical solutions can be found.

2.4.1.1 2D test cases: concentric circles

We will evaluate radiation between two concentric circles separated by a nonparticipating medium. Since theory tells us the radiative flux density should be equal all along the surface of each of these two circles for symmetry reasons, we can make detailed analyses of the accuracy of our algorithm. The two circles have a respective radius of $r_{in} = 0.4$ and $r_{out} = 0.6$ with an emissivity $\epsilon_{in} = \epsilon_{out} = 0.5$ for both surfaces. We set $T_{in} = 500\,\mathrm{K}$ and $T_{out} = 1000\,\mathrm{K}$. Since this is a simulation between two boundary surfaces, there is nothing to immerse. The mesh is uniform with a mesh size of 0.02, which amounts to a total of 8756 elements, and out of these 476 radiating facets. Figure 2.10 displays the setup.

In such a case, the only physical phenomenon that is considered is thermal radiation between two concentric circles at a fixed temperature. When steady state is reached, the net exchanged radiation is null (since the system is closed). The total amount of radiative flux $W_{tot,i\to o}$ leaving the inner circle and absorbed by the outer circle is (see [45]) with σ standing for the Stefan constant:

$$W_{tot,i\to o} = \frac{2\pi r_{out}\sigma(T_{in}^4 - T_{out}^4)}{(1/\epsilon_{in}\cdot r_{out}/r_{in} + 1/r_{out} - 1)} \tag{2.27}$$

The amount of radiative flux going from the outer circle to the inner circle is the opposite of this value. Dividing by the respective surface area of each circle gives

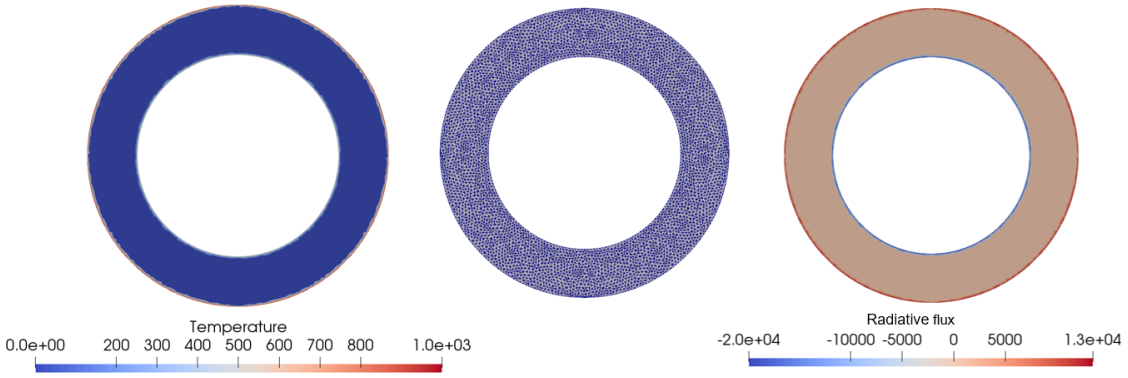


Figure 2.10: Pure radiation test case in concentric circles. The leftmost figure displays the enforced temperatures on the boundaries. The central figure displays the mesh employed. The rightmost figure displays the computed radiative fluxes at the radiating surfaces. Temperature is in K.

the analytical value of the radiative flux density to be expected on each point of the surface of each respective circle, $Q_{s,i\to o}$ and $Q_{s,o\to i}$:

$$\begin{cases} Q_{s,i\to o} = \dfrac{W_{tot,i\to o}}{2\pi r_{in}} = \dfrac{r_{out}\sigma(T_{in}^4 - T_{out}^4)}{r_{in}(1/\epsilon_{in}\cdot r_{out}/r_{in} + 1/r_{out} - 1)} \\ \\ Q_{s,o\to i} = \dfrac{W_{tot,o\to i}}{2\pi r_{out}} = \dfrac{r_{out}\sigma(T_{in}^4 - T_{out}^4)}{r_{out}(1/\epsilon_{in}\cdot r_{out}/r_{in} + 1/r_{out} - 1)} \end{cases} \tag{2.28}$$

We then compare the range of radiant exitance obtained over the inner and outer circles, and compare with the analytical values we deduce from Equation 2.28. First, we compute the average flux density, obtained by integrating the computed local flux density values over the circumference of each circle. These average values should match the analytical flux density value. Then, in order to determine dispersion, for each circle we look up the local flux density value of the mesh point where this value is farthest away from the expected value. We then compute the error between that value and the expected surface value. This gives us the maximum error commited by the solver at any given point on each circle. Table 2.1 sums up the results. We can see that the obtained results are within 1.2% of the theoretical values.

2.4.1.2 2D test cases: square within a square

We compute pure radiation between two homothetic squares separated by a nonparticipating medium. The outer square has an edge size of 1 and is set to $T_{out} = 1000\,\mathrm{K}$.

Source ($\mathrm{W\,m^{-1}}$)	Inner circle surface flux	Outer circle surface flux
Analytical result	-19935	13290
Present work	-19877 (-0.3%)	13401 (+0.8%)
Maximum error value	-19876 (-0.3%)	13445 (+1.2%)

Table 2.1: Comparison of net output radiative flux with analytical solution for concentric circles pure radiation case.

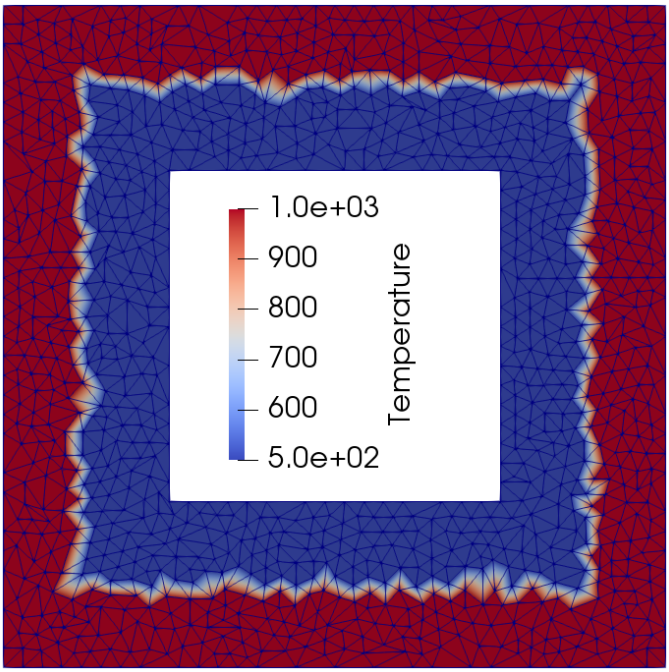


Figure 2.11: Pure radiation test case in homothetic squares.

The inner square has an edge size of 0.5 and is set to $T_{in} = 500\,\mathrm{K}$. Emissivity is set at $\epsilon = 0.5$ for both squares. The meshing is homogeneous with a mesh size of 0.04. The number of elements is 2314, with 161 facets participating in radiation. The setup can be seen in Figure 2.11.

In this simple configuration, we can not find an analytical solution describing the radiative flux density $Q_{radiation}$ exchanged at every point of each of the two squares. However, we can find a solution to the total radiative flux $W_{radiation}$ exchanged by each square towards the other. For this, we will consider each side of the squares to be a single emitting facet, for a total of 8 facets in the system. The outer square sides are numbered from 1 to 4 and the inner square sides from 5 to 8. Analytical formulations of the view factors in such a case can be processed by hand with the use of the crossed strings methods as per Equation 2.16 and displayed in Figure 2.6.

The view factor matrix (F_{ij}) is displayed in Equation 2.29. The square of 0s in the lower right corner indicates how facets of the inner square, from indices 5 to 8, do not see one another at all. All view factors between facets of line and column 5 to 8 are thus equal to 0. From here on, we then invert by hand the 8x8 radiosity matrix $\mathbf{M}$ as seen in Equation 2.21, and from this we deduce the radiosity vector $\mathbf{J}$ and finally the flux ϕ of all radiating facets as per Equation 2.23.

$$(F_{ij}) = \begin{pmatrix} 0 & 1-\frac{\sqrt{10}}{4} & 1-\frac{\sqrt{10}}{4} & \frac{\sqrt{10}-3}{2} & \frac{1-\frac{\sqrt{10}-\sqrt{2}}{2}}{4} & \frac{\sqrt{10}-\sqrt{2}}{4} & \frac{1-\frac{\sqrt{10}-\sqrt{2}}{2}}{4} & 0 \\ 1-\frac{\sqrt{10}}{4} & 0 & \frac{\sqrt{10}-3}{2} & 1-\frac{\sqrt{10}}{4} & \frac{\sqrt{10}-\sqrt{2}}{4} & \frac{1-\frac{\sqrt{10}-\sqrt{2}}{2}}{4} & 0 & \frac{1-\frac{\sqrt{10}-\sqrt{2}}{2}}{4} \\ 1-\frac{\sqrt{10}}{4} & \frac{\sqrt{10}-3}{2} & 0 & 1-\frac{\sqrt{10}}{4} & 0 & \frac{1-\frac{\sqrt{10}-\sqrt{2}}{2}}{4} & \frac{\sqrt{10}-\sqrt{2}}{4} & \frac{1-\frac{\sqrt{10}-\sqrt{2}}{2}}{4} \\ \frac{\sqrt{10}-3}{2} & 1-\frac{\sqrt{10}}{4} & 1-\frac{\sqrt{10}}{4} & 0 & \frac{1-\frac{\sqrt{10}-\sqrt{2}}{2}}{4} & 0 & \frac{1-\frac{\sqrt{10}-\sqrt{2}}{2}}{4} & \frac{\sqrt{10}-\sqrt{2}}{4} \\ \frac{1-\frac{\sqrt{10}-\sqrt{2}}{2}}{2} & \frac{\sqrt{10}-\sqrt{2}}{2} & 0 & \frac{1-\frac{\sqrt{10}-\sqrt{2}}{2}}{2} & 0 & 0 & 0 & 0 \\ \frac{\sqrt{10}-\sqrt{2}}{2} & \frac{1-\frac{\sqrt{10}-\sqrt{2}}{2}}{2} & \frac{1-\frac{\sqrt{10}-\sqrt{2}}{2}}{2} & 0 & 0 & 0 & 0 & 0 \\ \frac{1-\frac{\sqrt{10}-\sqrt{2}}{2}}{2} & 0 & \frac{\sqrt{10}-\sqrt{2}}{2} & \frac{1-\frac{\sqrt{10}-\sqrt{2}}{2}}{2} & 0 & 0 & 0 & 0 \\ 0 & \frac{1-\frac{\sqrt{10}-\sqrt{2}}{2}}{2} & \frac{1-\frac{\sqrt{10}-\sqrt{2}}{2}}{2} & \frac{\sqrt{10}-\sqrt{2}}{2} & 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.29}$$

The analytical calculation gives us the expected total radiative flux received each of the four facets (that is, each side) of each square. To that, we compare the flux density of our computed solution integrated over the four facets of each square. We thus only compare flux to flux, and not flux density to flux density, whereas that was the case for the concentric circles. The results are indicated in Table 2.2. Once again the computed solution stands very close to the analytical solution.

Source (W)	Inner square integrated flux	Outer square integrated flux
Analytical result	-42528	42528
Present work	-42139 (-0.9%)	42140 (-0.9%)

Table 2.2: Comparison of net output radiative flux with analytical solution for square within square pure radiation case.

2.4.1.3 3D test cases: cube within a cube

We now present a test case in pure radiation for the following 3D geometry: a cube of edge size 0.3 within a cube (which is a homothetic transformation of the first one) of edge size 0.6. The inner cube is set at temperature $T_{in} = 1000\,\mathrm{K}$ and the outer cube at $T_{out} = 500\,\mathrm{K}$. Emissiviy is set at $\epsilon = 0.5$ for both cubes. The solver computes a total of 164 participating facets. The geometry is shown Figure 2.12.

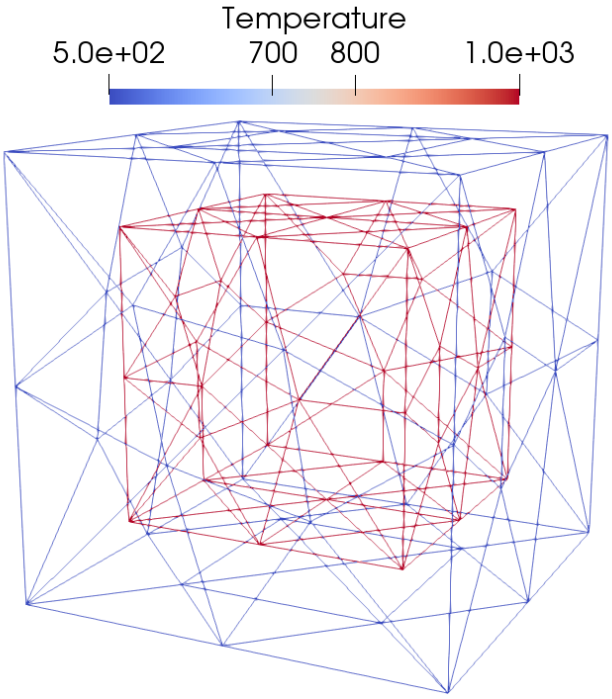


Figure 2.12: Pure radiation test case in cube within cube.

Similarly to the squares, we measure the total net radiative flux density integrated on respectively the inner cube and the outer cube. For this simple case, in order to compute the analytical solution of the problem, we need to compute 3D view factors between the facets of the two cubes. For this we will consider each cube to be composed of only 6 radiating facets: this means we will consider each side of each cube to be a single unique radiating facet. This way, our system is composed of 12 radiating facets, and thus we only need to set up a view factor matrix of size 12. In 3D, we can not use the crossed strings method to compute the view factor. However the "cube within cube" view factor 3D integrals of Equation 2.9 have been analytically solved in the litterature: for example in Howell [41]. From these values we can once again manually set up and invert the radiosity matrix, and thus obtain the analytical radiative flux emitted by each side of each cube.

Our simulation gives us the radiative flux density emitted by each of the 164 facets in the mesh, and we then integrate the value from all the facets composing each individual cube side, so that we obtain the total computed flux received by each side of each cube. We then compare these values to those devised analytically.

Source (W)	Inner cube flux	Outer cube flux
Analytical result	21747	-9665
Present work	21755	-9669
Relative difference (%)	0.04	0.04

Table 2.3: Comparison of net output radiative flux with analytical solution for cube within cube pure radiation case.

Table 2.3 shows the obtained results. The simulated fluxes are very close from the analytical fluxes.

2.4.2 Conduction–radiation

Now that we are confident with our radiation results for the considered geometries, we can simulate a multiphysics heat transfer problem. Once again we will first provide 2D examples before moving on to 3D test cases.

2.4.2.1 2D test cases: very long rods

We will resolve thermal conduction, coupled with radiation as a source term, on the following geometry: two parallel, very long solid rods of thickness $e_{rod} = 1\,\mathrm{m}$ separated by a gas channel of thickness $e_{gas} = 3\,\mathrm{m}$. The total thickness of the system is $= 5\,\mathrm{m}$, while the rods have a lentgh of $= 40\,\mathrm{m}$. There is no fluid circulation, meaning the convection term of Equation 2.26 is zero. The bottom side of the bottom rod sees a boundary condition $T_1 = 1000\,\mathrm{K}$ while the top side of the top rod sees $T_4 = 0\,\mathrm{K}$. Conduction operates through the width of the bottom rod, then through the air, and finally through the width of the top rod. The gas itself is considered transparent to radiation. Meanwhile, radiation also occurs between the top side of the bottom rod and the bottom side of the top rod. The chosen values for emissivity and conductivity are arbitrary: we take $\epsilon = 0.5$ for all radiating surfaces, $\lambda_{rod} = 25\,\mathrm{W\,K^{-1}\,m^{-1}}$ for the conductivity in the rods and $\lambda_{gas} = 10\,\mathrm{W\,K^{-1}\,m^{-1}}$ for the conductivity in the gas. While normal air has a conductivity of the order of $10^{-2}\,\mathrm{W\,K^{-1}\,m^{-1}}$, we purposefully inflate that value so that radiation does not completely dwarf conduction in that simulation and we have to make a more general analytical forumulation where conductive flux in gas is not simply neglected. Volumic masses are $\rho_{rod} = 7700\,\mathrm{kg\,m^{-3}}$ and $\rho_{gas} = 1\,\mathrm{kg\,m^{-3}}$ respectively, and specific heat capacity are $c_{p,rod} = 460\,\mathrm{J\,kg^{-1}\,K^{-1}}$ and $c_{p,gas} = 1000\,\mathrm{J\,kg^{-1}\,K^{-1}}$ respectively. The setup is displayed in Figure 2.13.

Remeshing takes place around the radiating facets, as explained in Part 1. We can afford a coarser mesh in the bulk of each domain, where only conduction takes place. This time we make use of the immersed mesh method. We start from a

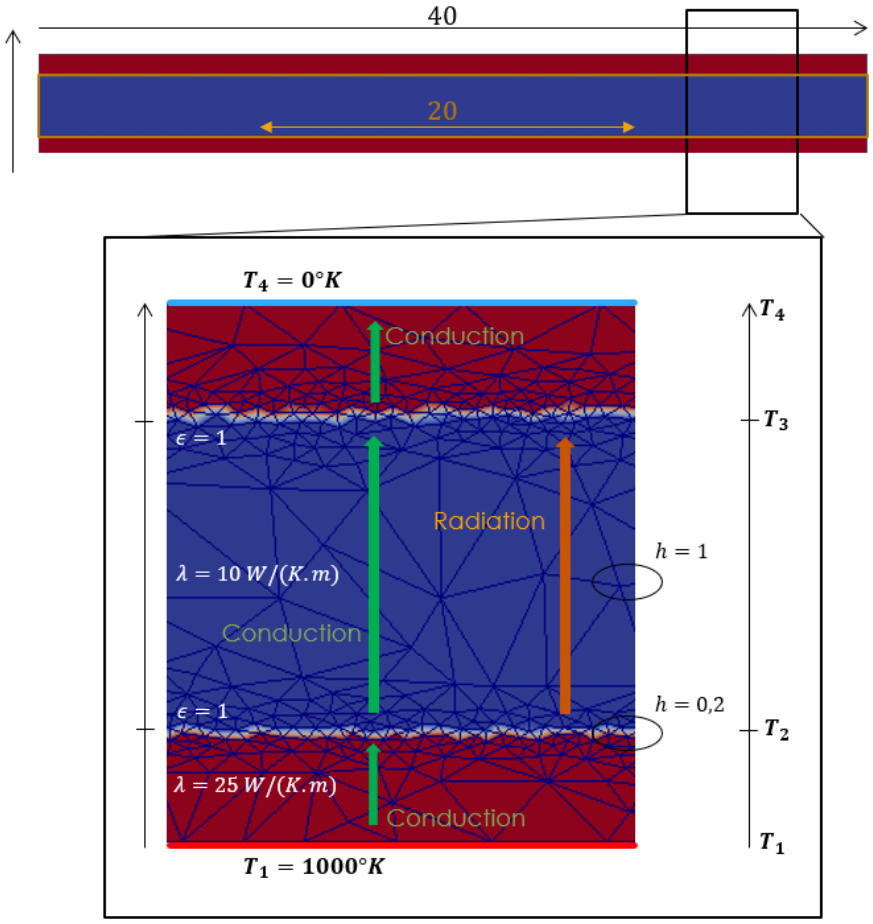


Figure 2.13: Conduction–radiation test case.

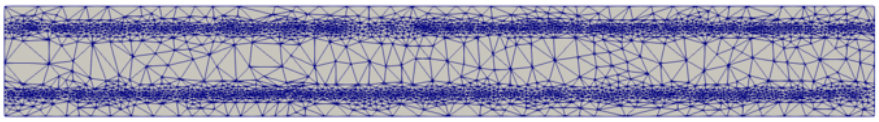


Figure 2.14: Mesh used for the infinitely long rods. It has been refined around the radiating surfaces.

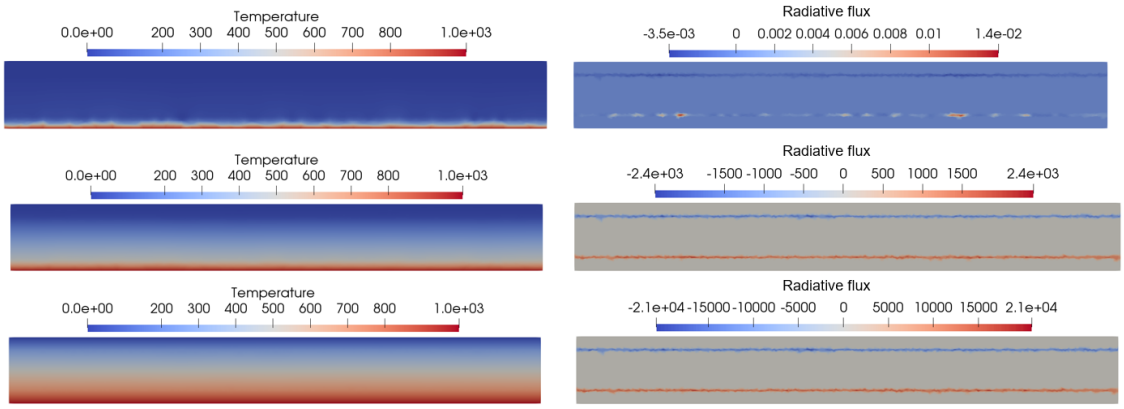


Figure 2.15: Thermal propagation and radiative flux in the rods. The first line shows the initial setup. The second line shows the state after 10 time stepss. The third line shows the state after 50 time stepss, when steady state is reached.

uniform mesh of 37816 elements, in which we immerse the two radiating planes. The mesh is reduced to 5642 elements after remeshing, with 1004 radiating facets. The mesh can be seen in Figure 2.14.

Since the rods'length is high compared to their width, when looking at what happens far from the edges, physics is essentially 1D in the direction of the width or the rods. It amounts to a 1D-thermal transfer problem for which an analytical solution can be devised in steady state. In particular, the temperatures T_2 and T_3 as seen in Figure 2.13 can be measured on the simulation after convergence and compared to the analytical solution. Figure 2.15 shows thermal propagation and radiative flux in the rods at various time steps. When the computation begins, only the bottom surface is at a temperature other than 0 K. In theory this means the radiating surfaces should output an absolute radiative flux of $0\,\mathrm{W\,m^{-1}}$. Due to numerical inaccuracies, very small non-zero fluxes are computed instead, though they do not adversely impact thermal propagation. After 10 time steps we can see the radiating surfaces have reached a non-zero temperature and start radiating flux towards the other. After 50 time steps, steady state is reached, and we can use the temperatures obtained at that stage for analytical comparison.

In order to devise the analytical solution to the problem, we find it convenient

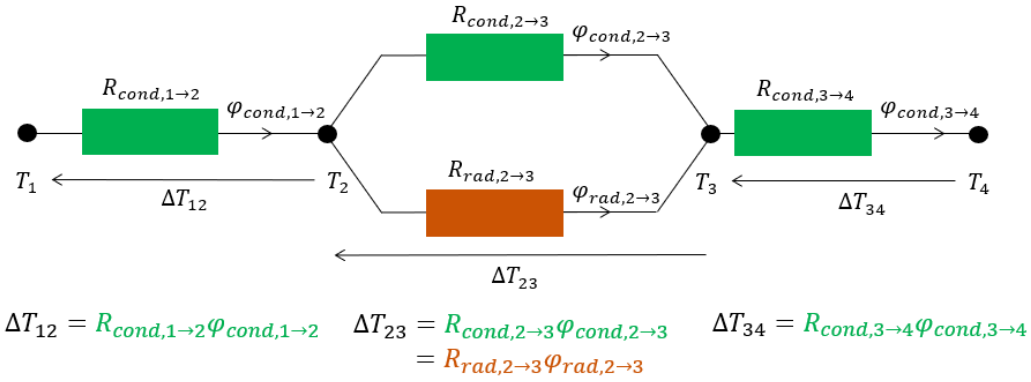


Figure 2.16: Electrical analogy of thermal transfer through the rods.

to use the electricity analogy. Figure 2.16 shows a graphical representation. Our 1D system is composed of four different potential points: T_1, T_2, T_3 and T_4. Out of these, T_1 and T_4 are known. T_1 and T_2, and T_3 and T_4 respectively, are distant from a thickness e_{rod} of conductivity λ_{rod}, so the thermal conduction resistance between these potentials is $R_{cond,1\to2}$ and $R_{cond,3\to4}$ respectively. The walls facing each other are considered of infinite surface in order to ensure the physics occurs in 1D, thus surface values will be considered. All terms of surface normally appearing in the formulae will be replaced by a set surface value of $1\,\mathrm{m}^2$.

The expressions of the resistances are thus defined as:

$$R_{cond,1\to2} = R_{cond,3\to4} = \frac{e_{rod}}{\lambda_{rod}} = 4 \times 10^{-2}\,\mathrm{K\,m^2\,W^{-1}} \tag{2.30}$$

These resistances are expressed in $\mathrm{K\,m^2\,W^{-1}}$, which is the inverse of $\mathrm{W\,K^{-1}\,m^{-2}}$, which is indeed the unit of a thermal conductivity. Then, T_2 and T_4 are connected by two parallel branchs. The branch accounting for the conduction thermal flux holds a thermal conduction resistance $R_{cond,2\to3}$ such as:

$$R_{cond,2\to3} = \frac{e_{gas}}{\lambda_{gas}} = 3 \times 10^{-1}\,\mathrm{K\,m^2\,W^{-1}} \tag{2.31}$$

And the branch accounting for the radiation thermal flux holds a thermal radiation resistance $R_{rad,2\to3}$ the expression of which can be deducted as follows. According to [45], the thermal surface exchanged radiative flux ϕ between two infinite, parallel walls of respective temperature T_2 and T_3 and emissivity ϵ_2 and ϵ_3 is

$$\phi = \frac{\sigma(T_2^4 - T_3^4)}{\frac{1}{\epsilon_2} + \frac{1}{\epsilon_3} - 1} \tag{2.32}$$

which can be rewritten as the expression of a potential difference divided by a thermal resistance as follows:

$$\phi = \frac{(T_2 - T_3)}{\frac{\frac{1}{\epsilon_2} + \frac{1}{\epsilon_3} - 1}{\sigma(T_2 + T_3)(T_2^2 + T_3^2)}} \tag{2.33}$$

We thus obtain a thermal radiation resistance the expression of which depends on both T_2 and T_3:

$$\begin{aligned} R_{rad,2\to3}(T_2, T_3) &= \frac{\frac{1}{\epsilon_2} + \frac{1}{\epsilon_3} - 1}{\sigma(T_2 + T_3)(T_2^2 + T_3^2)} \\ R_{rad,2\to3}(T_2, T_3) &= \frac{5.29 \cdot 10^7}{(T_2 + T_3)(T_2^2 + T_3^2)} \mathrm{K\,m^2\,W^{-1}} \end{aligned} \tag{2.34}$$

We can observe that its value decreases as T_1 and T_2 increase, when the difference between the two is fixed. This makes sense, as the difference between the fourth power of the two increases, making the radiative exchange more important and thus the resistance smaller. In addition, for T_1 and T_2 of an order of magnitude of 500 K, $R_{rad,1\to2}(T_1, T_2)$ would be of the order of $10^{-1}\,\mathrm{K\,m^2\,W^{-1}}$. This is of the same order of magnitude than $R_{cond,2\to3}$, though we have taken an extremely large (for a gaz) conductivity of $\lambda_{gas} = 10\,\mathrm{W\,K^{-1}\,m^{-1}}$. For a more realistic value of air conductivity of about $\lambda_{gas} = 10^{-2}\,\mathrm{W\,K^{-1}\,m^{-1}}$, the conduction resistance would be 100 greater than the radiation resistance and conduction in the gas medium would be negligible before radiation.

Continuing the electricity analogy, we can define an equivalent thermal resistance R_{eq} from T_1 to T_4 that takes into account the four previously defined resistances. Here, we take $\epsilon_1 = \epsilon_2 = \epsilon_{rod}$ (since the gas is transparent to radiation, its emissivity does not intervene).

$$\begin{aligned} R_{eq} &= R_{cond,1\to2} + \frac{1}{\frac{1}{R_{cond,2\to3}} + \frac{1}{R_{rad,2\to3}}} + R_{cond,3\to4} \\ R_{eq} &= 0.34 + \frac{5.29 \cdot 10^7}{(T_2 + T_3)(T_2^2 + T_3^2)} \mathrm{K\,m^2\,W^{-1}} \end{aligned} \tag{2.35}$$

When steady state is reached, since there is no local source or sink of flux, total thermal flux Φ_{tot} is constant throughout the system. We can thus write Ohm's law from one end of the system at $x = 0$ at $T = T_1$ to the other at $x = 5$ and $T = T_4$:

$$\Delta T = T_1 - T_4 = R_{eq}\Phi_{tot} \Leftrightarrow \frac{T_1 - T_4}{R_{eq}} - \Phi_{tot} = 0 \tag{2.36}$$

However, Φ_{tot} is not the only unknown here, since the expression of R_{eq} includes $R_{rad,2\to3}$, which itself depends upon both T_2 and T_3. We will thus express T_2 and T_3 from Φ_{tot} by writing Ohm's law between $x = 0$ at $T = T_1$ and $x = 1$ at $T = T_2$, and $x = 4$ at $T = T_3$ and $x = 5$ at $T = T_4$, respectively. We have seen that the only thermal resistance between T_1 and T_2 is the thermal conduction resistance, and its expression is composed of known constants:

$$\Delta T = T_1 - T_2 = \Phi_{tot} R_{cond,1\to2} = \Phi_{tot} \frac{e_{rod}}{\lambda_{rod}} \tag{2.37}$$

From which we can express T_2 with the only unknown Φ_{tot}:

$$T_2 = T_1 - \Phi_{tot} \frac{e_{rod}}{\lambda_{rod}} \tag{2.38}$$

And similarly, we obtain

$$T_3 = T_4 + \Phi_{tot} \frac{e_{rod}}{\lambda_{rod}} \tag{2.39}$$

This allows us to express $R_{rad,2\to3}$ as a function of Φ_{tot}:

$$R_{rad,2\to3} = \frac{\frac{1}{\epsilon_{rod}} + \frac{1}{\epsilon_{rod}} - 1}{\sigma(T_2 + T_3)(T_2^2 + T_3^2)} \tag{2.40}$$

$$R_{rad,2\to3} = \frac{\frac{1}{\epsilon_{rod}} + \frac{1}{\epsilon_{rod}} - 1}{\sigma((T_1 - \Phi_{tot}\frac{e_{rod}}{\lambda_{rod}}) + (T_4 + \Phi_{tot}\frac{e_{rod}}{\lambda_{rod}}))((T_1 - \Phi_{tot}\frac{e_{rod}}{\lambda_{rod}})^2 + (T_4 + \Phi_{tot}\frac{e_{rod}}{\lambda_{rod}})^2)} \tag{2.41}$$

We can now return to Equation 2.36 to find an equation with only one unknown, being Φ_{tot}:

$$\frac{T_1 - T_4}{R_{cond,1\to2} + \dfrac{1}{\dfrac{1}{R_{cond,2\to3}} + \dfrac{1}{R_{rad,2\to3}}} + R_{cond,3\to4}} - \Phi_{tot} = 0 \tag{2.42}$$

$$\frac{T_1 - T_4}{\dfrac{e_{rod}}{\lambda_{rod}} + \dfrac{1}{\dfrac{1}{\frac{e_{gas}}{\lambda_{gas}}} + \dfrac{1}{R_{rad,2\to3}(\Phi_{tot})}} + \dfrac{e_{rod}}{\lambda_{rod}}} - \Phi_{tot} = 0 \tag{2.43}$$

We will not develop the equation further for readability reasons. However, Equation 2.42 can be numerically solved to find the value of Φ_{tot} achieving the right-hand 0 value. Once Φ_{tot} is determined, we can express the numerical values of T_2 and T_3. From that point, the temperature at any point in the system can be written as

a piecewise linear function, which is a simple linear interpolation between ($x_1 = 0$, T_1) and ($x_2 = 1$, T_2), ($x_2 = 1$, T_2) and ($x_3 = 4$, T_3), and ($x_3 = 4$, T_3) and ($x_4 = 5$, T_4) respectively.

$$T(x) = \begin{cases} T_1 + (T_2 - T_1)\dfrac{x - x_1}{x_2 - x_1} & \text{if } x \in [x_1; x_2] \\ T_2 + (T_3 - T_2)\dfrac{x - x_2}{x_3 - x_2} & \text{if } x \in [x_2; x_3] \\ T_3 + (T_4 - T_3)\dfrac{x - x_3}{x_4 - x_3} & \text{if } x \in [x_3; x_4] \end{cases} \tag{2.44}$$

This, at last, can be plotted against the computed value for the temperature taken at a vertical slice right in the middle of Figure 2.18. The results are shown Table 2.4. Since the simulation cannot really depict a 1D test case, the measured values vary inside an interval depending on where on the horizontal axis is the measure done. We take the range of values observed within the interval of width 20, located far from the edges to avoid edge effects as depicted in Figure 2.13, and we indicate it in the table.

Source	$T_2(K)$	$T_3(K)$
Analytical result	729	271
Present work	729-732	267-271
Relative difference (%)	0.4	1.4

Table 2.4: Comparison of temperatures with analytical solution for infinitely long rods conduction–radiation test case.

The results are in very good accordance with the analytical predictions.

We can also plot the temperature variation along the width of the rod, both with analytical values and with our computed solution. This way we can assess the accuracy of the solution relatively to the position inside the rod. Figure 2.17 shows the temperature value inside the rod from bottom to top, depending on the distance x from the bottom only since the problem is 1D. This figure also displays the absolute temperature error between the computed solution and the analytical one. The worst error is of around 6 K at width $x = 3.90$, right before the interface between air and upper rod part. This errors amounts to 2% of the analytical temperature at this point, which remains a fairly good accuracy.

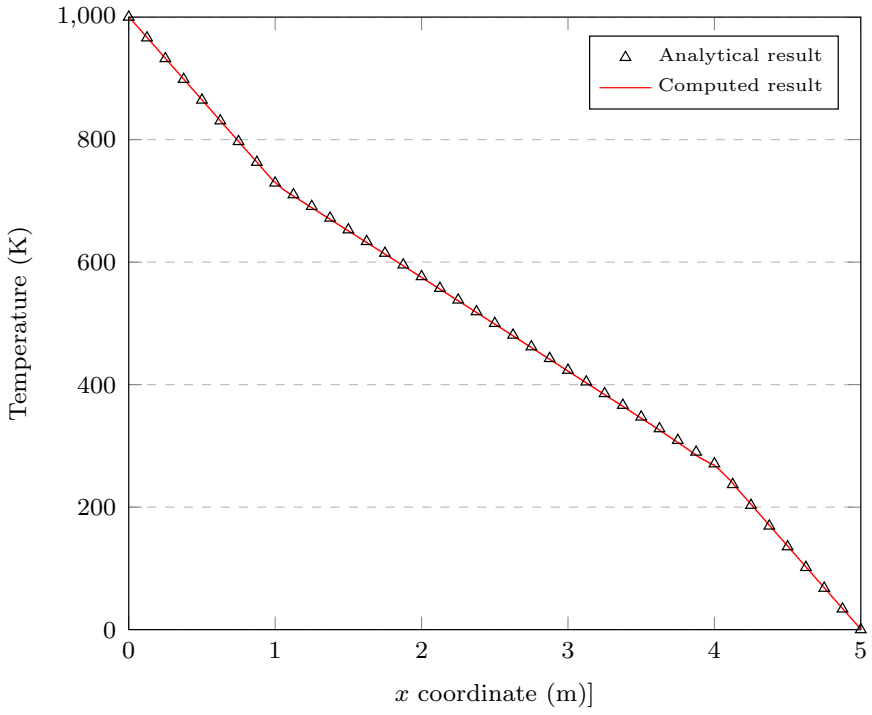


Figure 2.17: Progression of the temperature inside the rod from bottom (x=0) to top (x=5).

2.4.2.2 2D test cases: concentric crowns

The next geometry we assess is that of two concentric crowns separated by a fluid area. No physics is dependant upon the z coordinate, meaning the physics that takes place can be described in 2D. The inner crown is comprised between radii $r_1 = 0.8\,\mathrm{m}$ and $r_2 = 2\,\mathrm{m}$, and the outer crown between radii $r_3 = 4\,\mathrm{m}$ and $r_4 = 5\,\mathrm{m}$. The geometry is constructed as follows. A global domain shaped like a crown comprised between radii r_1 and r_4 is defined and meshed uniformly. Essentially, the domain is comprised between two concentric circles. We then immerse into that domain a mesh of a 2D crown comprised between radii r_2 and r_3. Once again this amounts to defining a domain comprised between two concentric circles. This immersion allows us to define three different spatial domains: between radii r_1 and r_2, between radii r_2 and r_3, and between radii r_3 and r_4. The mesh starts out from a uniform mesh of 8056 elements, and is then refined around the 0-isovalue of the level-set of the immersed crown down to 3966 elements. This amounts to refining the mesh around the r_3 and r_4 circles, since they form the implicit boundaries of the immersed slice. These circles are where radiation will take place. Figure 2.18 displays the setup.

Even if the refinement does not seem very circle-shaped around the interfaces, the process of extracting virtual facets ensures the radiation is computed around a proper set of facets forming the correct circles. The values of emissivity, conductivity, specific heat and volumic mass are the same as in the infinite rods case. The inner side of the inner cylinder standing at r_1 is set to $T_1 = 1000\,\mathrm{K}$ while the outer side of the outer cylinder standing at r_4 is set to $T_4 = 0\,\mathrm{K}$.

We let thermal propagation run until steady state is reached after 50 time steps of $2 \times 10^4\,\mathrm{s}$ each. Figure 2.19 shows the situation both for temperature and for emitted radiative flux density at different points of the simulation. As was the case for the rods, we can see a non-zero, but negligible, radiative flux density at the two radiating circles at the beginning of the simulation.

Once again, conduction occurs through the thickness of the inner crown (standing for an arbitrary metal) between r_1 and r_2. Then, it occurs again between r_2 and r_3 in the thickness between the two crown (standing as an arbitrary radiation-transparent gas), plus radiation occurs between the outer surface of the inner crown at r_2 and the inner surface of the outer crown at r_3. Conduction occurs again in the thickness of the outer crown between r_3 and r_4. The physical parameters are $\epsilon = 0.5$ for all radiating surfaces, $\lambda_{crown} = 25\,\mathrm{W\,K^{-1}\,m^{-1}}$ for the conductivity in the crown and $\lambda_{fluid} = 10\,\mathrm{W\,K^{-1}\,m^{-1}}$ for the conductivity in the medium between the crown. We then let the system stabilize towards a steady state in temperature. We can devise an analytical solution for the thermal situation of the problem, exactly how we did so for the infinitely long rods. The only difference relies in the expression of the thermal resistances. Figure 2.16 still describes the electrical analogy of thermal transfer in this system. We take a set length of 1 m for the crown in the z dimension, in order to

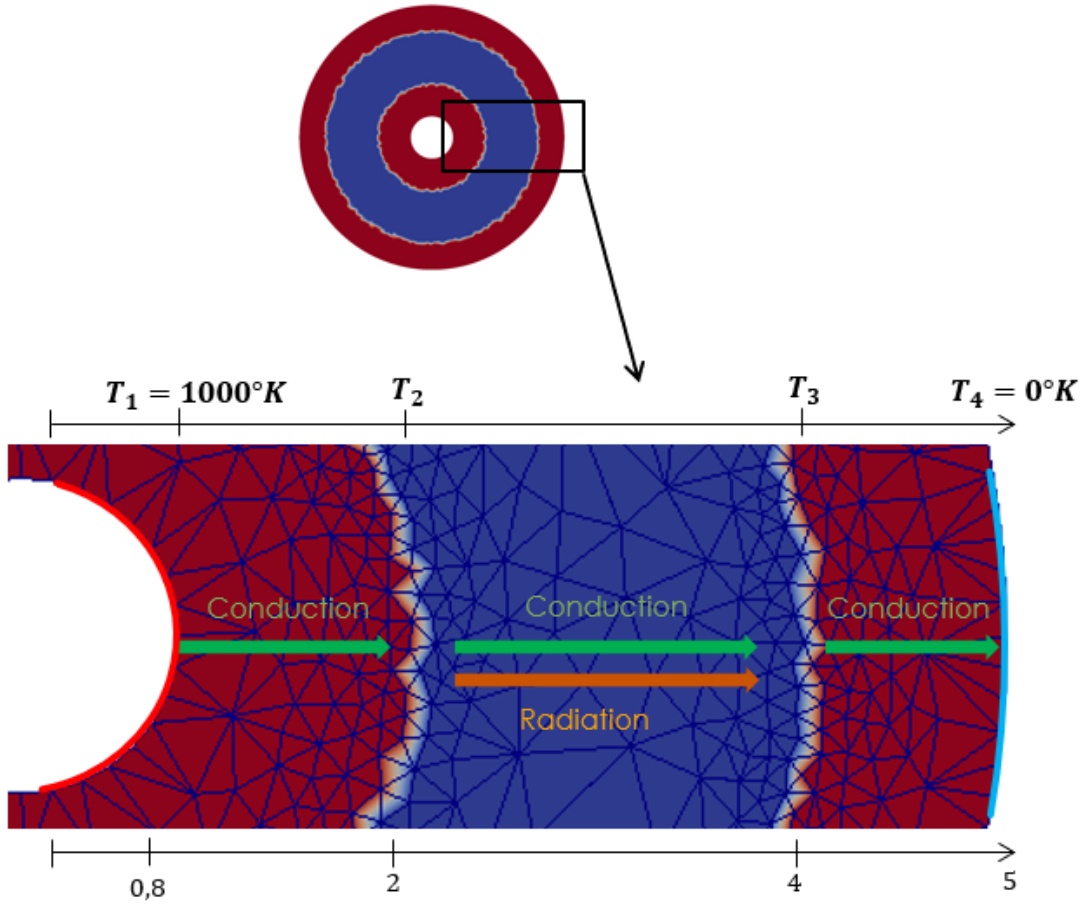


Figure 2.18: Immersion of a 2D crown in the domain. The two white circles stand for the 0-isovalue of the level-set defining the slice. The immersed item covers all the space between them. The mesh has been refined at the interface.

