Path integrals formulations leading to propagator evaluation for coupled linear physics in large geometric models

Léa Penazzi, Stéphane Blanco, Cyril Caliot, Christophe Coustet, Mouna El-Hafi, Richard Fournier, Jacques Gautrais, Morgan Sans, Anna Golijanek-J drzejczyk

Path integrals formulations leading to propagator evaluation for coupled linear physics in large geometric models.

L. Penazzi[a,b,*], S. Blanco[c], C. Caliot[d], C. Coustet[e], M. El Hafi[a], R. Fournier[c], J. Gautrais[f], A. Golijanek-Jędrzejczyk[g], M. Sans[a]

[a] *RAPSODEE, UMR CNRS 5302, IMT Mines Albi, Campus Jarlard, Albi, France*
[b] *PROMES-CNRS, UPR 851, 7 rue du Four Solaire, 66120 Font Romeu Odeillo, France*
[c] *LAPLACE, UMR CNRS 5213, Université Paul Sabatier, 118 Route de Narbonne - 31062 Toulouse, France*
[d] *CNRS, UPPA, E2S, LMAP, 1 Allée du Parc Montaury, Anglet, France*
[e] *Méso-Star - 8 rue des Pêchers, 31410 Longages, France*
[f] *Centre de Recherches sur la Cognition Animale (CRCA), Centre de Biologie Intégrative (CBI), Université de Toulouse, CNRS, Université Paul Sabatier - Toulouse, France*
[g] *Gdańsk University of Technology, Faculty of Electrical and Control Engineering, 11/12 Gabriela Narutowicza Street, 80-233 Gdańsk, Poland*

Abstract

Reformulating linear physics using second kind Fredholm equations is very standard practice. One of the straightforward consequences is that the resulting integrals can be expanded (when the Neumann expansion converges) and probabilized, leading to path statistics and Monte Carlo estimations. An essential feature of these algorithms is that they also allow to estimate propagators for all types of sources, including initial conditions. The resulting practice is a single Monte Carlo run, for one given set of sources, producing propagators that can later be used with any other set of sources for fast simulations, typically as parts of optimization, inversion, sensitivity analysis and command control algorithms. The present paper illustrates how this practice can be extended to problems involving several interacting physics, provided that their coupling is only at the boundary of the system or at interfaces between sub-parts, and may itself be given the form of a second kind Fredholm equation. A full practical implementation is described as part of the *Stardis* code, with the example of transfering heat via the coupling of radiation, reaction-diffusion and convection as typically expected in the multidisciplinary context of urban climate modeling. Besides, we show how recent advances in computer graphics indicate that these algorithms can be made numerically extremely efficient when facing large CAD geometries: computing the propagator becomes strictly independent of the geometry refinement, i.e. is identical whatever the number of triangles and tetraedra used to numerize the surface and volume descriptions. To the best of our knowledge this is the first report of propagator computations that remains practical for coupled physics in large CAD geometries.

Keywords: Propagator, Monte Carlo method, Large geometric models, Coupled heat transfer

PROGRAM SUMMARY

Program Title: Stardis 0.7.2 (built on `stardis-solver` 0.12.3)
CPC Library link to program files: (to be added by Technical Editor)
Developer's repository link: `https://www.meso-star.com/projects/stardis/stardis.html`
Code Ocean capsule: (to be added by Technical Editor)
Licensing provisions: GPLv3
Programming language: ANSI C
Supplementary material: (Zip folder added to submission)
Nature of problem: Estimating temperatures in coupled heat transfer systems involving large CAD and/or large numbers of spatially distributed sources.
Solution method: Stardis uses the Monte Carlo Method. Each emperature estimate constructs a propagator of each of the energy

sources within the system: initial temperature, temperature boundary conditions, volume powers and surface fluxes. The propagator can be stored for further use outside the Monte Carlo code.

Additional comments including restrictions and unusual features: The *Stardis* estimates of propagators are only reliable when radiative transfer can be linearized around a reference temperature. *Stardis* can deal with the nonlinearity of radiation when computing temperatures, but then nothing can be interpreted as a meaningfull propagator for any external usage.

1. Introduction

First simple illustration. Starting from G. Green's theory, the propagator concept was introduced by R. Feynman as a way to picture, in integral terms, the solution $O(\vec{x},t)$ of a field physics problem at a location $\vec{x}$ and time t, when this physics is linear : $O(\vec{x},t)$ is viewed as an integral over all sources $S(\vec{x}_S,t_S)$ at all locations $\vec{x}$ inside the domain $\mathcal{D}$ and all times preceding t (down to initial time t_I), multiplied by a scalar $\zeta(\vec{x},t,\vec{x}_S,t_S)$[1] :

$$O(\vec{x},t) = \int_{\mathcal{D}} d\vec{x}_S \int_{t_I}^{t} dt_S \; \zeta(\vec{x},t,\vec{x}_S,t_S)\, S(\vec{x}_S,t_S) \tag{1}$$

The propagator $\zeta(\vec{x},t,\vec{x}_S,t_S)$ indicates how each source impacts the solution, and invites for intuitions of the sources being propagated in space and time throughout the system, toward the considered location $\vec{x}$ and time t. Historically, this rewriting of G. Green's formalism is mainly significant with regards to the physical pictures it suggests. Here, we will concentrate on translating these pictures in pure computational terms: since $\zeta(\vec{x},t,\vec{x}_S,t_S)$ is independent of the source *values*, it can be numerically evaluated on its own. Then, any set of sources values can be plugged into Eq. (1) to compute the corresponding $O(\vec{x},t)$.

Let us illustrate this concept with a standard practice in radiative transfer, where the factors associated with the propagative point of view are named "shape factors" or "exchange surfaces" (depending on the context and the chosen formulations). Let us take a simple scene with stationary radiative transfer between a camera and two lamps of respective powers $\mathcal{P}_1$ and $\mathcal{P}_2$. Let O denote the radiative flux incident on a chosen pixel of the camera.

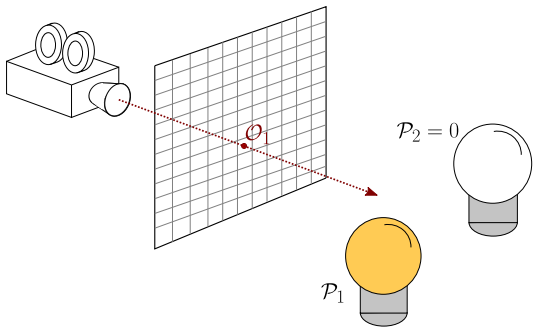

Figure 1: $\mathcal{P}_1$ light source on and $\mathcal{P}_2$ off.

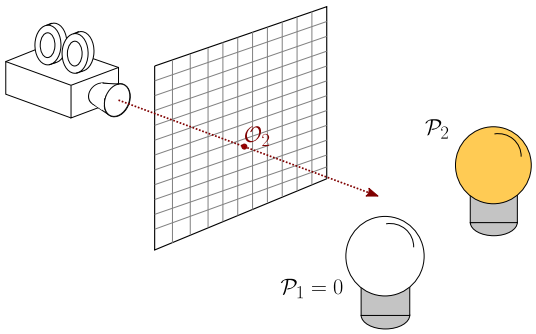

Figure 2: $\mathcal{P}_2$ light source on and $\mathcal{P}_1$ off.

Evaluating the shape factor ξ_1 between the first lamp and the target pixel can be obtained by switching on the first lamp alone and solving the radiative transfer equation to get the corresponding pixel flux O_1 (see Fig. 1). The shape factor associated with this first lamp is then $\xi_1 = \frac{O_1}{\mathcal{P}_1}$. The same radiative transfer equation can also be solved to evaluate the pixel flux O_2 when only the second lamp is on, and the shape factor of the second lamp is $\xi_2 = \frac{O_2}{\mathcal{P}_2}$ (see Fig. 2). Once these factors are known, since they are independent of the powers of the lamps, they can be used to evaluate the pixel flux $\tilde{O}$ for any other set of lamp powers $\tilde{\mathcal{P}}_1$ and $\tilde{\mathcal{P}}_2$ when the two lamps are on simultaneously:

$$\tilde{O} = \xi_1\tilde{\mathcal{P}}_1 + \xi_2\tilde{\mathcal{P}}_2 \tag{2}$$

[1] In such a formulation all source types are integrated over all times in $[t_I,t]$ and all locations in $\mathcal{D}$. Boundary sources are therefore spatial Diracs at $\partial\mathcal{D}$ and the initial conditions are translated into temporal Diracs at t_I.

The required number of shape factors can be huge, for example in the domain of infrared radiation where all surface and volume parts of the system can emit radiation, so the question of the efficient numerical evaluation of large numbers of shape factors has raised numerous technical questions, in particular for multiple scattering and multiple reflection configurations with semi-transparent materials. A classical approach is to use reverse Monte Carlo algorithms in which optical paths are tracked from the receptor (the target pixel in our example) backward to sources (the lamps). The procedure can be duplicated for each target of interest (e.g. the other pixels of the camera).

Such algorithms, initially designed to estimate O, can also be used to estimate each shape factor (from each lamp to the target pixel), with no additional computational cost: the paths sampled in one single computation can yield estimates of O as well as all the shape factors. In our example, the reverse Monte Carlo computation can then be made only once for a given set of the two lamps power, and does not need to be repeated for each lamp one by one. Once evaluated, the shape factors can be linearly combined with any sets of lamp powers. In this practice, there is one point of concern: we need to consider statistical correlations when evaluating the estimate uncertainty ("error bars") because the estimates are built using the same set of sampled optical paths over different sets of lamps power. Taking this point into account, the approach is then straightforward: only the shape factors required to estimate O are to be computed using a reverse Monte Carlo.

The question in broader terms. Path-integral statistics allow to address the very same question (a single Monte Carlo run computing entirely the propagator) now for advanced linear physics. Without coupling, i.e. when only one single physical phenomenon is at work, the most common approach to propagator computation consists in translating the corresponding partial differential equation (typically a Boltzmann-type transport equation, a reaction-diffusion equation, Maxwell equation, etc) into a second kind Fredholm equation and developping the solution as a Neumann expansion [1, 2]. The expansion is then interpreted in statistical terms to define path-integral statistics, i.e. writting the solution as the expectation of a random variable associated to a stochastic path. The corresponding path-integrals address only the solution for a fixed set of sources, but structurally the functional dependance to each source is linear and expressing the propagator using the very same path-integral statistics is straightforward. This extends the illustration of the above simple example. Even for advanced physics or more complex geometries, Monte Carlo algorithms may be designed so that they sample paths along which propagation information can be stored. Then, the propagation information can be directly used when addressing a new set of sources, instead of re-running the Monte Carlo.

In most applicative contexts, computing the propagator with such a Monte Carlo approach would be of great significance. However, this is not yet feasible because of coupling between different physics. Thus, our question becomes the following: can the path-integral approach be theoretically extended to coupled physics and does it remain computationnaly convenient? The present paper briefly initiates a possible answer to the theoretical part: Thanks to double randomization, the Fredholm approach can be extended by probabilizing the coupling. Then, the question of practicability is restricted to only one particular type of coupling: physics interacting via interfaces only (and not at all locations within the field). This restriction allows to stick to a very active field of research in both the physics and computer graphics communities: the design of recursive grids for acceleration of path tracing in complex geometries, leading essentially to Monte Carlo algorithms that are strictly insensitive to geometry refinement: same computation times for scenes described with hundreds of triangles, hundred of voxels, or billions of them [3]. By mentionning voxels, we want here to point out that the restriction to coupling via interfaces does not exclude advanced field descriptions, and even multiple physics at work at the same location in the field: only the coupling is not within the field. When randomizing the coupling at the interfaces, the features of the stochastic paths inside each subpart of the field are strictly preserved and the available schemes for accelerating their construction remain unchanged. This leads to Monte Carlo codes computing propagators inside systems involving coupled physics and displaying the same property of being insensitive to geometrical refinement.

The implementation reported here is made within the *Stardis* code. *Stardis* was recently used to explore ways toward the definition of new climate services for analysts and designers anticipating climate change in urban area. In this context, *Stardis* strength was its ability to deal simultaneously with all the spatial and temporal scales involved in the modeling of energy exchanges, from the milimeter scale of windows and heat seals to the kilometers extensions of cities, and from the minute scale of wind and solar fluctuations to typically fifty-year lifetimes of the ground installations to be planed. Radiative transfer inside and outside buildings could be modeled together with full three-dimension heat diffusion inside the solid structures, as well as convection inside each fluid cell, providing the first

reported attempt to model energy transfers in a complete city, without any compromise on its geometrical description [4]. The corresponding approach was name "teapot in a city" by reference to the "teapot in a stadium" paradigm in computer graphics where it was indeed shown that ray tracing Monte Carlo algorithms could deal with quasi-infinite scale ratios on an industrial basis when producing cinema and video-game images. Retaining *Stardis* for illustration of the present theoretical discussion of propagator estimation in complex systems, we want to highlight an essential idea to be added to the perspective statements of [4]: using Feynman-Kac strategy to extend the computer graphics strategies to other physics than light transport allows one to model the heat transfer in geometrically complex systems, but also the systematic computation of all the propagators associated to each energy source. We will concentrate on standard heat transfer physics, but the statement is valid for all the physical contexts where same path-integral Monte Carlo strategies were already reported, either using similitudes with slightly modified versions of *Stardis* (e.g. linear-Boltzmann equation coupled with reaction-diffusion-advection in porous structures [5]), or with independent codes using the same path-tracing libraries (e.g. electromagnetism, photosynthesis and molecular spectroscopy, in their linear parts [6, 7]).

The article is structured as follows:

- Sec. 2 provides the theoretical background.
- Sec. 3 describes the physics involved in the standard version of *Stardis*.
- Sec. 4 describes the reverse Monte Carlo path sampling strategy solving this model.
- Sec. 5 describes how *Stardis* stores the propagation data.
- Sec. 6 describes `stardis-solver`, an implementation of reference used by *Stardis*.
- Sec. 7 depicts simulation examples.
- Sec. 8 gives some hints towards a generalisation to non-uniform and time-dependent sources, before concluding remarks in Sec. 9.