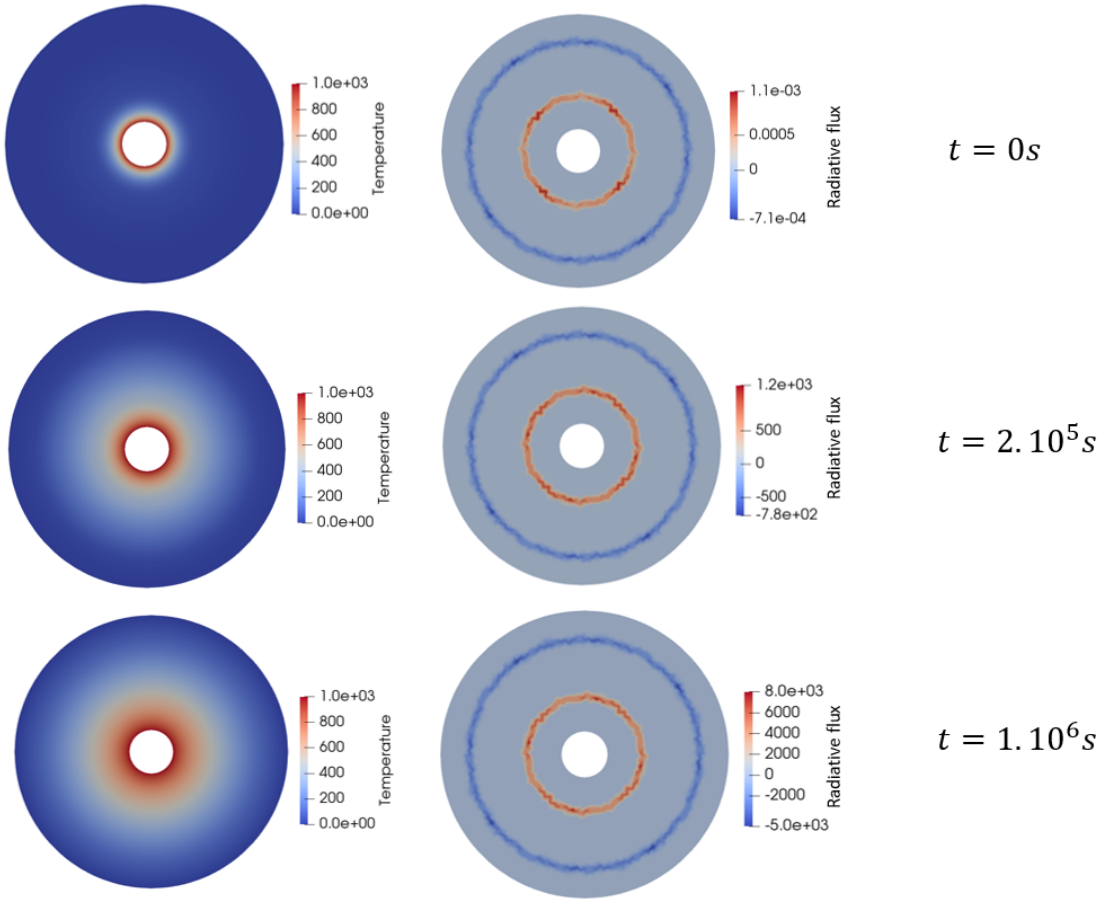


Figure 2.19: Thermal propagation and radiative flux in the rods. The first line shows the initial setup. The second line shows the state after 10 time steps. The third line shows the state after 50 time steps, when steady state is reached.

make the length disappear from the equations. Thus, we have the following values for the various thermal conduction resistances:

$$R_{cond,1\to2} = \frac{\ln(\frac{r_2}{r_1})}{2\pi\lambda_{crown}} = 5.8 \times 10^{-3}\,\mathrm{K\,m^2\,W^1} \tag{2.45}$$

$$R_{cond,2\to3} = \frac{\ln(\frac{r_3}{r_2})}{2\pi\lambda_{fluid}} = 1.1 \times 10^{-2}\,\mathrm{K\,m^2\,W^1} \tag{2.46}$$

$$R_{cond,3\to4} = \frac{\ln(\frac{r_4}{r_3})}{2\pi\lambda_{crown}} = 1.4 \times 10^{-3}\,\mathrm{K\,m^2\,W^1} \tag{2.47}$$

And the value of the thermal radiation resistance is [45]:

$$\begin{aligned} R_{rad,2\to3} &= \frac{\frac{1}{\epsilon_{rod}}\frac{r_3}{r_2} + \frac{1}{\epsilon_{rod}} - 1}{2\pi r_3\sigma(T_2+T_3)(T_2^2+T_3^2)} \\ R_{rad,2\to3} &= \frac{2.8\cdot10^7}{(T_2+T_3)(T_2^2+T_3^2)}\mathrm{K\,m^2\,W^1} \end{aligned} \tag{2.48}$$

These changes done, we can once again compare the analytical solution to the computed solution at the radiating interfaces. The results are indicated in Table 2.5 and show the computed solution matches very well with the analytical solution.

Source (K)	T_2	T_3
Analytical result	547	110
Present work	544-547	110-112
Relative difference (%)	0.7	1.6

Table 2.5: Comparison of temperature for concentric crown conduction–radiation test case.

We can also plot the temperature alongwith a radial progression through the crown, both for the analytical and computed solutions. This can be seen in Figure 2.20. Once again, we can see that our computed solution matches very well with the analytical solution.

2.4.2.3 3D test cases: concentric spheres

Moving on to 3D geometries, we model conduction and radiation between four concentric spheres. The whole 3D domain is comprised between two concentric spheres

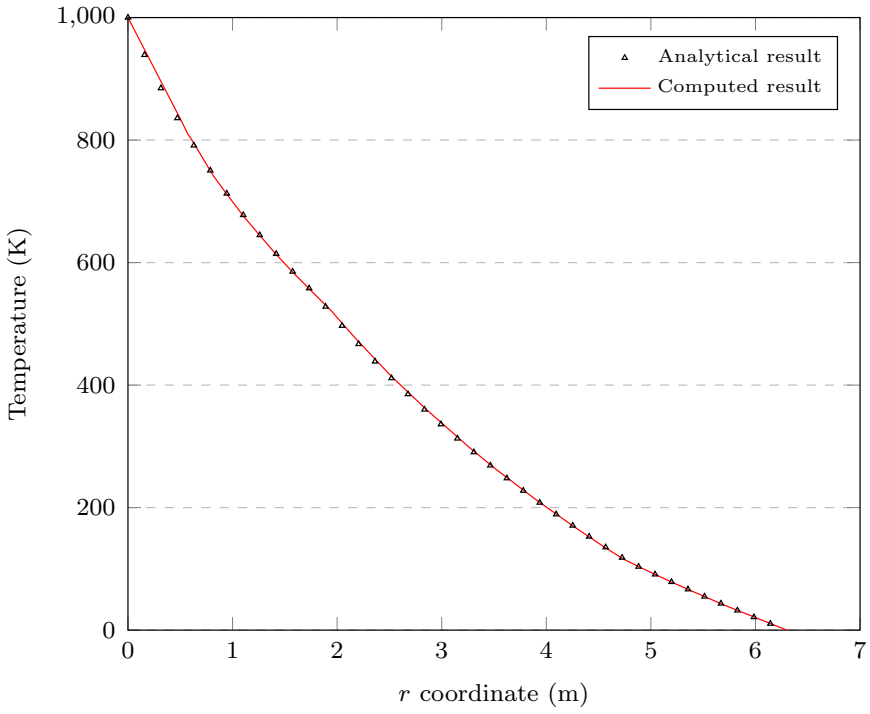


Figure 2.20: Progression of the temperature inside the rod from bottom (x=0) to top (x=5).

S_1 and S_4 of respective radii $r_1 = 4\,\text{m}$ and $r_4 = 10\,\text{m}$. Two surface meshes, one of a sphere S_2 of radius $r_2 = 6\,\text{m}$ and one of a sphere S_3 of radius $r_3 = 8\,\text{m}$, are immersed in the domain. The space between S_1 and S_2, and S_3 and S_4 respectively, is filled with an arbitrary solid material, through which only conduction occurs. The space between S_2 and S_3 is filled with an arbitrary, immobile gas, transparent to radiation. This means both conduction through that gas and radiation between the two spheres occur at that stage. The setup is displayed in Figure 2.21.

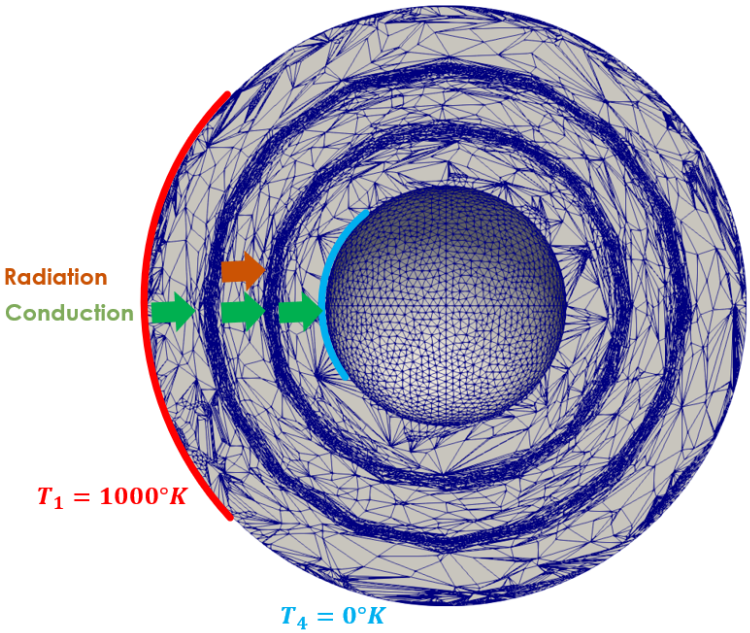


Figure 2.21: Immersion of two spheres in the spherical domain. The mesh has been refined around these two inner spheres. Conduction occurs in the bulk of the system, and radiation occurs between the two immersed spheres.

Starting from a uniform mesh with a mesh size of $h_{init} = 0.3$ and 635377 elements, we refine around the immersed inner spheres so as to achieve a mesh size around the interfaces $h_{final} = 0.03$ and 286434 elements. The outer sphere surface is set to $T_1 = 600\,\text{K}$ while the inner sphere surface is set to $T_4 = 0\,\text{K}$. At the beginning of the simulation, temperature is equal to 0 K except for the outer sphere. We let thermal transfer heat up the volume starting from the outer sphere towards the inner sphere and wait for thermal equilibrium to be reached. The equations are fairly similar to those of the 2D concentric cylinders test case. The expression of thermal conduction resistance $R_{cond,1\to2}$ between two concentric spheres of respective radius r_1 and r_2 (with $r_1 < r_2$) separated by a solid medium of constant and homogeneous thermal conductivity λ amounts to:

$$R_{cond,1\to2} = \frac{1}{4\pi\lambda}(\frac{1}{r_1} - \frac{1}{r_2}) \tag{2.49}$$

And as a result we have the following resistances between in our sphere:

$$\begin{aligned} R_{cond,1\to2} &= 2.4 \times 10^{-3}\,\mathrm{K\,m^2\,W^{-1}} \\ R_{cond,2\to3} &= 2 \times 10^{-3}\,\mathrm{K\,m^2\,W^{-1}} \\ R_{cond,2\to3} &= 1.6 \times 10^{-4}\,\mathrm{K\,m^2\,W^{-1}} \end{aligned} \tag{2.50}$$

The thermal radiation resistance between these same spheres, of respective surface emissivity ϵ_2 and ϵ_3, is close to the expression for concentric circles, Equation 2.48:

$$\begin{aligned} R_{rad,2\to3} &= \frac{\frac{1}{\epsilon_2}\frac{r_3}{r_2} + \frac{1}{\epsilon_3} - 1}{4\pi r_3^2\sigma(T_2+T_3)(T_2^2+T_3^2)} \\ R_{rad,2\to3} &= \frac{4.4\cdot10^6}{(T_2+T_3)(T_2^2+T_3^2)}\mathrm{K\,m^2\,W^{-1}} \end{aligned} \tag{2.51}$$

These changes aside, the same computations can be made than for the 2D test cases, and similarly, we can compare the achieved temperature at certain specific points between the analytical solution and the computed solution. We look at the temperature at the surfaces of spheres 2 and 3. Table 2.6 shows the gathered data.

Source (K)	T_2	T_3
Analytical result	355	491
Present work	338-344	490-493
Maximum relative difference (%)	-4.8	0.5

Table 2.6: Comparison of temperatures with analytical solution for concentric spheres conduction–radiation test case.

We can see that the results are in good agreement with the analytical values, though slightly less than for the 2D cases. And once again we can plot the temperature at a fixed radius and compare it to the analytical temperature. In theory, temperature should depend on radius only. The simulation data we use for comparison must not depend too much on local differences and mesh imperfections. This means we can not just draw an arbitrary radius segment through the sphere, gather the temperature along that segment, and plot it against the analytical value. Instead, we gather the temperature alongwith six different radius segments, and we

then take the mean of all these to obtain a representative result from our computed data. For two of these six segments, we choose the following segments that run along the x axis: from (r_1,0,0) to (r_4,0,0), and from (-r_4,0,0) to (-r_1,0,0). We proceed similarly to obtain four other segments from the y and z axes. At last, the data is plotted against analytical value and shown Figure 2.22.

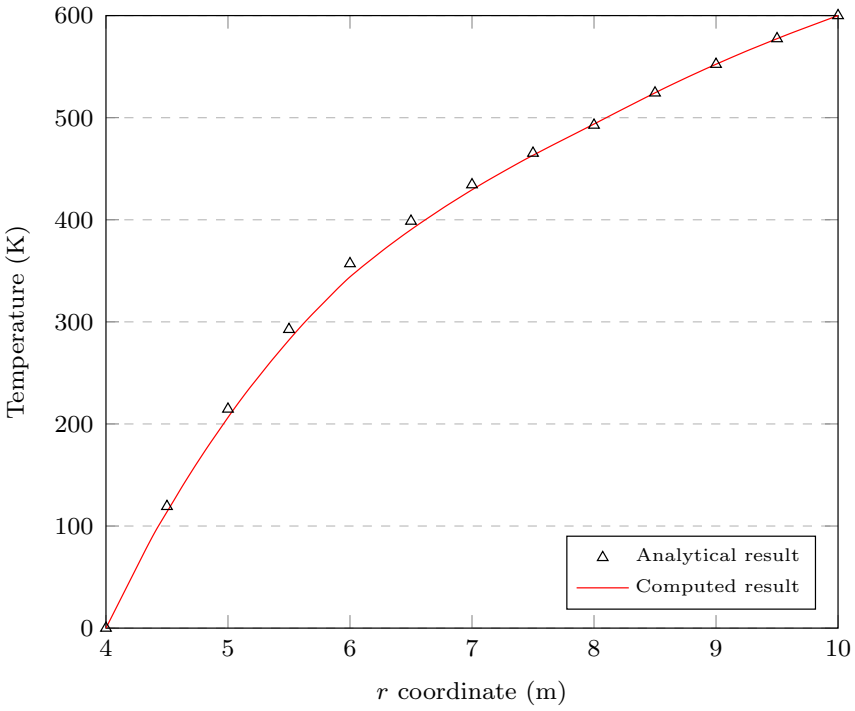


Figure 2.22: Progression of the temperature according to radius starting from the inner sphere surface at $r = 4$ to the outer sphere surface at $r = 10$.

Once again we can see the computed temperature matches very closely with the analytical temperature. This test case gives us confidence in our ability to model 3D geometries with reasonable accuracy.

2.5 Summary

- In this chapter, an immersed volume approach has been applied to conduction–radiation simulation cases. Radiating facets have been precisely extracted from the implicit nature of the boundaries between the different elements in the mesh.

- The S2S model validity in the case of silicon carbide ceramics has been carefully

studied and we have established that indeed, such a model can be reasonably applied for such a material.

- Some methods for view factor computation, including obstruction handling, have been exposed and we have developed the chosen adaptive integration method. The complete process of radiative flux computation through the use of radiosity has been detailed, as well as the coupling with thermal conduction in a complete thermal transfer model.
- 2D and 3D steady-state pure radiation and conduction—radiation simple test cases have been computed and the results have been compared to analytical solutions. The proposed solver achieves good accuracy with regard to the expected solutions.

The question of computation time, which has not been touched on so far, will be the focus of the following chapter.

Bibliography

[1] Q. Schmid, C. Mensah, E. Hachem, Y. Mesri, A versatile immersed surface-to-surface method for 2d radiative exchange: implementation and validation, International Journal of Heat and Mass Transfer (2018). 35

[2] E. Hachem, S. Feghali, R. Codina, T. Coupez, Immersed stress method for fluid–structure interaction using anisotropic mesh adaptation, International Journal for Numerical Methods in Engineering 94 (9) (2013) 805–825. 36

[3] C. Bahbah, M. Khalloufi, A. Larcher, Y. Mesri, T. Coupez, R. Valette, E. Hachem, Conservative and adaptive level-set method for the simulation of two-fluid flows, Computers and Fluids (2019). 36

[4] H.-K. Zhao, T. Chan, B. Merriman, S. Osher, A variational level set approach to multiphase motion, Journal of Computational Physics 127 (1) (1996) 179–195. 36

[5] E. M. Sparrow, Advances in heat transfer, Vol. 2, Elsevier, 1965. 37

[6] P. Hanrahan, D. Salzman, L. Aupperle, A rapid hierarchical radiosity algorithm, Computer Graphics 25 (1991). 37, 38

[7] C. M. Goral, K. E. Torrance, D. P. Greenberg, B. Battaile, Modeling the interaction of light between diffuse surfaces, in: SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques, 1984, pp. 213–222.

[8] D. R. Baum, H. E. Rushmeire, J.Winget, Improving radiosity solutions through the use of analytically determined form-factors, in: Proceedings of the 16st Annual Conference on Computer Graphics and Interactive Techniques, 1989, pp. 325–334. 37

[9] H. Hottel, A. Sarofim, Radiative transfer, AIChE Journal 15 (5) (1969) 794–796. 38, 49

[10] G. Walton, Calculation of obstructed view factors by adaptive integration, National Institute of Standard and Technology Internal Report (nov. 2002). 38

[11] M. Cohen, M. F. Greenberg, The hemi-cube: a radiosity solution for complex environments, in: SIGGRAPH'85 conference proceedings, Vol. 19, july 1985, pp. 31–39. 38

[12] A. Franck, W. Heidemann, K. Spindler, Modeling of the surface-to-surface radiation exchange using a Monte Carlo method, Journal of Physics: Conference Series 745 (2016). 38

[13] B. Py, Rayonnement thermique des métaux microrugueux ou dispersés, International Journal of Heat and Mass Transfer 10 (12) (1967) 1735–1736. 39

[14] M. A. Ordal, L. L. Long, R. J. Bell, S. E. Bell, R. R. Bell, R. W. A. Jr, C. A. Ward, Optical properties of the metals Al, Co, Cu, Au, Fe, Pb, Ni, Pd, Pt, Ag, Ti, and W in the infrared and far infrared, APPLIED OPTICS 22 (1983) 1099–1120. 39

[15] V. Prokopets, I. Shaikevitch, L. Robur, Optical properties of silicon-nitride-based ceramics with molybdenum and aluminium admixtures, Ukrainian Journal of Physics 50 (2005). 39

[16] C. Liebert, Emittance and absorptance of NASA ceramic thermal barrier coating system, NASA technical paper 1190 (june 1978). 39

[17] D. Law, D. Zhou, Developing Solid Oral Dosage Forms (Second Edition) - Chapter 3 - Solid-State Characterization and Techniques, Pharmaceutical Theory and Practice, 2017. 39

[18] H. Ebrahimi, A. Zamaniyan, J. S. Mohammadzadeh, A. A. Khalili, Zonal modeling of radiative heat transfer in industrial furnaces using simplified model for exchange area calculation, Applied Mathematical Modelling 37 (2013) 8004–8015. 39

[19] Y. Guo, S. Pang, Z. Luo, Y. Shuai, H. Tan, H. Qi, Measurement of directional spectral emissivity at high temperatures, International Journal of Thermophysics 40 (11 2018). 40, 43

[20] J. M. Jones, P. E. Mason, A. Williams, A compilation of data on the radiant emissivity of some materials at high temperatures, Journal of the Energy Institute (2019). 41, 43

[21] H. Watanabe, J. Ishii, H. Wakabayashi, T. Kumano, L. Hanssen, Chapter 9 - spectral emissivity measurements, in: T. A. Germer, J. C. Zwinkels, B. K. Tsai (Eds.), Spectrophotometry, Vol. 46 of Experimental Methods in the Physical Sciences, Academic Press, 2014, pp. 333–366. 41

[22] K. Berger, I. Reshetouski, M. Magnor, I. Ihrke, Refractive index dependent bidirectional scattering distribution functions, in: ACM SIGGRAPH 2011 Posters, SIGGRAPH '11, Association for Computing Machinery, New York, NY, USA, 2011. 41

[23] K. K. Sikka, M. A. Postlethwait, M. F. Modest, J. R. Hellmann, High temperature emittance of silicon carbide based materials. volume 1. high temperature normal spectral emittance of silicon carbide based materials. volume 2. effects of heat treatments on the emittance of silicon carbide based materials. topical report, june 1, 1989-december 31, 1993 (12 1993). 41, 42, 43

[24] M. Balat-Pichelin, A. Bousquet, Total hemispherical emissivity of sintered SiC up to 1850 K in high vacuum and in air at different pressures, Journal of the European Ceramic Society 38 (10) (2018) 3447–3456. 43

[25] T. E. Ariceta, Infrared spectral emissivity studies on metals and materials for solar thermal applications, Ph.D. thesis, Euskal Herriko Uniberstitatea (march 2016). 43

[26] M. Sun, I. K. Lloyd, Emissivity of silicon carbide composites as a function of temperature and microstructure, MRS Proceedings 410 (1995) 411. 43

[27] P. Schroder, P. Hanrahan, On the form factor between two polygons, ACM SIGGRAPH Computer Graphics (sept. 1993) 163–164. 48

[28] M. Coehn, D. Greenberg, The hemi-cube, a radiosity solution for complex environments, ACM SIGGRAPH Computer Graphics (jul. 1985).

[29] C. B. S. Woop, A.T. Afra, STBVH : A spatial-temporal BVH for efficient multi-segment motion blur, Proceedings of High Performance Graphics (jun. 2017).

[30] S. Mazumder, Methods to accelerate ray tracing in the Monte-Carlo method for surface-to-surface radiation transport, Journal of Heat Transfer (feb. 2006) 945–952. 48, 54

[31] M. Coehn, D. P. Greenberg, The hemi-cube, a radiosity solution for complex environments, ACM SIGGRAPH Computer Graphics (july 1985). 50

[32] B. Gaume, Réduction d'un problème d'auto-rayonnement par modes de branche : Application aux échanges thermiques dans un domaine multi-enceintes, Ph.D. thesis, Université Evry-Val d'Essonne (2016). 50, 51

[33] S. Chandran, G. N. B. Kiran, M. Phani, Anti-aliased hemicubes for radiosity, 2001. 51

[34] A. Frank, et al., Modeling of the surface-to-surface radiation exchange using a Monte Carlo method, Journal of Physics: Conference Series (2016). 51

[35] K. J. Daun, D. P. Morton, J. R. Howell, Smoothing Monte Carlo Exchange Factors through Constrained Maximum Likelihood Estimation, Journal of Heat Transfer 127 (10) (2005) 1124–1128. 51

[36] R. P. Taylor, R. Luck, B. K. Hodge, W. G. Steele, Uncertainty analysis of diffuse-gray radiation enclosure problems, Journal of Thermophysics and Heat Transfer 9 (1) (1995) 63–69. 52

[37] J. van Leersum, A method for determining a consistent set of radiation view factors from a set generated by a nonexact method, International Journal of Heat and Fluid Flow 10 (1) (1989) 83–85. 52

[38] G. Walton, Calculation of obstructed view factors by adaptive integration, National Institute of Standard and Technology Internal Report (november 2002). 52

[39] P. Schröder, P. Hanrahan, On the form factor between two polygons, ACM SIGGRAPH Computer Graphics (1993) 163–164. 53

[40] A. Ambirajan, S. Venkateshan, Accurate determination of diffuse view-factors between planar surfaces, International Journal of Heat and Mass Transfer 36 (8) (1993) 2203–2208. 53

[41] J. Howell, A catalog of radiation heat transfer configuration factors, http://www.thermalradiation.net/tablecon.html (1982). 53, 63

[42] M. Leblond, F. Rousselle, C. Renaud, Hybridization techniques for fast radiosity solvers, in: Proceedings Computer Graphics International 2000, 2000, pp. 269–278. 56

[43] R. Bramley, P. Shirley, Iterative methods for fast radiosity solutions, 1995. 56

[44] F. Favre, G. Colomer, O. Lehmkuhl, A. Oliva, Numerical simulations of conjugate convection combined with surface thermal radiation using an immersed-boundary method, Journal of Physics: Conference Series 745 (2016) 032017. 56

[45] D. Marchio, P. Reboux, Introduction aux transferts thermiques, Presses des Mines, 2008. 59, 67, 75

Chapter 3

Kd-tree acceleration structure

Contents

3.1 Résumé 86
3.2 Naive obstruction computation 87
3.2.1 Local half-space culling 88
3.3 Binary tree structures 88
3.3.1 Metanode kd-trees 89
3.3.2 Bounding volume kd-trees 126
3.4 Parallel computation 141
3.4.1 Radiosity preprocessing 141
3.4.2 Thermal transfer 147
3.5 Summary 150
Bibliography 150

3.1 Résumé

Cette section porte sur la problématique du temps de calcul des facteurs de vue. Les derniers calculs analytiques de la section précédente ont mis en évidence des temps de calcul incompatibles avec un usage industriel de notre modèle. Pour cela, nous développons diverses stratégies d'accélération. Le point bloquant du calcul du rayonnement est celui de l'obstruction des facteurs de vue, qui fait passer l'étape des facteurs de vue de $O(N^2)$ à $O(N^3)$ quand on a N faces rayonnantes.

La méthode naïve a pour principal problème d'évaluer une à une toutes les faces du système de manière indiscriminée quand il faut tester si l'une d'elles obstrue le couple de faces courant. Pour accélérer cela, nous introduisons les arbres kd. Ce sont des arbres binaires qui permettent un découpage spatial des barycentres de chaque face rayonnante. Ensuite, entre deux faces à étudier, nous les localisons dans l'arbre kd qui hiérarchise toutes les faces. Nous lançons un rayon de lumière les rejoignant, et faisons traverser l'arbre à ce rayon. Le parcours du rayon intersecte un certain nombre de sections de l'arbre, et nous évaluons l'obstruction, au fur et à mesure de l'avancée du rayon, uniquement avec les faces dont les barycentres sont contenus dans ces sections. Ceci nous permet de ne tester en moyenne que $O(\log N)$ faces par facteur de vue considéré, faisant ainsi passer la complexité totale de $O(N^3)$ à $O(N^2 \log N)$.

Nous élaborons une brève bibliographie de l'état de l'art sur les arbres kd et le lancer de rayon dans le cadre de la méthode des radiosités. Nous introduisons ensuite les deux variantes d'arbres kd que nous avons dévoppées : les arbres kd à métanoeuds et les arbres kd à volumes englobant. Les algorithmes spécifiques de création, de parcours, de lancer de rayon et de recherche du plus proche voisin sont explicités. Une brève comparaison des performances en temps de construction nous conduit à retenir les arbres kd à volume englobant.

Dans un dernier temps, nous évoquons la mise en calcul parallèle de notre nouvel algorithme de radiosité. Deux cas de rayonnement sont détaillés, effectués sur un nombre de coeurs allant jusqu'à 256. Nous examinons la performance en temps de calcul des différentes étapes du prétraitement de la radiosité à l'incrément zéro de chaque calcul radiatif, dans un premier temps. Puis nous examinons la performance du calcul des radiosités qui prend place à chaque pas de temps. Les résultats montrent que le calcul parallèle est performant jusqu'à 256 coeurs, sans perte de performance visible due aux temps de communication entre coeurs.

The previous chapter has explained how we ascertained the accuracy of the proposed solver, by comparing it to numerous analytical solutions. It also has briefly mentioned that good care must be given to view factor computation, not only for the purpose of accurate results, but also for quick computation. We will now detail the efficiency performance of the proposed solver. This chapter is organized as follows. The first section will detail the problem encountered when dealing with obstruction. The second section will develop the two data structures implemented to solve it. The last section will focus on the parallel capacity of the proposed solver.

3.2 Naive obstruction computation

We have mentioned in the previous chapter how the S2S method was designed in such a way that the view factor computation matrix takes $O(N^2)$ operations to be computed, for a system with N facets. Then, for each view factor F_{ij} standing for the exchange between facets F_i and F_j, one has to ensure there is no facet F_k standing in between. The naive way to proceed would be to check the view factor F_{ij} over all the remaining facets in the system one by one. This would take $O(N)$ operations and bring our total complexity to $O(N^3)$. Algorithm 1 details this. For the first part of this thesis, we were more focused on ensuring correct physical results and predictable behaviour from the proposed solver, which means we were content with such a naive approach. However, when we first wanted to evaluate how the solver was coping with the simple 3D geometries presented in the previous chapter, we met a roadblock. Modelling steady-state pure radiation between two concentric spheres in a system totalling 4468 facets on a single core took about 19 hours. These computation times were out of line with an expected industrial day-to-day use of the proposed solver, where meshes including between 100k and 10M radiating facets are expected to be used. We then decided to set up a more clever approach.

Algorithm 1 Naive algorithm for obstruction computing

```
1: procedure OBSTRUCTION(F_i, F_j)
2:     Initialize ListFacets                ▷ Retrieve the list of radiating facets
3:     Bool NotObstructed = true
4:     Int k = 0
5:     while (NotObstructed && k < n) do
6:         Facet F_k = ListFacets[k]
7:         NotObstructed = CheckObstruction(F_i, F_j, F_k)
       return NotObstructed
```

3.2.1 Local half-space culling

A preliminary step we undertake in order to reduce the number of obstruction operations is to preemptively determine, for any given facet, which facets out of the list of radiating facets are facing it. This is a $O(N^2)$ step. Since our base complexity is at least $O(N^2)$, this step does not significantly worsen the time performance of the solver.

1. We run through the list of facets. For the ongoing facet F_i, we compute the scalar product between its normal vector $\vec{n}_i$ and the vector from the origin of the system to its barycentre, $\vec{w}_i$, and we save the result. This gives us the signed distance of the barycentre of F_i from the origin.

2. Then, we run through all the facets again, and we compute the scalar product between $\vec{n}_i$ and the vector from the origin to the barycentre of ongoing facet F_j, $\vec{w}_j$. If this result is higher than the previously computed scalar product, this means F_j belongs in the half-space located in front of F_i, and so F_i, can see F_j.

3. We also compute the reciprocal value by swapping the roles of F_i and F_j. If that result is also higher than the saved scalar product for F_j, this means F_i also belongs in the half-space located in front of F_j, and so F_j can see F_i.

4. We then store F_j in the list of "facets that can be seen by F_i".

Algorithm 2 details the process. In effect, this means we can assign 0 to a number of the view factors in the i line of the view factor matrix. Experience shows that for typical industrial geometries, only 15% to 50% of the terms are left to be computed after this filter has been applied. Thus, the view factor matrix is efficiently culled. This process amounts to computing the sparsity of the matrix.

3.3 Binary tree structures

One of the means of accelerating the obstruction process is the use of a data structure for the facets. By sorting the facets into a binary tree, we can check all the facets we need for obstruction into $O(\log(N))$ operations, with $\log(N)$ standing in this chapter for $\log_2(N)$, thus turning our total complexity into $O(N^2 \log(N))$. Note that this means adding in the $O(N^2)$ required for local half-space culling, we still achieve a total complexity of $O(N^2 \log(N))$. For this, we have investigated two different binary trees: metanode kd-trees and BV kd-trees. We will introduce each in turn.

Algorithm 2 Local half-space culling algorithm

```
1: procedure CULLING(ListFacets)
2:     Initialize SignedDistanceList
3:     Initialize LocalVisibilityList
4:     for (F_i in ListFacets) do SignedDistanceList[i] = ScalarProduct(n_i,w_i)
5:     for (F_i in ListFacets) do
6:         for (F_j in ListFacets) do
7:             if (F_i==F_j) then
8:                 Continue                    ▷ Facets do not radiate with themselves
9:             if (ScalarProduct(n_i,w_j) > SignedDistanceList[i]    ▷ F_i can see F_j
               and ScalarProduct(n_j,w_i) > SignedDistanceList[j]) then    ▷ F_j can see F_i
10:                ListFacets[i].add(j)          ▷ Sparse storage of non-zero indices
```

3.3.1 Metanode kd-trees

As mentioned in the introduction, ray tracing in imagery shares a number of mathematical and physical tools with radiative transfer by the S2S method: instead of tracing rays of heat energy, the purpose is to trace rays of illumination, so a lot of tools can be applied to both. One of them is Binary spatial partioning (BSP). It is a concept that finds enormous applications to sort objects in space. In a BSP tree, the space is recursively subdivised into a set of hierarchically linked volumes (also called voxels). The very first idea of the BSP tree comes from Schumaker et al. [1] in the context of image rendering. They have remained closely associated with ray tracing for imagery purpose ([2], [3], [4], [5], [6]). Tracing a ray from a light source to the eye of the observer through a BSP tree means finding down the tree all the objects that are located in the voxels situated between the source and the eye.

3.3.1.1 Kd-tree introduction

The kd-tree structure has been devised by Bentley in 1975 [7] and is one particular sort of BSP tree where the voxels are always axis-aligned cuboïds. It is a binary tree that stores points in space, which in our case will be the facet barycentres. It stands for "k-dimensional tree". While the technically correct way of designing these trees in 2D, 3D, or more would be to say "2d-tree" or "3d-tree", usage has spread out to also call them "2D kd-tree" or "3D kd-tree", and both appellations can be found in the literature. Figure 3.1 shows a simple 2D kd tree. Kd-trees can actually sort vectors of dimension much higher than 2 or 3, though their sorting and access performance typically decrease very fast and special adaptations then need to be made. We will focus on 2D-trees and 3D-trees here. The use of a kd-tree

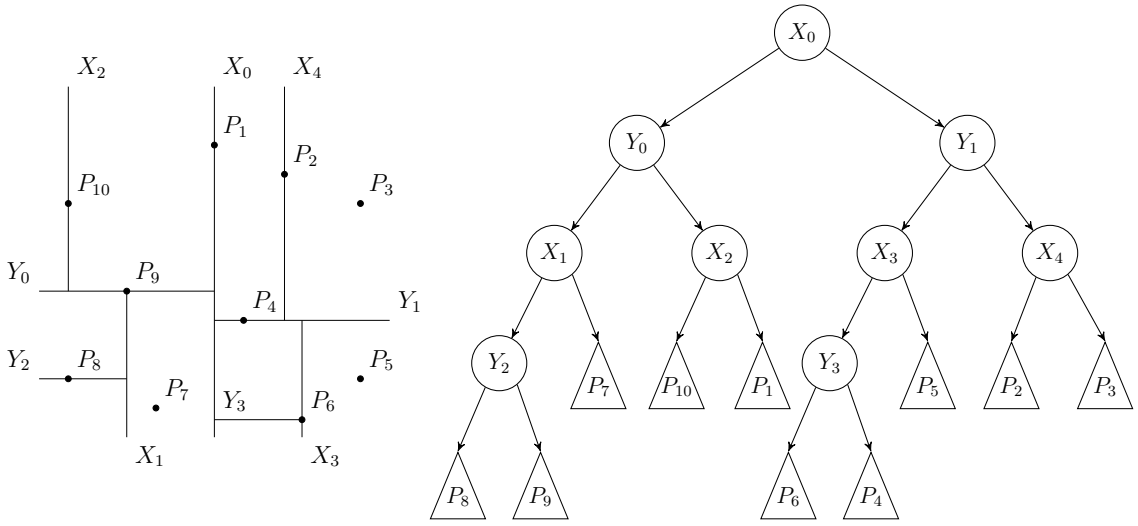


Figure 3.1: Sample 2D kd-tree. On the left, the spatial repartition of points and the $x-$ and $y-$ split values chosen. On the right, the binary representation of the tree. All data is stored in the leaves.

to accelerate radiosity computation has already been investigated by Bindick et al. [8]. while other forms of radiative heat transfer modelling such as the Monte-Carlo method can also make use of BSP trees [9, 10].

In this tree, the space containing the data to be sorted is recursively divided into a number of axis-aligned space partitions, each of them being assigned to a node in the tree. First, the set of point is divided into two sets of roughly equal size, separated by a split along the x coordinate. The value chosen for this x-split, X_0, is stored in the root node. In Figure 3.1, the X_0 value is chosen to be the x coordinate standing as the median of all the x coordinates of the set of points. The point to which the split coordinate belongs is sorted into the left subtree. On the left child, containing all the points in space the x-value of which is lower than X_0, we split again the points in two sets of equal size but along a y-split, creating a subtree. We store the Y_0 in the child node. We iterate this way by alternating splits along the x and the y coordinates of the points (for such a 2D tree). In Figure 3.1, we stop creating subtrees when a node contains a single point. This node is then assigned the label of a leaf node. At the end of our tree construction, all data is stored in the leaves of the tree. Alternatively, the points can be partially stored in the inner nodes instead of in the leaf nodes only. In this case, each time a split is made, the point of which coordinate the split was chosen is stored, alongside the split value itself, in the root node of that subtree and subsequently removed from the remaining subgroups of point. This leads to the kind of kd-trees shown in Figure 3.2.

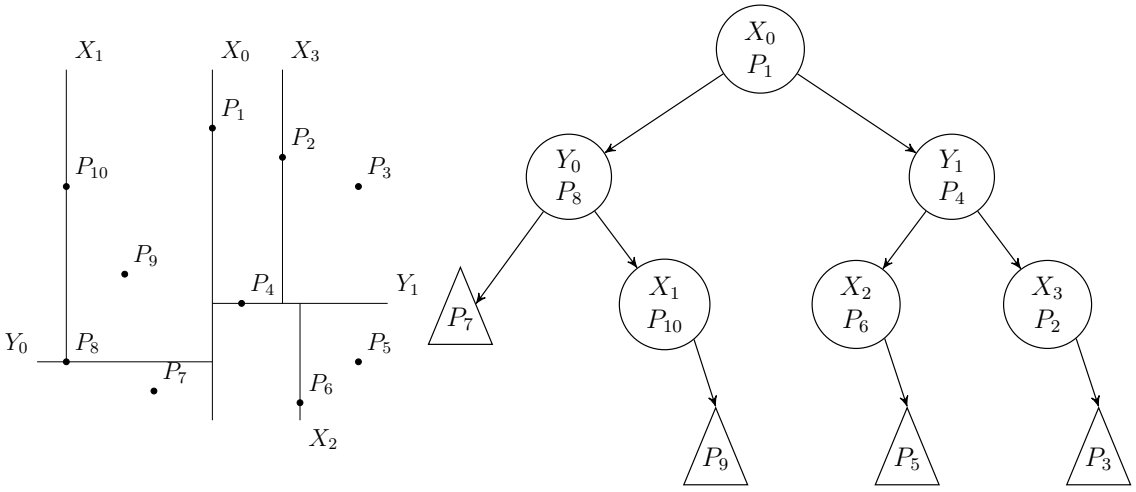


Figure 3.2: Sample 2D kd-tree. On the left, the spatial repartition of points and the $x-$ and $y-$ split values chosen. On the right, the binary representation of the tree. Data is stored in part in the inner nodes and in part in the leaves.

Locate operation One of the most basic operations on a kd-tree is to determine where a given point fits in the kd-tree, that is, in which leaf node it belongs. The recursive procedure is as follows to locate point A in the tree shown in Figure 3.1:

1. Look at the root node's split coordinate. Here, it is the x coordinate. Then, take that coordinate from point A and compare it to the value of the coordinate of the splitting plane of the node. So, compare $A.x$ and X_0. If $A.x$ is smaller, go to the left child. If not, go to the right child.

2. Now, look at the new root node's split coordinate. In our example it will be y. Compare $A.y$ to the coordinate of the split plane of the node we are in. It will be Y_0 if we have headed towards the left child of the root node, or Y_1 if towards the right child. Once again, if $A.y$ is smaller, go to the left child. If it is higher, to the right child.

3. Carry on until you reach a node that has no children. This means it is the leaf node into which A is located.

This process takes $O(\log(N))$ operations and the pseudocode can been seen on Algorithm 3.