2. Theory

2.1. From second-kind Fredholm equations to path-statistics

For the sake of exposition, consider a single model for the field of a quantity $\theta \equiv \theta(\vec{x},t)$ at location $\vec{x}$ and time t within a domain Ω of boundary $\partial\Omega$, where sources S are known, in the volume only (no source at the boundary) and where initial conditions are reported to $-\infty$. Let us further assume that this model can be formulated as a second kind Fredholm equation:

$$\theta(\vec{x},t) = \int_{-\infty}^{t} dt_b \int_{\partial\Omega} d\vec{x}_b \; G_b(\vec{x}_b,t_b|\vec{x},t)\; \theta(\vec{x}_b,t_b) + \int_{-\infty}^{t} dt' \int_{\Omega} d\vec{x}\,' \;\left[S(\vec{x}\,',t') + G(\vec{x}\,',t'|\vec{x},t)\; \theta(\vec{x}\,',t')\right] \tag{3}$$

where the Green functions G and G_b are known. To illustrate how such a Fredholm equation leads to a path-integral statistical description, let us further assume that $\theta(\vec{x},t)$ is known at the boundary: $\theta(\vec{x},t) = \theta_b(\vec{x},t)\;\forall \vec{x} \in \partial\Omega$ where θ_b is fixed (this condition will translate into a coupling issue in section 2.2). Then, assuming convergence as in a standard Neumann expansion, replacing $\theta(\vec{x}\,',t')$ by a recursive call to Eq. 3 yields:

$$\begin{aligned}\theta(\vec{x},t) = &\int_{-\infty}^{t} dt_{b,1} \int_{\partial\Omega} d\vec{x}_{b,1}\; G_{b,1}(\vec{x}_{b,1},t_{b,1}|\vec{x},t)\; \theta_b(\vec{x}_{b,1},t_{b,1}) \\ &+ \int_{-\infty}^{t} dt_1 \int_{\Omega} d\vec{x}_1 \left[S(\vec{x}_1,t_1) + G(\vec{x}_1,t_1|\vec{x},t)\; \left[\int_{-\infty}^{t_1} dt_{b,2} \int_{\partial\Omega} d\vec{x}_{b,2}\; G_{b,2}(\vec{x}_{b,2},t_{b,2}|\vec{x}_1,t_1)\; \theta_b(\vec{x}_{b,2},t_{b,2}) \right.\right. \\ &\left.\left. + \int_{-\infty}^{t_1} dt_2 \int_{\Omega} d\vec{x}_2 \left[S(\vec{x}_2,t_2) + G(\vec{x}_2,t_2|\vec{x}_1,t_1) \qquad ... \qquad \right]\right]\right]\end{aligned} \tag{4}$$

This expression can be, in turn, probabilized to define a random path backward in time, starting at $(\vec{x}, \vec{t})$ and ending at the boundary at a previous time:

$$\begin{aligned}
\theta(\vec{x},t) = P_b(\vec{x},t) \int_{-\infty}^{t} dt_{b,1} \int_{\partial\Omega} d\vec{x}_{b,1}\; p_b(\vec{x}_{b,1},t_{b,1}|\vec{x},t)\; \frac{G_{b,1}(\vec{x}_{b,1},t_{b,1}|\vec{x},t)\;\theta_b(\vec{x}_{b,1},t_{b,1})}{P_b(\vec{x},t)\;p_b(\vec{x}_{b,1},t_{b,1}|\vec{x},t)} \\
+ (1-P_b(\vec{x},t)) \int_{-\infty}^{t} dt_1 \int_{\Omega} d\vec{x}_1\; p(\vec{x}_1,t_1|\vec{x},t) \left[\frac{S(\vec{x}_1,t_1)}{(1-P_b(\vec{x},t))\;p(\vec{x}_1,t_1|\vec{x},t)} \right. \\
+ \frac{G(\vec{x}_1,t_1|\vec{x},t)}{(1-P_b(\vec{x},t))\;p(\vec{x}_1,t_1|\vec{x},t)} \left[P_b(\vec{x}_1,t_1) \int_{-\infty}^{t_1} dt_{b,2} \int_{\partial\Omega} d\vec{x}_{b,2}\; p_b(\vec{x}_{b,2},t_{b,2}|\vec{x}_1,t_1)\; \frac{G_{b,2}(\vec{x}_{b,2},t_{b,2}|\vec{x}_1,t_1)\;\theta_b(\vec{x}_{b,2},t_{b,2})}{P_b(\vec{x}_1,t_1)\;p_b(\vec{x}_{b,2},t_{b,2}|\vec{x}_1,t_1)} \right. \\
+ (1-P_b(\vec{x}_1,t_1)) \int_{-\infty}^{t_1} dt_2 \int_{\Omega} d\vec{x}_2\; p(\vec{x}_2,t_2|\vec{x}_1,t_1) \left[\frac{S(\vec{x}_2,t_2)}{(1-P_b(\vec{x}_1,t_1))\;p(\vec{x}_2,t_2|\vec{x}_1,t_1)} \right. \\
\left. \left. \left. + \frac{G(\vec{x}_2,t_2|\vec{x},t)}{(1-P_b(\vec{x}_1,t_1))\;p(\vec{x}_2,t_2|\vec{x}_1,t_1)} \quad ... \quad \right] \right] \right]
\end{aligned} \tag{5}$$

In this probabilistic reading of the Neumann expansion, the main sum between the integral over the boundary and the integral over the domain is translated into a Bernoulli test between the two integrals. When the boundary integral is retained, a location is sampled at the boundary and the process is stopped because θ has been set to be known at the boundary. When the domain integral is retained, a location is sampled inside the domain, and since θ is unknown, the process is continued from the last sampled location and time, up to a choice of the boundary integral.

In principle, the probabilities P_b (to select the boundary at the i-th step) and probability densities p_b and p (of the sampled location and time at the i-th step knowing the location and time of the preceding step) are arbitrary, but we can rely on the Monte Carlo literatures dedicated to each physic to indicate the meaningful choices in terms of variance reduction. Overall, in most linear physics where such converging Neumann expansions are available, the theoretical construction and numerical practice of sampling such paths are already available. In box 2.1, we briefly illustrate this starting point with a famous academic example using Feynman-Kac formula. Let us add some details regarding the two main physics addressed in the following sections: radiation physics and reaction-diffusion in confined domains.

Radiative transfer (or linear transport Bolzmann equation in neutronics, biology, etc) is straightforward. The only specificity is that the domain is in phase space and not only in geometrical space. Apart from this, expressing the linear Bolzmann equation in Fredholm terms, and probabilizing it, is very common and leads to elementary pictures such as multiple scattering or multiple reflection path tracing. In terms of of radiative transfer theory, θ is the specific intensity. Its value at a given location in a given direction can be viewed as an average of radiative energy transported along the line of sight. The probabilities P_b to reach the boundary are essentially Beer exponential extinction along the line and the probability density of volume collisions, p, is the spatial derivative of this exponential decrease (see Fig. 3). Translating radiative transfer in terms of path-statistics is straightforward and fully rigorous.

Regarding reaction-diffusion equations, the theoretical background is heavier as it involves Brownian motion in confined spaces for which very little can be done with no numerical approximations. When dealing with thermal diffusion with spatially distributed sources, our approach will therefore be approximate. This implies that Eq. 3 will only hold when a spatial discretization is applied after a Brownian motion. This is typically the case for walk-on-sphere algorithms that are among the most efficient available path-sampling approaches to confined diffusion [8, 9, 10]. At each step, a location is sampled on the smallest sphere tangenting the boundary and this location is projected on the boundary when its distance to the boundary is lower than a numerical parameter ϵ. The probability P_b is therefore the fraction of the sphere satisfying this condition (see Fig. 4) and p is a uniform distribution along the rest of the sphere. A slightly modified version of this algorithm will be used for essentially two reasons: 1) including heterogeneous sources and 2) designing an algorithm using line-boundary intersections (and not line-sphere intersections) so that we benefit from the path-tracing acceleration techniques of computer graphics when dealing with complex geometries. The principle is depicted in Fig. 5 and details will be provided in the following sections.

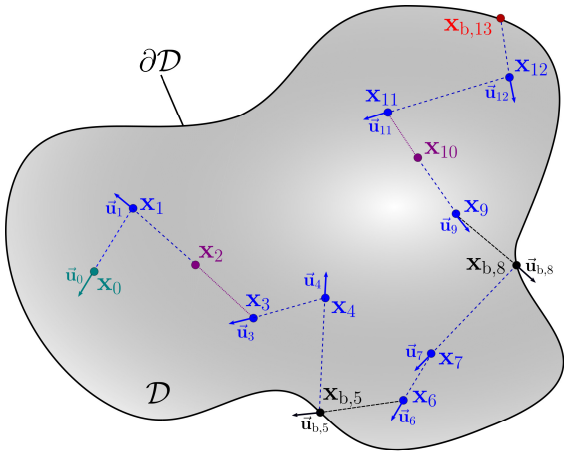

Figure 3: An heterogeneous, multiple scattering and emitting/absorbing medium $\mathcal{D}$ is bounded by a partially reflecting wall $\partial\mathcal{D}$. Physical images associated to the evaluation of a local radiative quantity at the location $\mathbf{x}_0$ (for instance, the monospectral radiative intensity) with a reverse Monte Carlo algorithm are as follows. Photons are followed from the location $\mathbf{x}_0$ in the unit direction $-\vec{\mathbf{u}}_0$ until the emission/absorption location, either in the volume $\mathcal{D}$ or on the wall $\partial\mathcal{D}$ (see $\mathbf{x}_{b,13}$). At each step j of the optical path within the medium, a free path l_j is sampled according to the exponential Beer law, enabling the computation of the next step location $\mathbf{x}_j = \mathbf{x}_{j-1} - l_j\vec{\mathbf{u}}_{j-1}$. If the location $\mathbf{x}_j$ is in the medium $\mathcal{D}$, the event may be an absorption, a scattering (see $\mathbf{x}_1$,$\mathbf{x}_3$,$\mathbf{x}_4$,$\mathbf{x}_6$,$\mathbf{x}_7$,$\mathbf{x}_9$,$\mathbf{x}_{11}$ and $\mathbf{x}_{12}$) or a null-collision (see $\mathbf{x}_2$ and $\mathbf{x}_{10}$). A null-collision event corresponds to a pure-forward scattering event in which $\vec{\mathbf{u}}_j = \vec{\mathbf{u}}_{j-1}$ (see box 2.1). If the location x_j reaches the wall $\partial\mathcal{D}$, the event may be an absorption (see $\mathbf{x}_{b,13}$) or a reflexion (see $\mathbf{x}_{b,5}$ and $\mathbf{x}_{b,8}$).

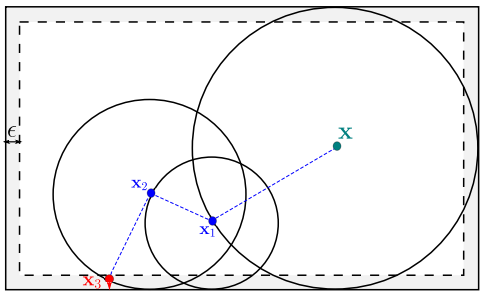

Figure 4: Illustration of the sampling of a random path using the random walk-on-sphere method to estimate density at location $\mathbf{x}$. In order to end the random walk, the boundary of the domain is thickened of a small value ϵ in which the final position $\mathbf{x}_3$ is projected on the boundary.

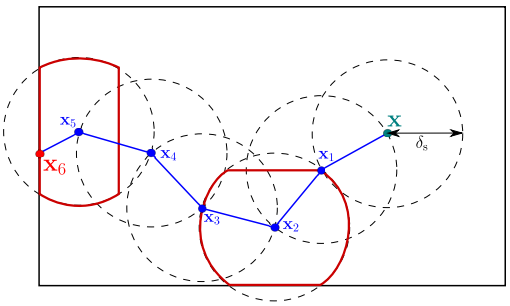

Figure 5: Illustration of a random path sampling compatible with path-tracing acceleration techniques of computer graphics. Each sphere has the same radius δ_s and is adjusted when getting close to the domain boundary.

Feynman-Kac

The most famous example of expressing the solution of a field physics partial differential equation as a second kind Fredholm equation is Feynman-Kac formula. We use it here to illustrate the class of physical problems addressed in the present article. Let us consider the following reaction-diffusion equation:

$$\underbrace{\frac{\partial\theta}{\partial t}}_{\text{temporal evolution}} = -\underbrace{\vec{\nabla}\cdot\left(-D\vec{\nabla}\theta\right)}_{\text{diffusion}} - \underbrace{\vec{\nabla}\cdot(\vec{\mathbf{v}}\theta)}_{\text{advection}} - \underbrace{\nu\,(\theta-\theta_0)}_{\text{reaction}} \tag{6}$$

where D is the diffusion coefficient, $\vec{\mathbf{v}}$ an advection speed and ν the reaction frequency. When this frequency is function of time and/or space, an overestimate $\hat{\nu}$ can be introduced [11] to write

$$\frac{\partial\theta}{\partial t} = -\vec{\nabla}\cdot\left(-D\vec{\nabla}\theta\right) - \vec{\nabla}\cdot(\vec{\mathbf{v}}\theta) - \hat{\nu}\left[\theta - \left(\frac{\nu}{\hat{\nu}}\theta_0 - \left(1-\frac{\nu}{\hat{\nu}}\right)\theta\right)\right] \tag{7}$$

and applying Feynman-Kac formula [12] gives

$$\theta(x,t) = \mathbb{E}\left[\underbrace{e^{-\hat{\nu}(t-T)}\theta_b\left(X_T,T\right)}_{\text{boundary}} + \underbrace{\int_T^t dt'\,\hat{\nu}e^{-\hat{\nu}(t-t')}\left(\frac{\nu\left(X_{t'},t'\right)}{\hat{\nu}}\theta_0\left(X_{t'},t'\right) + \left(1-\frac{\nu\left(X_{t'},t'\right)}{\hat{\nu}}\right)\theta\left(X_{t'},t'\right)\right)}_{\text{volume}}\right] \tag{8}$$

where $X_{t'}$ is the associated Weiner process backward in time, starting at location x at time t and first encountering the boundary at T. The solution $\theta(x,t)$ is herefore expressed as an expectation of a random expression that includes $\theta\left(X_{t'},t'\right)$, i.e. the solution of the very same problem at another location and another time. This implies that the integral formulation of this expectation is a second kind Fredholm equation of the general type that we used as starting point in Eq. 3:

$$\begin{aligned}\theta(x,t) = &\int_{-\infty}^{t} dt_b \int_{\partial\Omega} dx_b\, G_b(x_b,t_b|x,t)\,\theta_b(x_b,t_b)\\ &+\int_{-\infty}^{t} p_T(t_b)dt_b \int_{t_b}^{t} dt' \int_{\Omega} dx'\left[S\left(x',t'\right) + G(x',t'|x,t)\,\theta(x',t')\right]\end{aligned} \tag{9}$$

with

$$\begin{aligned}G_b(x_b,t_b|x,t) &= p_{X_T}(x_b,t_b)\,e^{-\hat{\nu}(t-t_b)}\\ G(x',t'|x,t) &= \hat{\nu}e^{-\hat{\nu}(t-t')}\,p_{X_{t'}}(x'|T=t_b)\left(1-\frac{\nu(x',t')}{\hat{\nu}}\right)\\ S\left(x',t'\right) &= e^{-\hat{\nu}(t-t')}\,p_{X_{t'}}(x'|T=t_b)\,\nu(x',t')\,\theta_0(x',t')\end{aligned} \tag{10}$$

where $p_{X_T}(x_b,t_b)$ is the probability density of the location x_b and time t_b at which the Wiener process encounters the boundary and $p_{X_{t'}}(x'|T=t_b)$ is the probability density that $X_{t'}=x'$ knowing $T=t_b$. The only noticeable difference with Eq. 3 is that the t' integral over $]-\infty,t]$ is expressed as a convolution product, requiring the sampling of the boundary encountering time also for the volume part of the Fredholm equation, which makes no difference as far as the following derivations are concerned.

2.2. Coupling via interfaces between sub-parts with distinct physics

The two examples above illustrate the broad variety of available path-statistics dealing with linear field physics (see Fig. 3 and Box 2.1). However, they hold when the addressed quantity θ is unique. Here, our objective is to discuss coupling in the particular case where coupling occurs via internal interfaces. Several physics are involved, defining field quantities θ_1,θ_2 ... θ_N only known at the boundary of the overall system. These quantities are not known at the

interfaces between two sub-parts of the system. At these interfaces, the coupling constraint is assumed to express each θ_i using linear integral operators over the connected fields, i.e. for each $\vec{x}_b \in \partial\Omega_i$ and each time t_b,

$$\theta_i(\vec{x}_b, t_b) = \sum_{j=1}^{N} \left\{ \int_{-\infty}^{t_b} dt_b' \int_{\partial\Omega_j} d\vec{x}_b{}' F_{bij}(\vec{x}_b{}', t_b'|\vec{x}_b, t_b)\, \theta_j(\vec{x}_b{}', t_b') + \int_{-\infty}^{t_b} dt \int_{\Omega_j} d\vec{x}\, F_{ij}(\vec{x}, t|\vec{x}_b, t_b)\, \theta_j(\vec{x}, t) \right\} \tag{11}$$

where Ω_i is the sub-part of Ω where θ_i is defined, $\partial\Omega_i$ its boundary (a part of which may belong to the boundary of the overall system), j is the index referring to a different sub-part of the system over the total N sub-parts, and F_{ij} and F_{bij} are known (they are null if θ_j is not defined in either of the two sub-parts separated by the considered interface). The approach consists in probabilizing this integral constraint exactly the same way the Fredholm equations are probabilized in their respective fields for each single physics (see Eq. 5):

$$\begin{aligned} \theta_i(\vec{x}_b, t_b) = \sum_{j=1}^{N} P_{ij} \Bigg\{ & P_{bij} \int_{-\infty}^{t_b} dt_b' \int_{\partial\Omega_j} d\vec{x}_b{}'\; p_{bij}(\vec{x}_b{}', t_b'|\vec{x}_b, t_b)\, \frac{F_{bij}(\vec{x}_b{}', t_b'|\vec{x}_b, t_b)\, \theta_j(\vec{x}_b{}', t_b')}{P_{bij} p_{bij}(\vec{x}_b{}', t_b'|\vec{x}_b, t_b)} \\ & + (1 - P_{bij}) \int_{-\infty}^{t_b} dt \int_{\Omega_j} d\vec{x}\; p_{ij}(\vec{x}, t|\vec{x}_b, t_b)\, \frac{F_{ij}(\vec{x}, t|\vec{x}_b, t_b)\, \theta_j(\vec{x}, t)}{(1 - P_{bij}) p_{ij}(\vec{x}, t|\vec{x}_b, t_b)} \Bigg\} \end{aligned} \tag{12}$$

where P_{ij} is the probability of selecting the θ_j branch when estimating θ_i at the interface (see Fig. 6). P_{ij}, P_{bij}, p_{ij} and p_{bij} are then the strict equivalent to P, P_b, p and p_b in Eq. 5.

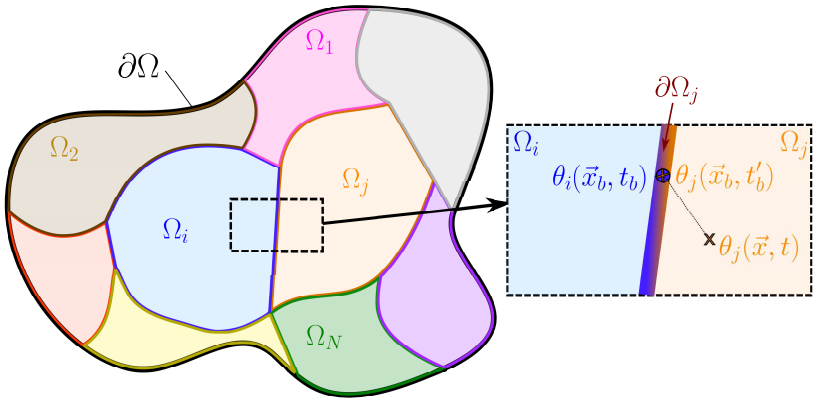

Figure 6: Ω domain is compound of N sub-domains such as $\Omega = \{\Omega_1, \Omega_2, ..., \Omega_N\}$ where $\theta_1, \theta_2, ..., \theta_N$ are only known at the boundary $\partial\Omega$ of the overall system. When estimating θ_i at a location $\vec{x}_b$ and a time t_b at the interface $\partial\Omega_j$ between Ω_i and Ω_j, selecting the θ_j branch is given by probability P_{ij}. If this branch is selected, then $\theta_i(\vec{x}_b, t_b)$ either takes the value $\theta_j(\vec{x}_b, t_b')$ with probability P_{bij} (same location, different time), or the value $\theta_j(\vec{x}, t)$ with probability probability $1 - P_{bij}$ (different location within domain Ω_j, different time). The path sampling goes on until a known temperature at the boundary $\partial\Omega$ is reached.

The parallel with the path-statistics described above for single physics is complete and suffices to recursively define statistical paths for the coupled problem:

- When estimating θ_i at a given location within Ω_i, the path starts as if the physics of θ_i was uncoupled and reaches a location at the boundary of Ω_i.
- At this stage, for an uncoupled problem, θ_i would be known and the path would end. However, the location reached on $\partial\Omega_i$ may either be on $\partial\Omega$ or located at an interface between Ω_i and other sub-parts of the system.
- If this location is on $\partial\Omega$, it means that the boundary of the overall system is reached, thus θ_i is known and the path ends.
- If this location is at an interface between Ω_i and other sub-parts of the system, then a new physics is sampled among those involved at this interface and the question is transformed into the estimation of θ_j either inside Ω_j or at the boundary $\partial\Omega_j$.

- This question is solved by the same algorithm, now for θ_j instead of θ_i.
- The path is therefore continued and this alternation between the coupled physics occurs at each encountered interface until $\partial\Omega$ is reached.

A plain picture of full paths comprised of successions of coupled sub-paths is therefore established. Each sub-path corresponds to one physics (same statistics as those of the corresponding uncoupled sub-path), the full path ends at the overall boundary $\partial\Omega$ with a known value θ_{bk} and the index k corresponds to the last visited physics (that of the last sub-path, reaching the boundary). Fig. 7 illustrates the general pattern for a coupled heat transfer situation in which the objective is to evaluate the probe temperature at a specific time and location[2].

We will prove practical illustrations in the following sections.

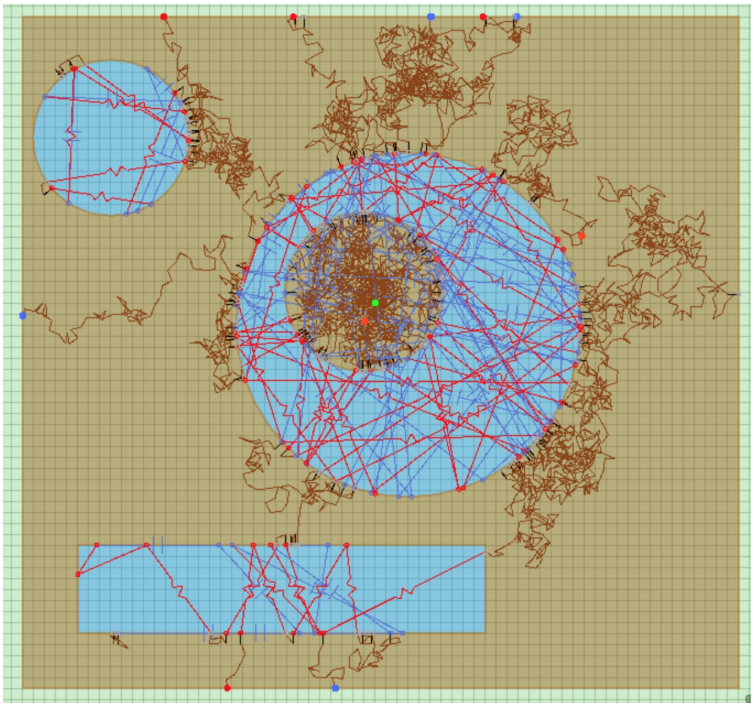

Figure 7: A coupled situation is illustrated in which fluid areas and solid areas, represented respectively in blue and brown surfaces, involve heat transfer by conduction, convection and radiation. The green dot in the central solid area is the probe position from which the paths start. Conductive sub-paths are represented in brown, radiative sub-paths are represented in red with a broken line symbol and convective sub-paths are represented in blue with a symbol indicating a capacity. In this example, the paths can either end in the domain, within the solid or fluid areas, if the initial condition has been reached, or on the boundary of the outer rectangle acting as boundary conditions. Ten paths are being displayed. They have been produced using a simplified version of *Stardis* that we use in didactic contexts (`http://www.edstar.cnrs.fr/prod/fr/training/tool/`).

2.3. Monte Carlo and the storage of propagators

At a first aim, the equations Eqs. (5) and (12) have been developed to evaluate the quantity θ with a reverse Monte Carlo algorithm. However, the essential point is this: by fully developing a formulation corresponding to Eq. (5) up to the coupling described by Eq. (12), the propagator ξ is made explicit and can be built up along the successive path samplings. This is exactly the concept described in introduction by the example with the two lamps. A major consequence is that the procedure to evaluate the propagator applies to a large class of physical problems, in complex geometries, with the same strategy as for evaluating a physics as simple as shape factors.

[2]Fig. 7 was created on an online application [13] solving two-dimensional coupled thermal problems. The implementation relies on the same ray-tracing acceleration libraries used for the complex geometries presented in the present article. The use of this application is described in a Bachelor's level learning scenario [14].

In other words, whatever the number of physics addressed, whatever the number of sub-zones and thus of sampled sub-paths, whatever the shapes of these sub-zones, the evaluation of the propagator ξ is performed recursively along the sampling of these random paths, successively crossing the various physics as diverse as they might be.

As a matter of fact, in Eq. 5, θ is expressed as an expectation along one path within the volume. In Eq. 12, θ_i is also expressed as an expectation in which coupling occurs at boundaries. If $\theta_j(x,t)$ from Eq. 12 is replaced by $\theta(x,t)$ of Eq. 5, recursivity appears in a single expectation. Hence, if both equations are coupled, they define a space of infinite dimension, either recursive in volume or at boundaries. Moreover, in this infinite-dimension expectation expression, sources from Green's theory are made explicit and each source contribution can be formally separated to express the propagator itself as an expectation. Hence, as all the sources are visited explicitly during one simulation (i.e. along the path sampling), all propagators (all source contributions) can be stored all along, without any significant additional computation effort. A simulation is performed "as if" θ was the quantity to be estimated for a given set of sources, and, more importantly, when a source is visited, the pre-factor of the corresponding source contribution is stored for later use with different sets of sources. In short, the propagators expectation needs to be estimated only once.

A main strength of the *Stardis* code used in the following practical illustration is its ability to deal with huge amounts of physical and geometrical data. In [4], the city is simulated with interacting buildings, each of them described in full details, room per room, and the main message is that the computation time is fully insensitive to the level of refinement of this description[3]. The same observation can be made concerning the computation of propagators: the computation times required for the estimation of the propagators, and for their use in external codes with new sets of sources, are also both insensitive to the refinement level.

3. Model

3.1. System description

At this stage the focus is set on radiation, diffusion and convection coupled inside systems typical of the heat-transfer engineering practice. The system is delimited with a system-boundary surface $\mathcal{S}$ that is split into $N_\mathcal{S}$ sub-surfaces $\mathcal{S}_i$. The internal volume Ω is split into N_Ω sub-volumes Ω_i of boundaries $\partial\Omega_i$ (see Fig. 8).

Each sub-volume is either a uniform opaque solid or a perfectly mixed transparent fluid. The contact between adjacent solid sub-volumes is perfect (although *Stardis* deals with thermal contact resistances, they will not be described here) and the boundary layers at solid-fluid interfaces are not described explicitly: they are summarized by a convective exchange coefficients. The thermal properties of a solid sub-volume Ω_i are the thermal conductivity λ_i, the mass density ρ_i and the mass thermal capacity c_i. For a fluid sub-volume, the fluid volume V_i is required with the thermal properties ρ_i and c_i. A power density ψ_i can also be prescribed inside each solid sub-volume. There cannot be two fluid sub-volumes adjacent to each other: a fluid sub-volume is always a fluid cell enclosed by solids.

The ensemble of all solid-fluid interfaces (between adjacent sub-volumes of different types) is noted $\mathcal{I}$. It is split into $N_\mathcal{I}$ sub-interfaces $\mathcal{I}_i$. The surface properties are uniform along each sub-interface: the convective exchange coefficient is noted h_i; the surface of the solid is grey of emissivity ϵ_i and reflection is modeled using a fraction α_i of specular reflection and a fraction $1-\alpha_i$ of diffuse reflection.

On each sub-surface $\mathcal{S}_i$, the boundary condition can be of the following types:

- *type-1* - $\mathcal{S}_i$ is along a solid sub-volume and the solid temperature is known at this boundary, noted $T_{\mathrm{B},i}$.
- *type-2* - $\mathcal{S}_i$ is along a solid sub-volume and the boundary flux density is known, noted $\varphi_{\mathrm{B},i}$.
- *type-3* - $\mathcal{S}_i$ is along a solid sub-volume, a transparent fluid is facing it, and the fluid temperature is known, noted $T_{\mathrm{BF},i}$. The boundary flux density is then the sum of the convective flux density and the radiative flux density, with uniform values of the convective exchange coefficient h_i, the emissivity ϵ_i and the specular/diffuse ratio α_i. At such a boundary, for incident directions that come from outside the system, the radiance temperature is known, noted θ_{BR}.