Axis-aligned bounding boxes Another useful concept for kd-trees is that of the axis-aligned bounding box (AABB). Each node is defined by four frontiers in 2D (eight in 3D). These frontiers are the axes that have split the space around the node. For example, in Figure 3.3, the node containing point P_4 is entirely contained within

Algorithm 3 Locate point into tree algorithm

```
1: procedure LOCATE(point, node) return node
2:     if (node.Leaf == true) then
         return node
3:     if ( thenpoint[node.splitCoordinate] < node.splitValue)
         return LOCATE(point, node.leftChild)
4:     else
         return LOCATE(point, node.rightChild)
```

the rectangle formed by the X_0, X_3, Y_1 and Y_3 split axes. The AABB of this node is, by definition, that rectangle. The root node's AABB is a rectangle defined by its two corners: the bottom-left corner has the lowest x and y coordinates found among all the points of the set, while the top-right corner has the highest. The AABB of the node containing point P_8, for example, is defined by the Y_2 split plane to the north, the X_1 split plane to the east, and the root node's AABB to the west and south. The knowledge of a node's AABB is not required for all uses of a kd-tree, but it will be for us. Depending on how often a particular kd-tree algorithm requires to access a node's AABB, it can be computed and stored into each node at tree generation or it can be computed on the fly during tree traversal. The former solution increases the weight of the tree but makes access faster, while the latter offers the reverse benefits and drawbacks. The root node's AABB needs to be computed directly from the set of points. All other nodes' AABBS can be deduced by the node's parents' split planes.

3.3.1.2 Kd-tree generation

Generation of a kd-tree is an operation that can be optimized in and of itself to ensure the kd-tree is of optimal performance. The most common procedure is to recursively build the tree by tackling the current set of points, creating a node and two subsets, and applying the procedure again to the subsets. Some important criteria to pay attention to are the following:

- At each layer of the tree, which coordinate should we take to execute the split? In the trees shown in Figures 3.1 and 3.2, we have periodically chosen to split on the x then y coordinates. Another common criterion is to choose the coordinate the variance of which across all the points in the current subset is highest. This can lead to splitting two times in a row on the same coordinate, or not having all nodes of a given layer split on the same coordinate.

- Once that coordinate has been determined, which point should we take as the split point? Figures 3.1 and 3.2 choose the point the split coordinate of which

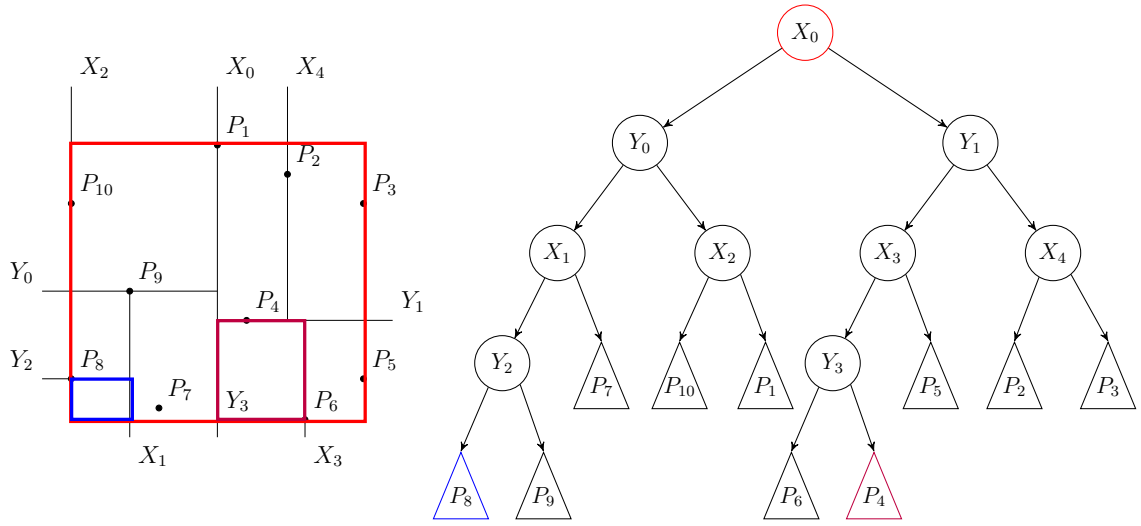


Figure 3.3: Example axis-aligned bounding boxes (AABBs). In red: the AABB of the root node. In blue: the AABB of the node containing point P_8. In purple: the AABB of the node containing point P_4.

is the median of the coordinates in this subset, but this is merely the simplest criterion. A lot of research has been invested in that question, because it heavily affects the quality of the tree.

- Which termination criterion to use? Do we want our nodes to contain a set maximum of points before assigning them to be leaves, which can be more than 1? Or do we want a set depth of our tree?

This means the most basic recursive tree generation algorithm looks like Algorithm 4. The termination criterion is a set number of points remaining in the node, and all the points are stored in the leaves.

The above choices are of critical importance for having a balanced and efficient tree. A very popular technique created by MacDonalds and Booth [11] is the Surface Area Heuristics (SAH). The idea is to evaluate the geometric probability and subsequent computation cost that any launched ray will cut through the AABB of any node of the tree. By summing these probabilities one obtains an approximate "cost" for ray tracing through the tree. The split plane origin and orientation are then taken so as to minimize this cost. Termination is decided when the heuristics estimates splitting again the current node would be costlier than letting it be a leaf. The use of this heuristics then determines all of the three above points as a locally optimal process for each node. Its computational cost is rather important, however, and many upgrades of this heuristics have later been proposed ([12], [13], [14]).

Algorithm 4 Naive recursive kd tree generation algorithm

```
1: procedure BUILD(currentNode, pointsList)
2:     if (pointsList.size ≤ Criterion) then
3:         currentNode.leaf = true
4:         currentNode.content = pointsList
5:     else
6:         splitPlane = findPlane(pointsList)        ▷ Finds the best coordinate and splitting point
7:         currentNode.leftChild = new node()
8:         currentNode.rightChild = new node()
9:         listLeft, listRight = Split(pointsList, splitPlane)
10:        Build(currentNode.leftChild,listLeft)
11:        Build(currentNode.rightChild,listRight)
```

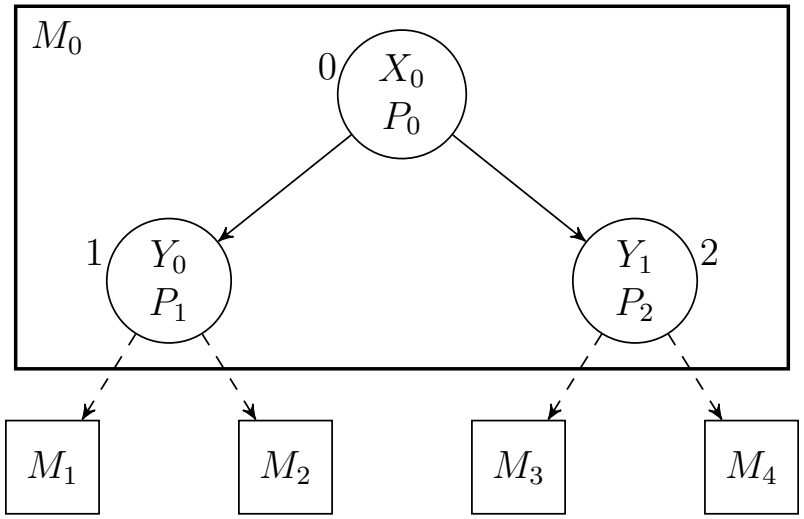
Metanode-based kd-tree We have settled for another criterion for our first implementation, which we estimated would offer different benefits. We decide to choose the split coordinates by rotating through x, y and z (in 3D), and to choose the split point with the median criterion. We then create a "metanode" which actually encompasses all these three splits, instead of creating one node per split as the standard way goes. In 2D, a metanode is composed of one node with a point and an x-axis split plane to which the point belong, and its two children nodes each storing a point and a y-axis split to which that point belongs. It thus contains three nodes and stores three points. In 3D, the metanode also encompasses, for each of the y-axis nodes, its two z-axis children nodes, each storing a point and a z-axis split plane. The 3D metanode thus contains seven regular nodes and stores seven points. It contains an array of the pointers to each of the inner nodes it stores. The metanode structures can be seen in Figure 3.4, including the index of each inner node's pointer in the metanode's array.

Algorithm 5 indicates, for a 2D case, the functions used to construct a metanode. The Bisect function sorts the current list of points according to a specified coordinate and then returns the median point of that list. The Split function creates a metanode by assigning to it the list of points it contains. For this, the metanode receives the pointers to the first and the last elements of the list, and that is enough to later access all the points. It bisects its list of points according to x, and then the two sublists created by this bisection according to y. Thus, the initial list of points has been split three times and defines four point list intervals. It generates its own AABB by extracting the min and max x and y coordinates of its list of points. Finally it creates its four children metanodes. It assigns to each child its corresponding point list interval by fetching the pointers to the first and last element of said list.

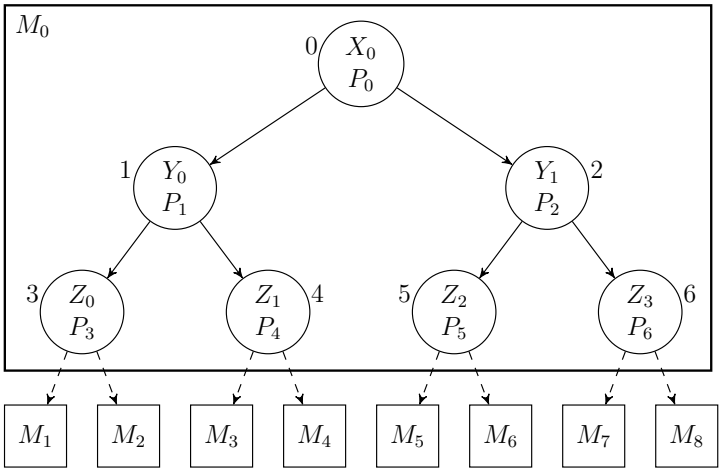
We will see that the complete build algorithm takes a vector containing the

Algorithm 5 Metanode construction algorithm

```
1: procedure BISECT(coordinate, firstPtr, lastPtr)
2:     length = lastPtr - firstPtr
3:     sort(firstPtr,lastPtr,coordinate)        ▷ We sort according to that coordinate
   return firstPtr+length/2                     ▷ We return the pointer to the median value
4:
5: procedure SPLIT(metanode,child)(2D)
6:     Initialize InnerNodes(metanode) ▷ This contains 3 nodes in 2D and 7 in 3D
7:     firstPtr = metanode.firstElement
8:     lastPtr = metanode.lastElement
9:     InnerNodes[0] = Bisect(x,firstPtr,lastPtr)  ▷ The first inner node split is by
   the x median
10:    metanode.AABB.update(x)   ▷ Now the pointers are x-sorted,so we update
   the AABB
11:    InnerNodes[1] = Bisect(y,firstPtr,InnerNodes[0])
12:    InnerNodes[2] = Bisect(y,InnerNodes[0]+1,lastPtr)     ▷ The next two inner
   nodes splits are by the y median of the subsets
13:    metanode.AABB.update(y)
14:    *child=metanode(firstElement=firstPtr,
                       lastElement=InnerNodes[1])
15:    *(child+1)=metanode(firstElement=InnerNodes[1]+1,
                           lastElement=InnerNodes[0])
16:    *(child+2)=metanode(firstElement=InnerNodes[0]+1,
                           lastElement=InnerNodes[2])
17:    *(child+3)=metanode(firstElement=InnerNodes[2]+1,
                           lastElement=lastPtr)
18:    for i from 0 to 4 do
19:        *(child+i).parent = metanode
20:        metanode.children[i] = child+i
```



(a) 2D metanode and its four children.



(b) 3D metanode and its eight children.

Figure 3.4: Metanode structure in the kd-trees. Each node contains a split plane, a point, and next to it the indix of its pointer in the metanode's array.

coordinates of all points as a parameter. This means all the addresses of the points are contiguously stored in memory. We take advantage of this to only manipulate pointers to the points during metanode construction. We can see the local subset of points the metanode is encompassing is never copied, nor it is stored anywhere in the metanode structure. All the metanode needs are pointers to the first and last element of its (initially unsorted) subset. All operations that need to be done on the points themselves are done by manipulating their pointers instead. Thus, the Bisect function that returns the median of the subset for a specific coordinate sorts all the adresses of the subset by increasing value of that coordinate (they

remain contiguously stored), and then returns the address located halfway through the range of local addresses. The Bisect function does not actually need the list of all addresses to process, just the addresses of the points located at the beginning and at the end. The root metanode is defined in the general build procedure with the first and last elements of the total set of points. The structure is thus lightweight, and the complete algorithm does not have the computational cost of copying or passing whole subsets of points to its subroutines. We have omitted the 3D version of the "Split" procedure because the writing would be rather tedious, and can be easily inferred from that of the 2D version.

The advantage of this strategy is to ensure all metanodes begin by an x split. This way, when we need to place a point through the tree, we automatically know we have to first look at its x coordinate, then the y, then the z. This means we successively determine where the point should go relatively to the first layer, then the second layer, and finally the third layer of the metanode. At the end of the process, our point has been directed towards the relevant child metanode, and the process can start again by an x-coordinate comparison. If we didn't have this metanode structure, in order to direct a point through any given node, we would have to retrieve the stored split coordinate of the node and then would we be able to compare the relevant point's coordinate to that of the splitting plane. This is what happens on line 3 of Algorithm 3. Our structure makes it so that there is no need to store node split coordinate, which means each node structure is slightly simpler, and also that directing a point through a node is faster, since we save the reading operation. Choosing to use the median criterion for split plane determination allows us to have a perfectly balanced tree, since this ensures us each subtree contains as close to possible as half the points contained in its parent's AABB. It also considerably speeds up tree generation compared to the use of the SAH method. One notes, however, that the use of the SAH method is compatible with our metanode structure, but due to time constraints we have chosen to first implement this simpler version of the kd-tree.

This choice of implementation also leads us to the kind of tree seen in Figure 3.2, where the points are stored partly in the branches and partly in the leaves. Having the data partially in the branches rather than completely in the leaves allows the tree to be a little shallower compared to an "all leaves" tree: one can compare between such a tree shown in Figure 3.1, which is four layers deep, and the tree in Figure 3.2 which is only three layers deep. More generally, a tree storing N points in the leaves only stores 2^k points in k layers, it needs a depth of at least $k \geq \log(N)$. A tree storing points in the branches too stores $\sum_0^k 2^i$ points in k layers, meaning it only needs a depth of at least $k \geq \log(N+1) - 1$. This tree thus contains one less layer than the "all leaves" tree, which amounts to about half the number of nodes, depending on the extent to which the last layer is complete.

Metanode-based generation algorithm The tree generation algorithm, contrary to the standard one, is not a recursive one where the set of all points is divided into two subsets by a splitting plane and the creation of a node, and then the algorithm is called again on each subset. We have chosen instead to proceed with an iterative algorithm. When the dimension of space is d, a metanode will store $2^d - 1$ points. This means a tree for N points will need, at maximum, a number of metanodes equal to $\lceil \frac{N}{2^d-1} \rceil$. And these metanodes will need a number of layers k such that $k \geq \log(\frac{N}{2^d-1} + 1) - 1$.

1. We preemptively create a kd-tree containing that amount of layers and metanodes (which means we possibly create some excess metanodes, up to $2^k - 2^{k-1} - 1$ if the last layer contains a single nonempty metanode).

2. We instantiate the root metanode.

3. We iteratively run through each row of pointers to corresponding rows of metanodes in the tree, starting with the first row (that only contains a pointer to the root node). We proceed to the d successive splits for each of pointers' corresponding metanodes (see Algorithm 5). We also instantiate each of these metanode's children, so that they are already defined when the procedure reaches their row.

4. At the end, we make one last pass on all the metanodes of the last complete row. Only at that moment do we check whether each of their children contains any point at all (in which case they are tagged as leaf metanodes) or not (in which case they are among the excess metanodes created and their pointer is set to null).

Checking for leaf status in a separate loop instead of during the splitting loop allows us to save the leaf checking operation for all but the last row of the tree. Meanwhile, preallocating the metanodes and sequentially, instead of recursively, running through them for splitting prevents the tree generation algorithm to generate overhead by adding a lot of calls to itself on the stack during a recursion. One can see Wald [15] for further discussion about limiting function call on the stack and generally keeping the algorithm lightweight. Algorithm 6 sums up the process.

Algorithm 6 kd-tree sequential build algorithm

```
1: k = dimension
2: procedure BUILD(pointsList) return kd-tree
3:     N = pointsList.size
4:     FullLayers = ⌊log(N) + 1⌋ ▷ Number of layers we know will be full
5:     MetanodeFullLayers = FullLayers / k
6:     NodesInMetanode = 2^k − 1
7:     Metanodes = 2^FullLayers − 1 / NodesInMetanode
8:     Initialize MetanodesVector(Metanodes) ▷ Preemptively allocate a vector of the max possible size
9:     pointer current = MetanodesVector[0] ▷ Operations will be made on incremented pointers
10:    pointer children = MetanodesVector[1]
11:    root = current
12:    root.firstElement = *pointsList[0]
13:    root.lastElement = *pointsList[N] ▷ We store pointers (not elements) into root metanode
14:    for (layer in MetanodeFullLayers) do ▷ Iterate through full layers for construction
15:        MetanodesInLayer = 2^(layer∗k)
16:        for (int r in MetanodesInLayer) do
17:            Split(current, children) ▷ We construct the inner nodes by median splitting
18:            current ++ ▷ To point at the next metanode in MetanodesVector
19:            children += NodesInMetanode ▷ Since Split instanciates NodesInMetanode at once
20:    for (metanode in MetanodeFullLayers[FullLayers]) do ▷ We examine the potential children of the metanodes of the last full layer
21:        for (child in metanode.children) do
22:            if (child is not empty) then
23:                CreateLeaf(child)
    return kd-tree(root)
```

Related work and performance evaluation Wald and Havran [12] have established that the theoretical optimal complexity on CPU for serial kd-tree generation is $O(N \log(N))$. They also demonstrate that the complexity of the standard, median-based split kd-tree building algorithm does attain such a complexity. After establishing a review of the existing methods thus far, they exhibit a SAH-based method that attains this lower bound complexity. Since our algorithm is a median-based one, we should achieve this theoretical complexity. Figure 3.5 indicates how

the algorithm scales with an increasing number of points in the kd-tree. In order to generate the dataset, we have chosen 3D meshes of various industrial systems, and have refined or made them coarser to increase the number of points. Though in theory we should attain $O(N \log(N))$ complexity, in reality we attain a complexity close to $O(N^{1,1648})$. This might be because of our second pass at the last tree layer to determine whether each node there is a leaf node or an excess node. All tests have been run on 1 Intel i7 6700 processor running at 3,4GHZ and equipped with 16GB of RAM. We have been able to generate a tree of up to 20 million points in $53s$ seconds, which shows the robustness of our algorithm.

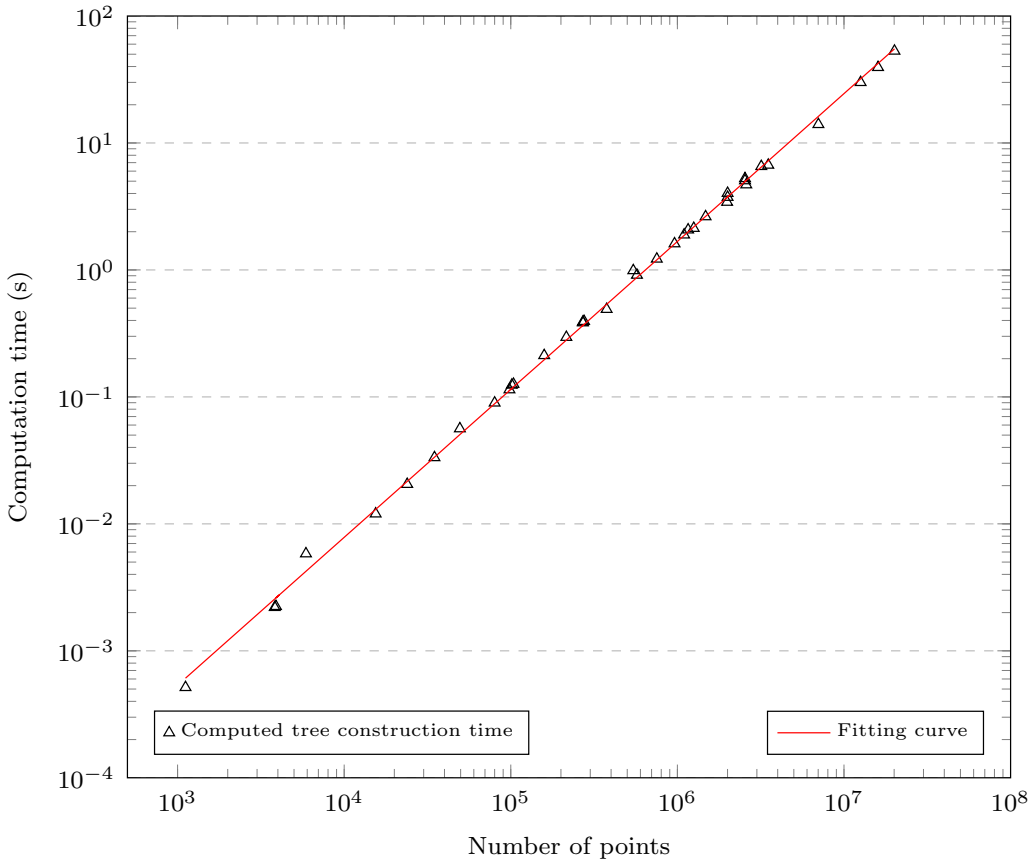


Figure 3.5: Complexity of kd-tree generation time depending on its number of points.

Later research has mainly focused on the construction of kd-tree using parallel algorithms running mostly on GPU, and they often are very different from the kind of CPU algorithm presented here. Since our generation algorithm is fully serial and runs on CPU, we do not compete with their performance. Li et al [16] have devised an algorithm for parallel construction on GPU, based on Morton code, that attains

a complexity of $O(\log(N))$. Liu et al [17] have refined the same type of algorithm with particular care to hardware implementation and memory allocation. Li et al [18] have provided a useful survey of the latest research in trying to improve ray tracing through kd-tree to the point of real-time rendering, including a survey of the kd-tree construction algorithms, on GPU and with parallel computation. Choi et al [19] have expanded and refined the state-of-the-art serial CPU algorithm of Wald and Havran [12] so that the SAH computation can efficiently take place in a parallel process. They have determined the SAH gives suboptimal splitting planes for the biggest nodes of the tree. Thus, they apply a different heuristic for the first eight layers of the tree. Shevtsov et al [13] have devised an algorithm that sorts blocks of space instead of points, leading to a degraded tree quality but a very good construction time: they can efficiently render 3D scenes in real time by rebuilding their tree from scratch at each frame. The last two papers include test cases with computation times, to which we will compare our results. Other papers do so too, but not all detail the hardware and configuration used, which makes comparisons across different installations hardly relevant. Besides, these two display the best results on single core CPU algorithms. The two papers use different geometries, but three are shared between them: the Stanford bunny, the Stanford dragon, and the happy buddha. These files are in open access on Stanford's website, and are often used in computer graphics papers as standard test cases against which one measures the performance of one's algorithm. Thus, we will do so too. Figure 3.6 displays these test cases.

In order to filter out the relative performances of the different architectures used from the time performances compared, we referred to the CPU UserBenchmarks online comparison tool. Shevtsov et al have used an Intel Core2 Duo clocking in at 3 GHz, and Choi et al. a Xeon X7550 at 2 GHz. The performance of our Intel Core 17 6700 at 3.4 GHz for single core operations is 113% faster than that of Shevtsov et al. and 123% faster than that of Choi et al., meaning we will divide their time results by 2.13 and 2.23 respectively to obtain adjusted comparative values. Table 3.1 shows the various construction times.

Test case	Stanford bunny	Stanford dragon	Smiling Buddha
Shevtsov et al. (adjusted)	0.104 (0.49)	0.751 (0.353)	0.696 (0.327)
Choi et al. (adjusted)	0.304 (0.136)	3.75 (1.68)	4.67 (2.09)
Present work	0.074	1.504	1.893

Table 3.1: Time to generate the facets kd-tree for these images (in s).

We can see Shevtsov et al. algorithm outperforms ours. But their tree has been lightened so as to be able to be rebuilt at each time step while ours does not

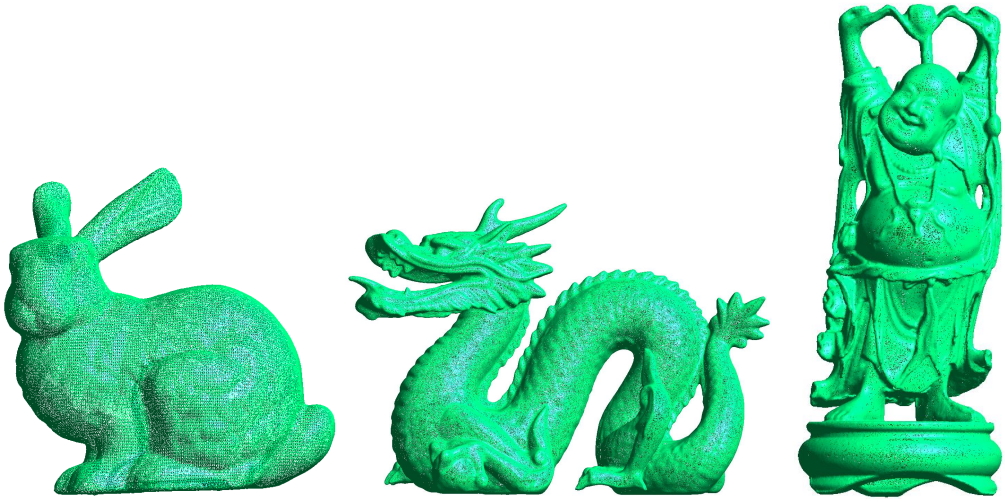


Figure 3.6: Test cases for our kd tree construction algorithm. From left to right: Stanford bunny, 69K facets; Stanford dragon, 863K facets; happy Buddha, 1087K facets. Images courtesy of Stanford University.

need so, since the industrial geometries we handle for radiation are not likely to *completely* transform from one time step to the next. Choi et al. algorithm displays performances that are slighty worse than ours'. Of course, their algorithm proceeds to the costly SAH and other heuristics operation in order to determine the next splitting plane, while we do not. This means it is expected that their algorithm will run slowlier. However, we are content to see our simple algorithm is on the same scale of magnitude that this state-of-the-art program. A construction operation taking about 2s to process a mesh of up to 1 million facets on a single core desktop computer processor is compatible with a daily use by an indutrial research team.

3.3.1.3 Kd-tree nearest neighbour search

The *nearest neighbour* (NN) search operation consists of taking a group of points and an input point, and to find which point among those of the group is closest to the input point. It is greatly enhanced when the group of points is sorted into a kd-tree. We don't directly use it in our ray tracing application, but it is interesting to look at how our tree performs with regard to this operation nonetheless for other applications. For example, the mesh immersion algorithm we use makes NN searches in order to find out, in parallel computation, to which facet of the receiving mesh each facet of the immersed mesh is closest. A lot of applications that call for the use of a kd-tree do so because they need to apply a NN search when the dimension

k is small before the size of the dataset: 3D points sorting for space mapping [20], data classification where the items don't have too many labels [21]. A good rule of thumb is that kd-tree NN search is efficient when the number N of items to be sorted in dimension k is such that $N >> 2^k$ [21]. Kingsford [22] indicates the average time complexity of an NN research in a kd-tree by the standard algorithm is close to $O(\log(N) + 2^k)$. We see that $N >> 2^k$ does not guarantee at all that $\log(N) >> 2^k$, and when the dimension is high the latter term will be dominant in the complexity. However, with some adaptations, kd-trees still see uses at higher dimensions, because they can be efficiently tuned to approximate NN searches [23], [24].

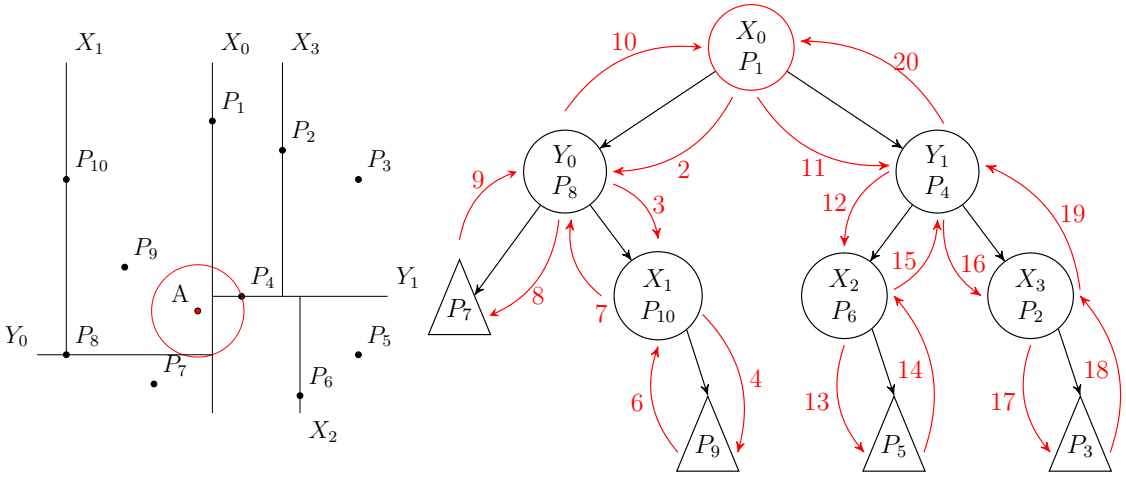


Figure 3.7: Standard KNN research for point A in the given kd-tree. The red circle on the spatial representation shows A's nearest neighbour is point P_4. The numbered arrows on the binary representation show the algorithm's path during the research. The starting point is the root node, in red.

KNN standard algorithm In a given kd-tree, the NN of a random point is not necessarily located in the leaf node that includes that point. This can be seen in Figure 3.7: point A's NN is point P_4, even if A is located in the leaf node containing point P_9. This means the algorithm must do more than merely locate the point in the tree as per Algorithm 3. Let us see through an example how the standard recursive KNN algorithm, introduced by Friedman et al. [25], proceeds. In Figure 3.7, we have located point A of coordinates $(-0.25; -0.25)$ in the tree space, and we will look for its NN using Euclidean distance. We will use a simple kd-tree with no metanode structure to make things simpler here.

1. We start at the root node. The current best NN candidate is set to NULL and the current best NN candidate distance is set to Infinity. Since the root node

is not empty, we look at the point in it: point P_1. We compute the distance between A and P_1. Since this distance is smaller than our current best NN candidate distance, we now store P_1 as the current best NN and the relevant distance as the current best NN distance.

2. Where should we now go in the tree? Similarly to how Algorithm 3 works, we compare A's relevant coordinate (the x coordinate for the root node) to the coordinate of the split plane of the node. Since $A.x < X_0$, we go to the left child. This is arrow 2. However, we have seen the best NN does not have to be located in the same subtree as point A. This means we first recursively call the algorithm on the left child, but then we will need to call it on the right child too.

3. We reach the P_8 node. Since it is not empty, we evaluate the distance from A to point P_8. It is smaller than the distance from A to point P_1, so we update our new best NN candidate to be P_8 and the new best NN candidate distance accordingly. This node's split coordinate is y. Since $A.y > Y_0$, we call the algorithm on the right child (This is arrow 3), and then the left child.

4. In the P_{10} node, we evaluate distance between point P_{10} and A and discover it is higher than the current best NN distance. Our current best NN thus remains P_8. This is an x-split node, and since $A.x > X_1$ we first call the algorithm on the right child (arrow 4) and then the left child.

5. In the P_9 leaf node, we check A against point P_9 and see it is closer to A than P_8 was. We update accordingly our best NN candidate and distance. Then, this node has no split plane. Thus, by convention, the algorithm is first called on its (null) right child, and then its (null) left child.

6. In the right child, that node is NULL, so the algorithm exits at once. The recursion on the right side of the P_9 leaf node is thus over. Since the left child is also NULL, the algorithm also exits at once there. This means the recursion on the right child of the P_{10} node is over. Now, we move back to that P_{10} node (arrow 6).

7. We enter the recursion of the left child of this node that has been added to the stack on Step 4. Since this is a NULL child, the recursion ends at once. We have now finished the recursion in the right child of the P_8 node. We thus move back to that node (arrow 7).

8. We enter the recursion in the left child of the P_8 node that has been added to the stack on Step 3: the P_7 node. And we will make an additional check here. We compute the distance between point A and the P_7 node's AABB.

This amounts to computing the distance from A to the split contained in the P_8 node. We compare that to our current best NN distance. This is because the distance from A to the P_7 node's AABB amounts to the smallest possible distance there can be between A and a point contained in the left child of the P_8 node (that would be the projection of A onto that child's AABB). If our current best NN distance is smaller than this, then no point in the left children can beat our current best NN candidate. Thus, there is no purpose in checking that subtree, and we can dismiss it at once. In our case, the distance from A to axis Y_0 is smaller than our current best NN distance, which is the distance from A to point P_9. This means we do need to proceed through the left child recursion (arrow 8). These checks were actually performed all along in each encountered node. But for the first tree descent, point A is always located *inside* the current node's AABB, meaning the distance returned is always negative and thus always smaller than the current best NN candidate distance.

9. We enter the P_7 node. We evaluate P_7 as a potential best NN candidate, and find out P_9 is still better. We call the recursion on the two children of that node and it returns immediately for both. The recursion on the left child of the P_8 node is now over. We move back to that node (arrow 9).

10. The two recursions on the two children of the P_8 node are done. This means the recursion on the left child of the root node is now over. We move back to the root node (arrow 10).

11. In the root node, we enter the recursion in its right child that has been added to the stack on Step 1 and thus move to the P_4 node (arrow 11).

12. We check the distance from A to the P_4 node's AABB, that is, distance from A to axis X_0, and find out it is smaller than our current best NN distance. This means we need to explore that child. We evaluate P_4 itself as a NN candidate and find it is indeed the best current NN candidate, so we update our current best NN candidate and distance. Then, this node is a y-split node. Since $A.y < Y_1$, we first call the recursion on the left child of the P_4 node (arrow 12).

13. In the P_6 node, P_6 does not prove to be a better candidate. Since $A.x < X_2$ the recursion is first called on the (null) left child of the node. It is then called on the P_5 leaf node (arrow 13).

14. We check the distance between A and the P_5 leaf node's AABB, that is, distance between A and split plane X_2. It is higher than our current best NN

candidate distance, we stop the recursion there and move back to the the P_6 node (arrow 14).

15. All recursions in the the P_6 node's children are over, we move back to the P_4 node (arrow 15).

16. Now we look at the P_6 node's right child's recursion. We enter the P_2 node (arrow 16).

17. In the P_2 node, we check for point P_2 and find out it is not a better NN candidate than P_4. Since $A.x < X_3$ we first call the recursion on the (null) left child, then on the right child, the P_3 leaf node (arrow 17).

18. We compute the distance from A to split plane X_3 and since it is higher than the current minimal NN distance we discard the leaf. That recursion is over too and we move back to the P_2 node (arrow 18).

19. All recursions in the P_2 node are done, so we move back to its parent the P_4 node (arrow 19).

20. Similarly, all recursions in the P_4 node are done, so we move back to its parent the root node (arrow 20).

21. All recursions in the root node are over. The tree is fully explored. The nearest neighbour is P_4.

The pseudocode for this algorithm is shown in Algorithm 7.

Though it is very simple, we can see a path of upgrade at once: during the course of the search, many interior nodes are checked multiple times. For example, the P_4 node is visited three times. In addition, in our simple example, almost all points have been evaluated as potential nearest neighbours: only points P_5 and P_3 have not. This is a symptom of the points set being very small. With a higher number of points in the tree, the P_5 and P_3 nodes and their children would have contained a lot more points each, and culling their subtrees would have resulted in more point checks avoided. Another issue is that the recursions incur a lot of overhead.

Semi-stack-based KNN algorithm Our implementation is a little different. First, it is adapted to our metanode structure. We have seen in Algorithm 7 that each time a new node is called, we check if the point it contains might be a better NN candidate than the current best one. In our algorithm, each time an inner node of a metanode is reached, we make the same check with the point stored there. We have mentioned an asset of this structure is the order in which the coordinates are checked. Algorithm 7 does not explicitly state how "nextCoordinate" is determined

Algorithm 7 Standard Nearest-Neighbour searching algorithm

```
1: GlobalVariable NN
2: GlobalVariable NNDistance
3: procedure NearestNeighbour(Point A, Node N, Coordinate C, Bounding-
   Box B)
4:     if (T == NULL or distance(A, B) > NNDistance) then
         return                    ▷ If the node is too far from A we don't explore it
5:     if (distance(A, N.point) < NNDistance) then
6:         NN = N.point
7:         NNDistance = distance(A, N.point)
8:     if (A[C] < N.splitPlane[C]) then          ▷ We first explore the closest child
9:         NearestNeighbour(A,N.leftChild,nextCoordinate,N.leftChild.AABB)
10:        NearestNeighbour(A,N.rightChild,nextCoordinate,N.rightChild.AABB)
11:    else
12:        NearestNeighbour(A,N.rightChild,nextCoordinate,N.rightChild.AABB)
13:        NearestNeighbour(A, N.leftChild, nextCoordinate, N.leftChild.AABB)
```

on lines 9 to 13: it depends on the node class implementation. Though the simplest way is to give to each node an attribute stating alongwith which coordinate the splitting plane is contained is made, this requires storing said coordinate, and then accessing it during the recursion. In our implementation, the locate subroutine called on any given metanode always calls the coordinates in the same pattern: x, then y in 2D, or x, then y, then z in 3D. Nowhere in the metanode structure is the coordinate of each split stored, and no access to such data is needed during the procedure.

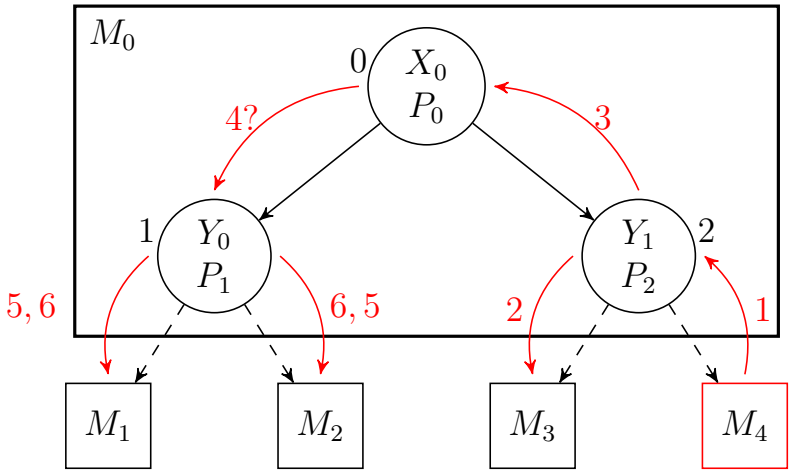
The second difference our implementation shows is the use of a stack to limit too much recursive calls, in the manner of the stack-based ray traversal algorithm shown in Figure 3.11a. At the beginning of the algorithm, when the point is first located in a leaf node, we push onto a stack each metanode we encounter. We do not recursively call the algorithm on all children of each metanode we have crossed: that would be 4 metanodes in 2D and 8 in 3D. Instead, after the leaf node has been reached, we pop the metanodes from the stack (it amounts to climbing back into the tree one layer at a time). Then, we determine which of the non-chosen children is the one closest to our point A. Only then do we call a recursive search down that child, in the manner of Algorithm 7. These two operations are handled by the Unwind procedure, which appears on line 12 of Algorithm 8.

Algorithm 8 Semi-stack-based Nearest-Neighbour searching algorithm

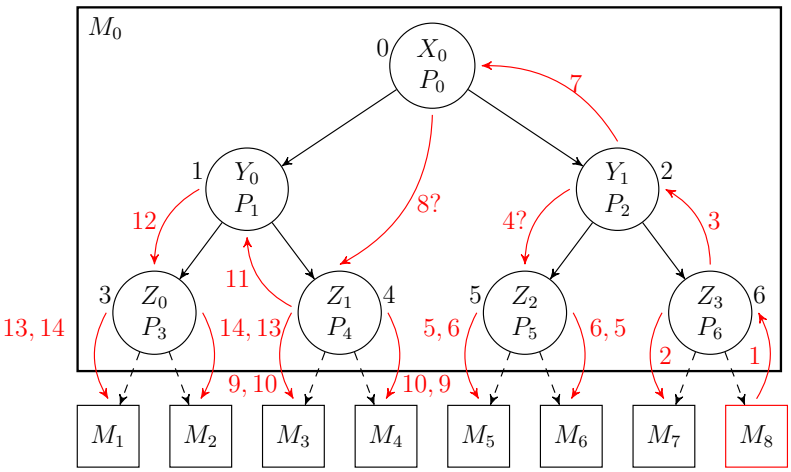
```
1: procedure NearestNeighbour(Point A, Metanode M)
2:     point NN = NULL
3:     double NNDistance = infinity
4:     Initialize stack
5:     while (N is not NULL) do
6:         stack.push(M) ‘
7:         M = MetanodeLocate(A, M)
8:     NN, NNDistance = CompareBestNeighbour(A, M) ▷ Examine all points in M for NN potential
9:     while (stack is not empty) do
10:        nextM= stack.pop ▷ nextM is the parent of M
11:        int childIndex = *M→child[0] - *nextM ▷ This indicates which child of nextM is M
12:        Unwind(A, nextM, NN, NNDistance, childIndex)
       return NN
```

Why have we chosen such a hybrid approach? The use of a stack to register the metanodes encountered when travelling down allows us not to make a recursive call to each child of the metanodes. Instead, when proceeding back through the stack, we call a recursion on maximum one other child of the metanode. This limits costly recursion operations. But then, why proceed to a classical recursive call in each of these other children, instead of pushing them on the stack as well? This is because we work under the hypothesis that the best NN candidate is not located too far from the leaf node in which point A is initially located during the first descent of the tree. Say we have a 3D mesh comprising 100k radiating facets; this is a typical industrial situation. Then we have $N >> 2^k$ as is required for a good use of the kd-tree NN research, but also $(\log(N) = 20) > (2^k = 8)$. In Kingsford's complexity of $O(\log(N) + 2^k)$, $\log(N)$ is the time complexity of the tree descent and 2^k the time complexity of the search in neighouring nodes. We see that in our situation, most of the time is spent travelling down the tree. As a result, we use the stack-based approach for the tree descent, where we want to avoid calling unnecessary recursions. But when exploring the neighbourhood, we assume not too much recursive calls will be needed, since many of them will be pruned as the standard recursive algorithm does. We assume calling a reasonably small number of recursions is then less costly than handling the stack of metanodes.