[3] A similar illustration is provided at `https://www.meso-star.com/projects/stardis/stardis.html` using a porous medium of increasing refinement levels.

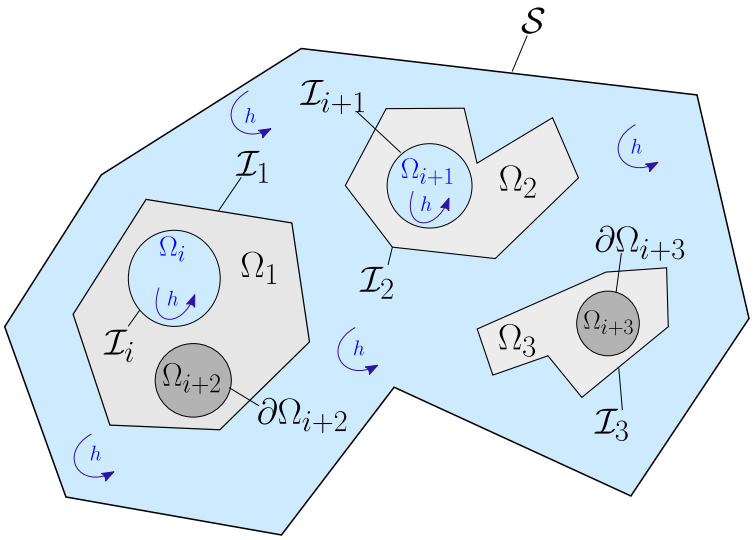

Figure 8: Sketch of the general configuration. The system boundary is $\mathcal{S}$. Three internal volumes Ω_1, Ω_2 and Ω_3 are represented. Inside Ω_1, there is a fluid inclusion Ω_i and a sub-solid Ω_{i+2}. $\mathcal{I}_i$ is the solid/fluid boundary between Ω_1 and Ω_i. $\partial\Omega_{i+2}$ is the solid boundary between Ω_{i+2} and Ω_1. Similarly, Ω_2 contains a fluid inclusion Ω_{i+1} and $\mathcal{I}_{i+1}$ is the solid/fluid boundary. The last volume Ω_3 contains a solid volume Ω_{i+3} and the solid boundary is $\partial\Omega_{i+3}$. Conduction occurs in the solid volumes, radiation and convection occur in the different fluid volumes. h is the convective heat transfer coefficient.

- *type-4 -* $\mathcal{S}_i$ is at the limit of a fluid sub-volume and the limit temperature is known, noted $T_{\mathrm{B},i}$. This temperature is to be interpreted as the one of a solid surface enclosing the fluid cell, with uniform values of the convective exchange coefficient h_i, the emissivity ϵ_i and the specular/diffuse ratio α_i.

3.2. Radiation

The solids are opaque, the fluids are transparent and photon transport is instantaneous: radiative heat transfer can be summarized to instantaneous exchanges between solid surfaces. At a location $\vec{y}$ at the surface of a solid sub-volume $\mathcal{D}_i$ facing a fluid, the radiative flux density $\varphi_R(\vec{y},t)$ is the difference between absorption of radiation in all incident directions $\vec{\omega}$ and emission by the solid due to its local temperature $T_i(\vec{y},t)$:

$$\varphi_R(\vec{y},t) = -\epsilon_i\left(\sigma T_i(\vec{y},t)^4 - \int_{\mathcal{H}_i(\vec{y})} |\vec{\omega}.\vec{n}_i(\vec{y})|\, I(\vec{\omega},\vec{y},t)\, d\vec{\omega}\right) \tag{13}$$

where $I(\vec{\omega},\vec{y},t)$ is the spectrally integrated intensity at $\vec{y}$ in direction $\vec{\omega}$, σ is the Stefan-Boltzmann constant, $\vec{n}_i(\vec{y})$ is the unit normal to the solid at $\vec{y}$ and $\mathcal{H}_i(\vec{y})$ is the hemisphere of all incident directions at $\vec{y}$.

It is assumed that radiative transfer can be linearized with respect to temperature around a given reference temperature T_{ref}, which means that $T_i^4 \approx T_{\mathrm{ref}}^4 + 4T_{\mathrm{ref}}^3(T_i - T_{\mathrm{ref}})$ leading to the expression $h_R = 4\epsilon_i\sigma T_{\mathrm{ref}}^3$. We then make the choice of translating the spectrally integrated intensity into a radiance temperature $\theta_R = \int_{\mathcal{D}_\Gamma} p_\gamma T(\vec{x}_\gamma) d\gamma$, i.e. a mean radiative temperature seen at the solid/fluid interface due to radiative exchanges through the fluid phase.

Observing that $\int_{\mathcal{H}_i(\vec{y})} \frac{|\vec{\omega}.\vec{n}_i(\vec{y})|}{\pi} d\vec{\omega} = 1$, Eq. (13) becomes

$$\varphi_R(\vec{y},t) = -h_R\left(T_i(\vec{y},t) - \int_{\mathcal{H}_i(\vec{y})} \frac{|\vec{\omega}.\vec{n}_i(\vec{y})|}{\pi}\, \theta_R(\vec{\omega},\vec{y},t)\, d\vec{\omega}\right) \tag{14}$$

We note $\vec{z} \equiv \vec{z}(\vec{y},-\vec{\omega})$ the location of first intersection with a solid sub-volume Ω_j of a straight line starting from $\vec{y}$ in direction $-\vec{\omega}$. If there is no intersection ($\vec{z}$ at infinity), then $\theta_R(\vec{\omega},\vec{y},t)$ equals the incident radiance $\theta_{\mathrm{BR}}(\vec{\omega},\vec{y},t)$ known at the system boundary. Otherwise, $\theta_R(\vec{\omega},\vec{y},t) = \theta_R(\vec{\omega},\vec{z},t)$ (pure transport) and $\theta_R(\vec{\omega},\vec{z},t)$ is modeled as the sum of the emission by the solid at temperature $T_j(\vec{z},t)$, the specular reflection of incoming radiation in direction $-\vec{\omega}_S$

where $-\vec{\omega}_S$ is the symmetric of $\vec{\omega}$ around $\vec{n}_j(\vec{z})$, and the diffuse reflection of radiation incident in all the directions $\vec{\omega}'$ of the incident hemisphere $\mathcal{H}_j(\vec{z})$ at $\vec{z}$. Altogether,

$$\begin{cases} \text{If } \vec{z} \text{ at } \infty : & \theta_R(\vec{\omega},\vec{y},t) = \theta_{\text{BR}}(\vec{\omega},\vec{y},t) \\ \text{If } \vec{z} \in \partial\mathcal{D}_j : & \theta_R(\vec{\omega},\vec{y},t) = \epsilon_j T_j(\vec{z},t) + (1-\epsilon_j)\alpha_j \theta_R(-\vec{\omega}_S,\vec{z},t) \\ & \qquad + (1-\epsilon_j)(1-\alpha_j) \displaystyle\int_{\mathcal{H}_j(\vec{z})} \frac{|\vec{\omega}'.\vec{n}_j(\vec{z})|}{\pi}\, \theta_R(\vec{\omega}',\vec{z},t)\, d\vec{\omega}' \end{cases} \tag{15}$$

3.3. Diffusion

At any location $\vec{x}$ inside a solid sub-volume $\mathcal{D}_i$, at any time t, the solid temperature $T_i \equiv T_i(\vec{x},t)$ is solution of the following heat equation,

$$\rho_i c_i \frac{\partial T_i}{\partial t} = \lambda_i \Delta T_i + \psi_i \tag{16}$$

where $\psi_i \equiv \psi_i(\vec{x},t)$ is the local value of the power density. The initial condition at time t_{I} is

$$T_i(\vec{x},t_{\text{I}}) = T_{\text{I},i}(\vec{x}) \tag{17}$$

At any location $\vec{y}$ at the boundary of $\mathcal{D}_i$ (i.e. $\vec{y} \in \partial\mathcal{D}_i$), at any time t, the modeling of the interface or the boundary condition is one of the following :

- If $\vec{y}$ is at an interface with another solid sub-volume Ω_j,

$$\lambda_i \vec{\nabla} T_i.\vec{n_i} = \lambda_j \vec{\nabla} T_j.\vec{n_i} \tag{18}$$

- If $\vec{y}$ is at an interface $\mathcal{I}_k$ with a fluid sub-volume $\mathcal{D}_j$ (with $h_R = 4\epsilon_k \sigma T_{\text{ref}}^3$),

$$-\lambda_i \vec{\nabla} T_i.\vec{n_i} = h_k(T_j - T_i) - h_R\left(T_i - \int_{\mathcal{H}_i(\vec{y})} \frac{|\vec{\omega}.\vec{n}_i(\vec{y})|}{\pi}\, \theta_R(\vec{\omega},\vec{y},t)\, d\vec{\omega}\right) \tag{19}$$

- If $\vec{y}$ is at the boundary of the system, in a sub-surface $\mathcal{S}_j$ with a *type-1* boundary condition,

$$T_i = T_{\text{B},j} \tag{20}$$

- If $\vec{y}$ is at the boundary of the system, in a sub-surface $\mathcal{S}_j$ with a *type-2* boundary condition,

$$-\lambda_i \vec{\nabla} T_i.\vec{n_i} = \varphi_{\text{B},j} \tag{21}$$

- If $\vec{y}$ is at the boundary of the system, in a sub-surface $\mathcal{S}_j$ with a *type-3* boundary condition (with $h_R = 4\epsilon_j \sigma T_{\text{ref}}^3$),

$$\begin{aligned} &-\lambda_i \vec{\nabla} T_i.\vec{n_i} = \\ &h_j(T_{\text{BF},j} - T_i) - h_R\left(T_i - \int_{\mathcal{H}_i(\vec{y})} \frac{|\vec{\omega}.\vec{n}_i(\vec{y})|}{\pi}\, \theta_R(\vec{\omega},\vec{y},t)\, d\vec{\omega}\right) \end{aligned} \tag{22}$$

3.4. Convection

Inside a fluid sub-volume $\mathcal{D}_i$, at any time t, the fluid temperature $T_i \equiv T_i(t)$ is uniform and its evolution equation is

$$\rho_i c_i V_i \frac{dT_i}{dt} = \int_{\partial\mathcal{D}_i} h(\vec{y})(T_S(\vec{y}) - T_i) d\vec{y} \tag{23}$$

where $h(\vec{y})$ and $T_S(\vec{y})$ are respectively the convective exchange coefficient and the surface temperature at $\vec{y}$ on one of the solid surfaces delimiting the fluid cell. If $\vec{y}$ is at an interface $\mathcal{I}_k$ with a solid sub-volume $\mathcal{D}_j$, then $h(\vec{y}) = h_k$ and $T_S(\vec{y}) = T_j(\vec{y},t)$. If $\vec{y}$ is at the boundary of the system, in a sub-surface $\mathcal{S}_j$ with a *type-4* boundary condition, then $h(\vec{y}) = h_j$ and $T_S(\vec{y}) = T_{\text{B},j}(\vec{y},t)$.

4. Path sampling and propagation

In this section, the reverse Monte Carlo algorithm is exposed, where sampling paths is driven by the model above, to estimate a local temperature at location $\vec{x}$ and time t, or a radiance temperature at location $\vec{x}$ and time t in direction $\vec{\omega}$.

When integrated quantities are required (an average temperature on a volume or a surface, a spatially and angularly integrated radiance for simulation of infrared camera pixels, ...), the only algorithmic change is that, prior to initiating a thermal path, $\vec{x}$ and/or $\vec{\omega}$ are sampled accordingly. Further description of such extensions will not be provided here.

4.1. A path sampling Monte Carlo algorithm

Depending on the choice of the quantity of interest, the estimate will yield either $\theta_R(\vec{\omega}, \vec{x}, t)$ for $\vec{x}$ inside a fluid sub-volume, either $T_i(\vec{x}, t)$ for $\vec{x}$ inside a solid sub-volume, or $T_i(., t)$ for any location inside a given fluid sub-volume. In each case, N thermal paths γ_j are sampled and each path is used to produce a Monte Carlo weight w_{γ_j}. These weights are then averaged to produce an estimate m of the addressed quantity, together with a standard error s associated to this estimate, that can be interpreted in term of a numerical uncertainty.

$$T_i(\vec{x}, t) \text{ or } T_i(., t) \text{ or } \theta_R(\vec{\omega}, \vec{x}, t) \approx m = \frac{1}{N} \sum_{j=1}^{N} w_{\gamma_j} \tag{24}$$

$$s = \frac{1}{\sqrt{N}} \left(\frac{1}{N} \sum_{j=1}^{N} w_{\gamma_j}^2 - m^2 \right)^{\frac{1}{2}} \tag{25}$$

The focus is here set on the calculation of w_γ for any path γ, highlighting its propagative nature and how the information about the sources is collected along the path.

A thermal path is structured as a succession of diffusive, convective and radiative sub-paths. From this point of view, the only difference between the paths used to evaluate $\theta_R(\vec{\omega}, \vec{x}, t)$ in a fluid, $T_i(\vec{x}, t)$ in a solid or $T_i(., t)$ in a fluid is that they start with a radiative sub-path, a diffusive sub-path or a convective sub-path respectively. Each sub-path can be therefore considered independently, only keeping in mind that: a) at the beginning of the first sub-path the Monte Carlo weight is initialized to $w_\gamma = 0$, and b) that the end of each sub-path is either the start of a new sub-path, or the end of the whole path γ. Each path γ ends at a location $\vec{x}_{\gamma,\text{end}}$, either inside the system at the initial time t_I or at the boundary at a time $t_{\gamma,\text{end}}$. When it ends with a known incident radiant temperature at the boundary, the corresponding incident direction is $\vec{\omega}_{\gamma,\text{end}}$.

As announced in the preceding theoretical part (Sec. 2), we start the description "as if" the only objective was the Monte Carlo estimation of temperatures or radiances, and not their associated propagators. This description is exhaustive as far as the path-sampling algorithms are concerned, but we do not recall the theoretical developments justifying them [4, 15]. We will focus successively on the sub-paths associated to each of the three physics considered (Secs. 4.1.1, 4.1.2 and 4.1.3), then on the probabilization of the coupling at the interfaces (Sec. 4.2). Once the whole path-sampling Monte Carlo algorithm is set, i.e. when the weight associated to each path is fully defined (Sec. 4.3), we will explain how propagators are constructed for each source by splitting the weight expression into parts and storing the corresponding data (Sec. 5).

4.1.1. Radiative sub-paths

Radiative sub-paths are constructed using a standard backward tracking multiple-reflection algorithm. Starting from $\vec{x}$ with the objective of evaluating $\theta_R(\vec{\omega}, \vec{x}, t)$, a ray is traced in the scene in direction $-\vec{\omega}$, looking for a first intersection $\vec{z}_1$ with a solid surface. If no intersection is found ($\vec{z}_1$ is at infinity), then our radiation model says that $\theta_R(\vec{\omega}, \vec{x}, t) = \theta_{\text{BR}}(\vec{\omega}, \vec{x}, t)$ where θ_{BR} is a known incident radiance temperature. In this case, the path γ is ended at location $\vec{x}_{\gamma,\text{end}} = \vec{x}$, the time $t_{\gamma,\text{end}} = t$ and the direction $\vec{\omega}_{\gamma,\text{end}} = \vec{\omega}$.

The Monte Carlo weight is increased by θ_{BR}:

$$w_\gamma \mathrel{+}= \theta_{\text{BR}}(\vec{\omega}_{\gamma,\text{end}}, \vec{x}_{\gamma,\text{end}}, t_{\gamma,\text{end}}) \tag{26}$$

Otherwise $\vec{z}_1$ belongs either to a sub-surface $\mathcal{S}_j$ or a sub-interface $\mathcal{S}_j$ where the emissivity ϵ_j and the specular/diffuse fraction α_j are known. A Bernoulli test of probability ϵ_j is made to decide whether absorption occurs. If the test is true, the radiative sub-path is ended at $\vec{z}_1$. If not, reflection occurs with another Bernoulli test to decide between specular or diffuse reflection, with a Lambertian sampling of the reflection direction in the latter case. The path tracing process is then continued from $\vec{z}_1$ in the direction of reflection $-\vec{\omega}_1$, etc, thus defining a succession of possible reflections at locations $\vec{z}_1$, $\vec{z}_2$, $\vec{z}_3$ ... until either no reflection is found or absorption occurs. If no reflection is found, the path is ended with the Monte Carlo weight increment of Eq. (26). When absorption occurs as a location $\vec{z}_k$, then there are two possible cases :

- If $\vec{z}_k$ belongs to a sub-surface at the system boundary, the temperature T_B is known at this surface (a radiative path travels necessarily in a fluid and only a *type-4* boundary condition can be encountered), the path γ is ended and the Monte Carlo weight is increased by T_B:

$$w_\gamma \mathrel{+}= T_\mathrm{B}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}) \tag{27}$$

  with $\vec{x}_{\gamma,\mathrm{end}} = \vec{z}_k$ and $t_{\gamma,\mathrm{end}} = t$.

- If $\vec{z}_k$ belongs to a sub-interface, then the encountered solid is inside the system and its temperature is unknown. The path γ must be continued with a new sub-path (with no change of w_γ) and a test is made to decide between the three heat transfer modes occurring at this interface: a radiative sub-path back into the fluid, a convective sub-path also in the fluid, or a diffusive path inside the solid. This test is the object of Sec. 4.2.

Radiative sub-paths : Summary

- A radiative sub-path is an instantaneous backward traced ray in a transparent fluid with multiple reflections at solid surfaces.
- If the sub-path encounters a known incident radiance or a known solid temperature, then γ is ended and the Monte Carlo weight is increased by θ_BR or T_B.
- Otherwise, the Monte Carlo weight is unchanged and at the absorption location, γ is continued with the start of another sub-path at the corresponding solid-fluid interface.

4.1.2. Diffusive sub-paths

Conductive sub-paths are approximate Brownian motions backward in time and space inside a solid. They are constructed as successions of jumps of arbitrary length δ and in isotropically sampled directions. Convergence towards the exact solution is obtained for $\delta \to 0$ (Brownian motion is only exact at the limit $\delta = 0$ [16]). However, as the computational time increases considerably when the value of δ decreases, a compromise is required between computational cost and precision. Hence, δ needs to be set sufficiently low to ensure a satisfactory accuracy on the obtained solution and sufficiently high to provide an appropriate computational time.

Starting from $\vec{x}$ with the objective of evaluating $T(\vec{x}, t)$, the first algorithmic step is the sampling of a backward time shift δt according to an exponential law of parameter $\tau_i = \frac{\delta^2 \rho_i c_i}{6\lambda_i}$, i.e.

$$\delta t = -\tau_i \ln(r) \tag{28}$$

where r is sampled uniformly on $[0, 1]$. If $t - \delta t < t_\mathrm{I}$ (the backward shift has crossed the initial time), then γ is ended and the Monte Carlo weight is increased by the initial temperature:

$$w_\gamma \mathrel{+}= T_\mathrm{I}(\vec{x}_{\gamma,\mathrm{end}}) \tag{29}$$

with $\vec{x}_{\gamma,\mathrm{end}} = \vec{x}$. Otherwise a direction $\vec{u}$ is sampled isotropically in the unit sphere, a jump is made from $\vec{x}$ to $\vec{x} + \delta\vec{u}$ and the Monte Carlo weight is increased to account for the local power density ψ:

$$w_\gamma \mathrel{+}= \beta_\psi(\vec{x})\psi(\vec{x}, t - \delta t) \tag{30}$$

with $\beta_\psi = \frac{\delta^2}{6\lambda_i}$.

In the vicinity of a solid surface, δ is adjusted depending on $\vec{u}$ so that $\vec{x} + \delta\vec{u}$ may either remain inside the solid or reach the solid surface exactly. While $\vec{x} + \delta\vec{u}$ remains in the solid, say $\mathcal{D}_i$, then the diffusive sub-path is repeatedly continued from location to location until reaching the surface $\partial\mathcal{D}_i$.

When $\vec{x} + \delta\vec{u}$ reaches $\partial\mathcal{D}_i$, say at a location $\vec{z}$ and time t_z, the diffusive sub-path is stopped. If $\vec{z}$ belongs to an interface with another solid sub-volume $\mathcal{D}_j$, then the temperature of the interface is unknown and γ must be continued with another diffusive sub-path, initiated either inside $\mathcal{D}_i$ or inside $\mathcal{D}_j$. If $\vec{z}$ belongs to an interface with a fluid sub-volume, then the temperature of the interface is also unknown and γ must also be continued, and the next sub-path can be either a diffusive one, back into $\mathcal{D}_i$, or a convective or a radiative one inside or through the fluid. The corresponding tests are described in Sec. 4.2.

If $\vec{z}$ is at the boundary of the system, then the algorithm depends on the boundary condition type:

- For a *type-1* boundary condition, the boundary temperature T_B is known and γ is stopped and the Monte Carlo weight is increased by T_B:
$$w_\gamma \mathrel{+}= T_\mathrm{B}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}) \tag{31}$$
with $\vec{x}_{\gamma,\mathrm{end}} = \vec{z}$ and $t_{\gamma,\mathrm{end}} = t_z$.
- For a *type-2* boundary condition, the location is shifted back into the solid sub-volume, of a distance δ along the normal, the diffusive sub-path is continued from this new location, and the Monte Carlo weight is increased to account for the value of the local flux density:
$$w_\gamma \mathrel{+}= \beta_\varphi(\vec{z})\varphi(\vec{z}, t_z) \tag{32}$$
with $\beta_\varphi = \frac{\delta}{\lambda_i}$.
- For a *type-3* boundary condition, neither the boundary temperature nor the flux density is known and γ is continued exactly the same way as for a solid-fluid interface inside the system (see Sec. 4.2). The only difference is that when the following sub-path is a convective one, then the fluid temperature T_BF is known and γ is ended. In this case, the Monte Carlo weight is increased by T_BF:
$$w_\gamma \mathrel{+}= T_\mathrm{BF}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}) \tag{33}$$
with $\vec{x}_{\gamma,\mathrm{end}} = \vec{z}$ and $t_{\gamma,\mathrm{end}} = t_z$ (t_z is the time at which the $\vec{z}$ location was reached).

Diffusive sub-paths : Summary

- A diffusive sub-path is a Brownian motion backward in time inside a solid sub-volume until it reaches either the initial time or the sub-volume boundary.
- If the initial time is reached, then γ is ended and the Monte Carlo weight is increased by T_I.
- If the system boundary is reached at location where the temperature is known, then γ is ended and the Monte Carlo weight is increased by T_B.
- If the system boundary is reached at location where the flux density is known, then a new diffusive sub-path is initiated, inside the same sub-volume, and the Monte Carlo weight is increased by $\beta_\varphi\varphi$.
- In all other cases, the diffusive sub-path has reached a location where neither the temperature nor the flux density is known, and a new sub-path (diffusive, convective or radiative) must be initiated from the corresponding interface.
- Along the path, the Monte Carlo weight is increased to account for the local value of the volume source density ψ.
- As Brownian motion is approximated with discrete jumps of length δ, the continuous effect of the source is replaced by a Monte Carlo weight increment of $\beta_\psi\psi$ at each jump.

4.1.3. Convective sub-paths

Convective sub-paths inside a fluid sub-volume $\mathcal{D}_i$ are independent of their initial location: the fluid cells are perfectly mixed so T_i is only a function of time, and the only required information is the time t at which the convective sub-path was initiated. From t, a backward time shift δt is sampled according to an exponential law of parameter $\frac{h_i V_i}{\rho_i c_i S_i}$, i.e.

$$\delta t = \frac{h_i V_i}{\rho_i c_i S_i} \ln(r) \tag{34}$$

where S_i and V_i are respectively the surface and volume of the fluid cavity, and r is sampled uniformly on $[0, 1]$. If $t - \delta t < t_\mathrm{I}$ (the backward shift has crossed the initial time), then γ is ended and the Monte Carlo weight is increased by the initial temperature:

$$w_\gamma \mathrel{+}= T_{\mathrm{I},i} \tag{35}$$

Otherwise a location $\vec{z}$ is sampled on $\partial\mathcal{D}_i$ according to probability density $p_{\vec{Z}}$ proportional to the local value of the convective exchange coefficient:

$$p_{\vec{Z}}(\vec{z}) = \frac{h(\vec{z})}{\int_{\partial\mathcal{D}_i} h(\vec{z}')d\vec{z}'} \tag{36}$$

and the time is shifted to $t_z = t - \delta t$.

If $\vec{z}$ is at the system boundary, then this corresponds necessarily to a *type-4* boundary condition and the boundary temperature T_B is known, so γ is ended and the Monte Carlo weight is increased by T_B:

$$w_\gamma \mathrel{+}= T_\mathrm{B}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}) \tag{37}$$

with $\vec{x}_{\gamma,\mathrm{end}} = \vec{z}$ and $t_{\gamma,\mathrm{end}} = t_z$. Otherwise $\vec{z}$ is at a solid-fluid interface inside the system and the interface temperature is unknown. γ is then continued with a new diffusive, convective or radiative sub-path as described in Sec. 4.2.

Convective sub-paths : Summary

- A convective sub-path inside a fluid sub-volume is only a backward exponential shift in time.
- If the initial time is reached, then γ is ended and the Monte Carlo weight is increased by T_I.
- Otherwise a location is sampled on one of the solid surfaces surrounding the fluid.
- If this location is at the boundary of the system and the surface temperature is known, then γ is ended and the Monte Carlo weight is increased by T_B.
- Otherwise the convective path has reached an interface where the temperature is unknown and a new sub-path (diffusive, convective or radiative) must be initiated from this interface.

4.2. Choosing the next sub-path at an interface

4.2.1. Solid-solid interface

When describing diffusive sub-paths, we encountered an algorithmic step where a location $\vec{z}$ was reached, at the interface between two solid sub-volumes $\mathcal{D}_i$ and $\mathcal{D}_j$, at time t_z. At this interface, the temperature was unknown and γ had to be continued. The same question is raised when γ needs to be started at such a location in order to evaluate $T(\vec{x}, t)$ with $\vec{x} = \vec{z}$ and $t = t_z$. This is achieved by first shifting $\vec{z}$ along the normal, either by a distance δ_i inside $\mathcal{D}_i$ or by a distance δ_j inside $\mathcal{D}_j$, where δ_i and δ_j are the values of the numerical parameter δ used inside $\mathcal{D}_i$ and inside $\mathcal{D}_j$ respectively (depending of their characteristic dimensions). Then a diffusive sub-path is started from this shifted location, still at the same date. Choosing the side is made by retaining $\mathcal{D}_i$ with probability $P_{\mathrm{cond},i}$ and $\mathcal{D}_j$ with probability $P_{\mathrm{cond},j}$:

$$\begin{aligned} P_{\mathrm{cond},i} &= \frac{\frac{\lambda_i}{\delta_i}}{\frac{\lambda_i}{\delta_i} + \frac{\lambda_j}{\delta_j}} \\ P_{\mathrm{cond},j} &= 1 - P_{\mathrm{cond},i} \end{aligned} \tag{38}$$

4.2.2. *Solid-fluid interface*

When describing each of the three sub-path types, we encountered the possibility that a location $\vec{z}$ is reached, at an interface $\mathcal{I}_k$ between a solid sub-volume $\mathcal{D}_i$ and a fluid sub-volume $\mathcal{D}_j$, at time t_z. At this interface the temperature is unknown and γ is to be continued. The same question is raised when γ needs to be started at such a location in order to evaluate $T(\vec{x}, t)$ with $\vec{x} = \vec{z}$ and $t = t_z$. The same question is also encountered when a diffusive sub-path reaches a location at the system boundary with *type-4* boundary condition. We focus the description on the case of an internal interface between $\mathcal{D}_i$ and $\mathcal{D}_j$.

At such an interface, all three heat transfer modes are present: diffusion inside $\mathcal{D}_i$, convection and radiation inside Ω_j. A test is therefore made to pick among a diffusive, a convective or a radiative sub-path. Conductive, convective and radiative sub-paths are picked with probabilities P_{cond}, P_{conv} and P_{ray} respectively, with

$$\begin{aligned} P_{\text{cond}} &= \frac{\frac{\lambda_i}{\delta_i}}{\frac{\lambda_i}{\delta_i} + h_k + h_R} \\ P_{\text{conv}} &= \frac{h_k}{\frac{\lambda_i}{\delta_i} + h_k + h_R} \\ P_{\text{ray}} &= 1 - P_{\text{cond}} - P_{\text{conv}} \end{aligned} \tag{39}$$

If diffusion is retained, then $\vec{z}$ is shifted by a distance δ_i along the normal inside $\mathcal{D}_i$ and the diffusive sub-path is initiated at this shifted location. For convection or radiation, the corresponding sub-path is initiated at $\vec{z}$ at time t_z.

Choosing the next sub-path at a solid-solid or solid-fluid interface : Summary

- Departing from a solid-solid or a solid-fluid interface is made by initiating a sub-path with a heat transfer mode that is sampled according to probabilities reflecting the flux continuity through the interface.
- When a diffusive sub-path is chosen, the starting location is shifted by a distance δ inside the solid.
- In all cases, there is no increment made to the Monte Carlo weight.