The Unwind procedure appearing on Algorithm 8 is detailed itself on Algorithm 9. It amounts to the local backtracking process that occurs inside a metanode. We have seen on Algorithm 7 that when the algorithm backtracks from one child, it evaluates if the other child has to be explored by comparing the current NN distance



(a) 2D metanode.



(b) 3D metanode.

Figure 3.8: Unwinding path of the algorithm inside a metanode.

to the split coordinate of the current node. However, when the algorithm backtracks from a child metanode to its parent metanode, there is not a single other child that has to be possibly explored. There are 3 other children in 2D, and 7 in 3D. The Unwind procedure determines which one of these the algorithm needs to explore. Figure 3.8 will help understand the process with an example where we start from metanode M_4 in 2D and M_8 in 3D (in red). When in 2D, during backtracking, the following process occurs.

1. We can see the Unwind method takes as input parameters an integer called childIndex, which is computed on line 11 of Algorithm 8. This is the difference between the pointer of the metanode we are leaving (metanode M_4 in Figure

Algorithm 9 Unwinding the inner tree of the metanode

```
1: GlobalVariable dimension
2:
3: When dimension = 2
4: procedure UNWIND<2>(Point A, Metanode M, Point NN, double NNDistance, int childindix)
5:     yindix = 1 + childIndix >> 1
6:     double result = Walk<1>(yIndix, A, M, NN, NNDistance, childIndix)   ▷ Direct y parent node
7:     result = Walk<0>(A, M, NN, NNDistance, childIndix)   ▷ We move up to the x parent node
8:     if (result < NNDistance ) then   ▷ We go to the other y node only if the x split is close enough
9:         farYIndix = yIndix ^3
10:        Walk<1>(farYIndix, A, M, NN, NNDistance, childIndix)
11:
12: When dimension = 3
13: procedure UNWIND<3>(Point A, Metanode M, Point NN, double NNDistance, int childIndix)
14:    zIndix = 3 + childIndix >> 1   ▷ Since the z nodes are indexed from 4 to 7
15:    Walk<2>(zIndix, A, M, NN, NNDistance, childIndix)   ▷ Direct z parent node
16:    yIndix = 1 + childIndix >> 2   ▷ Since the y nodes are two rows above
17:    double result = Walk<1>(yIndix, A, M, NN, NNDistance, childIndix)   ▷ Direct y parent node
18:    if (result < NNDistance ) then   ▷ We go to the other z node only if the y split is close enough
19:        farZIndix = 3 + (zIndix >> 1) ^1
20:        Walk<2>(farYIndix, A, M, NN, NNDistance, childIndix)
21:    result = Walk<0>(0, A, M, NN, NNDistance, childIndix) ▷ We move up to the x parent node
22:    if (result < NNDistance ) then   ▷ We go to the other y node only if the x split is close enough
23:        zIndixBis = 3 + (childIndix >> 1) ^2
24:        Walk<2>(zIndixBis, A, M, NN, NNDistance, childIndix)
25:        farYIndix = yIndix ^3
26:        Walk<1>(farYIndix, A, M, NN, NNDistance, childIndix)
27:        farZIndixBis = 3 + (childIndix >> 1) ^3
28:        Walk<2>(zIndixBis, A, M, NN, NNDistance, childIndix)
```

3.8a) and the pointer of the first child of the metanode we are in (metanode M_1 in the same example). Since all children metanodes of a given metanode are contiguously stored in memory, this pointer difference will return 3. From this the algorithm deduces the origin node is located three memory spaces away from M_1. This indicates the origin node is M_4.

2. We determine which of the two y-based inner nodes the M_4 metanode is a direct child of. For this we compute

$$\text{yIndex} = 1 + (\text{childIndex} >> 1) \tag{3.1}$$

Bitshifting childIndex to the right amounts to dividing it by 2 and taking the floor value, but is faster to execute. The yIndex value is 1 from metanodes M_1 and M_2 and 2 from metanodes M_3 and M_4. We can then access either the Y_0 or the Y_1 inner nodes, since they are stored respectively at the 1th and 2nd index of M_0's list of inner nodes. For M_4 we will reach the Y_1 inner node. This is noted arrow 1 in Figure 3.8.

3. We initiate the Walk<1> procedure on that node. This procedure is shown on Algorithm 10. The <1> means it is called for the 1st layer of the inner tree, that is the y layer (the x layer being the 0th). It evaluates point P_2 stored in the node for NN candidate. Then, it checks whether the current node is standing on the last layer of the current metanode; this check is made on line 36. In our case, the answer is yes. This means the current node's children are two metanodes rather than two inner nodes of the current metanode. As a result the procedure will call the Find recursive search procedure (detailed later) on the children, as we are entering here the recursive part of the whole algorithm. Walk calls the recursion on either on the left child first or the right child first of the Y_1 inner node, depending on the sign of $A.y - Y_1$. To get the pointers we need, we compute

$$\text{nextChildIndex} = (\text{yIndex} - 1) >> 1 + A.y > Y_1 \tag{3.2}$$

We call Find on that child. Then, we compute the XOR bitwise operation on nextChildIndex and 1:

$$\text{nextNextChildIndex} = \text{nextChildIndex}\hat{}1 \tag{3.3}$$

This returns nextNextChildIndex, the index of the other child to be explored, and we call Find on it.

We add a small caveat here: we check if the value of either nextChildIndex or nextNextChildIndex is equal to the value of childIndex. If it is, this means this child that should be explored is actually the one we just came from, so we skip it. In our example, if we have $A.y > Y_1$, then nextChildIndex returns 3 and nextNextChildIndex returns 2. So we first call Find on M_0.child[3], that is, we call Find on M_4 (but this is cancelled since that is the node we come from) and then on M_0.child[2], that is, on M_3. This is red arrow 2.

4. Once the Find recursive procedures are over, the Unwind procedure is not finished. Similarly to the backtracking process in Algorithm 7 shown in Figure 3.7, we climb up the next inner node of M_0: this is the X_0 node. It is red arrow 3. The Walk<0> procedure checks if point P_0 is a better NN candidate. Then, since the 0th layer of M_0's inner tree is not the last one, Walk<0> exits after having returned the distance from A to axis X_0. After this, similarly to the culling process in the standard KNN algorithm, we compare the distance from A to split plane X_0 to our current NN distance. If the NN distance is smaller, we have seen that means we can cull the Y_0 node and all its subtree from exploration. Then, the Unwind procedure is over in that metanode, and we pop the next one from the stack. If it is bigger, we go down to the Y_0 node. It is red arrow 4?. The ? indicates this descent might or might not take place, depending on the result of the culling check.

5. We call Walk<1> on that Y_0 node: it evaluates point P_1 and we call Find on children M_1 and M_2 in the order given by nextChildIndex and nextNextChildIndex. These are arrows "5,6" and "6,5". They are noted this way since sometimes the left child will be called first (and then be the 5th step of the algorithm) and the right child second (thus being the 6th step), and sometimes that will be the other way around. Then the Unwind procedure is done in metanode M_0.

The Unwind procedure in 3D can be seen in Figure 3.8b. It works as follows:

1. The procedure starts from metanode M_8. It reaches its parent inner node which is the Z_3 node. This is red arrow 1. In order to retrieve the indix of that node in metanode M_0's list of inner nodes, we compute

$$\text{zIndix} = 3 + (\text{childIndex} >> 1) \tag{3.4}$$

We bitshift to the right childIndex, which is comprised between 0 and 7, that gives us a number between 0 and 4. We add 3 to get the relevant inner node index, since the z inner nodes are indexed from 3 to 6. We then call Walk<2> on that Z_3 node. Since we are on the last layer of the inner tree, that means

children of the Z_3 inner node are other metanodes. One of these two children being our starting M_8 metanode, we call Find on the other, which is metanode M_7. This is arrow 2.

2. We move up to the parent y node. This is arrow 3. To access its inner node index we compute

$$\text{yIndix} = 1 + (\text{childIndex} >> 2) \tag{3.5}$$

Bitshifting two times to the right means childIndex gives a result from 0 to 1, to which we add 1. We call Walk<1> here and store the result: it returns distance from A to plane Y_1.

3. If that is smaller than the current NN distance, then Unwind moves down to the Z_3 inner node's far brother. The index is computed by

$$\text{farZIndix} = 3 + (\text{childIndex}\hat{}1) \tag{3.6}$$

In our example, farZIndix returns 5. We thus go to the 5th inner node of the list, which is the Z_2 inner node. This is arrow 4?. We execute Walk<2> and subsequently Find on the two children of that node, in an order depending on A. These are the arrows "5,6" and "6,5".

4. Once that is done we move up to the x inner node and call Walk<0>. This is arrow 7.

5. If Walk<0> returns a distance between A and X_0 that is greater than the current NN distance, Unwind is over. If not, we explore the other side of the metanode. We will have to explore all three nodes here. We directly go to the inner node which is closest to point A. Since we know point A is contained into the Z_3 inner node, it means its y coordinate is greater than Y_1. Then, a very simple heuristics is to consider its y coordinate will then also be greater than Y_0. We don't want to make a real comparison, since we want the Unwind procedure to be as fast as possible, and to only proceed with bit shifts if able. Thus:

$$\text{zIndexBis} = 3 + (\text{childIndex} >> 1)\hat{}1 \tag{3.7}$$

On the example, zIndexBis returns 4 so we move to the Z_1 inner node. This is the "8?" arrow.

6. We proceed to Walk<2> here, which entails calling Find on the metanodes M_3 and M_4. These are the "9,10" and "10,9" arrows.

7. We compute the index of the far y inner node to be accessed:

$$\text{farYIndex} = \text{childIndex}\hat{}3 \tag{3.8}$$

Here it returns 1, we move to the Y_0 inner node (it is arrow 11).

8. We execute Walk<1> here, and we then compute the index of the last inner node to be processed:

$$\text{farZIndexBis} = 3 + (\text{childIndex} >> 1)\hat{}3 \tag{3.9}$$

We thus go to the Z_0 inner node. This is arrow 12. We call Walk<2> here and then Find on children metanodes M_1 and M_2. These are arrows "13,14" and "14,13". The Unwind procedure is then over.

Algorithm 10 Walking down the subtrees

```
1: GlobalVariable dimension
2:
3: procedure WALK<K>(int dimIndix, Point A, Metanode M, Point NN, double
   NNDistance, int childIndix)
4:     Point N = M[dimIndix] ▷ We retrieve the point stored in the relevant inner
   node
5:     double dist = distance(A,N)
6:     if (dist < NNDistance) then
7:         NNDistance = dist
8:         NN = N
9:     if (k+1=dimension) then          ▷ If we are walking at the last layer of the
   metanode...
10:        int offset = (1 << (dimension -1)) - 1
11:        int nearChildIndex = ((dimIndix - offset) << 1) + (NN[k] < N[k])
12:        if (nearChildIndex != childIndix ) then ▷ Don't go back in the child we
   came from
13:            Find(A, M.child[nearChildIndex], NN, NNDistance)
14:        int farChildIndex = nearChildIndex ^1
15:        if (farChildIndex != childIndix ) then
16:            Find(A, M.child[farChildIndex], NN, NNDistance)
       return dist
```

A consequence of this semi-stack-based approach and the use of the metanode structure means Find, the recursive search procedure we call, is a little different from the one shown on Algorithm 7. When we recursively travel down the tree to explore all neighbouring metanodes, we want an approach that allows us to discard too far away metanodes and their subtrees without having to deal with the inner structure of the metanodes. For this we introduce the *inner axis-aligned bounding box* (inner AABB). It is defined, for each metanode, by the smallest axis-aligned rectangle that fully encloses all the points of the metanode. It is always included

inside the regular AABB of the metanode as defined in Figure 3.3. The AABB of a metanode is the AABB of the topmost inner node of its inner structure. The inner AABBs are displayed in Figure 3.9.

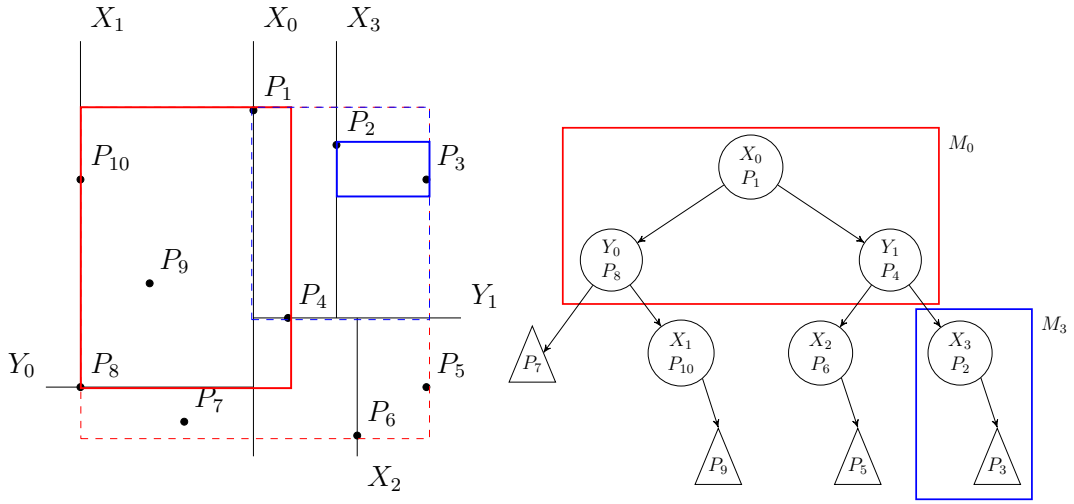


Figure 3.9: Inner and regular AABBs of metanodes M_0 and M_4. The inner AABBs are drawn in solid line and the regular AABBs in dashed line.

The Find procedures uses the inner AABBs the following way. When the procedure enters a metanode, it computes the distance between point A and the metanode's inner AABB. If this is bigger than the current best NN distance, then the procedure ends. This is similar to the process in the standard algorithm that computes distance to the regular AABBs, but the check is made when the procedure enters a node, instead of when it is about to exit one. Inner AABBs are much tighter than regular AABBS, as can be seen in Figure 3.9, though they contain the exact same amount of information. Filtering through inner AABBs rather than regular AABB thus saves us from needlessy entering metanodes the regular AABBs of which stand close to point A even when the points they contain do not. The inner AABB of a given metanode is, as the regular AABB, stored as a vector of two points standing for the corner of minimum coordinates and the corner of maximum coordinates. The Find procedure is detailed on Algorithm 11.

3.3.1.4 Kd-tree traversal by ray tracing

In order to check only the facets we need for obstruction among the chosen tree, we will traverse it through ray tracing. Ray tracing through an array of facets sorted into a tree is a classical process in radiosity computation for thermal applications [8] and image rendering [26]. Hapala and Havran [27] or Lira dos Santos et al. [28] have written reviews of ray tracing algorithms specifically designed for kd-tree

Algorithm 11 Recursive travel down procedure using inner AABBs

```
1: GlobalVariable dimension k
2: procedure FIND(Point A, Metanode M, Point NN, double NNDistance)
3:     if (M == NULL) then
           return
4:     for (int i from 1 to k) do
5:         if (A[i]+NNDistance<M.innerAABB[0][i] or A[i] >M.innerAABB[0][i]+NNDistance ) then
               return    ▷ A is too far from M's inner AABB by comparing axis by axis
6:     for (int i from 1 to 2^k) do
7:         Find(A, M.c[i], NN, NNDistance) ▷ We call the recursion on all children
8:     NN, NNDistance = CompareBestNeighbour(A, M)  ▷ Examine all points in M for NN potential
```

traversal. The former are more focused about CPU implementation while the latter detail GPU implementations.

Sequential traversal The simplest ray tracing through kd-trees algorithm has been devised by Kaplan [29] into what is known as the sequential traversal algorithm. Wald [15] has considerably honed this algorithm by proceeding to a lot of low-level optimizations, without changing its principle. We recall here that the kd-tree is used to organize the set of the barycentres of the radiating facets.

For each considered pair of facets, we shoot a ray starting from one facet and aimed at the other one. We create the ray at the barycentre of the starting facet. We then draw it towards the barycentre of the receiving facet. We check in which node's AABB the ray starting point currently lies. To do so, we simply locate the point in the tree as per Algorithm 3. We compute collision between the ray trajectory and the facets that are inside that node. If we are using a tree with points also stored in the interior nodes, we check for obstruction at each of these interior nodes we encounter while locating the point in the tree. If it turns out no facet collides with the ray in the current node, we extend this ray until it has completely gone through that node's AABB. We do so by "sliding" the starting point of the ray along the ray trajectory we have computed betwen the two facets. The endpoint remains the same. This means our ray becomes progressively shorter and shorter as we near the end facet. Our ray's starting point is now located into a different AABB corresponding to a different leaf node. In order to know which leaf node this is, we simply locate the starting point into the tree once again. We then check for obstruction with the facets contained into that leaf node. We carry on until we reach the node containing the receiving facet.

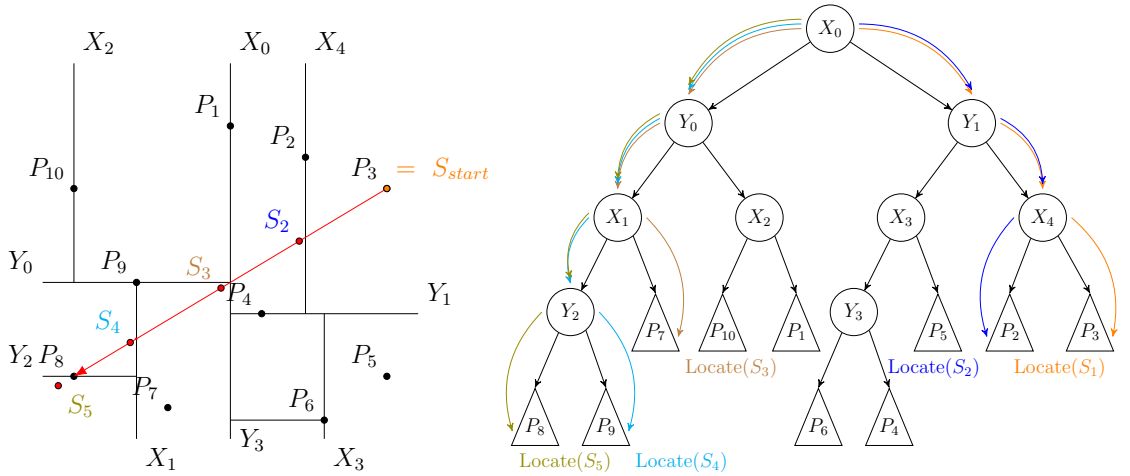


Figure 3.10: Sequential ray tracing from facet of barycentre P_3 to facet of barycentre P_5.

Let us see a step by step example to better understand the way it works, with the kd-tree containing all points in the leaves shown Figure 3.1. The facets are not represented: in order to make the figure clearer, we assume they are very small triangles compared to the distances between them, so we approximate them as their barycentres. Figure 3.10 shows obstruction being calculated between the facet of barycentre P_3 and the facet of barycentre P_8. A ray is traced between these barycentres. The starting point of the ray, S_{start}, is defined as being equal to point P_3. The ray length is computed and stored.

1. At first the starting point of the ray, S_{start}, is point P_3 itself. We locate that point in the tree: this is the orange arrow path in Figure 3.10. We find it is located in the P_3 leaf node. This node only contains point P_3 itself. We compute whether the facet of point P_3 is in the path of the ray. Of course, the starting facet of a ray should never be considered to obstruct that very same ray. A trick that can be used to avoid this useless intersection check is to slide the starting point of the ray ϵ away from point P_3. Then the starting facet will always be computed as being located *behind* the ray's starting point. This means the obstruction check always returns that the starting facet does not intersect the ray. This trick means the algorithm processes that facet exactly like any other, without the need for a special case in the code that would add indirection overhead.

2. All facets of the node have been checked. We move the starting point of the ray to the intersection between the ray and the P_3 node's AABB; and then just an ϵ further along the ray trajectory. We update the ray length accordingly. The new starting point of the ray, S_2, is now fully contained inside the next

leaf node's AABB.

3. We locate the starting point. We find we are now located in the P_2 leaf node. This is the blue arrow path in Figure 3.10. This node only contains the facet having P_2 as its barycentre. We verify if that facet is in the way of the ray. It is not. We have now evaluated all the facets of this node. We move the starting point an ϵ into the next AABB and update ray length.

4. We locate again the starting point (brown arrow path in Figure 3.10), S_3, we are in the P_7 leaf node. The P_7 facet does not intersect, we carry on, we move to the next AABB and update ray length.

5. We have reached the P_9 node with starting point S_4 (cyan arrow path in Figure 3.10). The P_9 facet does not intersect, we move to the next AABB and update ray length.

6. We locate the starting point (olive arrow path in Figure 3.10). We are now in the P_8 leaf node. However, this time the ray length check indicates the current length is negative. This means we have travelled all the way up to the arrival facet. The ray traversal then stops before checking intersection with the facets of the P_8 leaf node. This is wanted, since otherwise it would have computed intersection with the arrival facet, and that should always return false.

7. We have reached the end of the ray. No obstructions have been found.

The pseudocode for this algorithm is indicated in Algorithm 12. It also handles the case of a ray being shot from outside the tree. This can happen in image rendering when the ray correspond to a beam of light originating outside the scene to be rendered. To this end, after the ray is created, we check at which points it intersects with the AABB of the root node. The outputs are given in parametric coordinates relatively to the ray's starting and ending facets. If the ray does not actually intersect the tree, there are no such points, and we can declare at once there is no intersection. If the ray origin is contained in the tree's AABB, the entry point of the ray in the AABB will have a negative parametric coordinate, and we use the barycentre of that facet as the ray starting point as normal. But if the ray origin is outside the tree's AABB, then the entry point of the ray into the tree's AABB will have a parametric coordinate between 0 and 1, and we will take *that* point as the ray's starting point inside the tree.

This algorithm's main issue is that the tree is ran through from top to bottom, each time the ray enters a new leaf node. This means some interior nodes will be visited many times: for example, the root node is visited as many times as there are leaf nodes encountered. This is easily seen in Figure 3.10. And in the case of

Algorithm 12 Sequential algorithm for kd-tree traversal

```
1: procedure TREE TRAVERSAL(node RootNode, ray)
2:     bool obstruction = false
3:     entry, exit = RayBoxIntersection(ray, RootNode.AABB)
4:     if (entry, exit == null, null) then
        return obstruction
5:     if (entry < 0) then
6:         origin = ray.origin
7:     else
8:         origin = ray.origin + entry*ray.direction + epsilon
9:     currentNode = Locate(origin)
10:    while (currentNode.leaf == true) do
11:        entry, exit = RayBoxIntersection(ray, currentNode.AABB)
12:        ray.length -= exit
13:        if ray.length =< 0 then return obstruction
14:        for (Facet in currentNode) do
15:            if (Facet before entry or after exit) then
16:                continue    ▷ Do not check facets before ray start or after ray end
17:            obstruction = RayFacetIntersection(ray,Facet)
18:            if (obstruction) then
         return obstruction
19:            origin = ray.origin + exit*ray.direction + epsilon
20:            currentNode = Locate(origin)
       return obstruction
```

kd-trees storing points in the interior nodes, an additional structure needs to be set up to ensure the algorithm does not check for ray-facet intersection more than once for the facets contained in such often visited interior nodes. If the ray's origin and end/first obstruction are separated by a small number of leaf nodes, which statistically happens often enough in realistic image-rendering situations [30], then the complexity depends on the time it takes for the algorithm to make a few tree descents. This means the complexity is $O(\log(N))$. At worst, however, the origin and the end of the ray are far away, and the algorithm will need to make $O(N)$ descents before finding the leaf node containing the end of the ray (since there are $O(N)$ leaf nodes given that each leaf node stores a set number of facets). This means in the worst case the complexity degrades into $O(N \log(N))$.

Related work Subsequent version of the ray tracing algorithm strive to diminish this number of redundant passes. We will present a number of them here: standard sequential, stackless, backtrack, restart, and ropes. Janssen [31] has devised a stack-based ray traversal algorithm that visits each node of the tree that is on the ray path exactly once. His algorithm has been made more robust by Havran [30] into the so-called "standard sequential algorithm", whose improvements allow the algorithm to gain up to 65% efficiency. They search through the tree to locate all the leaf nodes that are intersected by the ray. When an interior node is found to be intersected by the ray, a check is made to determine which of its two children contains the ray. In case both of the children do, the child containing the ray's entry point into that node is explored first, and the child containing the ray's exit node is pushed into a stack. The search goes down recursively until it reaches a leaf, evaluates all facets into that leaf against the ray, and then pops the first node of the stack and proceeds to recursively go down from that node. This prevents any multiple traversal of the same node. The process can be seen in Figure 3.11a. This algorithm can not degrade into a complexity worse than $O(N)$, since in the worst case, every leaf and interior node will need to be visited, but never more than once each.

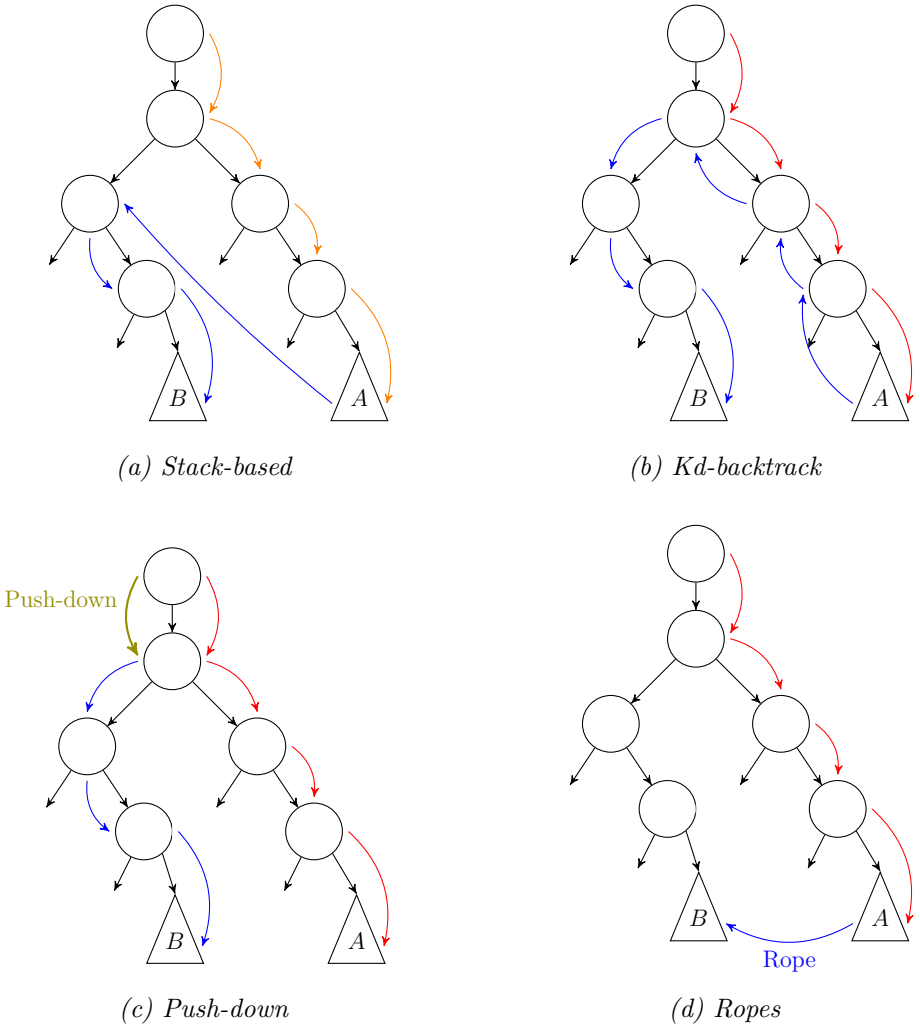
Foley and Sugerman [32] wanted to adapt the algorithm on GPU, and to accomodate GPU's limited memory availability compared to CPU, removed the stack from Havran's algorithm and essentially went back to Janssen's version in what they call the "stackless algorithm". They have then developped another version they called "kd-backtrack", in which when a leaf node has been attained, the algorithm then hops back up in the tree until it finds a node having a yet unexplored child. When the algorithm reaches a node's parent, it checks the node's "brother", that is, the other child node of that parent, to see if the ray intersects its AABB. If it does, it then goes down into that child until it reaches a leaf again, etc. This algorithm ensures each node is, at worst, visited two times, so the worst case complexity of the procedure is $O(N)$. Moreover, no stack is needed, which alleviates memory con-

sumption during processing. However, each node now needs to store a pointer to its parent, as well as its AABB. This means more memory is needed to store the tree. Figure 3.11b shows their algorithm.

Horn et al. [33] have gone back to Kaplan's algorithm and enhanced it through a process they call "push-down". Their algorithm progressively tracks the ray's current position leaf node after leaf node just like the original sequential algorithm does. However, for each tree search when trying to locate the ray's current position, they do not start back from the root node. Instead, they keep track of the lowest possible node that has children such that the ray crosses only one of the two children. Such a node is updated during each tree descent. Since the ray is fully contained within only one child of that node, this defines a subtree that entirely contains the ray. The node that contains this subtree is the effective root node from which all tree descents need to be performed. Each time the ray's next current position needs to be located in the tree, the algorithm initiates the locating procedure directly starting from this node. Figure 3.11c indicates that process: the tree descent is started from the second node, rather that the root node, after the push-down process shown by the olive arrow. This diminishes the number of redundant node passes from the standard sequential algorithm. They have also implemented a variant of the node stack called the "short-stack". They use a stack of to-be-visited nodes exactly as per Havran's algoithm. But instead of maintaining a stack the size of the tree depth, they keep a small, circular stack of fixed size. If a node were to be pushed on it while the stack is full, the node at the bottom of the stack gets flushed. If the algorithm ever needs to pop a node from the stack while it is empty, a full tree traversal process is called (including their "push-down" enhancement). This allows their short stack to work as a makeshift memory cache the algorithm can access, and it diminishes the number of tree traversals needed.

Popov et al. [34] have devised a rather different solution. In their "ropes" algorithm, the kd-tree is first built. Then, on a second pass, for each leaf node, the "ropes" algorithm determines which are the other leaf nodes the AABB of which are adjacent to the current node's AABB. Then, that node receives as local data a pointer (the "rope") to each of its neighbouring leaf nodes. When tree traversal occurs, when the ray is about to leave a given leaf node, the algorithm computes the exit point of the ray from that leaf node's AABB. Using the coordinates of that point, the algorithm then determines towards which of the neighbouring leaf nodes the ray is about to go, and simply retrieves the corresponding pointer from the leaf node's structure. Figure 3.11d indicates that traversal with the rope pointer from point A node to point B node. The ray can thus travel from leaf node to leaf node, and there is no need to ever proceed to a full tree traversal apart from the first one needed to locate the ray origin. This ray tracing algorithm visits the fewest nodes of all the versions presented here. However, the tree itself becomes three times heavier

than the standard kd-tree, since each leaf node must store six additional pointers. Besides, the tree construction algorithm becomes longer as well as a consequence. Lira dos Santos et al. [28] have further upgraded the algorithm into what they call the "ropes++" algorithm by noticing the computation of which neighbouring AABB the ray is about to enter does not need the exact computation of the ray exit point from the current node's AABB.



(a) Stack-based *(b) Kd-backtrack*

(c) Push-down *(d) Ropes*

Figure 3.11: Sequential ray tracing from facet of barycentre A to facet of barycentre B for various ray tracing algorithms.

Metanode-adapted kd-backtrack In our case, since the points are not exclusively contained in the leaves, we cannot make use of a ropes-type algorithm. This is why we have chosen to implement a back-track-like algorithm, adapted to our

metanode structure. The locate procedure displayed on Algorithm 13 is performed efficiently inside a metanode: we have devised a branchless algorithm that, given a point and a pointer to a metanode as inputs, returns a pointer to the child of the metanode into which the input point is located. To perform a full locate operation, we keep calling Algorithm 13 onto itself as long as the input node is not a leaf node, and that is a fully branchless operation. This helps our algorithm, since we have seen kd-backtrack makes a lot of travels (up and then) down through the tree.

Algorithm 13 Branchless locate in metanode algorithm

```
1:  GlobalVariable N = dimension
2:  procedure MetanodeLocate(Point A, *Metanode M)
3:      int i = 0
4:      int k = 0      ▷ Stores the offset of the current inner node from the leftmost node of its layer
5:      bool s = 0  ▷ Is 0 if point A should go to the left of current inner node, 1 if to the right
6:      *Node n = M→ nodes  ▷ We retrieve the first node of the metanode's inner node list
7:      for (int d from 0 to N) do                    ▷ Each pass is for 1 coordinate
8:          s = n[k][d]<A[d]
9:          i = i << 1 +s  ▷ Bitshift: i is doubled, so goes to next layer of the inner tree
10:         k = k << 1 +s
11:         n += 1 << d      ▷ n jumps to the leftmost node of the next inner layer
        return M→children[i]
```

When collision has to be checked between a ray and a facet, we use the efficient algorithm developed by Möller and Trumbore [35]. It computes the barycentric coordinates of the intersection point between the ray and the facet plane relatively to the facet vertices coordinates, one by one. As soon as one of them is not comprised between 0 and 1, it returns a miss.

The metanodes's addresses are stored in a vector of pointers. The zeroth layer of the tree (that is, the root node) occupies the space of indix 0 in the vector. The first layer occupies the spaces of index 1 to 4 (in 2D) or 1 to 8 (in 3D). The k-th layer thus occupies the spaces of index $2^{2(k-1)+1}$ to 2^{2k}. This means all the addresses are stored contiguously in memory, so that the many calls to node's parent or node's child during the tree traversal do not generate too much overhead. Algorithm 14 details this process.

Algorithm 14 Metanode-adapted kd-backtrack

```
1: procedure NODE COLLISION(ray, node)
2:     Obstruction = false
3:     while (Obstruction is false) do
4:         for (facet in node) do
5:             Obstruction = RayFacetIntersection(ray, facet)
       return Obstruction
6:
7: procedure TREE TRAVERSAL(ray, node)
8:     (starting point, endpoint) = intersection(ray, node)
9:     if (starting point is null) then
10:        if (node is root) then
        return false                                  ▷ The ray is exiting the tree
11:        else
12:            Tree traversal(ray, node.parent)      ▷ The ray backtracks up the tree
13:    Obstruction = Node collision(ray, node)
14:    if (Obstruction is true) then
        return Obstruction
15:    if (node is not a leaf) then
16:        child = MetanodeLocate(starting point, node)
17:        Tree traversal(ray, child)  ▷ The ray climbs down the tree, for 1 layer at
   a time
18:    else
19:        if (endpoint - starting point > ray.length) then return Obstruction  ▷
   The ray terminates within this leaf
20:        Update ray origin(ray, endpoint + ray.direction*epsilon)  ▷ The ray is
   pushed a little inside the next node
        return Tree traversal(ray, node.parent)  ▷ The ray backtracks up again
```

Performance evaluation We have stated at the beginning of this chapter that with the previous, naive obstruction method, processing a pure radiation test case between two concentric spheres totalling 4452 facets amounted to 19 hours of serial single-core computation. With our kd-tree, repeating the same test case led to a computation time of 3 minutes. Obviously, this is a great step forward, and more in line with the expected use for the proposed solver. However, how does it compare to the other existing kd-tree ray tracers in the literature? A lot of modern literature like Lira dos Santos et al. [28] or Li et al. [18] refers to GPU-based kd-trees, to which we cannot compare ours since it is CPU-based. We can refer to Wald's thesis [15] where he indicates a useful metric for his standard traversal algorithm: the number of rays launched per second by the algorithm. The total number of rays that needs to be launched to compute the obstruction matrix of a given problem solely depends on the meshing of the geometry. Each visible pair of facets needs one ray tracing from one facet to the other. Local half-space culling as seen on Subsection 2.1 means we trace a ray only for facet pairs in the sparsity pattern of the view factor matrix. Starting from N^2 facet pairs, we end up with dN^2 where d is the density of the matrix. A high number of launched rays per second either means the algorithm is very quick to trace a given ray through the tree (as with Wald's standard traversal algorithm), or that it can terminate the ray after a small number of steps (as is the case for the ropes algorithm), or both. Besides, this metric tends to vary within no more than an order of magnitude from one geometry to the other, meaning it is relatively consistent across all geometries.

Wald's algorithm achieves $5.82\,\mathrm{Mrays\,s^{-1}}$ to render a 84k facets scene, while our algorithm achieves $0.04\,\mathrm{Mrays\,s^{-1}}$ to render a 52k facets scene. This is without accounting for the performance difference between the processors and the compilers used. Wald has indicated using a 2002 Intel Pentium 4 CPU, 2.2 GHz, and we have not found a direct performance comparison to our 2015 Intel core I6-6700, 3.4 GHz; but it can only set the performance comparison further back against us.

This tells us our kd-tree, while using clever structures like the metanodes, is still far from being fully optimized with regard to the state-of-the-art. We suspect not making use of a SAH imposes a severe penalty on the performance of the kd-tree construction. The expected gain in time by the use of the metanode structure clearly does not compensate the lack of flexibility of choosing the optimal split plane at each step of tree construction. In addition, our ray traversal could be improved a lot too: we have no means of avoiding checking the same nodes several times as the algorithm goes up and down, and its recursive formlation is cumbersome.

All things considered, given the limited amount of time available to develop the structure, the metanode-based kd-tree is a tentative step forward that could be the subject of a lot of improvement. Our relatively simple and sturdy kd-tree structure and ray traversal algorithms are much faster than the previous naive approach, and

in that regard they have achieved our aim of making the simulation usable on a reasonable time scale for industrial purposes.

3.3.2 Bounding volume kd-trees

Despite the performance improvement of the metanode kd-tree, the structure has proved too unwieldy for our immediate needs. The speedup seemed to come from the "kd-tree" part, rather than from the "metanode" part in spite of our initial assessments. Thus we decided to build a cleaner kd-tree structure from scratch, without using the metanode structure. This part will detail some possible refinements and variants of the methods used both for generation and tree traversal. Due to time constraints, we have not incorporated in our code many of these upgrades. This will be the focus of ulterior work.

Bounding volumes We also decided to handle facets by encasing them into AABBs that are defined by bounding all vertices of the facets. This is an idea called the Bounding Volume (BV) technique to accelerate ray tracing. In our case, the BV of each facet will be exactly equivalent to the inner AABB of a node containing all the vertices of said facet, as shown in Figure 3.9. Ray-AABB intersection algorithms are much simpler and faster than general ray-triangle or ray-segment intersections. The idea is thus to generate a binary tree of bounding volumes recursively containing one another. With such a Bounding Volume Hierarchy (BVH), the ray is launched through the BHV. When it has been located in a leaf BV and the ray-AABB collision returns true, only then the real ray-facet collision algorithm with the facet contained inside the AABB is launched. This technique allows us to compute the more costly ray-facet collision algorithm only when it is really needed. The BHV structure was first devised by Rubin and Whitted [36]. It is actually very close to the kd-tree structure: it partitions objects instead of partitioning space, but produces a binary tree with similar properties. One of the differences is that two leaves of a BHV can partially overlap if they each contain AABBs of two objects standing close to each other. Meister et al. [6] provide a very thorough survey of the latest (as of 2021) techniques and methods related to ray tracing in BHVs. Here we decided to implement an even simpler version of the BVH. Taking a list of radiating facets as input, we create a BV for each of them. After this, however, we will operate for tree construction not on the BVs themselves, but on their barycentres. This means we are once again tasked with sorting a list of points. Compared to the previous kd-tree, we sort AABB barycentres instead of facet barycentres, though in general for a given facet these two points are located very close to each other. The end result is that we obtain a sort of hybrid kd-tree using BVs. Henceforth it will be called the *BV kd-tree*.

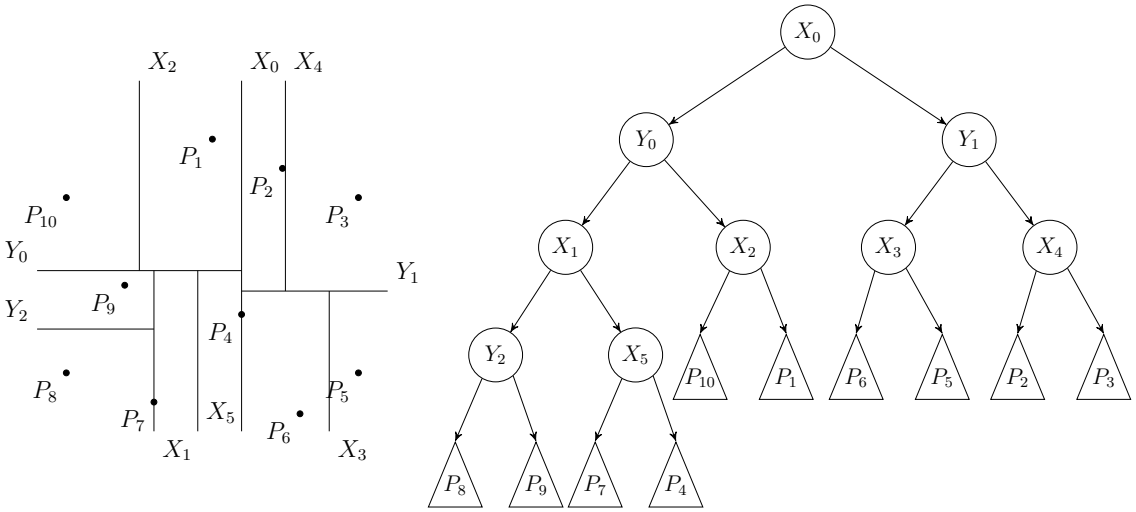


Figure 3.12: BV kd-tree. On the left, the spatial repartition of points and the $x-$ and $y-$ split values chosen. On the right, the binary representation of the tree. Data is stored in the leaves.