4.3. *The Monte Carlo weight*

As mentioned above, each path γ ends at a location $\vec{x}_{\gamma,\text{end}}$, either inside the system at the initial time t_I or at the boundary at a time $t_{\gamma,\text{end}}$. When it ends with a known incident radiant temperature at the boundary, the corresponding incident direction is $\vec{\omega}_{\gamma,\text{end}}$. In each case, the Monte Carlo weight is increased by a temperature value that can be $T_\text{I}(\vec{x}_{\gamma,\text{end}})$, $T_\text{B}(\vec{x}_{\gamma,\text{end}}, t_{\gamma,\text{end}})$, $T_\text{BF}(\vec{x}_{\gamma,\text{end}}, t_{\gamma,\text{end}})$ or $\theta_\text{BR}(\vec{\omega}_{\gamma,\text{end}}, \vec{x}_{\gamma,\text{end}}, t_{\gamma,\text{end}})$. Let $\mu_{\gamma,\text{end}}$ denote the type of ending, from 0 to 3 in the order of this list.

We can then define $T_{\gamma,\text{end}} \equiv T_{\gamma,\text{end}}(\vec{x}_{\gamma,\text{end}}, t_{\gamma,\text{end}}, \vec{\omega}_{\gamma,\text{end}}, \mu_{\gamma,\text{end}})$ as :

$$\begin{aligned} T_{\gamma,\text{end}}(\vec{x}_{\gamma,\text{end}}, t_{\gamma,\text{end}}, \vec{\omega}_{\gamma,\text{end}}, 0) &= T_\text{I}(\vec{x}_{\gamma,\text{end}}) \\ T_{\gamma,\text{end}}(\vec{x}_{\gamma,\text{end}}, t_{\gamma,\text{end}}, \vec{\omega}_{\gamma,\text{end}}, 1) &= T_\text{B}(\vec{x}_{\gamma,\text{end}}, t_{\gamma,\text{end}}) \\ T_{\gamma,\text{end}}(\vec{x}_{\gamma,\text{end}}, t_{\gamma,\text{end}}, \vec{\omega}_{\gamma,\text{end}}, 2) &= T_\text{BF}(\vec{x}_{\gamma,\text{end}}, t_{\gamma,\text{end}}) \\ T_{\gamma,\text{end}}(\vec{x}_{\gamma,\text{end}}, t_{\gamma,\text{end}}, \vec{\omega}_{\gamma,\text{end}}, 3) &= \theta_\text{BR}(\vec{\omega}_{\gamma,\text{end}}, \vec{x}_{\gamma,\text{end}}, t_{\gamma,\text{end}}) \end{aligned} \tag{40}$$

Along γ, diffusive sub-paths may have crossed solid sub-volumes with a volume power source ψ, and we have seen that the Monte Carlo weight was increased by $\beta_\psi \psi$ at each discrete jump location. Let N_ψ denote the number of such locations, and $\vec{x}_{\gamma,\psi}(k)$ and $t_{\gamma,\psi}(k)$ the location and time of the k-th of these Monte Carlo weight increments. Similarly, diffusive sub-paths may have visited boundary locations where the flux density φ is known, and we have seen that the Monte Carlo weight was increased by $\beta_\varphi \varphi$ at each such visit. Let N_φ denote the number of such visits, and $\vec{x}_{\gamma,\varphi}(k)$ and $t_{\gamma,\varphi}(k)$ the location and time of the k-th of these Monte Carlo weight increments.

With these notations, the complete expression of the Monte Carlo weight associated to γ is

$$\begin{aligned} w_\gamma = & \sum_{k=1}^{N_\psi} \beta_\psi(\vec{x}_{\gamma,\psi}(k))\, \psi\left(\vec{x}_{\gamma,\psi}(k), t_{\gamma,\psi}(k)\right) \\ & + \sum_{k=1}^{N_\varphi} \beta_\varphi(\vec{x}_{\gamma,\varphi}(k))\, \varphi\left(\vec{x}_{\gamma,\varphi}(k), t_{\gamma,\varphi}(k)\right) \\ & + T_{\gamma,\mathrm{end}}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}, \vec{\omega}_{\gamma,\mathrm{end}}, \mu_{\gamma,\mathrm{end}}) \end{aligned} \tag{41}$$

The reading of this weight expression illustrates the relation between such path sampling Monte Carlo algorithms and Green's theory. From Green's point of view, the temperature solution of the problem at $(\vec{x}*, t*)$, or the radiance temperature at $(\vec{\omega}*, \vec{x}*, t*)$ results from an integral of all the sources, T_I, T_B, T_BF, θ_BR, ψ and φ, at $(\vec{x}, t, \vec{\omega})$ multiplied by a propagator density that is independent of the source values. From Monte Carlo point of view, it results from the average of a large number of weights that carry the same sources from $(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}, \vec{\omega}_{\gamma,\mathrm{end}})$ or from $(\vec{x}_{\gamma,\psi}(k), t_{\gamma,\psi}(k))$ and $(\vec{x}_{\gamma,\varphi}(k), t_{\gamma,\varphi}(k))$, multiplied by factors that are independent of the source values: the factor is 1 for T_I, T_B, T_BF and θ_BR, it is β_ψ for ψ and β_φ for φ. The paths sample $\zeta(\vec{x}, t, \vec{x}_S, t_S) S(\vec{x}_S, t_S)$ in a backward manner.

5. Stardis : storing the propagation data

5.1. Illustration of the principle in the case of two sources

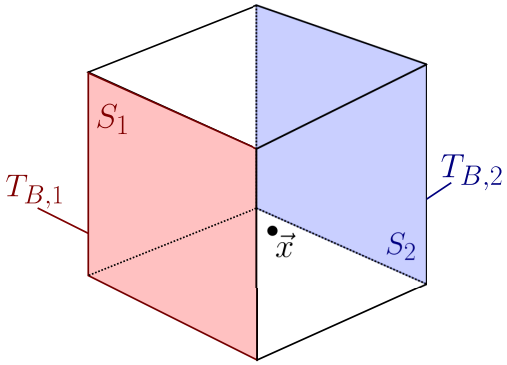

Figure 9: Schemes of the solid cube with two isothermal faces S_1 and S_2, respectively at temperature $T_{\mathrm{B},1}$ and $T_{\mathrm{B},2}$, with a location $\vec{x}$ inside the solid volume.

Let us start by considering a simple configuration akin to the radiative transfer example with two lamps of power $\mathcal{P}_1$ and $\mathcal{P}_2$ viewed from a camera pixel.

Here, the heat transfer mode is diffusion inside a cubic solid with two isothermal faces $\mathcal{S}_1$ and $\mathcal{S}_2$, facing each other, at temperatures $T_{\mathrm{B},1}$ and $T_{\mathrm{B},2}$, the other four faces being adiabatic (see Fig. 9). The addressed quantity is the stationary temperature $T(\vec{x})$ at a location $\vec{x}$ inside the solid. In terms of Green's theory, as the problem is stationary, no propagator is required for the initial condition (reported to $-\infty$). There are no volume sources and the only imposed surface flux is null (at the adiabatic faces). The only sources are therefore $T_{\mathrm{B},1}$ and $T_{\mathrm{B},2}$ and we note $\zeta_{\mathrm{B},1}(\vec{x})$ and $\zeta_{\mathrm{B},2}(\vec{x})$ the corresponding propagators:

$$T(\vec{x}) = \zeta_{\mathrm{B},1}(\vec{x}) T_{\mathrm{B},1} + \zeta_{\mathrm{B},2}(\vec{x}) T_{\mathrm{B},2} \tag{42}$$

Considering the expression of the Monte Carlo weight in Eq. (41), how can we provide an estimate for $\zeta_{\mathrm{B},1}(\vec{x})$ and one for $\zeta_{\mathrm{B},2}(\vec{x})$ using the same thermal paths as those used to estimate $T(\vec{x})$?

In the expression of the Monte Carlo weight of the preceding section, for this simple case, the sums over N_ψ and N_φ vanish (no surface flux, no volume flux), $\vec{\omega}_{\gamma,\mathrm{end}}$ is not used (we estimate a local temperature and not a radiance

temperature), $t_{\gamma,\text{end}}$ is unused (the problem is stationary) and $\mu_{\gamma,\text{end}} = 1$ (the path can only end at $\mathcal{S}_1$ or $\mathcal{S}_2$, i.e. at a boundary with a known solid temperature). Eq. (41) reduces to

$$w_\gamma = T_{\gamma,\text{end}}(\vec{x}_{\gamma,\text{end}}, 1) \tag{43}$$

with $T_{\gamma,\text{end}}(\vec{x}_{\gamma,\text{end}}, 1) = T_{\text{B},1}$ if $\vec{x}_{\gamma,\text{end}} \in \mathcal{S}_1$ and $T_{\gamma,\text{end}}(\vec{x}_{\gamma,\text{end}}, 1) = T_{\text{B},2}$ if $\vec{x}_{\gamma,\text{end}} \in \mathcal{S}_2$.

In the spirit of the example used in introduction, let us address $\zeta_{\text{B},1}(\vec{x})$ by "turning off" the second source, i.e. $T_{\text{B},2} = 0$, so that $\zeta_{\text{B},1}(\vec{x}) = \frac{T(\vec{x})}{T_{\text{B},1}}$.

This defines the Monte Carlo weight to be used for estimating $\zeta_{\text{B},1}(\vec{x})$ as : $w_\gamma^{\text{B},1} = \frac{w_\gamma}{T_{\text{B},1}} = \frac{T_{\gamma,\text{end}}(\vec{x}_{\gamma,\text{end}},1)}{T_{\text{B},1}}$
with $T_{\gamma,\text{end}}(\vec{x}_{\gamma,\text{end}}, 1) = T_{\text{B},1}$ if $\vec{x}_{\gamma,\text{end}} \in \mathcal{S}_1$ and $T_{\gamma,\text{end}}(\vec{x}_{\gamma,\text{end}}, 1) = 0$ if $\vec{x}_{\gamma,\text{end}} \in \mathcal{S}_2$,
namely, $w_\gamma^{\text{B},1} = 1$ if $\vec{x}_{\gamma,\text{end}} \in \mathcal{S}_1$ and $w_\gamma^{\text{B},1} = 0$ if $\vec{x}_{\gamma,\text{end}} \in \mathcal{S}_2$.
This writes

$$w_\gamma^{\text{B},1} = \mathcal{H}(\vec{x}_{\gamma,\text{end}} \in \mathcal{S}_1) \tag{44}$$

where $\mathcal{H}$ is a test function, taking the value 1 if the condition is valid and 0 otherwise. Similarly,

$$w_\gamma^{\text{B},2} = \mathcal{H}(\vec{x}_{\gamma,\text{end}} \in \mathcal{S}_2) \tag{45}$$

In algorithmic terms,

- N paths γ_j are sampled;
- $w_{\gamma_j}^{\text{B},1}$ is computed for each path (1 if $\mathcal{S}_1$ is reached, 0 otherwise);
- $w_{\gamma_j}^{\text{B},2}$ is computed for each path (1 if $\mathcal{S}_2$ is reached, 0 otherwise);
- the Monte Carlo estimate of $\zeta_{\text{B},1}(\vec{x})$ is $m_{\text{B},1} = \frac{1}{N}\sum_{j=1}^{N} w_{\gamma_j}^{\text{B},1}$;
- the Monte Carlo estimate of $\zeta_{\text{B},2}(\vec{x})$ is $m_{\text{B},2} = \frac{1}{N}\sum_{j=1}^{N} w_{\gamma_j}^{\text{B},2}$.

Then, Eq. (42) can be used to estimate the results for any set of source values.

5.2. Implementation in Stardis

In *Stardis*, we consider scenes where boundary conditions can be split into a set of constant and uniform sources: $\theta_{i\text{BR}}$ does not depend on location, time and direction, and for each geometrical element i, $T_{\text{B},i}$, $\varphi_{\text{B},i}$ and ψ_i are constant and uniform, $T_{\text{I},i}$ does not depend on location. λ_i, ρ_i, c_i, h_i, ϵ_i and α_i are also considered uniform over element i.

Under these assumptions, $\zeta(\vec{x}, t, \vec{x}_S, t_S)$ can be aggregated by geometrical element as done above in Eqs. (44) and (45). For this, we use the test function to build one weight expression for each geometrical element i and each type of source, following:

$$\begin{aligned}
w_\gamma^{\text{I},i} &= \mathcal{H}(\vec{x}_{\gamma,\text{end}} \in \Omega_i)\mathcal{H}(\mu_{\gamma,\text{end}} = 0)\\
w_\gamma^{\text{B},i} &= \mathcal{H}(\vec{x}_{\gamma,\text{end}} \in \mathcal{S}_i)\mathcal{H}(\mu_{\gamma,\text{end}} = 1)\\
w_\gamma^{\text{BF},i} &= \mathcal{H}(\vec{x}_{\gamma,\text{end}} \in \mathcal{S}_i)\mathcal{H}(\mu_{\gamma,\text{end}} = 2)\\
w_\gamma^{\text{BR}} &= \mathcal{H}(\mu_{\gamma,\text{end}} = 3)\\
w_\gamma^{\psi,i} &= \sum_{k=1}^{N_\psi} \mathcal{H}(\vec{x}_{\gamma,\psi}(k) \in \Omega_i)\beta_\psi(\vec{x}_{\gamma,\psi}(k))\\
w_\gamma^{\varphi,i} &= \sum_{k=1}^{N_\varphi} \mathcal{H}(\vec{x}_{\gamma,\varphi}(k) \in \mathcal{S}_i)\beta_\varphi(\vec{x}_{\gamma,\varphi}(k))
\end{aligned} \tag{46}$$

These weights are then just a splitting of the weights given by Eq. 41 and only them need to be stored while solving the latter. The Monte Carlo algorithm is therefore modified to estimate:

$$
\begin{aligned}
m_{\mathrm{I},i} &= \frac{1}{N}\sum_{j=1}^{N} w_{\gamma_j}^{\mathrm{I},i} \\
m_{\mathrm{B},i} &= \frac{1}{N}\sum_{j=1}^{N} w_{\gamma_j}^{\mathrm{B},i} \\
m_{\mathrm{BF},i} &= \frac{1}{N}\sum_{j=1}^{N} w_{\gamma_j}^{\mathrm{BF},i} \\
m_{\mathrm{BR}} &= \frac{1}{N}\sum_{j=1}^{N} w_{\gamma_j}^{\mathrm{BR}} \\
m_{\psi,i} &= \frac{1}{N}\sum_{j=1}^{N} w_{\gamma_j}^{\psi,i} \\
m_{\varphi,i} &= \frac{1}{N}\sum_{j=1}^{N} w_{\gamma_j}^{\varphi,i}
\end{aligned}
\tag{47}
$$

Finally, once these estimates have been constructed, they can be used to estimate the addressed quantity for any set of sources:

$$
\begin{aligned}
T(\vec{x},t) \text{ or } T(t) \text{ or } \theta_R(\vec{\omega},\vec{x},t) \approx m = &\sum_{i=1}^{N_\omega} m_{\mathrm{I},i}\, T_{\mathrm{I},i} \\
&+ \sum_{i=1}^{N_S} m_{\mathrm{B},i}\, T_{\mathrm{B},i} \\
&+ \sum_{i=1}^{N_S} m_{\mathrm{BF},i}\, T_{\mathrm{BF},i} \\
&+ m_{\mathrm{BR}}\, \theta_{\mathrm{BR}} \\
&+ \sum_{i=1}^{N_\omega} m_{\psi,i}\, \psi_i \\
&+ \sum_{i=1}^{N_S} m_{\varphi,i}\, \varphi_i
\end{aligned}
\tag{48}
$$

5.3. Uncertainty estimation

Since thermal paths are all used to estimate the different propagators, each propagator estimate is correlated to another. Furthermore, surfacic and volumetric sources are encountered along thermal paths and therefore each path does not visit only one single source (the final temperature). Altogether, the question of quantifying the uncertainty associated to $T(\vec{x},t)$ requires some attention as soon as it is computed using the data stored along a Monte Carlo ran with another set of sources.

If, in any applicative context, there was some need for estimating the uncertainty of each propagator (and not only of the resulting uncertainty on $T(\vec{x},t)$), then, from Eq. (47), it appears clearly that a Monte Carlo approach is used for estimating each propagator as an average of dedicated weights. We could therefore compute an uncertainty associated to each of these estimates by computing the standard error of the Monte Carlo weights for each propagator. This would indeed provide a faithful information about the uncertainty with which each propagator is known for a given number of sampled thermal-paths. However, this information would be insufficient to estimate the uncertainty of the

finally addressed quantity ($T(\vec{x},t)$ in Eq. 48): since the same set of thermal-paths has been used to estimate all the propagators (in one single Monte Carlo run), the propagator estimates are correlated. Estimating the uncertainty of $T(\vec{x},t)$ then requires to take into account the correlation matrix for all the propagators.

Leaving aside the idea of quantifying the uncertainty of each propagator and computing only the uncertainty on $T(\vec{x},t)$ gives us more freedom on how the stored data can be aggregated. All the required information is stored along each path (i.e. which source and how many times this source is encountered). When evaluating the temperature with the Green function, we could therefore recalculate the Monte Carlo weights and hence, recalculate at the same time the square weight to evaluate the uncertainty as in the case of a regular Monte Carlo computation (see Eq. (25)). The quantification would be made exactly as if the Monte Carlo had been re-run, with the same samples but other source values. This would include all the above mentioned correlations. However, operating this way may be cumbersome if a high amount of information has been stored (large number of sources, for instance). But then, we can simply gather

the information into squared sums and cross–sums for volumetric/surfacic sources:

$$
\begin{aligned}
ssq_{\mathrm{I},i} &= \sum_{j=1}^{N} \left(w_{\gamma_j}^{\mathrm{I},i}\right)^2 \\
ssq_{\mathrm{B},i} &= \sum_{j=1}^{N} \left(w_{\gamma_j}^{\mathrm{B},i}\right)^2 \\
ssq_{\mathrm{BF},i} &= \sum_{j=1}^{N} \left(w_{\gamma_j}^{\mathrm{BF},i}\right)^2 \\
ssq_{\mathrm{BR}} &= \sum_{j=1}^{N} \left(w_{\gamma_j}^{\mathrm{BR}}\right)^2 \\
ssq_{\psi,i} &= \sum_{j=1}^{N} \left(w_{\gamma_j}^{\psi,i}\right)^2 \\
ssq_{\varphi,i} &= \sum_{j=1}^{N} \left(w_{\gamma_j}^{\varphi,i}\right)^2 \\
sc_{\mathrm{I},i,\psi,k} &= \sum_{j=1}^{N} w_{\gamma_j}^{\mathrm{I},i} w_{\gamma_j}^{\psi,k} \\
sc_{\mathrm{I},i,\varphi,k} &= \sum_{j=1}^{N} w_{\gamma_j}^{\mathrm{I},i} w_{\gamma_j}^{\varphi,k} \\
sc_{\mathrm{B},i,\psi,k} &= \sum_{j=1}^{N} w_{\gamma_j}^{\mathrm{B},i} w_{\gamma_j}^{\psi,k} \\
sc_{\mathrm{B},i,\varphi,k} &= \sum_{j=1}^{N} w_{\gamma_j}^{\mathrm{B},i} w_{\gamma_j}^{\varphi,k} \\
sc_{\mathrm{BF},i,\psi,k} &= \sum_{j=1}^{N} w_{\gamma_j}^{\mathrm{BF},i} w_{\gamma_j}^{\psi,k} \\
sc_{\mathrm{BF},i,\varphi,k} &= \sum_{j=1}^{N} w_{\gamma_j}^{\mathrm{BF},i} w_{\gamma_j}^{\varphi,k} \\
sc_{\mathrm{BR},\psi,k} &= \sum_{j=1}^{N} w_{\gamma_j}^{\mathrm{BR}} w_{\gamma_j}^{\psi,k} \\
sc_{\mathrm{BR},\varphi,k} &= \sum_{j=1}^{N} w_{\gamma_j}^{\mathrm{BR}} w_{\gamma_j}^{\varphi,k} \\
sc_{\psi,i,\varphi,k} &= \sum_{j=1}^{N} w_{\gamma_j}^{\psi,i} w_{\gamma_j}^{\varphi,k}
\end{aligned}
\tag{49}
$$

We then compute s as:

$$
\begin{aligned}
s = \frac{1}{\sqrt{N}}\Bigg(\frac{1}{N}\Bigg(
&\sum_{i=1}^{N_\omega}\left[ssq_{\mathrm{I},i}\, T_{\mathrm{I},i}^2 + \sum_{k=1}^{N_\omega} sc_{\mathrm{I},i,\psi,k}\, T_{\mathrm{I},i}\, \psi_k + \sum_{k=1}^{N_S} sc_{\mathrm{I},i,\varphi,k}\, T_{\mathrm{I},i}\, \varphi_k\right] \\
&+\sum_{i=1}^{N_S}\left[ssq_{\mathrm{B},i}\, T_{\mathrm{B},i}^2 + \sum_{k=1}^{N_\omega} sc_{\mathrm{B},i,\psi,k}\, T_{\mathrm{B},i}\, \psi_k + \sum_{k=1}^{N_S} sc_{\mathrm{B},i,\varphi,k}\, T_{\mathrm{B},i}\, \varphi_k\right] \\
&+\sum_{i=1}^{N_S}\left[ssq_{\mathrm{BF},i}\, T_{\mathrm{BF},i}^2 + \sum_{k=1}^{N_\omega} sc_{\mathrm{BF},i,\psi,k}\, T_{\mathrm{BF},i}\, \psi_k + \sum_{k=1}^{N_S} sc_{\mathrm{BF},i,\varphi,k}\, T_{\mathrm{BF},i}\, \varphi_k\right] \\
&+ssq_{\mathrm{BR}}\, \theta_{\mathrm{BR}}^2 + \sum_{k=1}^{N_\omega} sc_{\mathrm{BR},\psi,k}\, \theta_{\mathrm{BR}}\, \psi_k + \sum_{k=1}^{N_S} sc_{\mathrm{BR},\varphi,k}\, \theta_{\mathrm{BR}}\, \varphi_k \\
&+\sum_{i=1}^{N_\omega} ssq_{\psi,i}\, \psi_i^2 \\
&+\sum_{i=1}^{N_S} ssq_{\varphi,i}\, \varphi_i^2 \\
&+\sum_{i=1}^{N_\omega}\sum_{k=1}^{N_S} sc_{\psi,i,\varphi,k}\, \psi_i\, \varphi_k\Bigg) \\
&-m^2\Bigg)^{1/2}
\end{aligned}
\tag{50}
$$

6. Implementation

The implementation of `stardis-solver`, that is presented here, is a reference implementation suitable for execution with conventional computing resources (low-end personal computer).

The source code of the solver is designed to be easy to understand and suitable for training purposes. Users can then rely on this implementation and make it evolve according to their needs.

The current implementation is a compromise between the different possibilities described in Sec. 5. This compromise consists in:

- Grouping the terms related to volume power densities and heat flux densities, restraining heat flux and power to be uniform over time and space.
- Keeping all positions and times for other sources (initial temperature, ambient radiation temperature, fluid temperature, imposed temperature), allowing these sources to vary, either over time or space or both.

Although most of the propagators that we compute in practice are integrals of the Green function over the system parts where the sources are uniform, we make use of the post-fix *green* for the corresponding parts of the code.

6.1. Code structure

Code structure is briefly presented to help the reader understand the topics related to the Green function. All the data structures and functions described thereafter in a literate programming-inspired way [17], are located in the file `sdis_green.c`. The file is structured as follows:

```
<sdis_green.c> =
  <license>
  <inclusions>
  <secondary types and functions>
  <green data structures> (1)
  <helper functions>
  <build functions> (2)
  <evaluation functions> (3)
```

The three main parts of interest and detailed below are:

(1) Data structures used to store the Green function (see Sec. 6.2).

(2) Functions used to fill up these data structures in the construction of the Green function (see Sec. 6.3).

(3) Functions using these data structures to evaluate the temperature for a given set of source values (see Sec. 6.4).

The source code for the data structures and functions described in these three sections are grouped in Appendix A at the end of the document.

6.2. Data structures

When building the Green function, the Monte-Carlo weights are not computed, but the data needed to compute them is stored, path by path, for later use. This storage requires two different types: one to store the data collected along individual Green paths, and another one to store the Green function itself, including the data of the sampled Green paths, as well as all the shared data referenced by the paths (materials, interfaces, ...):

```
<green data structures> =
  <green path data structure>
  <green function data structure>
```

Path storage. Green paths are constructed by `stardis-solver`, following the very same algorithm as when evaluating a temperature. The difference between temperature computation and the construction of the Green function is that when a path is sampled, some of the information is stored in the data structure corresponding to the Green path instead of being used on the fly to compute a temperature (see List. 2). Then, each path sampled by the solver results in a Green path data structure storing the information as follows:

```
<green path data structure> =
  struct green_path {
    <path elapsed time>
    <flux density terms collection>
    <power density terms collection>
    <end of path>
    <miscellaneous variables>
  };
```

Elapsed time is trivially a double:

```
<path elapsed time> =
  double elapsed_time;
```

Flux and power terms encountered along the path are partially merged and stored in dynamic arrays. Merging is done by material and interface: all contributions along the path are accumulated and stored as a single term associated with a given material or interface.

```
<flux density terms collection> =
  struct darray_flux_term flux_terms;
<power density terms collection> =
  struct darray_power_term power_terms;
```

As flux terms can only appear at interfaces, merged flux terms consist of an interface identifier, the involved side and the corresponding cumulated flux value. On the other hand, power terms appear in media, thus merged power terms consist of a medium identifier and the cumulated power value.

```
struct flux_term {
  double term;
  unsigned id; /* Identifier of the interface */
  enum sdis_side side;
};

struct power_term {
  double term;
  unsigned id; /* Identifier of the medium */
};
```

The end of the path can be of three types: at a boundary (`fragments`), in a volume (`vertex`), or a radiative exchange with the surrounding environment. This end of the path is represented by an union which is interpreted according to the value of the field `limit_type`, which also allows to interpret `limit_id` as being an identifier of medium (case in volume) or interface (case at boundary); note that the radiative case requires neither an union member nor a `limit_id`:

```
<end of path> =
  union {
    struct sdis_rwalk_vertex vertex;
    struct sdis_interface_fragment fragment;
  } limit;
  unsigned limit_id;
  enum sdis_green_path_end_type end_type;
```

Green function storage. The main structure is used to store everything allowing the later evaluation of a temperature estimator. This includes the description of the sampled paths as well as all the shared data referenced by the sampled paths (see List. 1).

```
<green function data structure> =
  struct sdis_green_function {
    <media collection>
    <interfaces collection>
    <paths collection>
    <miscellaneous variables>
  };
```

Collections of media and interfaces accumulate the media and interfaces that have been visited when constructing the Green function. Individual paths can then reference this shared information. These collections are hash tables, i.e. associative containers that favor fast and constant time lookup, to ensure unique storage of only the media and interfaces visited by the paths:

```
<media collection> =
  struct htable_medium media;
<interfaces collection> =
  struct htable_interf interfaces;
```

The paths collection is the set of the paths sampled for the construction of the Green function. It is a dynamic array that is well suited for iterative storage and path iteration:

```
<paths collection> =
  struct darray_green_path paths;
```

6.3. Building propagation information

Various functions are needed to fill in the Green function's data structures. They can be divided in two groups:

```
<build functions> =
  <functions building the green function>
  <functions building green paths>
```

Functions building the Green function itself do not need further description, as they are limited to collections management. Functions building green paths are divided in two groups:

```
<functions building green paths> =
  <functions that store path ending>
  <functions that accumulate data along a path>
```

6.3.1. Storing path ending

Since there are three different ways to end a path, there are three different functions that can be called to store information about the end of paths (see List. 3):

```
<functions that store path ending> =
  <store path's end at an interface>
  <store path's end in a medium>
  <store radiative path's end>
```

Let's start by describing the first function:

```
<store path's end at an interface> =
  res_T
  green_path_set_limit_interface_fragment
    (struct green_path_handle* handle,
     struct sdis_interface* interf,
     const struct sdis_interface_fragment* frag,
     const double elapsed_time)
  {
    res_T res = RES_OK;
    <check input arguments>
    <register interface 'interf' against the green function>
    <store path duration>
    <store the location at interface>
    <store identifier of interface 'interf'>
    <store the path ends up at an interface>
    return RES_OK;
  }
```

The current path ends at an interface that must be available at the time the Green function is evaluated. We start by making sure that is the case, returning an error if the process fails.

```
<register interface 'interf' against the green function> =
  res = ensure_interface_registration(handle->green, interf);
  if(res != RES_OK) return res;
```

Then the elapsed time is stored:

```
<store path duration> =
  handle->path->elapsed_time = elapsed_time;
```

Then the information related to the location at interface is stored (including position, normal, parametric coordinates and time):

```
<store the location at interface> =
  handle->path->limit.fragment = *frag;
```

Then the interface identifier is stored:

```
<store identifier of interface 'interf'> =
  handle->path->limit_id = interface_get_id(interf);
```