Points are not stored in the inner nodes of the subtrees, as was shown in Figure 3.2, but are automatically stored in the leaves similarly to Figure 3.1. Figure 3.12 shows the same dataset as in Figure 3.1, but sorted into a BV kd-tree tree instead of a metanode kd-tree. We have seen that a metanode kd-tree will always be a balanced tree, but the subspaces it splits can be of very different sizes, and sometimes concentrated into a single region of space if the points are not evenly distributed. On the contrary, a segment tree will ensure space will be split into subspaces of regular volume and that evenly partition the total space. But the tree can be inbalanced if the points are all concentrated into a region of the space. BV trees have already been investigated for the purpose of ray tracing in general and for obstruction computation in radiative transfer applications in particular [9].

Design choices For the BV kd-tree we made the following choices for construction. Table 3.2 indicates the differences between these choices and those we made for the metanode kd-tree.

- The points are now only stored in the leaves. The metanode kd-tree with its points stored in the inner nodes presents two main differences in behaviour: it is on average one layer shallower than the BV kd-tree (a positive performance factor), and its ray tracing algorithm makes multiple redundant ray-facet intersection checks in these inner nodes (a negative performance factor). We es-

Binary tree	Metanode kd-tree	BV kd-tree
Facet representation point	Facet's barycentre	Facet's AABB's barycentre
Position of points	In inner nodes and leaves (1 per leaf)	In leaves (1 per leaf)
Termination criterion	As soon as the last metanode layer is not full	When a node contains a single point
Split coordinate choice	Periodically x then y then z (within a metanode)	Longest coordinate of the current node's AABB
Split plane choice	Plane containing point standing at median of point coordinates	Plane containing barycentre of current node's AABB

Table 3.2: Construction choices for metanode kd-tree and BV kd-tree.

timate the net effect of these two differences is that the "inner nodes+leaves" structure is on the whole less efficient than the "only leaves" structure.

- For the termination criterion, since there is no metanode structure anymore, we went back to the simplest possible criterion which is to stop when the last node only contains a single point.
- The choice of rotating through the coordinates in order was needed in order to build up the metanode structure. However, we suspect this unwieldy process has been an overall detrimental factor to performance. Hence we chose the simplest possible heuristics which is to split on the longest coordinate of the current AABB.
- The split plane chosen for the metanode kd-tree was that containing the point being stored in the current inner node. Since there is no point stored in the inner node anymore we are not constrained to choosing a split plane containing a point. We just take the split plane cutting the AABB in half for the chosen coordinate.

Choosing, as the split coordinate, the longest coordinate of the current node's AABB means we will sometimes pick the same coordinate twice in a row. This can be shown in Figure 3.12 where on the third row, another x split (split X_5) is the child of the split X_1. The node structure for the BV kd-tree we use contains the following attributes:

- A two-dimensional array storing the two opposite corners (bottom left and top right) of the inner AABB containing the node (as shown in Figure 3.9). We have chosen to use doubles for point coordinates. The first item of this array

is an array of size N storing the lower corner of the inner AABB while the second item is another array of size N storing the upper corner.

- An integer value called "rank" that is set to -1 if the node is an inner node. If it is a leaf node, then its value is positive. It serves as an indicator of the nature of the node as inner or leaf.
- An array of size two storing two pointers to other nodes. The first item stores the pointer to the left child of the current node, the second item the pointer to the right child.

We can see the node structure is lighter than for the metanode kd-tree. There is no need to store the pointer to the node's parent, nor do we store *both* the regular AABB and the inner AABB of the node. We do not store the split coordinate or the plane of the node. This means our BV kd-tree cannot be used to locate a given point in it, as per Algorithm 3, because we cannot determine into which child of a given node the point to be located should go. This is not a problem for ray tracing, however. In addition to this, the BV kd-tree itself has the following global variables:

- An integer storing the total number of leaf nodes contained in the tree
- A pointer to the root node
- An array of pointers to each node
- An array of pointers to each node's AABB
- A heap for intermediate computation during tree construction

3.3.2.1 Kd-tree generation

Generating our BV kd-tree is not too different from generating a kd-tree with the standard recursive algorithm shown on Algorithm 4. Since there is no metanode structure to handle here, we have taken the standard recursive generation algorithm, shown for example by Havran [30], and turned it from recursive to iterative by the use of a stack. The procedure undergoes the following steps.

1. A first pass through all the radiating facets is made to compute, for each facet, the tightest AABB that can include it. This amounts to computing the inner AABB that can contain all the vertices of the facet.
2. During this first pass we also create at once all the leaf nodes of the tree. For this, we create one node for each AABB, labelled as a leaf node. We update the "rank" attribute of each leaf node, so that it is equal to the index of the

facet in the facet array that has been passed as parameter to the Init function. As long as it is not equal to -1 anymore, the node is now labeled a leaf node.

3. We create a stack of node intervals. We push on that stack the complete node list we have generated at the previous step.

4. While the stack is not empty, we pop the first node interval. We create a new node which is a copy of the first node in the interval. We run through the node interval and for each node we retrieve the coordinates of the corners defining its AABB. We compare them to the AABB of the new node created. We update the AABB of the new node with the maximum (for its positive corner) and the minimum (for its negative corner) of the two sets of AABB coordinates being compared. This amounts to inflating the new node's AABB so that it can fully contain the current node's. At then end of the run, the new node's AABB contains the entirety of the node interval's AABBs.

5. We want to proceed to a bisection. As for the kd-trees, we choose a split coordinate according to the criterion shown in Table 3.2. We evaluate the length of the new, enclosing node's AABBs on each coordinate, and we pick the coordinate in which this length is maximum. Then, to actually determine the splitting plane's position we select the englobing AABB's barycentre's split coordinate value. For example, if the chosen split coordinate is y then we will pick the $y = y_{barycentre}$ as the splitting plane.

6. We sort the nodes in the node interval according to the value of their AABB's barycentre on the split coordinate. Nodes that have a corresponding value lower than that of the englobing node should be placed to the left; others to the right. In case this split would produce an ordering where no node is either on the left or on the right of the englobing node, we instead split by choosing the median node of the interval. This is just to avoid an edge case.

7. If on either side of the englobing node, only a single node remains, then that node is assigned as a child of the englobing node on that side. If two or more nodes remain, then the subinterval they define is pushed on the stack for later splitting.

The pseudocode for the tree generation is shown on Algorithm 15. Let us see through an example how the process is done. We will take the point distribution shown on the left of Figure 3.12 and indicate how we end up with the tree shown on the right of that same figure. The points are displayed again in Figure 3.13. We will suppose here that each point is actually the barycentre of a radiating facet. Since we are in 2D, facets are actually segments. These are the black lines shown in Figure 3.13.

Algorithm 15 BV kd-tree generation

```
1: procedure INITIATE(pointsNumber, listPoints)
2:     Initialize listAABB
3:     Initialize listNodes
4:     for (i in pointsNumber) do ▷ We create the AABBs for all points (Step 1)
5:         listAABB[i] = generateAABB(listPoints[i])
6:         listNodes[i] = new node
7:         listNodes[i].AABB = listAABB[i]
8:         listNodes[i].rank=i
9:     Initialize stackIntervals
10:    root = new node()
11:    firstInterval = subInterval(listNodes[0],listNodes[pointsNumber],root) ▷
   The first interval starts at the first point, ends at the last point and has root as
   parent
12:    stackIntervals.add(firstInterval)
13:    while (stackIntervals is not empty) do
14:        currInterval = stackIntervals.pop()
15:        firstNode, lastNode, parentNode = currInterval.first, currInterval.last,
   currInterval.parent
16:        parentNode.AABB = firstNode.AABB
17:        for (node in currInterval) do parentNode.AABB = en-
   globe(parentNode.AABB, node.AABB)
18:        splitPlane = getSplitPlane(parentNode)
19:        startPointer, endPointer = firstNode, lastNode
20:        while (startPointer <= endPointer) do
21:            if (startPointer and endPointer are both on the wrong side of split-
   Plane) then
22:                swap(*startPointer,*endPointer)
23:            if (startPointer is on the correct side of splitPlane) then
24:                startPointer++
25:            if (endPointer is on the correct side of splitPlane) then
26:                endPointer−−
27:        if (startPointer == firstNode ) then ▷ Only the lefmost node is on the
   left of the split plane
28:            parentNode.leftChild = firstNode
29:        else()
30:            parentNode.leftChild = new node()
31:            leftInterval = subInterval(firstNode,startPointer,parentNode.leftChild)
32:            stackIntervals.add(leftInterval)
33:        if (endPointer == lastNode ) then
34:            parentNode.rightChild = lastNode
35:        else()                                    131
36:            parentNode.rightChild = new node()
37:            rightInterval = subInterval(endPointer,lastNode,parentNode.rightChild)
38:            stackIntervals.add(rightInterval)
```

1. During Step 1, we compute the inner AABBs that encapsulate both vertices that define these segments: these are the red rectangles that are shown in Figure 3.13a.

2. During Step 2, we create one leaf node for each facet to be sorted, in the order in which the facets are presented to the tree. This is shown on the right of Figure 3.13b.

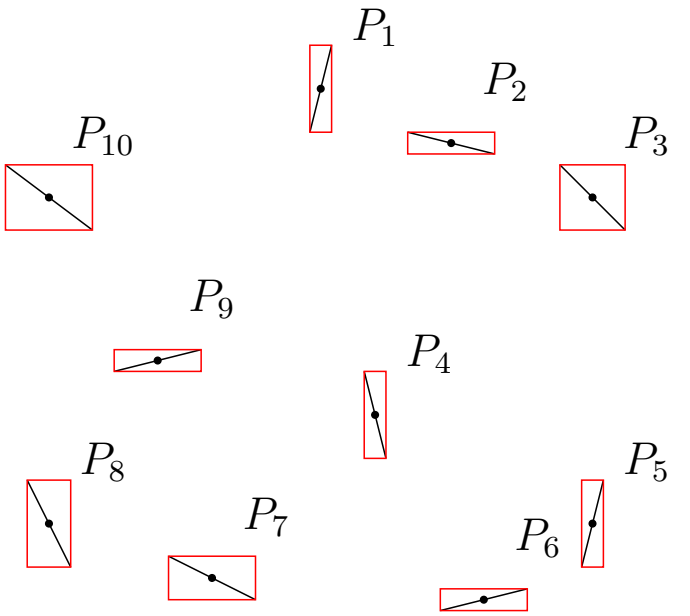
3. In Step 3, a stack is created to store the node intervals to be processed, and the current node interval we see in Figure 3.13b is pushed on it.

4. In Step 4, the first node interval of the stash is popped (in our example, we retrieve the node interval we just pushed). A new node is created: node N_0. It is a copy of the first node of the current interval: here, this is the node containing the P_1 point. Then, N_0's AABB is compared to the AABB of each node of the interval. Its AABB is resized to fit any AABB it encounters that was not fully contained within it before. Figure 3.14 shows this process with N_0's AABB getting bigger (in Figure 3.14a) as it runs through the node interval (in Figure 3.14b). At the end of the step, N_0's AABB fully contains all the AABBs of all the nodes of the interval.

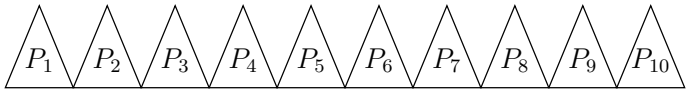
5. In Step 5 we compute the length of N_0's AABB according to each coordinate. We obtain $\Delta x = 5.7$ and $\Delta y = 5.2$. Our heuristics then tells us to choose x as the split coordinate. The x coordinates of the barycentre of the AABB (let us note this $\tilde{x}$) returns $\tilde{x} = 0.5$. Thus, this will be our split plane.

6. For Step 6, we now need to sort the nodes of the interval. We use a double sorting technique where we simultaneously start from the beginning and the end of the interval. We create an iterator starting at the P_1 node and another iterator starting at the P_{10} node. For each iterator, we evaluate the $\tilde{x}$ value of the underlying node. For the iterator that has started at the left of the interval, if the result is smaller than the split plane, that means this node is located in the correct half-plane. We then increment the iterator towards the middle of the interval. For the iterator that has started at the end, the reverse is true: we increment it towards the middle of the interval if the returned value is bigger than the split plane. When an iterator arrives at a node that is located in the wrong half-plane, it stops. The other iterator continues as normal. If later on the second iterator *also* stops because of a node located in the wrong half-plane, we swap them. This is shown in Figure 3.15a. This amounts to swapping the underlying nodes, which are now both located in the proper half-plane. When the two iterators have reached each other somewhere in the middle of the node interval, the sorting is done.

7. In Figure 3.15b, once the sorting is done, the left subinterval ranges from the P_1 node to the P_7 node, and the right subinterval ranges from the P_6 node to the P_2 node. These subintervals are then pushed on the stack of waiting subintervals. Then, the algorithm resumes at Step 4.



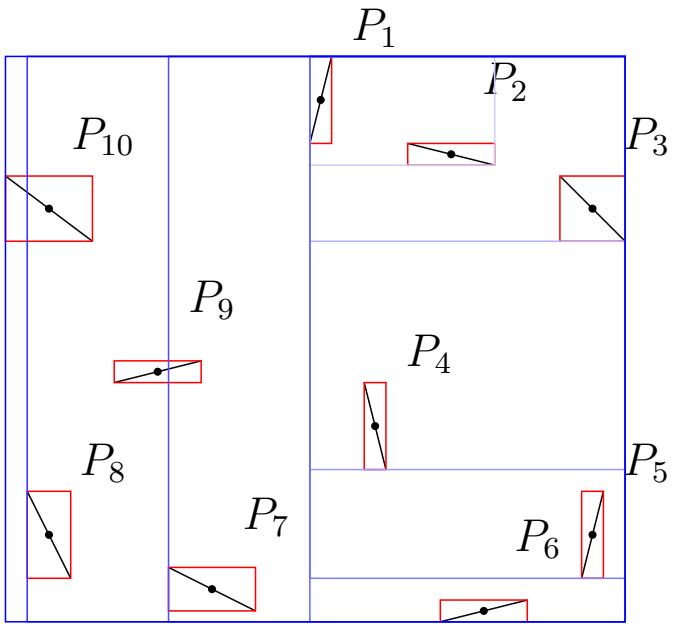
(a) Axis-aligned bounding boxes enclosing each facet (Step 1).



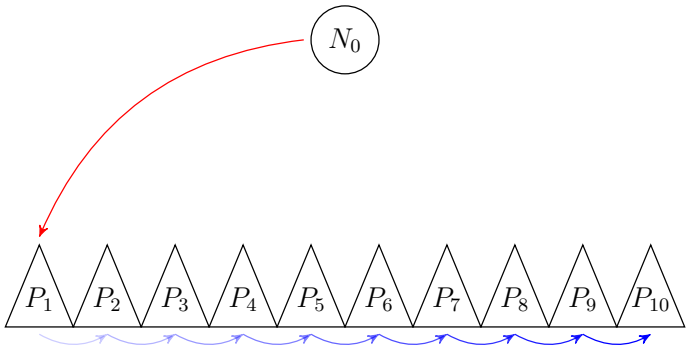
(b) Leaf nodes each containing one facet barycentre (Step 2).

Figure 3.13: Binary tree construction process, steps 1 and 2 of Algorithm 15.

Performance evaluation We have established the computation time for BV kd-tree construction for the same meshes than for the metanode kd-tree. The result can be seen in Figure 3.16. We have also displayed the ratio of BV tree time on metanode tree time for each of the considered meshes. As we can see, the BV kd-tree consistently outperforms the metanode kd-tree, with a construction time between 30% and 70% faster across the considered range of number of points. Given that the tree might have to be reconstructed a lot of times if the mesh changes a lot during computation, this is a time saving we can expect to benefit from several times. We also observe that the algorithmic complexity of the construction does not change, as

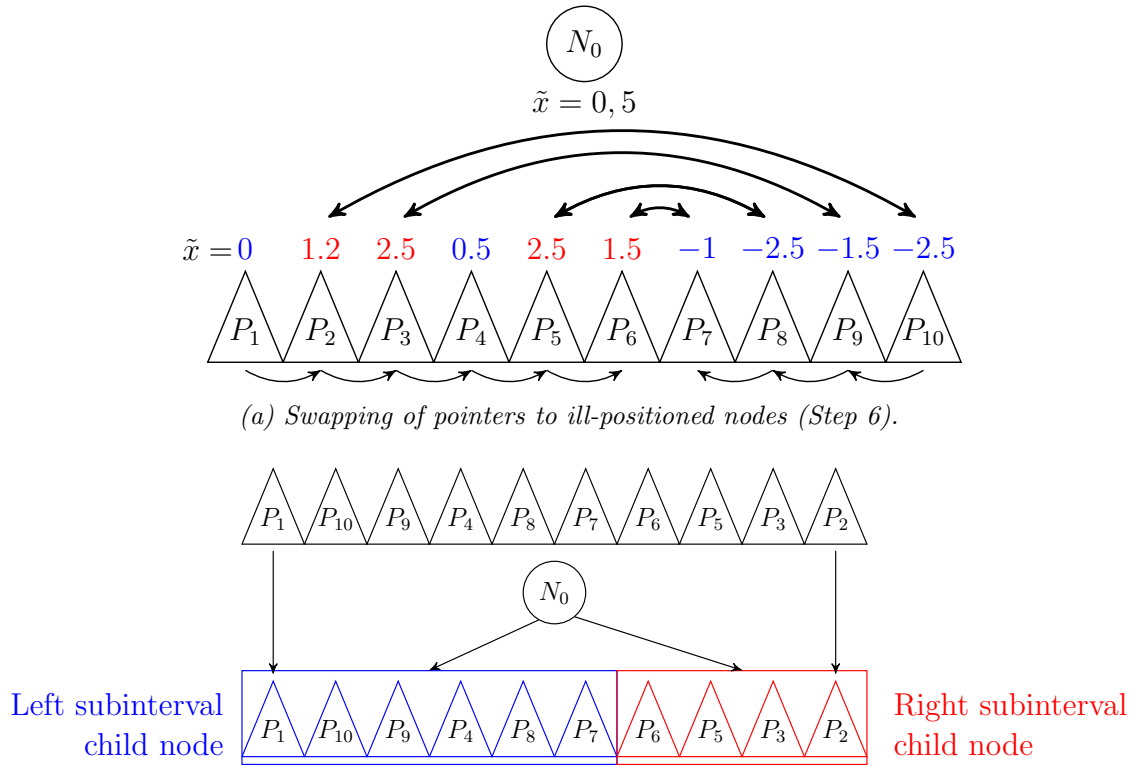


(a) AABB of node N_0 being updated to progressively contain the AABB of all nodes of the interval.



(b) Node N_0 is compared to all nodes of the interval.

Figure 3.14: Binary tree construction process. Step 4 of Algorithm 15 is shown.



(a) Swapping of pointers to ill-positioned nodes (Step 6).

(b) Creation of two node subintervals (Step 7).

Figure 3.15: Binary tree construction process. Steps 6 and 7 of Algorithm 15 are illustrated on it.

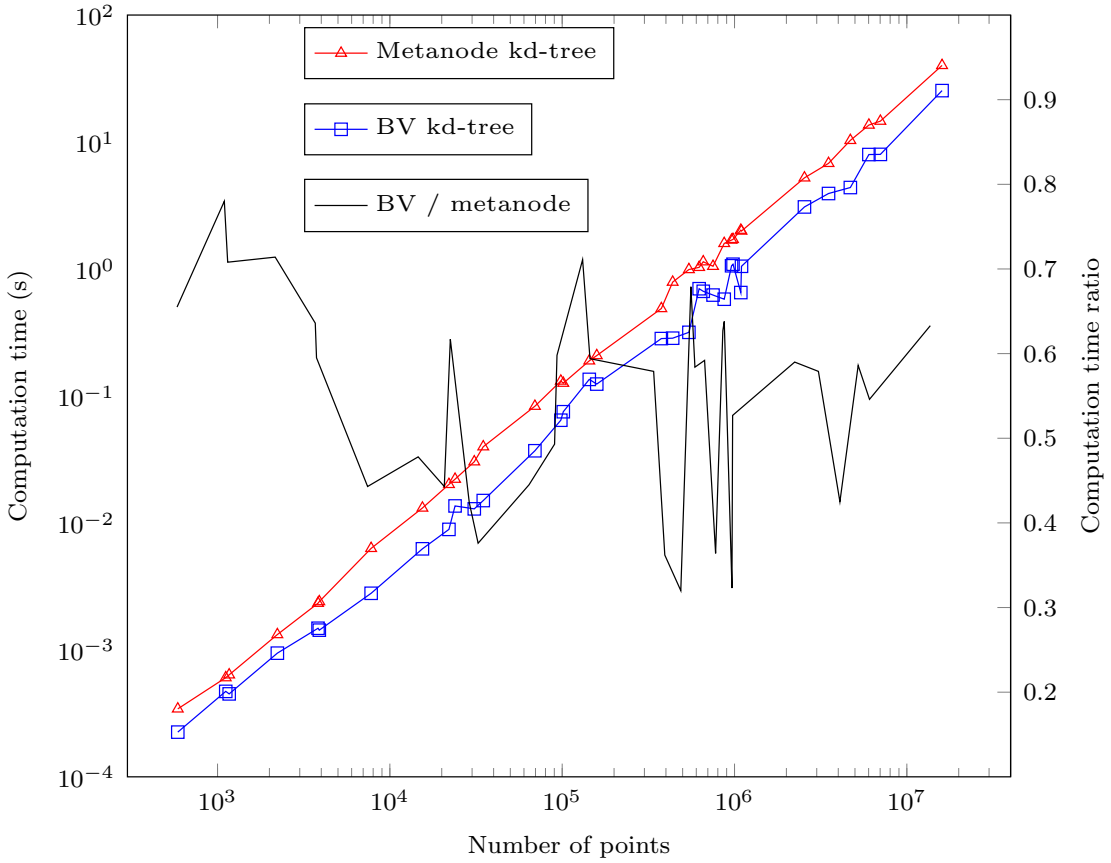


Figure 3.16: Complexity of BV kd-tree generation time depending on its number of points.

the two curves remain almost parallel. This speedup is due to the lighter structure of the BV kd-tree. Since we have devised this second type of kd-tree because we found the metanode kd-tree was a little too cumbersome, we are satisfied to indeed observe a performance enhancement.

3.3.2.2 Kd-tree traversal by ray tracing

The algorithm for kd-tree traversal we have selected is close to the stack-based algorithm for kd-tree traversal shown in Figure 3.11. It is inspired from the standard recursive BSP tree traversal algorithm formulated as a stack-based iterative version found in Appendix B of Havran's thesis [30]. The pseudocode for the tree traversal algorithm we have used is shown on Algorithm 16.

Algorithm 16 BV kd-tree traversal

```
1: Global variable listLeafNodes
2: procedure TREE TRAVERSAL(int facetIndix1, double * rayOrigin, int facetIndix2, double * rayEnd)
3:     double * ray = rayEnd - rayOrigin
4:     double * invertedRay = rayOrigin - rayEnd
5:     node nodeOrigin, nodeEnd = listLeafNodes[facetIndix1], listLeafNodes[facetIndix2]
6:     Initialize stackNodes
7:     stackNodes.add(root)
8:     while (stackNodes is not empty) do
9:         curNode = stackNodes.pop()
10:        if (collide(rayOrigin, ray, curNode.AABB) & collide(rayEnd, invertedRay, curNode.AABB)) then
11:            if (curNode is not a leaf) then
12:                stackNodes.add(curNode.leftChild)
13:                stackNodes.add(curNode.rightChild)
14:            else
15:                if (no overlap(curNode.AABB, nodeOrigin.AABB) & no overlap(curNode.AABB, nodeEnd.AABB)) then return true
    return false
```

Between facets F_i and F_j, we add the root node of the tree onto a stack. While the stack is not empty, we pop its first element and we launch a ray through its AABB. If we find the ray intersects it, if the node is a leaf node then the intersection returns true. If it is an inner node, we push its two children onto the stack. If it is a leaf node, we check whether its AABB intersects with either the AABB of the starting node or the AABB of the end node. If either test returns true, since there is only

one facet per node, this means the leaf node we are currently in actually is either the starting node or the leaf node. Therefore, the facet this leaf node contains is either the starting facet or the end facet of the ray, and it should be discarded as a potential obstructing facet. If both these checks return true, then the algorithm returns that an obstruction has indeed taken place. To avoid the situation where a ray might run inside the plane separating two AABBs, at the moment of checking ray-AABB collision we slightly inflate the AABB by an epsilon. This ensures that in such a situation, the ray will be considered colliding with both AABBs, rather than with none of them. This prevents the algorithm from halting when it should not.

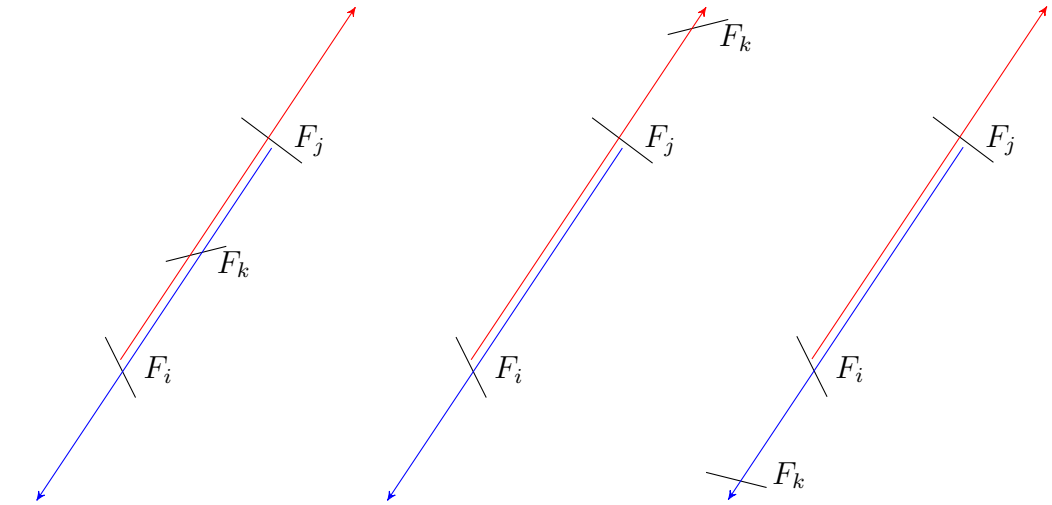
Design comparison with the metanode ray tracing We notice that this algorithm does not require the knowledge of the splitting plane of any node - just their AABB and children. This justifies us not storing this value into each node. As already mentioned, this means that compared to the metanode kd-tree traversal algorithm, we can not proceed to a Find algorithm, as shown on Algorithm 11, to determine where in the tree the ray's starting point is situated. However, since we have numbered the leaf nodes, we do not have to first make a tree descent to locate the starting facet of the ray. Since the tree stores as a global variable an array of all the leaf nodes, we can directly fetch the required leaf nodes by passing the index of the facets they contain to the tree. This contributes to making the BV kd-tree traversal faster compared to the metanode kd-tree traversal, since we save one tree descent this way. We also think the algorithm will be faster than the metanode kd-tree's because since it can only travel down in the tree, and never up, there is no way a given node can ever be visited more than once. This ensures we will not waste time evaluating nodes where we already have been. Finally, it is formulated as an iterative algorithm, and not as a recursive algorithm as was the metanode kd-tree ray traversal, which is a more efficient practise for algorithm efficiency as already mentioned.

Heuristics for which child to traverse In many image rendering applications, it matters a lot to determine all of the possible facets obstructing the (F_i, F_j) couple, and to return the one that is closest to the ray's starting facet F_i. In these cases, additional checks are made on the children of the current node to determine which one of them is traversed first by the current ray (it is then pushed first on the stack). These are called *first-hit traversals*. These involve comparing the position of the splitting plane of the node to the trajectory of the ray. On the other hand, in radiative transfer, we only wish to know whether the (F_i, F_j) couple is obstructed at all, and the particular position of the obstructing facet is irrelevant to us. In this *any-hit traversal*, we do not need these heuristics to sort out the two children of the

current node. This is why we simply push both of them on the stack without additional computation. Besides, since we do not store the splitting plane, we actually cannot proceed to such heuristics anyway. For this reason, the pseudocode shown on Algorithm 16 is actually a simplified version of the algorithm found in Havran's thesis. On the other hand, some more refined heuristics have been designed for any-hit traversal too. Nah and Manocha [37] have devised a traversal order where the child having the largest total area is pushed on the stack first, reasoning that it should have a higher chance of containing an obstructing facet. They have also devised two other heuristics, one targeting first the nodes with the smallest number of leaves, another the nodes with the higher average or maximum surface of the facets they contain. They conclude that different scene configurations require different heuristics for optimal ray traversal. Ize and Hansen [38] have devised a variation of the SAH already mentioned for the kd-tree, but adapted to determining the cost of traversing one child rather than the other. This is based on an estimated visibility function and called the Ray Tracing Surface Area Heuristics (RTSAH), which is applicable for both BV and metanode kd-trees. Feltman et al. [39] have devised their own version of the RTSAH but instead of using estimated visibility functions, they sample a small set of rays through the scene to obtain a first approximation of which areas are more occluded and which are more lighted. From this they extract occlusion probability estimations to construct their so-called Shadow Ray Distribution Heuristics (SRDH).

Ray and inverse ray launching It can happen that facet F_k is located on the path of the ray from F_i to F_j, but it is actually behind facet F_j. In that case, analysis of the ray length would show us that the intersection between the ray and F_k happens farther on the ray path than the ray length. Thus, this intersection would be invalidated. This implies computing the ray length at all facet comparisons, making additional checks to compute the position of the intersection between the ray and F_k. We have chosen another workaround to eliminate F_k as an intercepting facet. We also throw the ray from F_j to F_i. If F_k is indeed located behind F_j, then this "reversed ray" will never see it. Thus, to determine whether a given node's AABB properly intersects the ray *between* F_i and F_j, we do not make ray length analyses. We instead check the AABB both against the ray from F_i and F_j and the ray from F_j to F_i. Only if both checks return an interception do we proceed further down the algorithm. Figure 3.17 sums up this process in the case of 2D facets. We assume the extra ray launch cost is worht it in regard of the extra bookkeeping and operations that were required to handle ray length.

Bounded volume approximation The collide function we use for all nodes, even leaf nodes, checks the AABB of the node against the ray. For this, we use the



(a) Real obstruction: F_k between F_i and F_j. *(b) Improper obstruction: F_k behind F_j.* *(c) Improper obstruction: F_k behind F_i.*

Figure 3.17: Checking of the status of potentially obstructing facet by a double ray tracing in 2D. The twos rays in red and blue should be aligned but have been slightly moved away from one another for better visibility.

efficient ray-AABB intersection algorithm devised by Williams et al. [40]. We have mentioned how BHV ray tracing usually proceeds to a proper ray-facet intersection once a ray-AABB intersection has been determined. We have decided to lighten the process even more by approximating a facet to its BV. If a ray-AABB intersection is found, we stop here and automatically consider the ray-facet intersection to be true too. This means, however, that we compute an intersection with an object slightly bigger than the facet it contains. Possibly, in some cases the ray might intersect the AABB but not the facet itself. This situation is shown in Figure 3.18.

Given that the facet size is usually very small before the distance separating the facets radiating together, the portion of (solid in 3D) angle in which a ray must find itself for such an error to occur is very small too, meaning this error is not very frequent. Besides, this error should, on average, occur evenly on all facets around the system. The total radiative flux correction we use as seen in Chapter 2 has the effect of correcting the flux on all participating facets by a constant amount. This means if we make a mistake that is roughly constant over all radiating facets, the flux correction will erase it more or less entirely. On the other hand, the ray-AABB intersection algorithm is simpler and faster than the ray-triangle intersection algorithm from Möller et al. [35] that we use in the kd-tree. We thus estimate the tradeoff between accuracy and computation efficiency to be in our advantage.

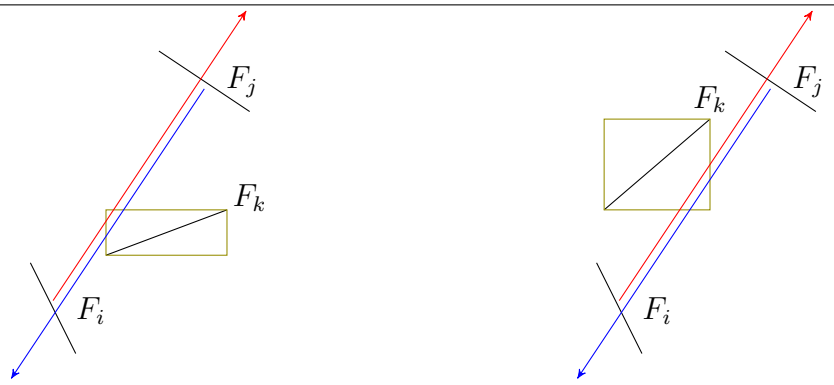


Figure 3.18: Examples in 2D of facet F_k not obstructing between F_i and F_j, but their AABB is, leading to a false positive obstruction.

3.4 Parallel computation

We have tried out the proposed solver on some simple 3D test cases involving multiple processors, in order to see how it behaves in parallel computation. An order of magnitude of the number of processors that can be routinely used by an industrial user is about 200. This means we want to guarantee the proposed solver scales properly up to at least that amount of processors. To this end we will evaluate how obstruction is handled within S2S during the first time step of the simulation, and then how S2S as a whole behaves during total thermal transfer resolution.

3.4.1 Radiosity preprocessing

In a given system where radiation is involved, we have seen that computation of the radiating facets and the view factor matrix needs to be done at the beginning of the simulation and then each time the items move or are deformed. It is at that moment that obstruction is computed. During parallel solving, each core is responsible for computing a block of lines of the radiosity matrix. Computing line i of the radiosity matrix means the core needs to compute all the F_{ij} view factors, j varying from 1 to the total number of facets. That stands for the geometric interaction between facet F_i and each and everyone of the other radiating facets of the system. We have decided that the core needs to hold in local memory the entirety of the topology of the radiating facets in the system. This way, it can compute the entirety of line i in a purely local fashion. If computing view factor F_{ij}, it does not need to communicate with the core responsible for facet F_j to obtain the geometry information of F_j it needs. This design choice places a heavy burden on local memory and prevents us from fully benefitting from the reciprocity law of view factors as per Equation 2.12. But we expect that the lack of need for communication between partitions leads to

a good scalability of that part of the process, as per Amdahl's law. In theory we could keep the scalability performance of that part of the resolution up to a number of processors equal to the number of facets, when each core is responsible for a single facet.

One could question the relevance of trying to optimize the parallel behaviour of an operation that will be performed only once by the solver for a simulation that might last for thousands of time steps. Optimizing the parallel behaviour of the S2S solver during each of these time steps would seem more sensible. The answer is that obstruction is so time-consuming that it can happen this sole stage of the computation takes a non insignificant amount of time even for lengthy simulations. Besides, if the system moves (say, we decide to take into account the shrinking ceramics suffers during sintering) then we have to repeat this operation several times over the course of the complete simulation. We will now see how the S2S solver handle radiosity preprocessing in parallel computation for two different test geometries.

Hat test case Figure 3.19 displays a 3D hat-shaped piece used by Safran Aircraft Engines. The hat is initially heated to 1160 °C and then put in a vacuum cube-shaped environment with adiabatic walls at 20 °C and left to cooling for 30 min. We use 1800 time steps of 1 s each.

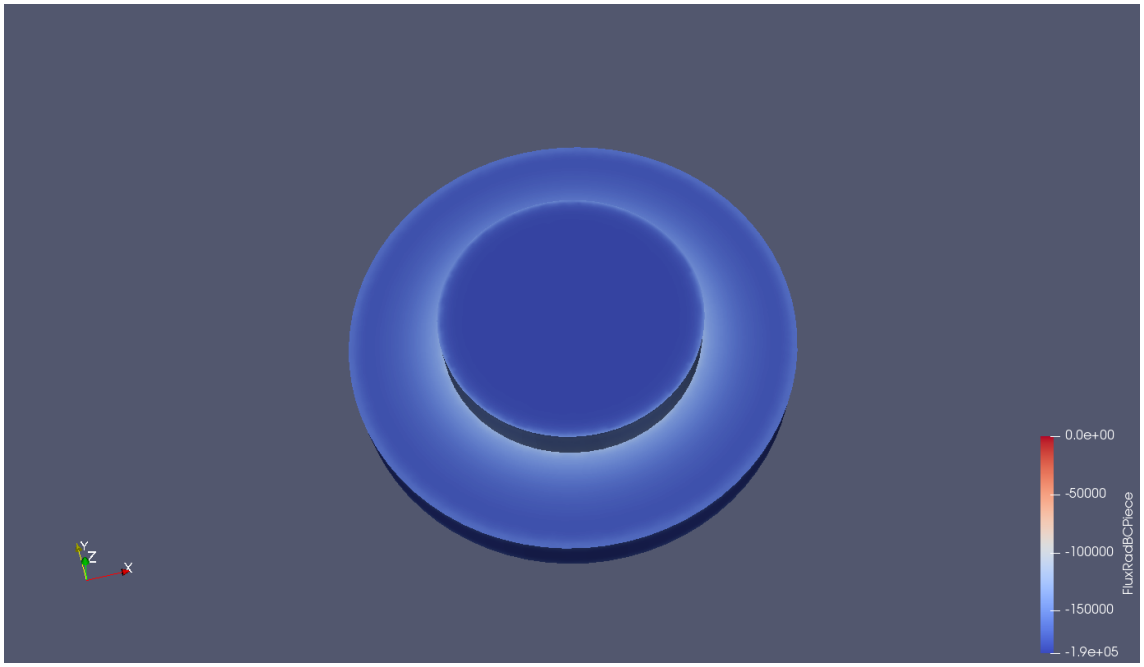


Figure 3.19: Radiative flux density on the surface of the hat (1160 °C) at the beginning of the simulation.

The outer surface of the hat loses heat through radiation, while the bulk of the hat loses heat by conduction to make up for the lost thermal flux. All the surface of the hat and the surface of the boundary contribute to radiation, and they are split into a total of 59992 radiating facets. We do not have an analytical solution to compare our results with. However, we can see in Figure 3.19 the radiative flux density received by the surface of the hat at $t = 0$. It is in negatives values, which makes sense since the hat is supposed to lose energy by radiation. As we can expect, the area near the junction between the two sections of the hat loses a smaller amount of radiation. This is due to the fact that at this place, radiation towards the outer environment is partially blocked by the surfaces of the hat that extend both under and behind the junction. A lot of view factors are set to 0 through obstruction by the hat itself.

The calculations we have made with this hat-shaped piece include many more thermal results and comparisons, but that will be studied in-depth in the next chapter. This section will only study the efficiency performance of the simulations.

The simulation is started several times with an increasing number of processors used, from 1 to 256. Figure 3.20 indicates how much time each part of the solver takes to compute radiation for the first time step, depending on the number of processors used.

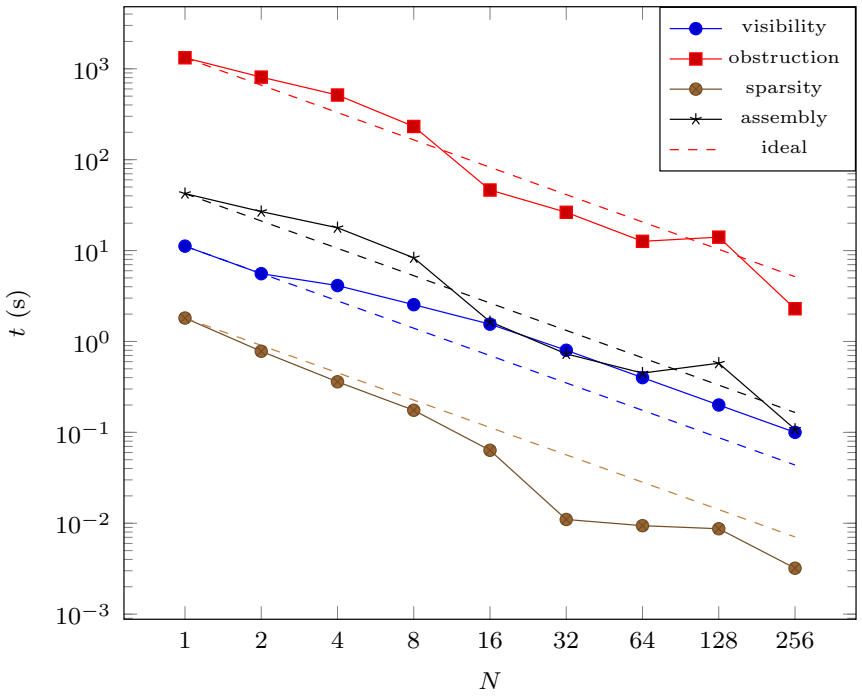


Figure 3.20: Computation time for the different stages of S2S resolution for the hat during radiosity preprocessing as a function of the number of cores.

Accordingly, computation time diminishes as the number of processors increases. Not all sections of radiation computation take the same amount of time or scale the same way. We can see that, as mentioned at the beginning of this chapter, obstruction stands for the first computation time budget of the complete S2S resolution by far, hence our efforts to curb it. The dashed lines appearing over each curve are the computation time when $N = 1$, then divided by N. It stands for ideal behavour of the solver if our process is correctly parallelized: working with N processors should be, in theory, N times faster than what we have with a single processor. We can observe obstruction solving behaves closely to the $1/N$ curve. We even observe an increase in performance when the number of processors reaches $N = 256$. We suspect this might be a caching effect: the memory footprint of the partition to be handled by each core is now small enough to be stored in local memory, which means we save global memory access time. Since obstruction is so overwhelmingly more time-consuming than either preliminar facet visibility, sparsity matrix computation or matrix assembly, its parallel behaviour sets the parallel behaviour of the complete S2S resolution. This means that even if visibility computation time curve seems to stray away from its ideal curve, it will not affect the global performance of the solver. Hence, we can conclude from this test case that for this very simple geometry, for about 60000 radiating facets and up to 256 processors, the expected time gain from parallel computation is attained.

Ferrule test case Another test case for parallel performance is that of radiation in a ferrule used by Framatome, modelled as a hollow cylinder. It can be seen in Figure 3.21.

It is set into a cube but near to one of the walls, so as to create a strong local gradient of exchanged flux density on the wall and the ferrule. One can seen in Figure 3.21 the emitted flux density of the system: as expected, the portion of wall directly facing the ferrule displays an emitted flux density that is much higher than elsewhere on the wall. The radiative surfaces of the system are split into 56456 facets. It is initally set to 1000 °C and left to cooling the same way the hat was. Like the hat, the ferrule test case thermal results will be analyzed in depth in the next chapter, and this section will simply focus on the efficiency performance of the test case.

We proceed to the same performance analysis for radiosity preprocessing as for the hat, from 1 to 64 processors used. The results are displayed in Figure 3.22. Once again, we can see obstruction dominating the process and following a correct linear behaviour. Visibility computation scaling slightly degrades when the number of cores increases but remains an order of magnitude faster than obstruction computation. Overall, this second test case does not invalidate the results obtained for the hat. We can conclude that for these very simple shapes and a moderate number of

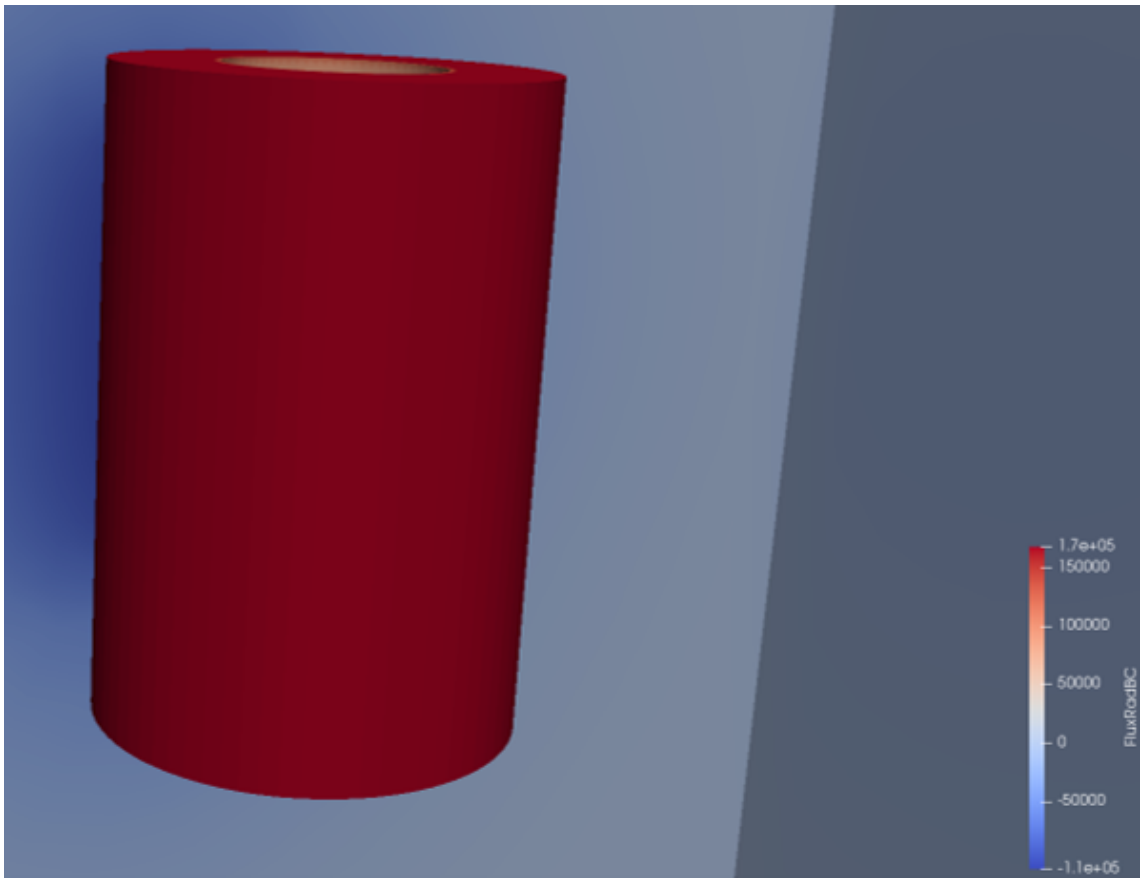


*Figure 3.21: Radiative emitted flux density on the surface of the ferrule (*1000 °C*) at the beginning of the simulation*

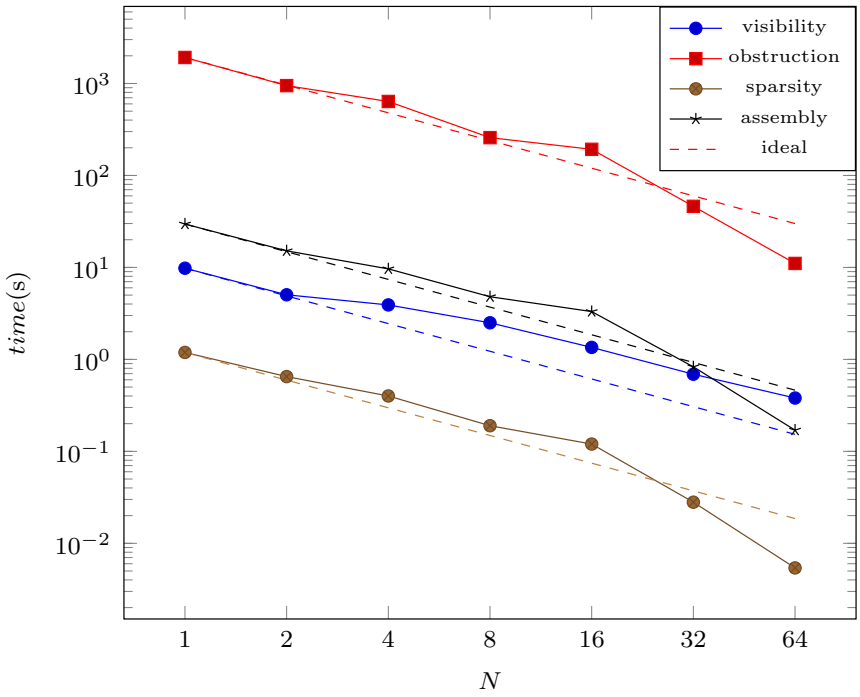


Figure 3.22: Computation time for the different stages of S2S resolution for the ferrule during radiosity preprocessing as a function of the number of cores.

processors and facets, the radiosity preprocessing step seems to behave as expected in parallel computation.

3.4.2 Thermal transfer

Once the preprocessing of radiosity is done, we can see how S2S solving as a whole behaves when compared to the other stages of thermal transfer solving. For the hat and the ferrule test cases, there was no convection involved. Thermal transfer was only composed of radiative exchange between the part and the enclosure, and conduction in the bulk of the part triggered by the cooling of the outer portion of the part due to that radiative loss. The two forms of thermal transfer have been weakly coupled as explained in Chapter 2. At each time step, radiative flux density is resolved, then that field is projected onto the finite elements of the system to participate as an added energy source in the heat equation. Then, thermal conduction is resolved. We will compare the average of the amount of time the solver spends for radiation and conduction respectively during each time step of the simulation.

During each time step, the heat equation is solved in a purely local way by each processor for the elements it is responsible of, with a minimal amount of communication with neighbouring processors. This is why it is efficiently parallelized. That is not the case of S2S. It requires a global communication from all processors to all processors two time during resolution.

1. At the beginning of S2S resolution for a given time step, the temperature of each facet has changed compared to the temperature they had at the previous time step. This is because thermal conduction has been solvedin the meantime. As a result, each processor has updated the temperature of the finite elements it is responsible of. However, during parallel setup, finite element partitioning and radiating facet partitioning are done completely separately. A given processor is responsible of both a partition of finite elements and a partition of radiating surfaces, so that it can contribute to parallel computation in both the S2S phase and the conduction phase. But the set of finite elements and the set of radiating facets, being determined independantly, have no *a priori* topological relationship. This means a given processor has no particular reason to be responsible of both a certain set of finite elements and the exact set of radiating facets that belong to these elements. As a result, when S2S is about to begin, each processor needs to update the temperature field of the facets it is responsible of. In the most general case, the finite elements corresponding to these facets are spread out between several *other* processors. This means there needs to be an all-to-all synchronisation of the temperature field at the beginning of S2S resolution. Besides, if other local values have

changed, such as emissivity if we later implement a variable emissivity model for example, these values need to be communicated globally at that moment too.

2. After S2S resolution, conduction resolution will begin. For that phase, each processor needs to know the radiative flux density that is received by the facets related to each finite element it is responsible of. For the exact same reason as described in the first point, each processor needs to be given that information from all the other processors. This is why at the end of S2S resolution, once each processor is done computing the net flux density received by each facet is it responsible of, the flux densities are synchronised back over all processors in another all-to-all communication.

These two all-to-all communications impose an added computation time on the process, which increases in proportion to the total computation time as the number of processors increases. For this reason, we expect our complete S2S solver to behave in a slightly degraded way as the number of processors increases.

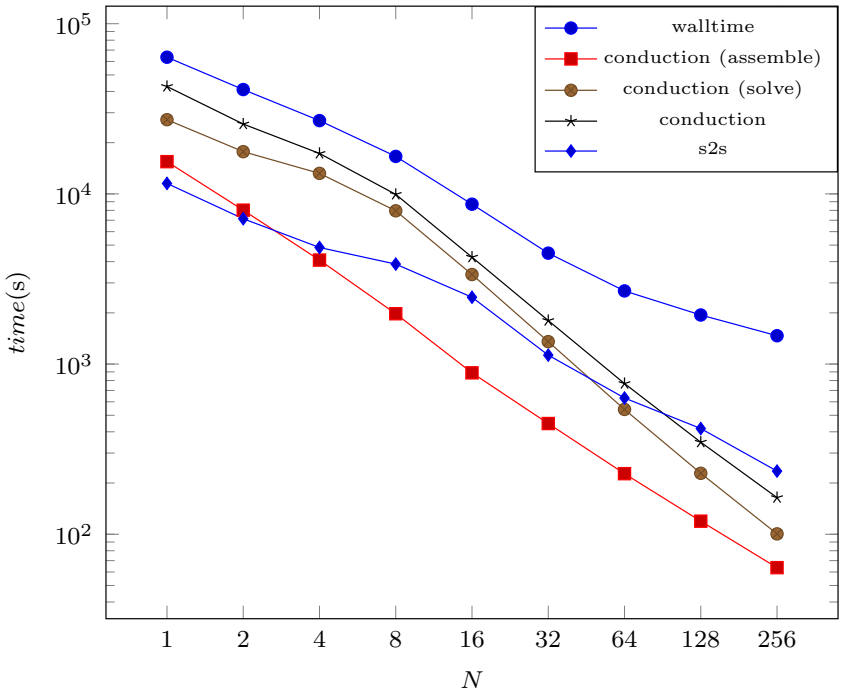


Figure 3.23: Computation time as a function of the number of cores used for thermal transfer in the hat.

Hat test case Figure 3.23 indicates the average time spent over all time steps for all stages of thermal transfer solving, for up to 128 processors used. We can see

conduction scaling linearly with the number of cores used in a satisfactory way. On the other hand, S2S scales a little less well, as was expected. It becomes more time-consuming than conduction starting from 128 processors used. Despite this, S2S still behaves relatively well. After an initial offset from the ideal trajectory, starting from 16 processors the behaviour follows a trajectory that is parallel to the ideal one, which means it is properly linear. We do not see the plateau of performance appearing yet, even when each processor is responsible for only 200 facets or so.

Ferrule test case The same study is conducted for the ferrule, up to 64 processors. Figure 3.24 shows the average computation time of each time step for the different parts of thermal transfer modelling in the ferrule. The ferrule behaves the same way the hat does: S2S processing scales slightly less well than conduction. We see the same offset from ideal trajectory up until 16 processors where the behaviour becomes parallel to the ideal one.

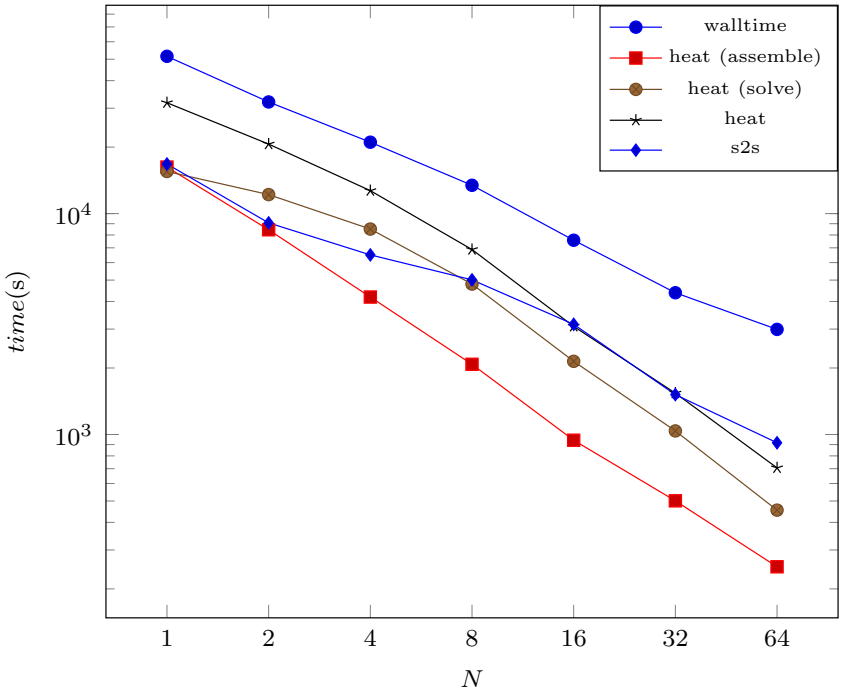


Figure 3.24: Computation time as a function of the number of cores used for thermal transfer in the ferrule.

3.5 Summary

- In order to overcome the roadblock of computation time, we have set up acceleration structures. The use of a metanode kd-tree-based ray tracing solution to determine facet obstruction has efficiently diminished our time complexity from the naive approach.
- After a litterature review, a complete kd-tree building and ray tracing algorithm has been developed.
- Nearest-neighbour search has also been implemented and performs adequately compared to the litterature.
- Another, more effective and light-weight, type of kd-tree has been developed, the BV kd-tree.
- The scalability of the ray tracing algorithm in the BV kd-tree for up to 256 cores has been established. The algorithm performs well even at high numbers of cores.
- Even if the established structures are simple and devoid of high-end optimization, they have enabled us to overcome the computation time problem.

We will now evaluate the solver's performance on more realistic and computer-heavy test cases.

Bibliography

[1] R. Schmacher, B. Brand, M. G. Gilliland, W. Sharp, Study for applying computer-generated images to visual simulation: (510842009-001), 1969. 89

[2] N. Chin, A walk through BSP trees, 1995. 89

[3] V. Havran, T. Kopal, J. Bittner, J. Žára, Fast robust BSP tree traversal algorithm for ray tracing, Journal of Graphics Tools 2 (4) (1997) 15–23. 89

[4] K. Sung, P. Shirley, Vi.1 - RAY tracing with the BSP tree, in: D. KIRK (Ed.), Graphics Gems III (IBM Version), Morgan Kaufmann, San Francisco, 1992, pp. 271–274. 89

[5] J. Arvo, Linear-time voxel walking for octrees, 1988.
URL https://www.acemap.info/paper/175904380 89

[6] J. Kobrtek, T. Milet, M. Tóth, A. Herout, Comparison of modern omnidirectional precise shadowing techniques versus ray tracing, Computer Graphics Forum. 89, 126

[7] J. Bentley, Multidimensional binary search trees used for associative searching, Communications of the ACM 18 (1975) 509–517. 89

[8] S. Bindick, M. Stiebler, M. Krafczyk, Fast kd-tree based hierarchical radiosity for radiative heat transport problems, Internation Journal for Numerical Methods in Engineering 86 (september 2014) 1082–1100. 90, 115

[9] S. Mazumder, Methods to accelerate ray tracing in the Monte-Carlo method for surface-to-surface radiation transport, Journal of Heat Transfer (feb. 2006) 945–952. 90, 127

[10] C. Wang, J. D. Eldredge, Application of a variance reduction technique to surface-to-surface Monte Carlo radiation exchange calculations, INTERNATIONAL JOURNAL OF HEAT AND MASS TRANSFER 131 (2019) 424–431. 90

[11] J. D. MacDonalds, K. S. Booth, Heuristics for ray tracing using space subdivision, Visual Computer 6 (1990) 153–165. 93

[12] I. Vald, V. Havran, On building fast kd-trees for ray tracing, and on doing that in O(N log N), IEEE, Salt Lake City, UT, USA, 2006, pp. 61–69. 93, 99, 101

[13] M. Shevtsov, A. Soupikov, Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes, Computer Graphics Forum 26 (2007) 395–404. 93, 101

[14] V. Havran, Heuristic ray shooting algorithms (2000). 93

[15] I. Wald, Realtime ray tracing and interactive global illumination, Ph.D. thesis (01 2004). 98, 116, 125

[16] Z. Li, Y. Leng, M. Gu, Path compression kd-trees with multi-layer parallel construction: A case study on ray tracing, San Francisco, CA, USA, 2017. 100

[17] X. Liu, Y. Deng, Y. Ni, Z. Li, Fasttree: A hardware kd-tree construction acceleration engine for real-time ray tracing, Grenoble, France, 2015, pp. 199–213. 101

[18] Y. Deng, Y. Ni, Z. Li, S. Mu, W. Zhang, Toward real-time ray tracing: A survey on hardware acceleration and microarchitecture techniques, ACM Computing Surveys 50 (4) (2017) 467–493. 101, 125

[19] B. Choi, R. Komuravelli, V. lu, H. Sung, R. Jr, S. Adve, J. Hart, Parallel sah k-d tree construction, 2010, pp. 77–86. 101

[20] R. Pinkham, S. Zeng, Z. Zhang, Quicknn: Memory and performance optimization of k-d tree based nearest neighbor search for 3d point clouds, in: 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2020, pp. 180–192. 103

[21] W. Hou, D. Li, C. Xu, H. Zhang, T. Li, An advanced k nearest neighbor classification algorithm based on kd-tree, in: 2018 IEEE International Conference of Safety Produce Informatization (IICSPI), 2018, pp. 902–905. 103

[22] C. Kingsford, kd-trees, https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/kdtrees.pdf. 103

[23] B. Fan, Q. Kong, B. Zhang, H. Liu, C. Pan, J. Lu, Efficient nearest neighbor search in high dimensional hamming space, Pattern Recognition 99 (2020) 107082. 103

[24] J. Jo, J. Seo, J.-D. Fekete, A progressive k-d tree for approximate k-nearest neighbors, in: 2017 IEEE Workshop on Data Systems for Interactive Analysis (DSIA), 2017, pp. 1–5. 103

[25] J. Friedman, J. Bentley, R. Finkel, An algorithm for finding best matches in logarithmic expected time, ACM Trans. Math. Softw. 3 (1977) 209–226. 103

[26] P. Hanrahan, D. Salzman, L. Aupperle, A rapid hierarchical radiosity algorithm, Computer Graphics 25 (1991). 115

[27] M. Hapala, V. Havran, Review: Kd-tree traversal algorithms for ray tracing, Computer Graphics forum 30 (1) (2011) 199–213. 115

[28] A. Lira dos Santos, J. Teixeira, T. Farias, V. Teichrieb, J. Kelner, Understanding the efficiency of kd-tree ray-traversal techniques over a GPGPU architecture, International Journal of Parallel Programming 40 (06 2011). 115, 122, 125

[29] M. Kaplan, Space-tracing: a constant time ray-tracer, in: SIGGRAPH '85: Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques, 1985. 116

[30] V. Havran, Heuristic ray shooting algorithms, Ph.D. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague (2000). 120, 129, 137

[31] F. W. Jansen, Data structures for ray tracing, Springer-Verlag, 1986. 120

[32] T. Foley, J. Sugerman, Kd-tree acceleration structures for a GPU raytracer, Graphics Hardware (2005) 15–22. 120

[33] D. Horn, J. Sugerman, M. Houston, P. Hanrahan, Interactive k-d tree GPU raytracing, in: I3D '07: Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games, 2007, pp. 167–174. 121

[34] S. Popov, J. Günther, H.-P. Seidel, P. Slusallek, Stackless kd-tree traversal for high performance GPU ray tracing, Computer Graphics forum 26 (3) (2007) 415–424. 121

[35] T. Möller, B. Trumbore, Fast, minimum storage ray-triangle intersection, Journal of Graphics Tools 2 (1) (1997) 21–28. 123, 140

[36] S. Rubin, T. Whitted, A 3-dimensional representation for fast rendering of complex scenes, Vol. 14, 1980. 126

[37] N. J.-H., M. D., Sato: Surface area traversal order for shadow ray tracing., Computer Graphics Forum 6 (2014) 167–177. 139

[38] T. Ize, C. Hansen, Rtsah traversal order for occlusion rays, Computer Graphics Forum 30 (2011) 297–305. 139

[39] N. Feltman, M. Lee, K. Fatahalian, Srdh: Specializing bvh construction and traversal order using representative shadow ray sets, Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics (2012). 139

[40] A. Williams, S. Barrus, R. K. Morley, P. Shirley, An efficient and robust ray-box intersection algorithm, Journal of Graphics GPU and Game Tools 10 (1) (2005) 49–54. 140

Chapter 4

Industrial applications

Contents

4.1 Résumé 156
4.2 Sintering oven 157
4.3 Simulation meshing and setup 161
4.3.1 Ferrule setup 161
4.3.2 Three loads setup 163
4.3.3 Hat setup 166
4.4 Results and discussion 168
4.4.1 Steady state ferrule results 168
4.4.2 Ferrule temperature evolution 170
4.4.3 Steady state three loads results 175
4.4.4 Three loads temperature evolution 178
4.4.5 Steady state hat results 182
4.4.6 Hat temperature evolution 184
4.4.7 Discussions 189
4.5 Summary 191
Bibliography 192

4.1 Résumé

Après avoir validé la précision de notre solveur sur des cas analytiques et amélioré sa vitesse d'exécution à l'aide d'arbres kd, nous effectuons trois simulations de transfert thermique conducto-radiatif sur des cas industriels réels. Nous commençons par rappeler les caractéristiques physiques et géométriques du four de frittage utilisé pour les pièces de TAS, mais aussi des cycles de frittage qui sont couramment effectués. Les trois cas industriels que nous simulons se passent dans des conditions spatiales et de température proches de celles présentes dans le four.

Nous détaillons ensuite les conditions d'étude de nos trois cas : le refroidissement radiatif d'une ferrule, le chauffage radiatif de trois blocs situés les uns à côté des autres dans un four (ce dernier cas reproduisant exactement les paramètres de frittage du four de TAS) et le refroidissement radiatif d'un disque. Les paramètres de maillage, de température et de simulation sont décrits.

Pour chaque cas, le profil de température et de flux radiatif est montré à l'issue du régime permanent. Les courbes d'évolution de la température et du flux radiatif sont aussi montrées au cours du régime transitoire. Les résultats sont commentés et évalués par rapport au comportement physique attendu.
L'étude des ferrules montre le caractère important de l'auto-rayonnement dans les pièces creuses. Le refroidissement purement radiatif ne permet pas d'évacuer efficacement la chaleur dans les zones de surface qui sont principalement exposées à d'autres zones de même température.
L'étude des trois blocs révèle deux choses. Le caractère localisé du rayonnement produit par la paroi latérale du four, ainsi que l'obstruction mutuelle des blocs, induisent des gradients importants de flux thermique reçu et de température. Comme ce sont les conditions réelles de frittage, cela nous incite à conclure qu'il vaut mieux espacer plusieurs pièces frittées en même temps pour limiter leur obstruction mutuelle.
Enfin, l'étude du disque est menée, et pour ce cas nous disposons de résultats expérimentaux de l'expérience de refroidissement. Nous comparons notre simulation aux sondes de température expérimentales. Nous dégageons des groupes locaux de comportement où la simulation diffère de l'expérience selon certaines tendances. Le modèle Surface-to-Surface parvient efficacement à capturer le caractère localement changeant du rayonnement en fonction de la géométrie, et les résultats simulés sont en très bon accord avec l'expérience.

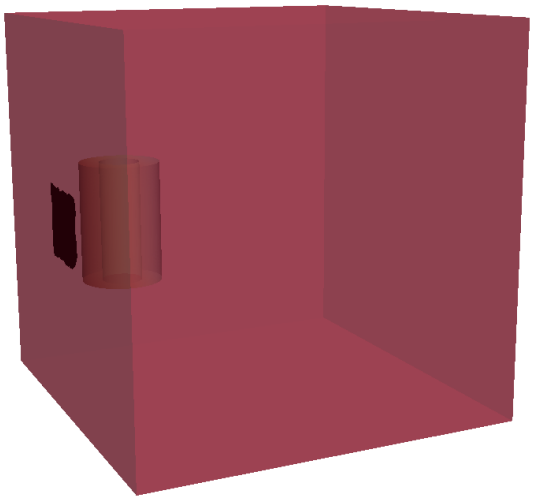
Now that we have devised a robust S2S thermal radiation solver and tested it both in accuracy against analytical test cases and in efficiency against multi-processor simulations, we can test it on more realistic industrial configurations. This chapter will detail three different configurations involving an industrial process where thermal radiative transfer plays an important role and where the S2S formulation as detailed in Chapter 2 subsection 2.3.2 are valid. The simulations will be run with geometries for specific objects that have been provided to us by industrial companies. For confidentiality reasons, they approach but do not exactly match the real geometries these companies use. However, they still provide a very useful benchmark for the proposed solver. Many medium-to-big companies can routinely afford to launch a simulation of a given industrial process on a few dozens of processors. This way, if we can ensure the proposed solver outputs stable, accurate and relatively fast results when launched on parallel with up to 200 processors or so, this means we have reasonably achieved a first version of a usable solver for radiative transfer applications. Of course the choice of prameters for the S2S model means we are aware the proposed solver is applicable to a specific set of industrial processes, and this thesis does not claim to have developed the end-all and be-all radiative solver.

Figure 4.1 displays the three geometries we will use for our simulations. Figures 4.1a and 4.1c have already been seen in Chapter 3: they are the ferrule and the hat, respectively. Figure 4.1b shows an assortment of two elongated bricks and one cylinder standing next to each other. Figure 4.1a is interesting as it presents a lot of autoradiation, since it is hollow. Figure 4.1b shows how different items standing close to each other radiate between themselves and obstruct radiative exchange with the outer environment. Figure 4.1c is particularly interesting since we have been provided experimental measurements during the cooling process of that shape. We will thus be able to evaluate how accurate the proposed solver is when faced with a real industrial situation.

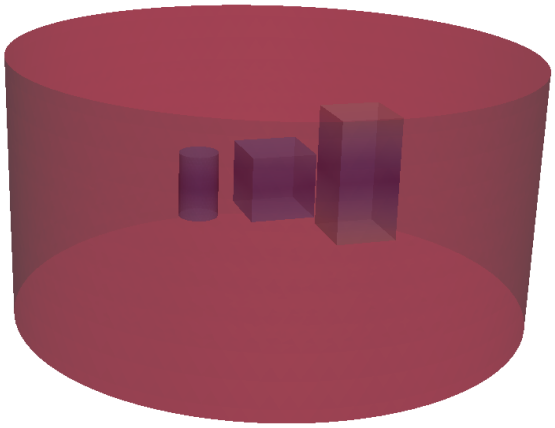
We will now proceed to detailing the context of simulation for each of these test cases, and afterwards we will detail the results of the computation for each. The results of this chapter have been obtained in collaboration with the company Sciences and Computers Consultants (SCC), to whom we are very grateful for their cooperation. These reference and industrial test cases have been pivotal in the validation of the proposed solver.

4.2 Sintering oven

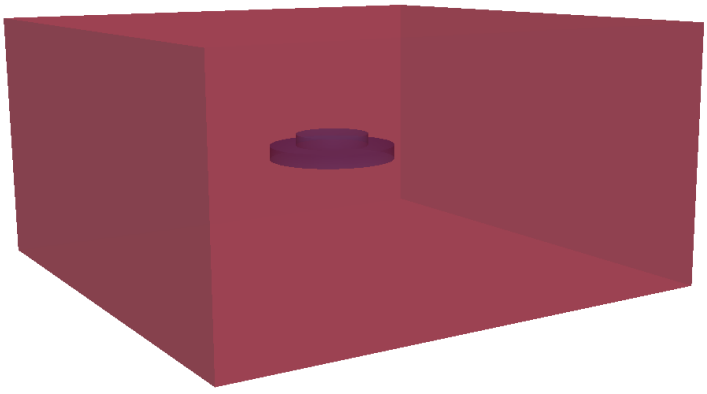
We do not possess all the detailed information of the furnace configurations that have been used by the industrial partners. However, we have been able to visit a ceramics manufacturer which possesses ceramic sintering ovens. Their configuration



(a) Ferrule



(b) Multiple elongated pieces



(c) Hat

Figure 4.1: All three geometries used for an end-to-end simulation test case.

can give us broad insights into the specifics of ceramic sintering and subsequent cooling. These ovens look like cylindric or parallelepipedic boxes of various sizes (from 50 cm to 4 m high). A modelling can be seen on Figure 4.2 (already shown in Chapter 1 but replaced here for convenience).

Figure 4.2: Vacuum induction-based sintering oven ([1]).

We have been able to look at the interior of one of the cylinder-shaped ovens. Inside each oven a series of horizontal discs is inserted at various heights, thus defining several stacked compartments that all behave as small inner ovens. The ceramic items are placed on each of these disks. This setup allows to sinter several different items that can be composed of different ceramics while preventing the samples to pollute each other or obstruct each other too much from the radiating walls. The lateral walls are laced with electrical coils. During oven operation, the coils heat the bulk of the wall through induction. The inner face of the lateral walls is coated with graphite, which has a very high emissivity (upwards of 0.9). Graphite is thus heated and then inwardly radiates heat towards the ceramics located within the oven. The lid and the bottom of the oven are water-cooled. A pump is located at the bottom of the oven. It is used to maintain vacuum, or to exert a given gas pressure, depending on which ceramics is being sintered. For example, silicon carbide ceramics needs to be sintered under a vaccum, while silicon nitride ceramics

needs 10 bar of nitrogen in order to replace the nitrogen that leaks from the ceramics itself during sintering. Figure 4.3 illustrates the setup we have witnessed.

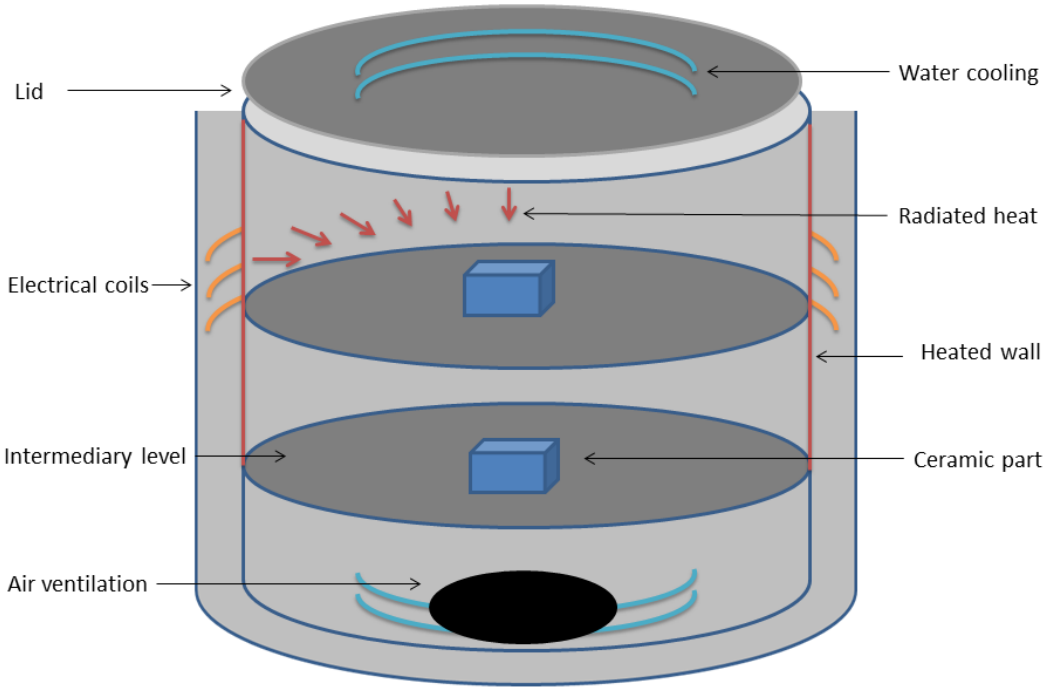


Figure 4.3: Schematics of a cylinder-shaped ceramic sintering oven.

As we had already stated in Chapter 1, typical sintering temperatures reach about 1600 °C to 2200 °C, depending on the type of ceramics and the size of the parts. While thermal transfer through radiation between the walls and the ceramics is dominant at these temperatures, conduction also occurs between the ceramics and the support on which they are set. In the case of nitrogen-based sintering, convection takes place as heated nitrogen interacts with the ceramics. A proper modelling of the sintering process would thus need to take into account all the modes of thermal transfer. A sintering process can take as long as 24 to 48 hours, during which the temperature ramp-up steadily processes during the first few hours. The oven is thus maintained at the desired temperature for an extended period of time, before being left to cool down. For a given ceramic piece to be sintered, it could be set inside the oven a number of ways: lying on the side, standing on top, at the centre of the inner oven, near the walls...these configurations matter a lot in the amount of radiative heating the piece receives.

While all these considerations and this oven are specific to the process of ceramic sintering as performed by TAS' supplier, they give us broad orders of magnitude of the types of physical values one can expect in industrial processes involving radiation. The following test cases are all situations of radiative cooling or radiative heating, where a hot part is left in a cooler environment or a left part in a locally hot environment. This is similar to what happens at the ramp-up or at the end of a sintering process, and these simulations will instruct us on how the load behaves during such a transient state situation. The three loads test case mimicks this specific oven as its environment, while the two other test cases' domains share the same broad characteristics (size, materials, etc.).

4.3 Simulation meshing and setup

We will now detail how we have modeled the geometries shown in the previous section.

4.3.1 Ferrule setup

The ferrule as shown on Figure 4.1a is a hollow cylinder situated inside a cubic domain. It is not located in the middle of the domain, but very next to one of the walls. The ferrule as well as a specific portion of the wall located in front of the ferrule are heated to 1000 °C, while the rest of the enclosure is not. When simulation begins, the ferrule is left to cool off in the environment through radiation and conduction in its bulk. There is a vacuum in the box, which explains why we do not consider convection. This setup, while not found as-is during sintering, teaches us how the ferrule will react to a strongl heterogeneousness in the environment temperature.

The simulation is a transient-state conduction-radiation 3D test case, similarly to the case of the radiating spheres presented in the analytical results of Chapter 2 subsubsection 2.4.2.3 (except they were in steady state). The aim is to assess the physical realism of the simulation of radiation cooling with a hollow object. There is a great amount of self-radiation and obstruction in this test case; given that obstruction is the most complicated part of radiation to handle, we estimate this geometry to be an interesting test for the proposed solver. The specific physical and geometrical information for this setup is given in Table 4.1.

The problem is solved using a body-fitted mesh for the ferrule. Since we only need to solve conduction in the ferrule, here, there is no need for a volumic mesh for anything other than the ferrule. This means there is no global volumic mesh

Geometry	Cylinder	Cubic domain	Heated square
Dimensions (m)	Inner radius: 0,5 Outer radius: 1 Height: 3	Edge size: 10	Height: 2 Width: 2 Thickness: 0.1
Centre position (m)	(-3.5 ; 0 ; 0)	N/A	(-5 ; 0 ; 0)
Emissivity	0.99	0.99	0.99
Starting temperature (°C)	1000	20	1000
Conductivity ($\mathrm{W\,m^{-1}\,K^{-1}}$)	6.5	N/A	N/A
Density ($\mathrm{kg\,m^{-3}}$)	8200	N/A	N/A

Table 4.1: Geometrical and physical values of the ferrule test case.

for the environment. We simply define a surfacic mesh for the box. Each surface element of the two meshes is a participating facet for S2S radiation. The mesh is highly refined at the location on the wall which is heated, as well as on the side of the ferrule facing the wall. Figure 4.4 shows the obtained mesh. The simulation is then ascribed to computing 1000 s of real-time cooling, with time steps of 1 s each. Table 4.2 sums up these informations.

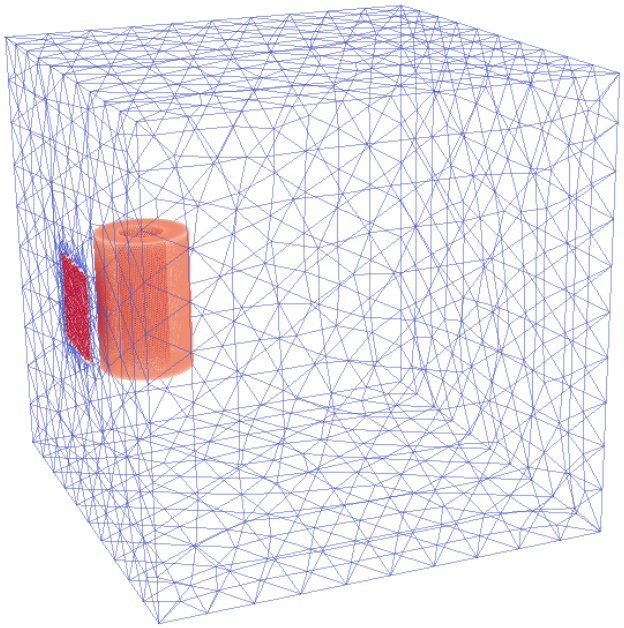


Figure 4.4: Ferrule and cubic domain mesh.

Mesh size (m)	Domain: 0.5 Cylinder and heated square: 0.05
Number of radiating facets	108685
Physical process duration (s)	1000
Time step (s)	1
Output file saving frequency	50

Table 4.2: Numerical values of the simulation of the ferrule cooling.

We do not have access to experimental results of the process our simulation is emulating. However, the good quality of our analytical comparisons gives us confidence in the physical plausibility of the results we are going to obtain from this simulation. Hence, while we cannot outright tell the degree of thermal accuracy our result will present, if the simulation behaves in a satisfactory way we will use this result as a benchmark test case comparison with later, improved versions of the proposed S2S solver.

What we expect is to witness that the inner surface of the cylinder cools down much slower than the outer surface, as the inner facets are almost completely obstructed from exchanging radiation with the cold facets of the environment wall. We also expect to see the side of the cylinder facing the heated square to cool down much slower than the other side, since it is directly exposed to strong thermal radiation.

4.3.2 Three loads setup

The three loads as shown on Figure 4.1b are a cylinder, a cube and a parallelepiped. They are set upright in the middle of a cylindrical domain. In this simulation, the domain is considered under a vacuum, meaning once again we solve for conduction and radiation but not convection. The system is set at an initial temperature of 20 °C, except for a band circling around the lateral edge of the domain which is instead set to 1200 °C. That band's temperature is set as a boundary condition throughout the simulation. The band is modeled as a very thin hollow cylinder superimposed with the domain at a specific height. In the meantime, the ceramic parts inside progressively heat as the simulation progresses. The aim of this setup is to model what happens in the type of sintering oven that has been displayed on Figure 4.3. We reproduce the lateral-only heat source, the vacuum, the shape of the oven, and the possible obstruction and interaction between the multiple items being sintered at the same time in the same inner oven. Tables 4.3 and 4.4 displays the geometrical and physical values used during this simulation.

Geometry	Cylinder	Cube	Parallelepiped
Dimensions (m)	Radius: 0.15 Height: 0.5	Edge size: 0.5	Height: 1 Width: 0.5 Thickness: 0.5
Centre position (m)	(-0.6 ; 0 ; 0.75)	(0 ; 0 ; 1)	(0.7 ; 0 ; 1)
Emissivity	0.99	0.99	0.99
Starting temperature (°C)	20	20	20
Conductivity ($\mathrm{W\,m^{-1}\,K^{-1}}$)	6.5	6.5	6.5
Density ($\mathrm{kg\,m^{-3}}$)	8200	8200	8200

Table 4.3: Geometrical and physical values of the three parts test case.

Geometry	Heated wall section	Cylindric domain
Dimensions (m)	Inner radius: 1.95 Outer radius: 2.05 Height: 0.666	Radius: 2 Height: 2
Centre position (m)	(0 ; 0 ; 1)	N/A
Emissivity	0.99	0.99
Starting temperature (°C)	1200	20

Table 4.4: Geometrical and physical values of the three parts test case (continued).

We once again use a body-fitted mesh approach with a volumic mesh for each of the three parts, and a surfacic mesh for the environment. The mesh is refined on the heating band of the oven as well as on the surface of each of the three parts. It can be seen in Table 4.5. The computation simulates 1000 s of cooling with time steps of 1 s. This information can be found in Table 4.5.

Mesh size (m)	Domain: 0.1 Cube, parallelepiped and heated band: 0.025 Cylinder: 0.01
Number of radiating facets	325401
Physical process duration (s)	1000
Time step (s)	1
Output file saving frequency	50

Table 4.5: Numerical values of the simulation of the ferrule cooling.

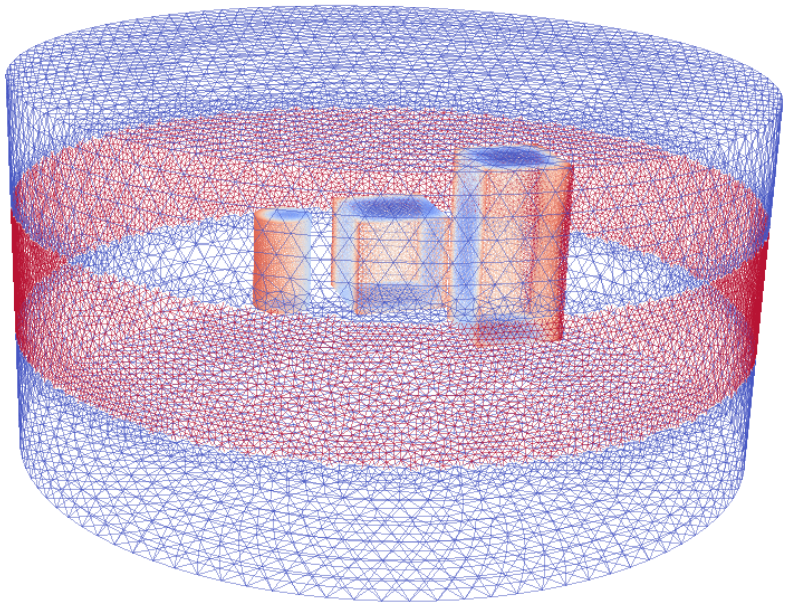


Figure 4.5: Three parts and cylindric domain mesh.

While a sintering process can take upwards of two days, we did not want to try to simulate such a long amount of physical time. The simulation would probably exceed the 128 Gb available RAM we have on the computer that was used for these simulations. In addition, we do not know the exact thermal ramp-up that is used for ceramic sintering, so we could not have simulated the real process anyway. However, we estimate that this 1000 s heating at 1200 °C can serve as a basis to model the hours-long steady state situation of the sintering once the ramp-up is over.

We do not have access to experimental measures for the physical process we are trying to model here, but we expect to receive them in an upcoming partnership with an industrial company. The results of this simulation, if satisfactory, will also be used as benchmark results to evaluate the performance of later S2S solvers we will develop. For example, we will pay close attention to thermal propagation in the bulk of the loads, since discussion with a ceramic maker has indicated this might be one of the most impacting factors in a proper and default-free sintering process. In particular, we expect to observe the central cube heating much less than the two other loads since they obstruct it quite a lot from radiation emitted by the heated band.

4.3.3 Hat setup

The hat on Figure 4.1c has already been introduced in Chapter 3 subsection 3.4.1. In this chapter we only focused on the time performance of the proposed solver in parallel computation with a varying number of processors. We will now detail the physical model of the hat as we have done for the previous two test cases. The hat is plunged in a parallelepipedic environment filled with air. It is initially completely heated to a temperature of 1160 °C while the environment is set for the complete simulation at 20 °C. It is then left to cooling for 30 min. Air is taken as nonparticipating in radiation, as is satisfactory with the hypotheses taken in Chapter 2. But Navier-Stokes is resolved in the volumic mesh defining the complete domain, and then the air velocity field accounts for a non-null convective term in the heat equation solved at each time step. This means this simulation does take into account all three forms of thermal transfer. The physical and geometrical information for this setup can be found in Table 4.6.

This time, since we take convection into account, we need a volumic mesh for the volume. This means, of course, that we also solve conduction through the air in the environment. We use a body-fitted mesh for the hat. The mesh is refined for the hat compared to that used in in the parallelism study: from about 60k to 250k facets. We compute a physical duration of 1800 s, with a time step of 1s. This information can be found in Table 4.7. The surface mesh itself can be seen on Figure 4.6.

Geometry	Hat	Parallelepipedic domain
Dimensions (m)	Upper radius: 0.145 Upper height: 0.05 Lower radius: 0.25 Lower height: 0.055	Height: 1 Width: 2 Thickness: 2
Centre position (m)	(0 ; 0 ; 0)	N/A
Emissivity	0.99	0.99
Starting temperature (°C)	1160	20
Conductivity ($\mathrm{W\,m^{-1}\,K^{-1}}$)	6.5	N/A
Density ($\mathrm{kg\,m^{-3}}$)	8200	N/A

Table 4.6: Geometrical and physical values of the hat test case.

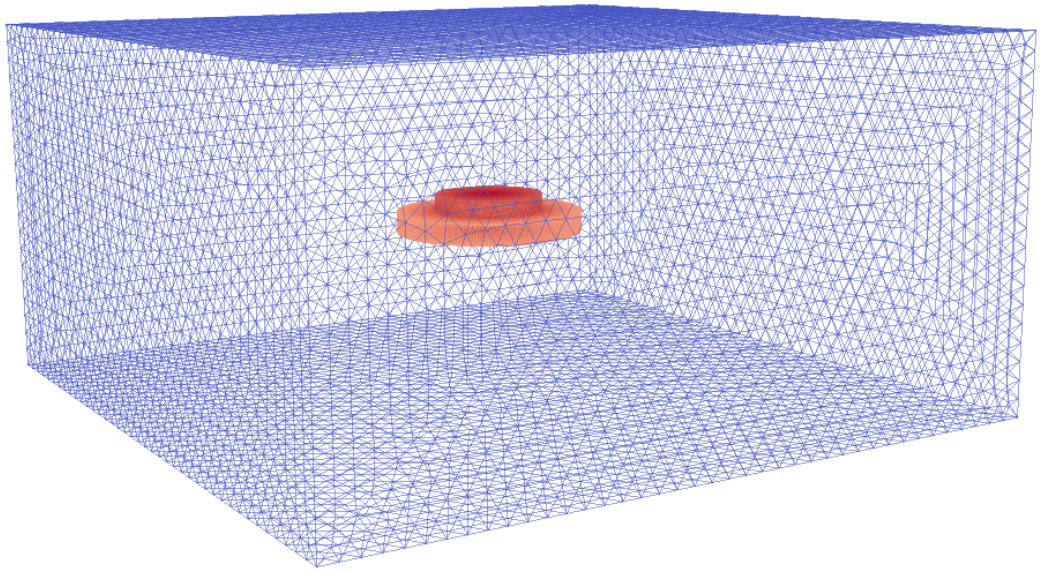


Figure 4.6: Hat and parallelepipedic domain mesh.

Mesh size (m)	Domain: 0.1 Hat: 0.025 Cylinder: 0.01
Number of radiating facets	251789
Physical process duration (s)	1800
Time step (s)	1
Output file saving frequency	60

Table 4.7: Numerical values of the simulation of the ferrule cooling.

This test case is of great interest to us because Safran Aicraft Engines, the company crafting the hat, has provided us with experimental results of its cooling process. This means we will be able to directly compare our results with real values. This is the only set of experimental data we have been able to obtain for comparison purposes in this thesis. After having ensured analytical comparisons with simple test cases were favourable, comparing with experimental data was the missing step in our validation process. While the process here is not a sintering operation (we have not been able to obtain experimental data for this process), this radiative cooling process is similar enough to still bring us useful insights.

4.4 Results and discussion

We will now display the results of the simulations for the three test cases we have previously introduced. We will comment each and detail what they teach us about the capabilities of our S2S solver.

4.4.1 Steady state ferrule results

The results of the simulation for the ferrule will be studied both at the final time step and during the course of the simulation. At the end of the simulation, we extract the temperature and the radiant exitance on the surface of the ferrule in order to have a snapshot of the thermal transfers that have been taking place. Figure 4.7 shows the flux on the left and the temperature on the right after 1000 s of physical cooling.

The first observation we can make is that the received flux density by the ferrule is negative everywhere. This means the ferrule actually emits more radiation than it receives. This makes sense, since the ferrule is the hottest item in the domain. Then, flux density is at its lowest magnitude (in absolute value) for the portion of the ferrule that was facing the heated square. This is expected, as that means

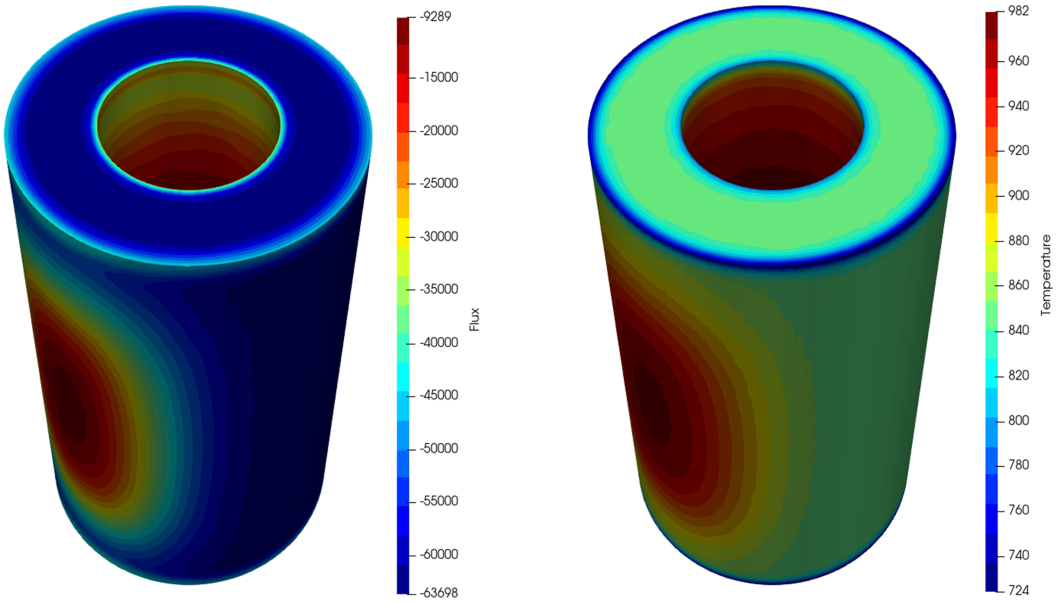


Figure 4.7: Flux density (on the left) and temperature (on the right) at the surface of the ferrule after 1000 s.

that portion of the ferrule has mostly exchanged with a body the same temperature than itself. For the same reasons, we can witness the received flux density for the surfaces of the ferrule located on its inner side are significantly lower (in absolute value) than that of the surfaces located on its outer side. The deeper inside the ferrule we go, the lower the received flux density. As expected, a radiating facet on the inner side but located near one of the ends of the ferrule will still "see" a fair number of radiating facets belonging to the environment wall and standing at 20 °C. But a radiating facet located on the inner surface near the middle of the length of the ferrule will mostly exchange with other radiating facets of the inner surface standing at a temperature very close to its own. The inverse reasoning holds true: facets having the highest (in absolute value) amount of received flux density are those standing on the exterior side of the ferrule that faces directly away from the heated patch on the wall. Indeed, these facets do not suffer any obstruction from others of the ferrule, and do not exchange at all with the heated patch on the wall. The ferrule being convex at this point, they only exchange with facets of the cool environment.

More surprisingly, the edges of the circles, both interior and exterior, that define the flat ends of the cylinder, exhibit an amount of radiant exitance that is significantly lower (in absolute value) than that of the flat surface of the end of the ferrule immediately next to them. One could even suppose that, in particular for the ex-

terior edges, these areas have three-quarters of a space available before them for radiation, with only a quarter of space occupied by the ferrule behind them. On that basis, they should be able to output much more flux density than the facets located in the flat area near them, where the ferrule blocks off the complete half-space behind them. One has to recall, however, that the individual facets are modeled as 2D surfaces. This means they can never radiate in the half-plane behind them, regardless of whether that half-space is actually occupied by obstructing matter or not. But if one looks at the isotherm on the right of Figure 4.7, one sees the temperature at these outer and inner edges is significantly lower than that of the flat surfaces around them. This, in turn, means these surfaces produce a significantly lower black-body radiation. This justifies these lower received flux density values.

Figure 4.8 displays the temperature field at the end of the simulation inside the ferrule. One can see the bulk of the ferrule is still at a temperature nearing 1000 °C: this means that after almost 15 min of cooling, it is still almost at starting temperature. This is true even for the side of the ferrule that was opposed to the heated patch: we can see, on the right, that the surface temperature of the ferrule is nearing 850 °C, which is much lower than the surface temperature of the left side of the ferrule which exceeds 940 °C at the areas facing the heated patch. The first conclusion is that relying exclusively on radiation to ensure the cooling of the ferrule entails the process will last for hours. Besides, there is a strong thermal gradient (approximatively 1280 $\mathrm{K\,m^{-1}}$) through the ferrule as we can see temperature varying from almost 1000 °C (in the bulk) to about 680 °C (at the outer top edge) within about 25 cm. That might have an impact on ceramic shrinking. This shrinking occurs during sintering as temperature increases; that would mean different degrees of shrinking within a short distance inside the ceramics. That would likely entail unwanted residual constraints and an heterogeneity of grain size. We will see the extent to which taking convection into account in the hat test case will allow a piece of similar size to cool down faster and/or at a more homogeneous rate.

4.4.2 Ferrule temperature evolution

These considerations have been made by looking at the state of the ferrule at the very end of the simulation. Let us see how the evolution of thermal transfer looks like in transient state. To this end, the ferrule has been instrumented with 18 numerical probes at different locations in order to retrieve temperature in various points of interest on the geometry. Figures 4.9a and 4.9b indicate where the probes are located in 3D and 2D view, respectively. The probes correspond to a combination of zones of interest where radiation is expected to play different roles: at the open top and bottom (probes 1 to 6 and 13 to 18), at the interior (probes 9 and 10), in the bulk of the ferrule (probes 8 and 11), facing (probes 1 to 3, 7 to 9, 13 to 15) or opposing (probes 4 to 6, 10 to 12, 16 to 18) the heated square.

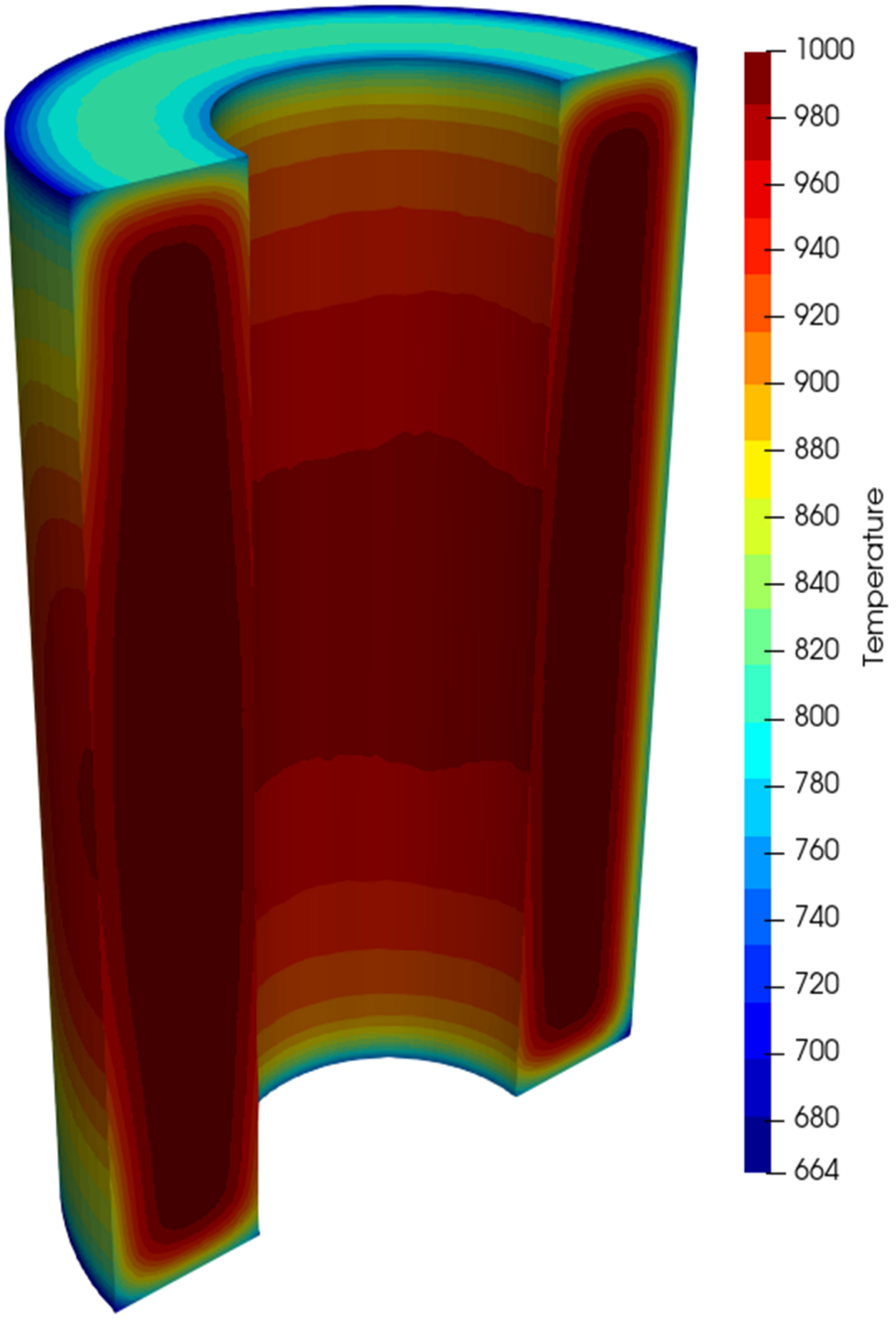
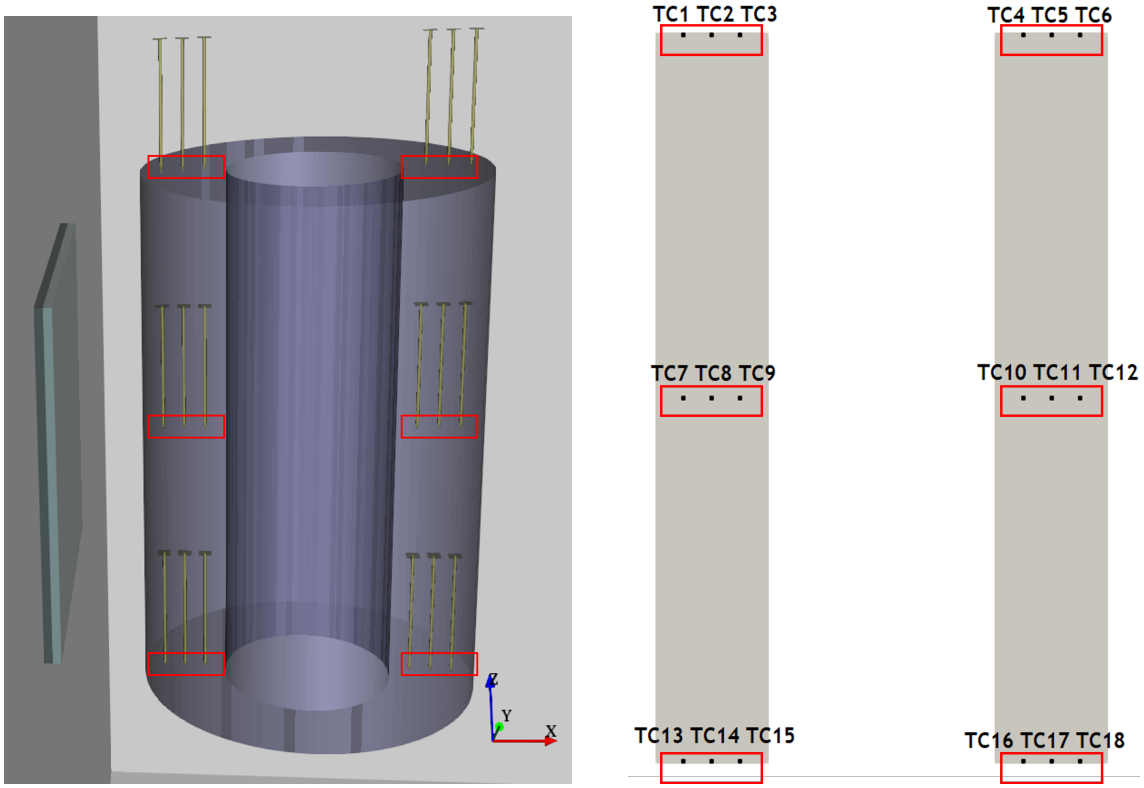


Figure 4.8: Temperature field inside the ferrule at the end of the simulation.



(a) Thermal probes in 3D view. *(b) Thermal probes in 2D view.*

Figure 4.9: Location of the numerical thermal probes on the ferrule. Temperature is collected at the end of the nail-like probes displayed on the left that is in a red rectangle. The points are reported in the same red rectangles on the right.

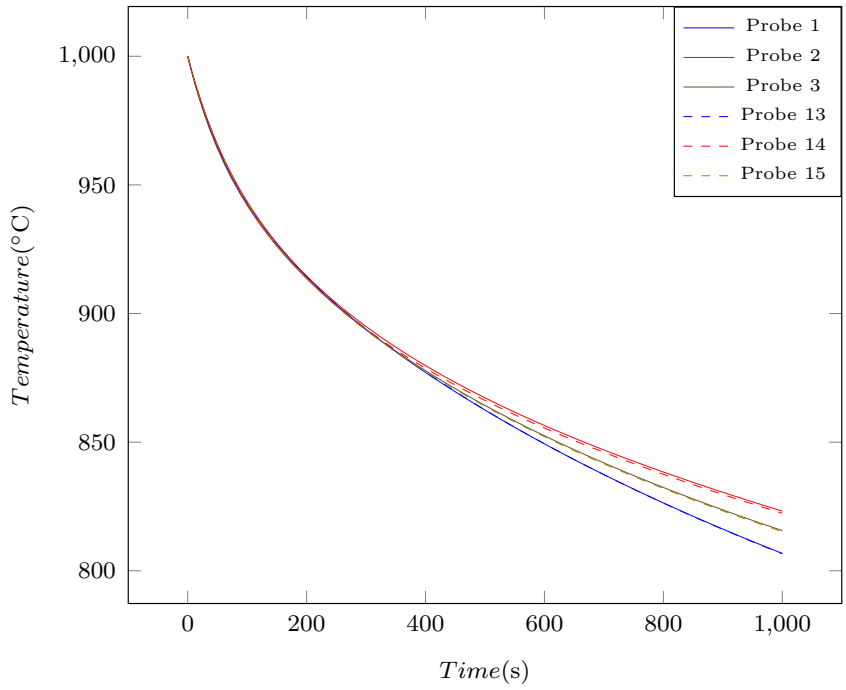


Figure 4.10: Temperature against time for probes 1 to 3 and 13 to 15.

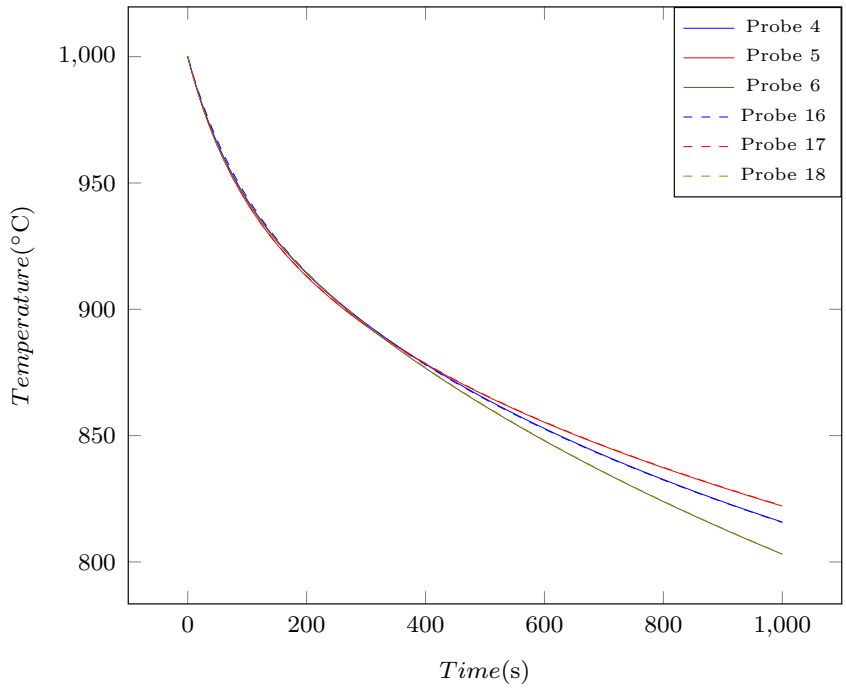


Figure 4.11: Temperature against time for probes 4 to 6 and 16 to 18.

Figure 4.10 displays temperature against time for probes 1 to 3 and 13 to 15. Since the heated patch on the wall is located mid-height to the ferrule, the complete system is symmetrical across the horizontal plane $z = 0$ cutting the ferrule in half. That means probes 1 and 13, 2 and 14, 3 and 15 form respective pairs of symmetric probes. As a consequence, we should expect each probe of a given pair to display the same temperature profile as its "twin probe". On Figure 4.10 probes 1 to 3 are shown on continuous lines while probes 13 to 15 are shown on dashed lines. Probe pair (1,13) is in blue, probe pair (2,14) in red and probe pair (3,15) in olive. We can see that for each colour, the respective continuous and dashed lines are nearly completely superimposed. This indicates that we indeed observe a symmetric temperature behaviour for the pairs of symmetric probes. Figure 4.11 displays the same temperature against time graph for symmetric probe pairs (4,16), (5, 17) and (6,18). On that figure too we can observe how the continuous and dashed line of any given colour are superimposed. This shows symmetry of physical behaviour is indeed obtained in the simulation.

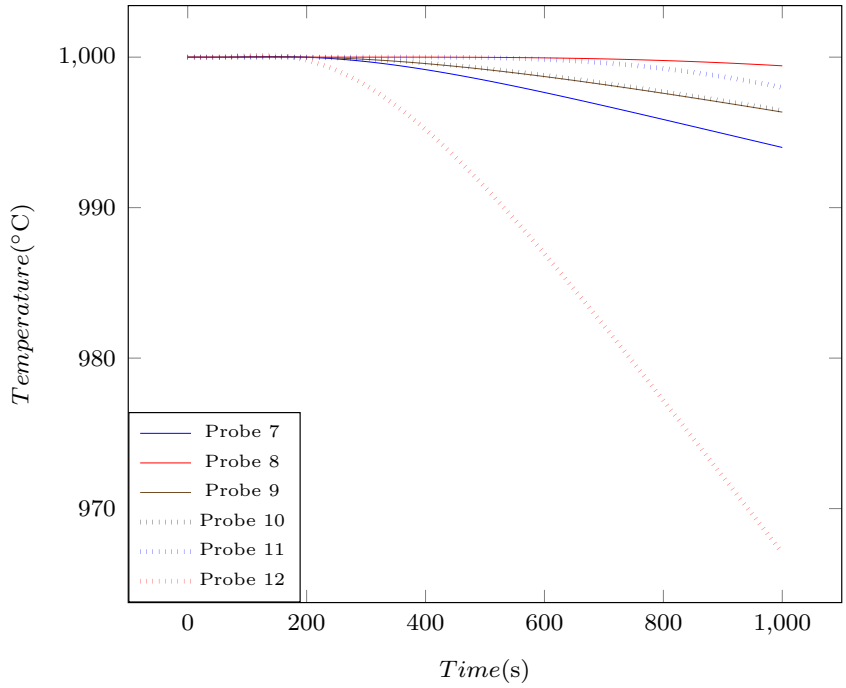


Figure 4.12: Temperature against time for probes 7 to 12

Figure 4.12 displays temperature against time for probes 7 to 13, that is, all the probes located in the $z = 0$ plane ordered from closest to farthest from the heated patch on the wall. For clarity reasons, probes 10 to 12 are displayed with dotted line to better separate them from probes 7 to 9. There is no symmetry at work here so there is no purpose in establishing pairs of "twin probes". We can see

that probes 9 and 10, however, display almost perfectly superimposed lines. This is despite the fact that probe 9 is under a greater influence from the heated patch than probe 10. Probes 9 and 10 being located on the inner side of the ferrule, this illustrates the consequence of all the facets located deep in the inner side of the ferrule exchanging radiation almost exclusively with other facets also located on that same inner side. The intense radiative exchange in that area is smoothing out all temperature difference and maintaining a state of local quasi-equilibrium. Probe 7, being directly exposed to the exchange with the heated patch, only loses about 6 °C at the end of the simulation. On the other hand, probe 12 shows a very significant cooling, as it exchanges with environment facets standing at 20 °C. We can notice its temperature does not start to diminish before about 200 s though. This might be due to the important thermal inertia of the ferrule starting out at 1200 °C.

On the other hand, let us examine probes 8 and 11. They are located in the bulk of the ferrule, probe 8 being on the side of the heated patch. Probe 8 is thus located between probe 7 and probe 9, none of them having lost more than 6 °C. Since it is in the bulk of the ferrule, it can only lose temperature through conduction, but given the very low thermal gradient it actually still stands at 999.4 °C at the end of the simulation. On the other side, probe 11 is located between probe 10 and probe 12, the latter being quite significantly cooled down to 967 °C over the simulation. We remark that despite this much cooler side, and as a result much higher thermal gradient, probe 11 has barely cooled itself anyway. It still stands at 998 °C at the end of the simulation. This shows how radiation is limited in its capacity to evacuate heat from the bulk of the material. A clear consequence is that a material filled with holes increases its total radiating surfaces and cools down faster, as can be seen for probes 9 and 10.

4.4.3 Steady state three loads results

We will now proceed to studying the results of the simulation of the three loads in the cylindrical oven. We display the temperature and the received flux density of the three loads at the end of the simulation after 1000 s of physical heating. Let us review temperature first. We recall that the simulation has started at 20 °C for the loads and the entire environment, apart from a band on the side of the environment heating at 1200 °C. Figure 4.13 displays the temperature field.

We can observe the temperature field is much more heterogeneous than what we saw for the ferrule. The effect of obstruction can be seen very clearly: the side of the parallelepiped facing the cube shows an area that has heated up to about 90 °C while immediately next to this the surface has heated to about 200 °C. We can see the cube has effectively obstructed a lot of the radiation that could reach that area of the parallelepiped. It has heated nonetheless through conduction received by other areas of the solid that have heated more. Another example is the top area of each

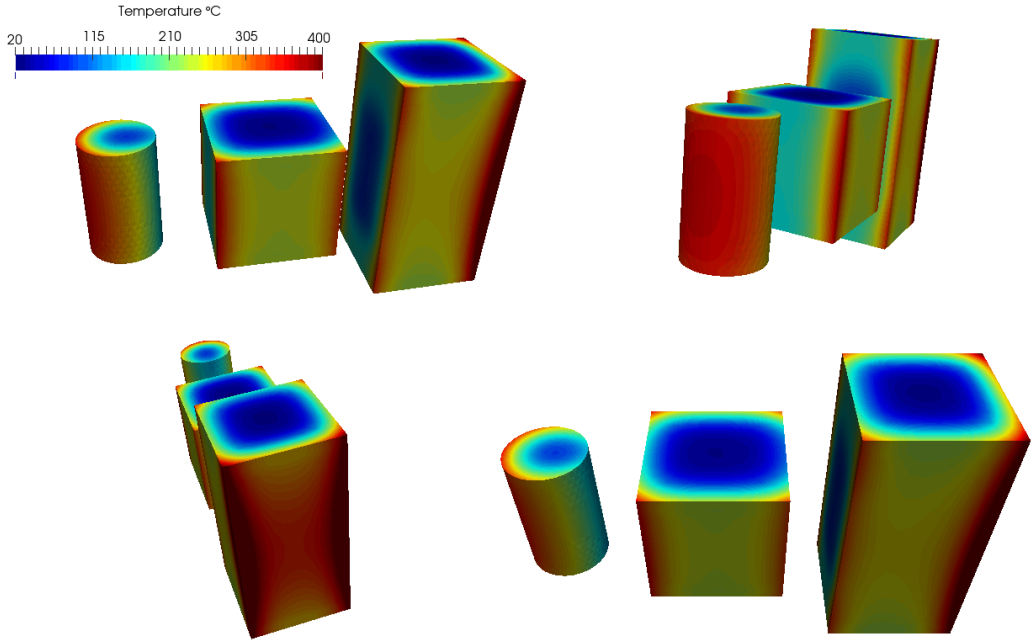


Figure 4.13: Temperature for the three parts after 1000 s *of heating.*

of the three solids. The band of heated wall only extended to the exact same height than that of the parallelepiped. As such, there was no direct view from the top of the parallelepiped to the heated band, and there was a very small, very inclined solid angle of vision from the tops of the cylinder and cube to the heated band. The face of the cylinder that was oriented towards the heated band presents a temperature profile that reminds us of that of the ferrule in the same configuration. The vertical edges of the cube and parallelepiped are the hottest areas of these solids. This can be explained by the fact they have received the maximum amount of radiation from the heated band, the same that the middle area of the sides of these solids have received. But then, they are in contact with a quarter-space of solid, instead of a half-space of solid compared to the middle area of the sides. As a result, they have been less able to transport heat inside their respective solids by conduction. This products a thermal gradient that is even stronger than what we saw in the ferrule: for the parallelepiped, we see temperature varying from about 20 °C to 400 °C within 20 cm, for about 1900 K m^{-1}.

Figure 4.14 shows the flux density of the loads at the end of the simulation. We can notice something quite peculiar, that was not present for the ferrule. Some areas receive a positive amount of flux density, while some others a negative amount. This means that within a same solid, some areas are net emitters of radiation while some

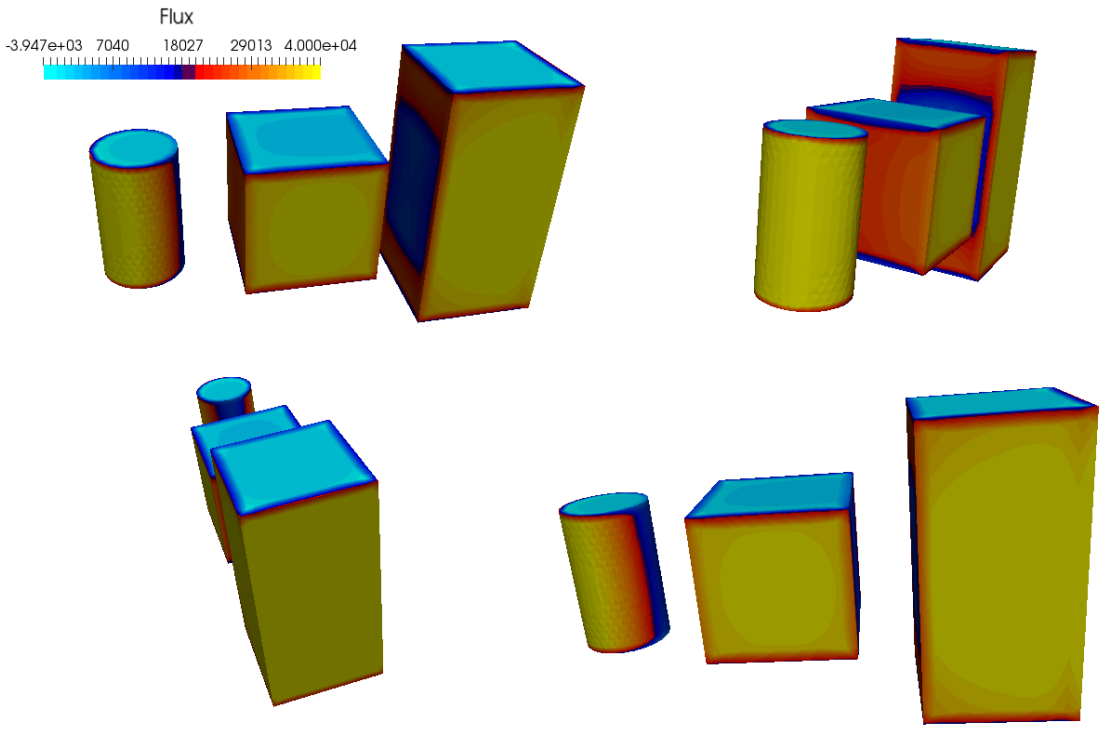


Figure 4.14: Received flux density for the three parts after 1000 s *of heating.*

others are net receivers. While odd, this situation can be explained as follows. The areas that are net emitters of radiation are the top surface of each solid. As we have seen, these surfaces have zero or next to zero exchange with the heated band on the wall. This means they almost exclusively exchange with the "ceiling" of the oven, which is set to 20 °C. However, by conduction, these surfaces are heated above 20 °C themselves. This means they end up exchanging almost exclusively with facets that are colder than they are. The result is that there is a net loss of thermal radiation of the pieces through their top surface towards the ceiling of the oven. All the while, the parts do gain thermal radiation from their lateral surfaces. This is why we can see very high flux density gradients around the edges of the top surfaces of the solids: we go from about $-4\,\mathrm{kW\,m^{-2}}$ to about $20\,\mathrm{kW\,m^{-2}}$ within a few centimeters.

This complicated and strongly heterogeneous flux density field leads us to believe the flux density and the temperature field inside the loads must be very complicated to map and display strong gradients too. Given that some parts of the solids cool down while others heat up at the same time, this situation is more complex than the ferrule where cooling was happening everywhere, even if at different paces. The most immediate solution to smoothing off the process would be to put the pieces further apart, or to put only a single piece in each inner oven, if such a slower sintering rate would not be too much time-consuming. Besides, ensuring the heated band on the wall is significantly wider than the pieces are tall would make sure even the top and the bottom of the pieces have view factors with the heated band high enough to be net absorbers of radiation instead of net emitters.

4.4.4 Three loads temperature evolution

We will now study the evolution of temperature during the simulation with the aid of numerical probes. 9 probes have been placed, 3 for each solid. Figure 4.15 indicates the location of the probes. Since we have placed the probes in the bulk of the solids, we hope to gain insight we could not obtain through the surface temperature display of Figure 4.13.

Let us observe the results solid by solid. Figure 4.16 indicates the temperature graph for the three probes in the cube. We can see that even after 1000 s of heating, the cube has not heated that much. Even when placed in an oven where the heated band is at 1200 °C, the cube itself does not reach a temperature higher than 40 °C. Probe 1, being located deep in the bulk of the cube, displays a temperature increase of only 2 °C. Since it is far from the radiation-receiving external surface, it takes a long time for conduction to propagate up to the location of Probe 1. We can see its temperature starting to increase only after about 600 s of heating. This phenomenon is likely amplified by the fact the top of the cube is leaking heat through radiation, as was seen in the previous paragraph: in order to ensure flux conservation, the conduction flux is probably attracted to the top of the cube and thus is deviated

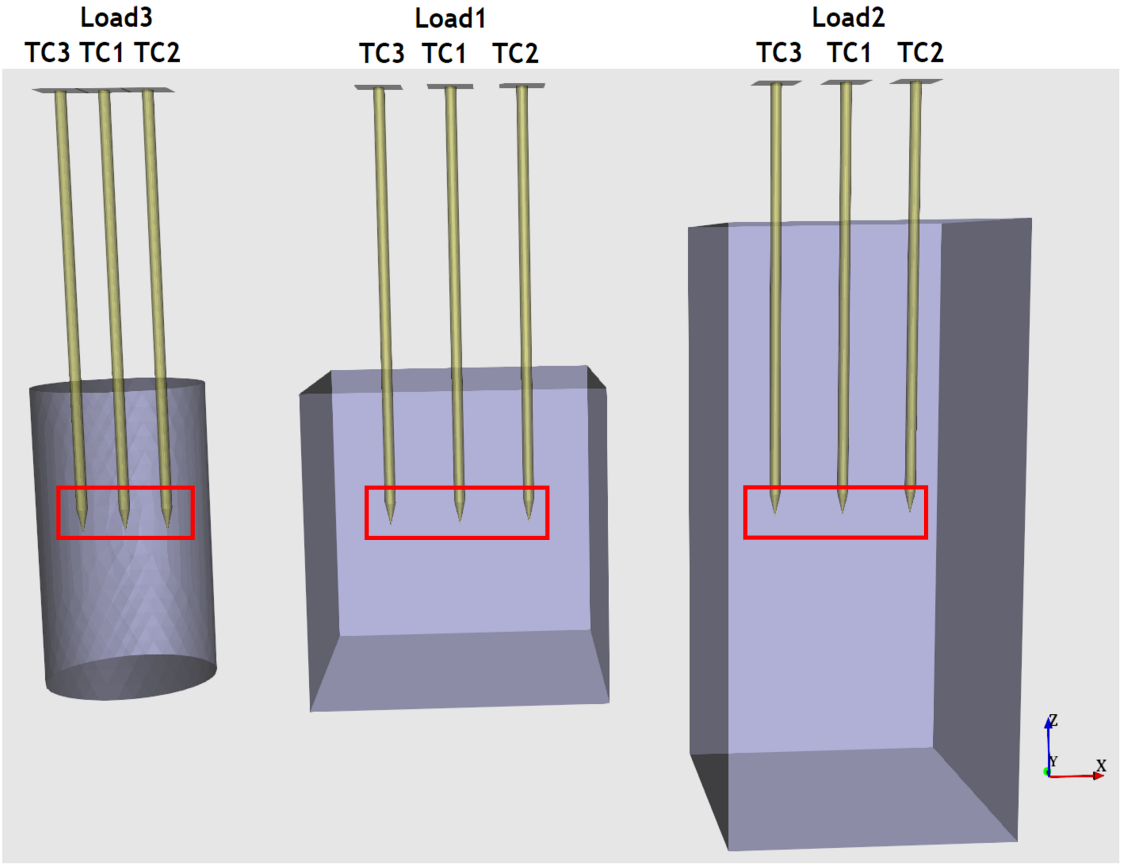


Figure 4.15: Location of the numerical thermal probes on the 3 parts. Temperature is collected at the end of the nail-like probes in a red rectangle.

from reaching the centre, where Probe 1 is located. Probe 2 and 3 display higher temperature increases than Probe 1, and their temperature starts to increase after a shorter period of time (about 200 s). This is expected since they are located closer to the external surface of the cube than Probe 1, so they receive more conduction flux. Probe 3 is at the side of the cube where the cylinder is located. It is at this probe that the temperature rises the highest. Since the cylinder is much smaller than the parallelepiped located on the other side, it obstructs the cube face near it a lot less. This explains the fact Probe 3 heats faster than Probe 2.

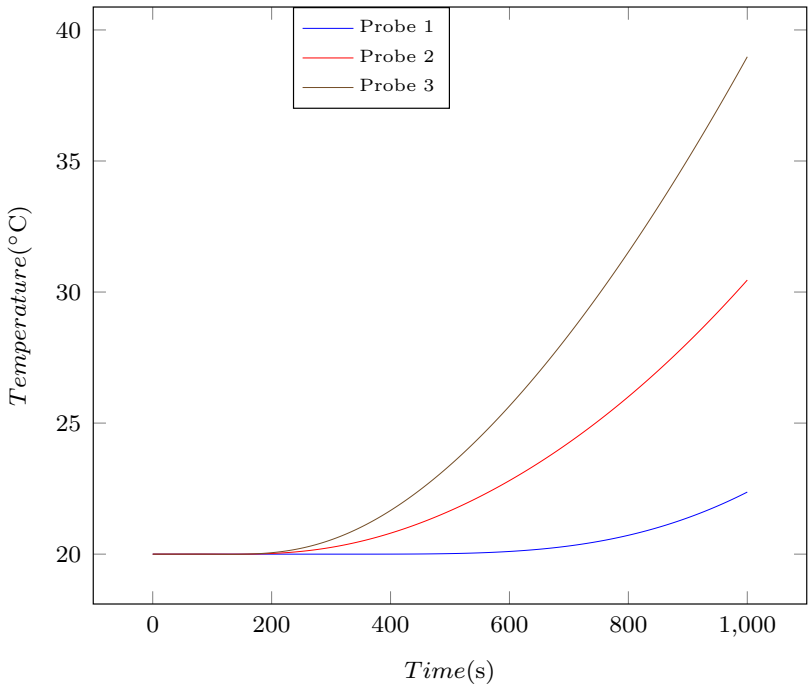


Figure 4.16: Temperature against time for probes 1 to 3 in the cube.

The parallelepiped's temperature against time graph is shown on Figure 4.17. Similarly to what we see in the cube, Probe 1 sees almost no heating at all for the first 600 s of simulation. Even at the end, it only stands at 23 °C. Probe 3 sees almost the same trajectory as Probe 2 of the cube: they both start to heat at around 200 s and end at 30 °C. This can be explained by the fact that Probe 3 is located in the side of the parallelepiped facing the cube, while Probe 2 is located in the side of the cube facing the parallelepiped. There is a lot of radiative exchange between the surfaces of the two solids that face each other, and as a result the temperature they reach in these areas are closely matched. On the other hand, Probe 2 is heated to a much higher degree than what the cube exhibits. Probe 2 is located on the side of the parallelepiped that is not obstructed by any item, and is directly facing the heated band of the wall. This explains this important thermal discrepancy.

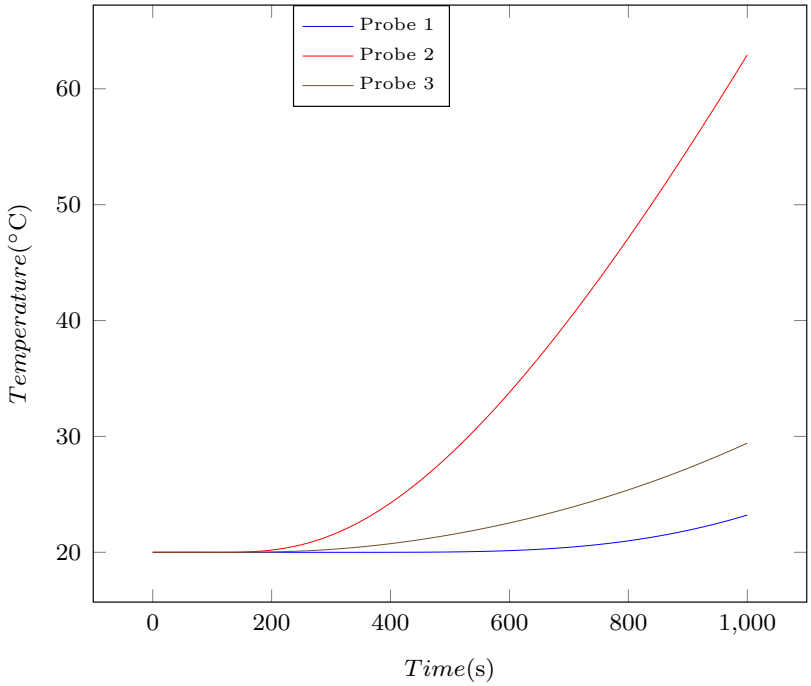


Figure 4.17: Temperature against time for probes 1 to 3 in the parallelepiped.

The temperature against time graph for the cylinder is shown on Figure 4.18. One instantly notices how the temperatures reached are much higher than for the cube and parallelepiped. The reason for this is that the cylinder is smaller than the other two loads. This means its surface to volume ratio is higher and its thermal inertia lower. For the other two loads, it took 200 s at the fastest for thermal flux to reach Probes 2 or 3 (the lateral probes), and 600 s to reach Probe 1. For the cylinder, we can see this happening as soon as 100 s for Probes 2 or 3, and 300 s for Probe 1. This indicates how thermal flux penetrates its bulk faster. This explains how Probe 3 reaches 180 °C at the end of the simulation where the hottest probe of the other two loads, Probe 3 of the parallelepiped, only reaches 63 °C. The second interesting behaviour we observe is that for the cylinder, Probes 1 and 2 follow a very closely matched pattern. This is because Probe 2 is located on the side of the cylinder that is obstructed by both the cube and the parallelepiped. As a result it receives a small amount of radiative flux density. This means this probe almost behaves as if it was located deep in the bulk of the solid: its behaviour is dictated by the conduction flux it receives more than by the radiative flux density.

The cylinder shows us the impact of the size of the object under sintering. Its volume is about one tenth that of the parallelepiped: this order of magnitude then creates a very different temperature profile in its bulk during the simulation. This is the case even if its position in the oven relatively to its environment is the same

than that of the parallelepiped: a load placed at the end of an alignment of items, standing upright, offset from the centre of the inner oven.

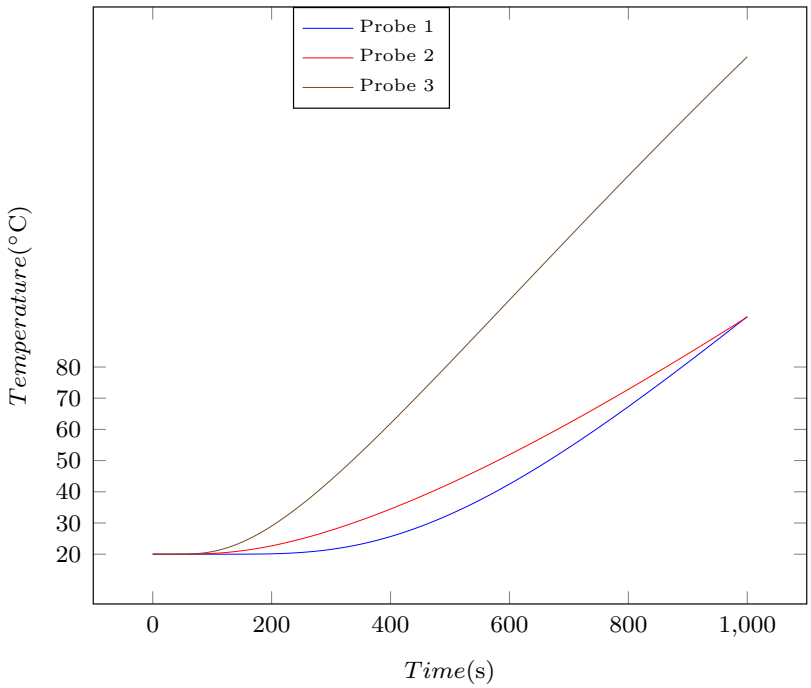


Figure 4.18: Temperature against time for probes 1 to 3 in the cylinder.

One last comment we can make about this test case is how the cube is the load where the temperature difference between all probes is smallest, due to its very obstructed position. This means, for sintering ovens, that if several pieces are to be sintered at once, choosing which one to put at the centre is a matter of some importance. It might be that if pieces of varying sensibility are sintered at the same time; it would then be advisable to put the most sensible piece at the centre. This would help smooth out the thermal gradient inside its bulk.

4.4.5 Steady state hat results

Our final test case is the hat. This subsection will develop the results obtained through simulation and compare them to the experimental results we have been given. The experimental values we have relate to the dynamics of thermal transfer in the hat, but we can still make some general observations by looking at the radiant exitance and temperature field on the hat at the end of the cooling. Figure 4.19 displays it.

We can see the hat temperature and flux follow the rotational symmetry of the geometry. We can also notice the coldest area of the hat is the lateral surface of its

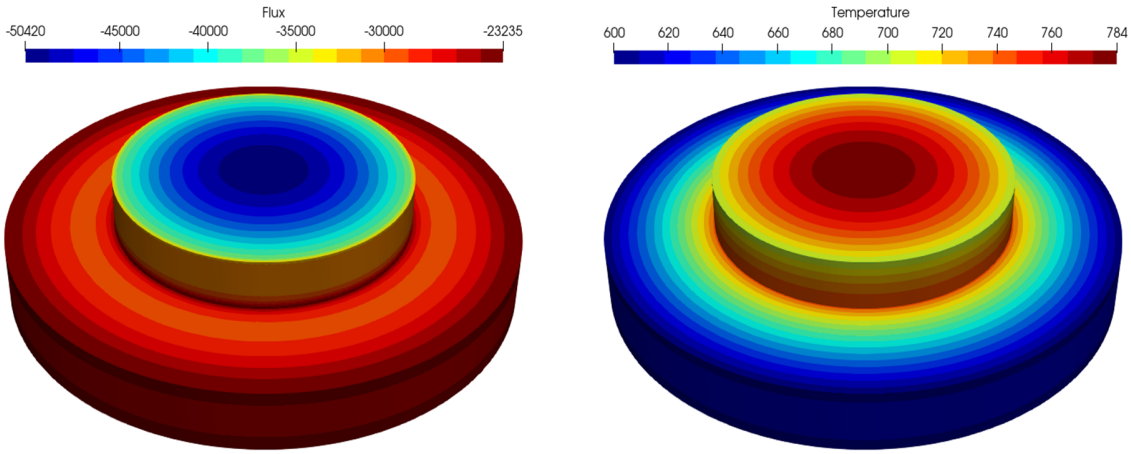


Figure 4.19: Radiant exitance and temperature fields on the hat after 1800 s *of cooling.*

lower cylinder. Since the hat is thinner at that place, there is less thermal inertia to combat and less heat brought by conduction from the bulk of the solid to replace heat lost by radiation. In addition to this, this area's surfaces are fully oriented towards the environment. By contrast, the facets located on the top of the lower cylinder can see and exchange radiation with the facets located on the side of the upper cylinder. This means these facets have less capacity to lose heat by radiation. We can see the coldest area within the lateral surface of the lower cylinder is actually composed of two thin rims circling around the top and bottom disks composing the lower cylinder. We think this is due to cooling by convection. Elements in this zone are backed with a quarter-space filled with hot hat and a three-quarter-space filled with cold air to exchange with. By contrast, elements right in the middle of the lateral area of the lower cylinder are located between a half-space of hat and only a half-space of cold air. The hottest part of the hat is predictably at the top centre of the top cylinder, where thermal inertia is the highest. We have seen in two previous test cases how the thickness of an item will severly impact its ability to cool down by radiation. One last thing of note is the fact there is a thin band on the top of the lower cylinder surface circling around the base of the top cylinder where temperature is higher than in the close vicinity. This is because facets there are very restricted for radiative exchange: they only have a quarter-space to emit radiation with the environment, and another quarter-space (out of the half-space available for radiation) is occupied by some other hot hat facets.

The flux density figure does not present any complicated interpretation situations. We see that the hottest facets are those who receive the lowest amount of flux density - that is, they emit the highest amount.

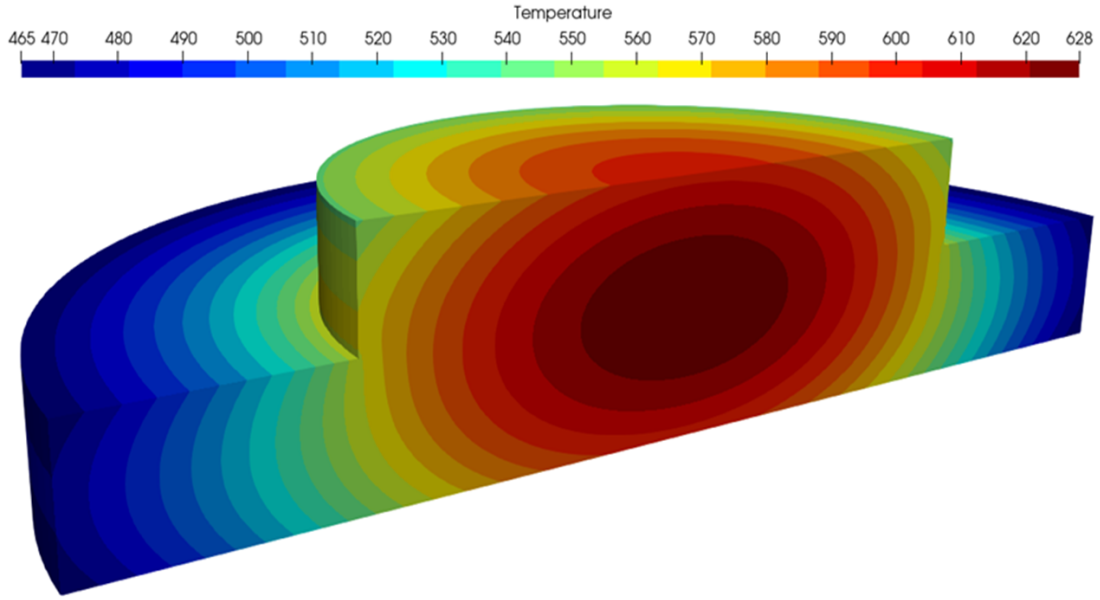


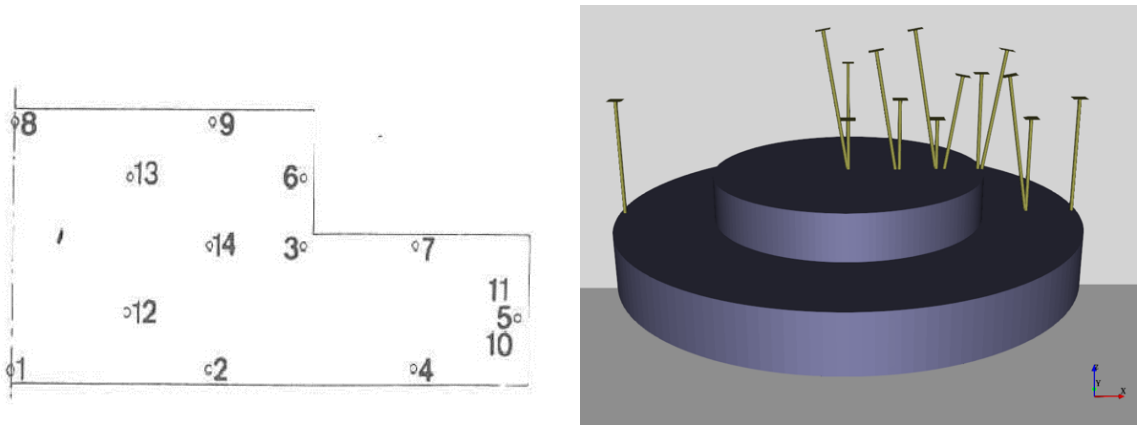
Figure 4.20: Bulk temperature fields inside the hat after 1800 s *of cooling.*

Figure 4.20 indicates on a sectional view of the hat the temperature field in its bulk at the end of the simulation. The temperature field is axisymmetric and logically the hottest part is right at the centre of the hat. We do not observe such strong thermal gradients as was the case for the ferrule: temperature varies of about 160 °C within 25 cm for a thermal gradient of about 640 $\mathrm{K\,m^{-1}}$, which is about half the thermal gradient we have witnessed there. We think this is the influence of convection, which participates in cooling the hat from every side equally. This leads us to think that ceramics sintered under a vacuum might be more at risk of important thermal gradients and possibly subsequent defaults that ceramics sintered under nitrogen atmosphere.

4.4.6 Hat temperature evolution

We will examine the results of the temperature probes that have been placed in the hat. SAE has provided us with the location of the physical probes used for the experimental measurement. We have thus located our numerical probes at the same location so as to be able to compare probe against probe. Figure 4.21 indicates the position of the probes. Figure 4.21a is the 2D probe chart that has been given to us by SAE while Figure 4.21b is a 3D view from above of the numerical probes in the simulated hat. 14 probes have been installed. Probes 10 and 11 cannot be seen on Figure 4.21a. This is because Probe 10 is Probe 5, but rotated by a half-turn around the Oz axis, around the hat. Probe 11 is Probe 5 but rotated a quarter-turn

around the Oz axis. Due to the axisymmetry of the hat, Probes 10 and 11 should behave the same as Probe 5. This is why they are indicated as being at the same spot than Probe 5 on Figure 4.21a. Four types of probe behaviour can be observed and we will study them in turn.



(a) 2D view *(b) 3D view*

Figure 4.21: Physical and numerical temperature probes placed in the hat.

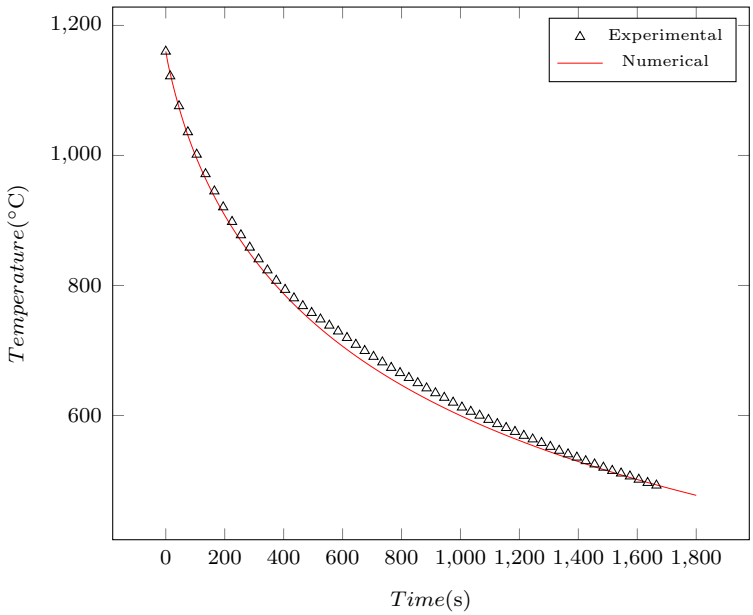


Figure 4.22: Hat probe 5.

Accurate probes Probes 3, 5, 7, 10 and 11 present a behaviour where the numerical computation matches the experimental data closely. They will be called the *accurate probes*. Figure 4.22 shows the behaviour of Probe 5. We can see the experimental temperature starts out with a strong decrease, before the cooling slows down after about 500 s of real time. This can be explained by the hat initially shedding a lot of heat through radiation. As its temperature diminishes, the amount of radiation these probes lose decreases rapidly (since it is roughly proportional to T^4). When temperature reaches about 700 °C, we can make the hypothesis conduction and convection are not dwarfed by radiation anymore and start to play a significant role in cooling, which explains why the cooling takes a more linear approach. The simulation manages to capture these evolutions well.

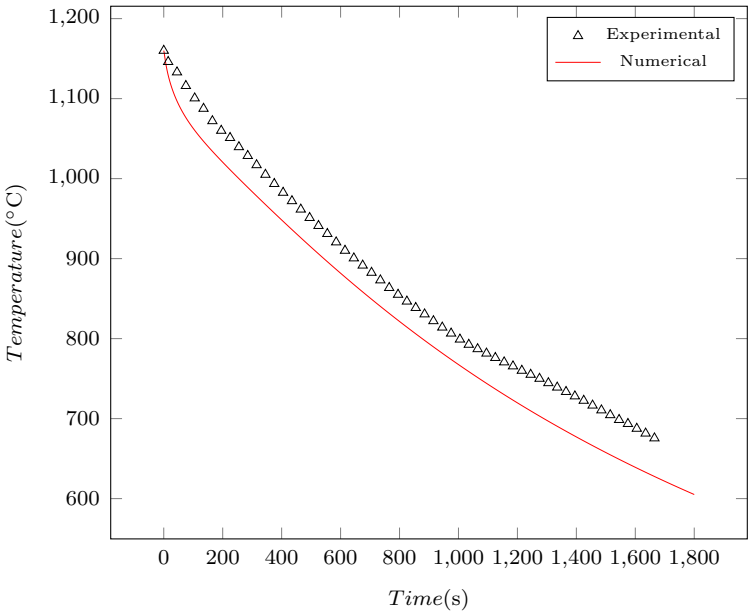


Figure 4.23: Hat probe 1.

Overcooling probes Probes 1, 2, 4, 8 and 9 will be called the *overcooling probes* as the simulation there is slightly colder than the experimental data. Probe 1 behaviour can be seen on Figure 4.23. Notably, the cooling starts with a higher slope. After about 100 s, numerical cooling slows down to run parallel to the experimental data. However, the excess loss of temperature that has been gained at the beginning of the simulation is never compensated. The simulation computes a temperature that is about 40 °C too cold after 100 s (1060 °C vs 1100 °C). From there on that difference remains more or less constant till the end of the computation: the difference at 1635 s is 49 °C (632°C for numerical data vs 681 °C for experimental data).

Since the temperature evolution was quite well rendered for the accurate probes, let us see what differs between the two sets that might explain the difference. The overcooling probes are all located either on the top surface or the bottom surface of the hat. By contrast, the accurate probes are located on the side of the lower disk of the hat or near the free top surface of the lower disk. The radiation conditions between the probes are fairly similar: in particular probes 5, 10 and 11 present zero self-obstruction, similarly to all the overcooled probes. This means the difference we observe might not be due to an error of radiation modelling.

There is one difference as far as conduction is concerned: the accurate probes are located in a zone where thermal inertia is lower, due to the fact they are further away from the bulk of the hat located under the top disk. The accurate probe's starting temperature slope is more abrupt than that of the overcooled probes. That might be because the overcooling probes are in a position where they can receive more conduction flux from the bulk of the hat to immediately replace the heat lost by radiation. As a result the overcooled probes' starting slope is smoothed out by this effect. However, while our simulations do exhibit the strong starting slope for the accurate probes, they *also* show a strong starting slope for the overcooled probes instead of the experimental smooth slope. Probe 3 is sorted among the accurate probes, even if it is located quite near the bulk of the hat, which would appear to contradict our hypothesis. However, it exhibits a thermal behaviour that is, among all accurate probes, closest to what we see in the overcooling probes. Therefore, we could make the assumption that we have somehow underestimated the thermal inertia of the hat. It might be because the real hat presents local variations of density and conductivity that our modeled hat fails to account for.

Good start probes Probes 12 and 14 are the two probes situated deepest in the bulk of the hat, and they exhibit a very similar pattern. Figure 4.24 displays the temperature profile of Probe 14. These two probes present a mix of the behaviour of the overcooling probes and the accurate probes. The dynamics of the very first time steps is correctly rendered, but then a slight overcooling takes place over time. They will thus be called the *good start probes*. On these two probes, one can see the experimental temperature starts out by almost zero cooling for the first 50 s. This is due to the fact these probes do not lose heat either by radiation or by convection. They have to wait for their surroundings to cool down before they lose heat by conduction. Once that process starts, it goes on with a slope that is much smoother than that of the surface probes. It is only starting from about 1000s that the simulation diverges from the experimental data by keeping the same slope, while the experimental temperature starts to lower at a slower pace.

One can conclude from this that conduction seems to be correctly rendered by the solver for these inner probes. Then, almost all the probes around them are from

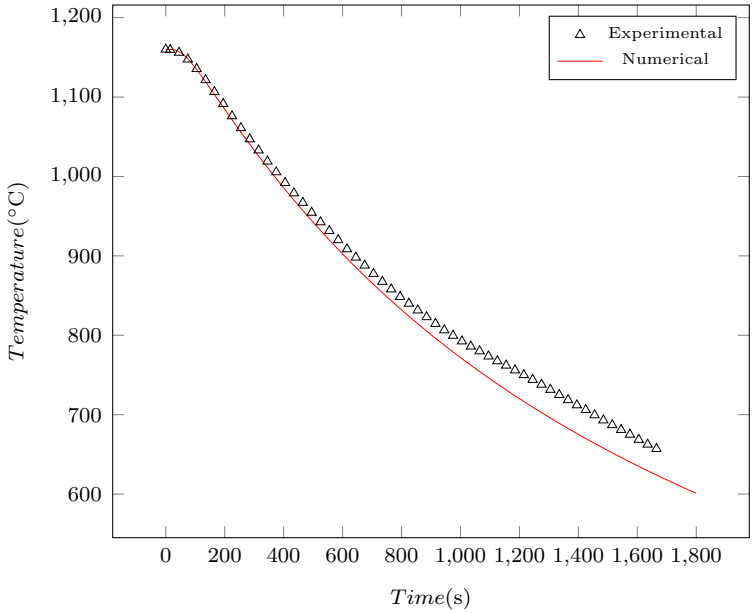


Figure 4.24: Hat probe 14.

the overcooling probes group. This overcooling propagates towards the centre of the hat after a sufficient amount of time has passed. On the whole, these probes's numerical results match reasonably well with the experimental ones. At 1635 s, Probe 12's numerical result is 46 °C too cold (635 °C vs 681 °C) while Probe 14's is 33 °C too cold (624 °C vs 657 °C).

Undercooling probes Probes 6 and 13 present a different profile: the temperature of the simulation exceeds that of the experience during the first 700 s or so, before adjusting itself closely with the experimental data. They will be called the *undercooling probes*. Figure 4.25 displays this behaviour for Probe 6. What we observe is the experimental probes display a sudden cooling right at the beginning of the experience, before the slope of the temperature becomes more gentle and the temperature diminishes slowlier. Probes 6 and 13 exhibit this strong thermal gradient at the beginning to a lesser degree. As a result, they do not cool enough at the start of the simulation. Over time, however, they present a temperature slope that is slightly steeper than that of the experimental data. They end up catching up with the measurements at around 800 s.

This result might seem a little baffling at first, since it displays the opposite behaviour from that of the overcooling probes. For Probe 6, the one difference there is compared to the overcooling probes is its radiating position. Near Probe 6, the radiating surfaces of the lateral area of the top disk are in exchange with the radiating

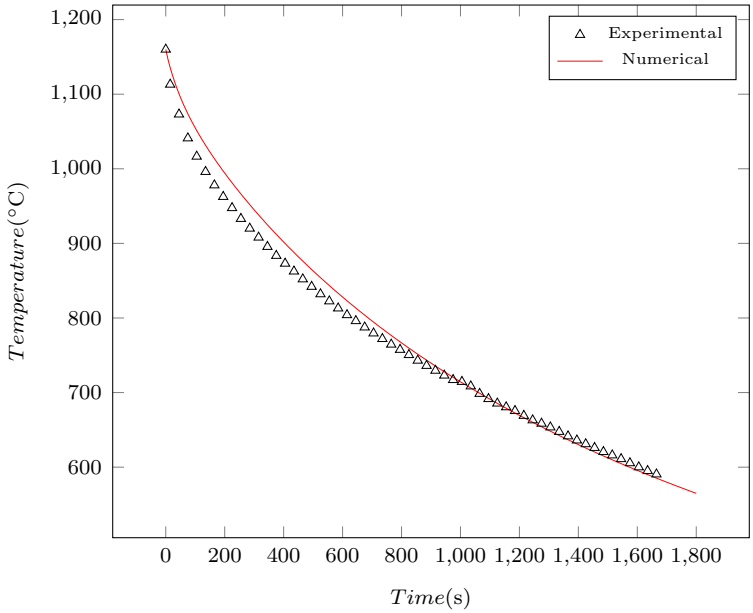


Figure 4.25: Hat probe 6.

surfaces of the top area of the bottom disk, near Probe 7. As a result Probe 6 should exhibit less emitted flux density - as was shown on Figure 4.19. It might be that we overestimate the view factors between the areas of Probe 6 and Probe 7. That would lead to Probe 6 not emitting enough radiation, and subsequently not cooling enough to reach what is seen on the experimental data. Since Probe 13 is located directly beneath Probe 6 in the bulk of the hat, a direct consequence is then Probe 13 not losing enough heat to Probe 6 by conduction, and in turn not cooling enough either. This hypothesis is all the more convincing that near 750 °C, the graph of simulated temperature for Probe 6 reaches that of experimental temperature. We have stated in the paragraph for accurate probes that radiation was likely to be dominant over 700 °C, and after that conduction and convection would play a non-negligible role for the cooling of exterior probes. If we suppose that Probe 6's radiation is the source of the discrepancy between experimental and numerical data, then that would explain why this discrepancy disappears when radiation starts to play a less dominant role.

4.4.7 Discussions

These three test cases have taught us several things. The ferrule test case has indicated the important role played by the inner surface for radiation cooling. The ferrule cools off both from the inside and from the outside, with temperature being at a sort of spatial equilibrium inside the hole. But cooling through radiation only is slow, and the ferrule remains a very hot object even after 1000 s. We have also

seen how the outer surface oriented towards the empty oven cools down significantly faster than the rest of the cylinder, which creates important thermal gradients. This shows the great impact of the relative position of the ferrule in the oven. Moreover, radiative cooling was not important enough to significantly affect the bulk of the thick ferrule, which shows that convective cooling is needed too in order to speed up the process.

The three loads test case has showed how mutual obstruction affects radiative heating. The cube located between the cylinder and the parallelepiped is obstructed by these two solids from receiving much heat from the heated band in the oven. It then heats up very slowly, as it mostly receives radiation from its neighbours. Since it allows it to heat more homogeneously, it might actually be desirable for some delicate ceramics to be sintered to be surrounded by other parts to reproduce this effect. The impact of load dimension has also been noted. The cylinder, being smaller than the two other parts, receives a higher ratio of radiative flux relatively to its volume, and it reaches temperatures that are a lot hotter than that of the two others. In addition, the highly localized nature of the heated band has been noted: the fact rays emitted from there could barely reach the top surface of the loads meant these areas remained almost at starting temperature for the whole simulation. That in turn led to very high thermal gradients on parts such as the parallelepiped.

The hat test case indicated how on the whole our simulation was in agreement with the numerical data. The maximum absolute error after 18 000 s of simulation was 54 °C for Probe 8 (627 °C instead of 681 °C). While non-negligible, this difference remains within 10% of the measured temperature of the hat. One also has to remember that we have taken fully uniform physical values and a perfectly geometrical shape for the hat. The real hat used for the experiment most likely differs by its local surface condition, defaults in its shape, etc. that certainly account for some part of the measured error. Besides, the general allure of the temperature graphs was reasonably the same between the numerical and experimental probes. We also witnessed thermal gradients that were lower than in the ferrule, and we suspect this is due to the presence of convection in this simulation. Observing where the discrepancies lie lead us to believe there might be something to upgrade in the computation of view factors at a close distance, as they might be slightly overestimated. The S2S model has been able to be satisfactorily combined with conduction and convection in this test case. We have also been able to observe how even a piece with a shape as simple as a hat can offer a diversity of thermal behaviours depending on where to place the probe. This has comforted us in our choice of using a radiation model able to capture local tendencies and geometry variation such as the S2S method, rather than a global model that averages out radiation such as the P_1 model.

4.5 Summary

- The solver has been evaluated on three more realistic industrial test cases in radiative cooling process lasting for 15 min to 30 min of real-time: a ferrule, three loads next to one another, and a hat.
- The ferrule and three loads have behaved in a physically-logical way and their result will be taken as a qualitative benchmark for future solvers.
- The importance of relative positioning of the load inside the oven and of interference between multiple loads has been observed. Strong thermal gradients have been seen in specific areas of the loads depending on their shape.
- The hat test case simulation matches reasonably well with the experimental data provided. This confirms the capabilities of the solver.
- The presence of convection in the hat test case seems to smooth out the thermal gradients inside the loads when compared to the convection-less ferrule test case.
- There seems to be some work to be done to enhance conduction modelling and close distance view factors computation.

Bibliography

[1] F.-A. GmbH, Vakuum-induktions-sinteranlage. 159

Chapter 5

Conclusions and Perspectives

Contents

5.1 Conclusions 194
5.2 Perspectives 195

5.1 Conclusions

The initial problem this thesis was trying to address was, as brought forth by TAS, how can we model ceramic sintering so that ceramic design can discard the costly and lengthy trial-and-error approach? In this extensive subject of research, our specific scope was set on how to understand thermal transfer between the oven and the ceramic during sintering as a preliminary step for a future modelling of sintering as a microstructural level. We have provided an answer to this question by focusing on a careful modelling of radiative thermal transfer in sintering through the use of the S2S method. We have enriched this method by the possible use of an immersed mesh approach to make it more adaptable to industrial needs, and have accelerated it by the addition of kd-tree ray tracing to make it usable on typical industrial simulation resources.

In order to achieve this result, we have first studied the specifics of radiative heat transfer in the context of industrial applications and seen the spécific challenges it poses. We have collected in the litterature a number of radiative transfer models that have been developed over the years to answer that question. Considering the requirements of great accuracy as brought forth by TAS and the particular conditions under which ceramic sintering takes places, we have chosen to develop the S2S radiative transfer method as it appeared to us it was the method most able to fulfill these requirements. This method itself presents many variants as it comes to view factor computations or resolution scheme, so a second phase of bibliographic research has oriented our choice towards choosing the most relevant options. In order to achieve an adaptable model that could gracefully handle moving or deforming pieces, we have used an immersed mesh method to be used during preprocessing of S2S radiation. The imperative of solver accuracy was then established by confronting the proposed solver to analytical results in several simple test cases. Upon realizing the time performance of the proposed solver was incompatible with our objectives, we decided to set up a kd-tree acceleration structure. A certain type of simple kd-tree was chosen after some research. As the results were still not satisfactory, we chose a second type of kd-tree and implemented a ray-tracing obstruction algorithm to accelerate radiosity computation. The scalability of the proposed solver on up to 256 cores has been deemed satisfactory. We then tried the proposed solver at more complex radiative cases inspired by industrial situations and assessed it was behaving as expected. By comparison with experimental data from an industrial test case of radiative cooling, we have validated the accuracy of the solver.

Our work has enabled us to conclude several things about radiative cooling or heating processes. The relative position, dimension, thickness and shape all have an important impact on the temperature and thermal gradient fields in the oven. The contribution of conduction and convection at different temperature stages has also been observed. Since our model is able to capture local variations in temperature

well, it means it is a useful analysis tool for ceramic sintering design. The S2S model can then provide reasonably accurate temperature and thermal gradient fields as input parameters for the next stage of the projet supported by Thales: devising a model of ceramic sintering microstructure evolution. The good scalability of the model will allow the company to simulate fine-meshed object for long durations. On a more general level, this work has shown how locally-dependant and locally-impacting radiative effects can be in the high-temperature cases we have seen. This presents a strong incentive for future similar industrial processes simulations to take care to model radiation through a proper local model such as S2S instead of a more globally averaging model.

The S2S model is a well-known and studied thermal radiation model, but few references we have been able to find have coupled it with immersed mesh methods as we have. On the other hand, the kd-tree ray-tracing-accelerated radiosity model is a classical tool of image rendering, but to the best of our knowledge ray-tracing has been scantily applied to radiosity in the context of thermal transfer. As far as we can know, this study is the first to propose an S2S model enriched with both immersed mesh method and kd-tree ray tracing acceleration. This constitutes the main contribution of this work. While the immersed mesh approach makes it simpler to model moving or deforming geometries, the subsequents requirements of frequent recomputation of the radiosity matrix hurt the viability of using S2S for such cases. But the additions of kd-tree ray tracing and good parallel performance have greatly improved the feasability of this approach.

5.2 Perspectives

The S2S model is still contained in a very specific domain of application. There are a lot of processes where the rather stringent set of hypotheses we need to make is not ensured: partially transparent materials, interfering medium, specular reflections, variable emissivity...or more simply, it can be that the computational power at hand cannot handle the heavy matrix manipulations involved. In addition, even when the S2S model is, on paper, a perfect match to the considered problem, *the proposed* S2S model remains limited. We have only evaluated its performance on a small number of analytical test cases, all of simple shapes, and on a single experimental dataset for a simple shape. In addition all these cases were for unmoving, undeformable item. This means while the immersed mesh implementation *will allow* for an efficient modelling of moving items at a later stage of development, we have not actually tested this feature yet. Besides, while proven to be reasonably fast and robust for short simulations and a moderate degree of parallelism, it is unknown how the solver will handle simulations of a 48-hour long sintering, or if it is able to scale up to massively parallel computations with a thousand cores or more. This is why we

remain cautious in the assertions of solver accuracy and adaptability we state.

As far as immersed meshes are concerned, testing how facet reconstruction and view factor computation are handled for the case of a shrinking ceramic would be very interesting. A possibility would be to update the radiating facets at a given frequency, and not necessarily at all time steps, provided the shrinking would be slow enough. Determining which view factors do not need to be updated, if feasible, would be a great improvement in that regard. Another possibility could be to proceed to local remeshing between time steps. We could compute the gradient of flux density on the radiating surfaces and decide we need to refine the mesh and thus split the faces where that gradient is too high.

For the numerical matrix inversion scheme, we have mentioned the Chebyshev polynomials-based method, which seemed promising in our configurations. Proceeding to more extensive tests on that regard, and maybe switching from GRMES to a more relevant numerical method, is another venue of improvement. For the view factor computation, we have not set up the proper adaptive integration technique of Walton, and getting back to that could lead in improved accuracy and speed. And for parallel computation, the choice of two all-to-all communications was the simplest decision to accomodate the independant nature of elements and facet partitioning but carries a scalability penalty. It might be interesting to explore whether it is feasible to force a given core to handle both a given set of elements and the corresponding set of facets.

The matter of kd-tree ray tracing alone is liable to a vast array of improvements. For tree construction, the choice of using an SAH for choosing the split plane would lead to a tree of better quality and thus faster ray tracing. For ray tracing itself, we could use a heuristics for ordering through which child we travel down first. The matter of memory footprint has not been touched on, but clever node and ray structures exist to diminish it. The NN search is also a very simple one and could be extended to the approximate NN search. These are but a few of the possible paths for further research on our topic.

RÉSUMÉ

Une modélisation soigneuse des transferts thermiques par rayonnement est importante pour simuler correctement un grand nombre de processus industriels. Cette thèse se concentre sur la modélisation du transfert thermique par rayonnement sous certaines hypothèses: matériaux gris et homogènes, milieu non participant. Dans ce cadre, l'approche Surface to Surface a été retenue car elle permet une modélisation souple et précise de ces échanges. Nous utilisons un maillage immergé pour définir implicitement les surfaces rayonnantes par leur fonction levelset. Nous couplons le transfert radiatif avec les autres formes de transfert thermique dans une modélisation éléments finis P1. Après avoir validé notre approche sur des cas analytiques simples, nous accélérons le calcul de l'obstruction par la méthode du lancer de rayons. A cette fin, nous ordonnons les facettes rayonnantes dans un kd-tree. Pour conclure, nous montrons la capacité de montée en échelle de notre solveur en calcul parallèle et nous établissons des simulations de refroidissement de processus industriels réels.

MOTS CLÉS

Transfert radiatif, maillage immergé, méthode S2S, arbre kd

ABSTRACT

Properly taking into account thermal radiative transfer matters a great deal to properly model numerous industrial processes. This thesis focuses on setting up a proper modelling for it in certain industrial configurations where some hypotheses are met: grey, homogeneous materials and non-transmitting medium. The Surface-to-surface approach we have chosen allows for such an accurate solving of thermal transfer. We make use of immersed meshes and implicit object definitions by the levelset function to allow for an adaptable model and an implicit definition of the radiating surfaces. We couple radiative transfer with other forms of thermal transfer in a P1 finite elements methods. After validating our model on numerous simple test cases, we set up ray tracing to accelerate obstruction computation and organize the radiating facets into a kd-tree. We conclude by ensuring our solver is highly scalable on parallel computing and show simulation cases of real industrial processes.

KEYWORDS

Radiative transfer, immersed mesh, S2S method, kd-tree