Finally, the type of the path ending is stored:

```
<store the path ends up at an interface> =
  handle->path->end_type = SDIS_GREEN_PATH_END_AT_INTERFACE;
```

The other two functions are built using the same pattern and are sketched thereafter:

```
  <store path's end in a medium> =
  res_T
  green_path_set_limit_interface_fragment
    (struct green_path_handle* handle,
     struct sdis_medium* mdm,
     const struct sdis_rwalk_vertex* vert,
     const double elapsed_time)
  {
    res_T res = RES_OK;
    <check input arguments>
    <register medium 'mdm' against the green function>
    <store path duration>
    <store the location in medium>
    <store identifier of medium 'mdm'>
    <store the path ends up in a medium>
    return RES_OK;
  }
```

```
  <store radiative path's end> =
  res_T
  green_path_set_limit_interface_fragment
    (struct green_path_handle* handle,
     const double elapsed_time)
  {
    res_T res = RES_OK;
    <check input arguments>
    <store path duration>
    <store the path ends up radiative>
    return RES_OK;
  }
```

6.3.2. Accumulating data along a path

The data accumulated along a path is volume power density terms and flux density terms.

```
<functions that accumulate data along a path> =
  <register a power term>
  <register a flux term>
```

It is performed through the following two functions, that share the same pattern:

```
<register a power term> =
  res_T
  green_path_add_power_term
    (struct green_path_handle* handle,
     struct sdis_medium* mdm,
     const struct sdis_rwalk_vertex* vtx,
     const double val)
  {
    <local variables>
    <check input arguments>
    <register medium 'mdm' against the green function>
    <search for a power term associated to 'mdm'>
    <if a power term exist for 'mdm'> {
      <add 'val' to this power term>
    } else {
      <register 'val' as the power term of 'mdm'>
    }
    <finalize the add_power_term function>
  }
```

```
<register the accumulated flux term> =
  res_T
  green_path_add_flux_term
    (struct green_path_handle* handle,
     struct sdis_interface* interf,
     const struct sdis_interface_fragment* frag,
     const double val)
  {
    <local variables>
    <check input arguments>
    <register the interface 'interf' against the green function>
    <search for a flux term associated to 'interf'>
    <if a flux term exist for 'interf'> {
      <add 'val' to this flux term>
    } else {
      <register 'val' as the flux term of 'interf'>
    }
    <finalize the add_flux_term function>
  }
```

6.3.3. Examples of Green paths vs. Monte Carlo paths

Two examples of paths sampled by the *Stardis* probe temperature solver are shown in Figs. 10 and 11. For each example, the archived information for the construction of the Green path is shown in Tabs. 1 and 2.

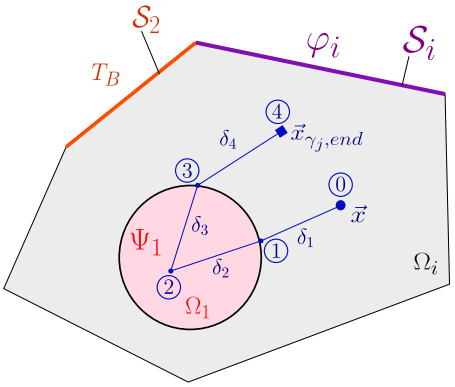

Figure 10: First path example.

Step number	Path position	$\beta_\psi(\vec{x}_{\gamma,\psi}(1))$ value	$\mu_{\gamma_j,\text{end}}$ identifier
1	$\vec{x}_{\gamma_j} \in \Omega_i$	0	-
2	$\vec{x}_{\gamma_j} \in \Omega_1$	$\frac{\delta_2^2}{6\lambda_1}$	-
3	$\vec{x}_{\gamma_j} \in \Omega_1$	$\frac{\delta_2^2}{6\lambda_1} + \frac{\delta_3^2}{6\lambda_1}$	-
4	$\vec{x}_{\gamma_j,\text{end}} \in \Omega_i$	$\frac{\delta_2^2}{6\lambda_1} + \frac{\delta_3^2}{6\lambda_1}$	0

Table 1: Successive data stored (framed in green rectangles and referred as *green path*) during Monte Carlo calculation for the first Green path.

The first path starts at the probe point $\vec{x}$. Contributions $\beta_\psi(\vec{x}_{\gamma,\psi}(1))$ related to the power density term Ψ_1 in the medium Ω_1 will accumulate when the path crosses the medium, i.e. during the second and third jump (δ_2 and δ_3). Path ends with the initial temperature condition T_I associated to the medium Ω_i (see Fig. 10).

Monte Carlo weight would be as follows:

$$w_{\gamma_j} = \psi_1 \beta_\psi(\vec{x}_{\gamma,\psi}(1)) + T_{\gamma,\text{end}}(\vec{x}_{\gamma,\text{end}}, t_{\gamma,\text{end}}, \vec{\omega}_{\gamma,\text{end}}, 0) \tag{51}$$

Data stored for the Green path (framed in green in Tab. 1) are: successive positions $\vec{x}_{\gamma_j}$, power density term contribution $\beta_\psi(\vec{x}_{\gamma,\psi}(1))$ along the path and the identifier of the boundary condition encountered $\mu_{\gamma,\text{end}} = 0$ ($T_{i,\text{I}}$: initial condition *type-0*).

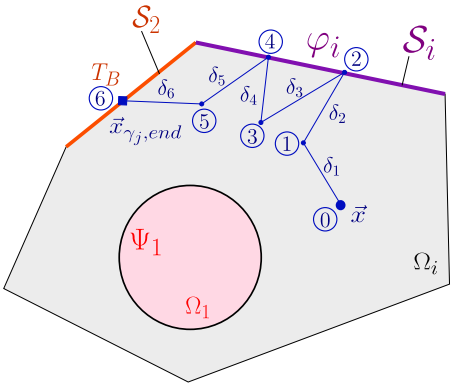

Figure 11: Second path example.

Step number	Path position	$\beta_\varphi(\vec{x}_{\gamma,\varphi}(i))$ value	$\mu_{\gamma_j,\text{end}}$ identifier
1	$\vec{x}_{\gamma_j} \in \Omega_i$	0	-
2	$\vec{x}_{\gamma_j} \in \mathcal{S}_i$	0	-
3	$\vec{x}_{\gamma_j} \in \Omega_i$	$\frac{\delta_2^2}{\lambda_i}$	-
4	$\vec{x}_{\gamma_j} \in \mathcal{S}_i$	$\frac{\delta_2^2}{\lambda_i}$	-
5	$\vec{x}_{\gamma_j} \in \Omega_i$	$\frac{\delta_2^2}{\lambda_i} + \frac{\delta_4^2}{\lambda_i}$	-
6	$\vec{x}_{\gamma_j,\text{end}} \in \mathcal{S}_2$	$\frac{\delta_2^2}{\lambda_i} + \frac{\delta_4^2}{\lambda_i}$	1

Table 2: Successive data stored (framed in green rectangles and referred as *green path*) during Monte Carlo calculation for the second Green path.

The second path starts at the probe point $\vec{x}$. Contributions $\beta_\varphi(\vec{x}_{\gamma,\varphi}(i))$ related to flux density φ_i at $\mathcal{S}_i$ interface will accumulate when the path hits this interface, i.e. on the second and fourth jump (δ_2 et δ_4). The path ends at the interface $\mathcal{S}_2$ with the temperature imposed on this interface T_B (see Fig. 11).

Monte Carlo weight would be as follows:

$$w_{\gamma_j} = \varphi_i \beta_\varphi(\vec{x}_{\gamma,\varphi}(i)) + T_{\gamma,\text{end}}(\vec{x}_{\gamma,\text{end}}, t_{\gamma,\text{end}}, \vec{\omega}_{\gamma,\text{end}}, 1) \tag{52}$$

Data stored for the Green path (framed in green in Tab. 2) are: successive positions $\vec{x}_{\gamma_j}$, density flux contribution $\beta_\varphi(\vec{x}_{\gamma,\varphi}(i))$ along the path and the identifier of the boundary condition encountered $\mu_{\gamma,\text{end}} = 1$ (T_B : *type-1* boundary condition).

6.4. Using propagation information

Once a set of sampled paths is stored in the Green function, dedicated evaluation functions are needed to apply the Green function to a set of source values (see List. 5). These functions are presented thereafter. Note that when using helper functions which are also used for standard Monte-Carlo computations (`solid_get_volumic_power` , `interface_side_get_flux`), they have to provide a vertex, even though the computation is time and space independent in the Green function context.

```
<evaluation functions> =
    res_T
    sdis_green_function_solve
      (struct sdis_green_function* green,
       struct sdis_estimator** out_estimator)
    {
      <local variables>
      <check input arguments>
      <create the estimator>

      <for each green path stored into 'green'> {
        <compute the weight of this green path>
        <accumulate the resulting weight>
      }
      <setup the estimator>
      <finalize the solve function>
    }
```

The computation of the weight associated to a path is done with a dedicated function (see List. 4):

```
<compute the weight of this green path> =
  double w; /* Monte Carlo weight to compute */
  res = green_function_solve_path(green, ipath, &w);
  <handle error code returned in 'res'>
```

The dedicated function simply takes into account the different contributions that have been stored in the current path (power, flux and end of path), and uses them to produce the Monte-Carlo weight of the path:

```
<evaluation functions> +=
  res_T
  green_function_solve_path
    (struct sdis_green_function* green,
     const size_t ipath,
     double* weight)
  {
    <local variables>
    <check input arguments>

    <compute the power collected along the path>
    <compute the flux collected along the path>
    <fetch the end of path>
    <compute the overall Monte-Carlo weight>

    <finalize the solve_path fuction>
  }
```

The volume power along the path is computed by considering each medium encountered along the path and accumulating the corresponding volume power contribution. Each volume power term `power_terms[i]` (see Eq. (46)),

previously accumulated by the function `green_path_add_power_term` is multiplied by the given volume power density value `solid_get_volumic_power(medium, &vtx)`. The total volume power contribution obtained for the path is `power`:

```
<Compute the volume power collected along the path> =
  power = 0;
  n = darray_power_term_size_get(&path->power_terms);
  power_terms = darray_power_term_cdata_get(&path->power_terms);

  FOR_EACH(i, 0, n) {
    vtx.time = INF;
    medium = green_function_fetch_medium(green, power_terms[i].id);
    power += power_terms[i].term * solid_get_volumic_power(medium, &vtx);
  }
```

The flux along the path is computed by considering each interface encountered along the path and accumulating the corresponding flux contribution.

Each flux terms `flux_terms[i]` (see Eq. (46)), previously accumulated by the function `green_path_add_flux_term` is multiplied by the given flux density value `interface_side_get_flux(interf, &frag)`. The flux term obtained for the path is `flux`:

```
<Compute the flux collected along the path> =
  flux = 0;
  n = darray_flux_term_size_get(&path->flux_terms);
  flux_terms = darray_flux_term_cdata_get(&path->flux_terms);

  FOR_EACH(i, 0, n) {
    frag.time = INF;
    frag.side = flux_terms[i].side;
    interf = green_function_fetch_interf(green, flux_terms[i].id);
    flux += flux_terms[i].term * interface_side_get_flux(interf, &frag);
  }
```

The temperature at the end of the path, depending on the type of end, is the given interface temperature value `interface_side_get_temperature(interf, &frag)`, the given temperature value of the medium `medium_get_temperature` or the given ambient radiative temperature value `sdis_scene_get_ambient_radiative_temperature(scn, &end_temperature)`:

```
<fetch the end of path> =
  switch(path->end_type) {
    case SDIS_GREEN_PATH_END_AT_INTERFACE:
      interf = green_function_fetch_interf(green, path->limit_id);
      frag = path->limit.fragment;
      end_temperature = interface_side_get_temperature(interf, &frag);
      break;
    case SDIS_GREEN_PATH_END_IN_VOLUME:
      medium = green_function_fetch_medium(green, path->limit_id);
      vtx = path->limit.vertex;
      end_temperature = medium_get_temperature(medium, &vtx);
      break;
    case SDIS_GREEN_PATH_END_RADIATIVE:
      SDIS(green_function_get_scene(green, &scn));
      SDIS(scene_get_ambient_radiative_temperature(scn, &end_temperature));
      if(end_temperature <  0) { /* Cannot be negative if used */
        res = RES_BAD_ARG;
        goto error;
      }
      break;
    default: FATAL("Unreachable code.\n"); break;
  }
```

Path weight `weight`, computed from the source values, is then simply the sum of the different contributions: `power`, `flux` and `end_temperature`.

```
<compute the overall Monte-Carlo weight> =
  *weight = power + flux + end_temperature;
```

7. Simulation examples

This section illustrates a typical use of the storage and use of the propagation information described in the previous sections. The geometrical and physical descriptions of the configurations used for this illustration, as well as the *Stardis* input files, are available in the enclosed zip file. The objective is essentially to show that a computation performed using the stored propagation information recovers the result that would be obtained with a complete MC run (with a particular attention to the associated statistical errorbars) and to illustrate the benefits in terms of computation times. As far as validation is concerned, we concentrate on the parts of the code constructing and using the propagators, not on the main code itself: *Stardis* is already validated elsewhere [18, 19]. However, we still reproduce parts of this validation by providing, together with each simulation example, a systematic comparison with the solution computed with a standard deterministic solver [20] (referred as *COMSOL Multiphysics*® hereafter and "Deterministic" in figures).

Two academic configurations are considered. They are designed as simplified versions of porous media, one with open porosities (see Fig. C.13 a)), the other with closed porosities (see Fig. C.13 c)). This benchmark has been already used by Sans et. al [21] for the purpose of validating Monte Carlo simulations of coupled diffusion-radiation heat transfer. The open-porosity configuration corresponds to a heterogeneous 3D honeycomb that can be assimilated to a porous medium with open channels, like a heat exchanger configuration. The closed-porosity configuration has 22 enclosed cavities and may be assimilated to an insulation material.

The physical assumptions are those of Sec. 3.1 with same values for λ, ρ, c and ψ throughout the whole solid phase and same values for h and ϵ along all the solid-fluid interfaces. For open porosity, there is one single imposed fluid temperature T_{BF} (*type-3* boundary condition), the same inside the channels and outside the system. For closed porosity, T_{BF} is only imposed outside the system (fluid cells temperatures are free). In both cases, the incoming

radiance temperature outside the system θ_{BR} is uniform and isotropic, the solid temperature T_{B} is imposed at the top surface (*type-1* boundary condition) and a flux density φ_B is imposed at the bottom surface (*type-2* boundary condition). The initial temperature T_{I} is uniform.

The estimated quantity is the temperature $T(\vec{x}_c, t)$ at the center $\vec{x}_c$ of the geometry for a given observation time t as a function depending on the six available sources:

- the initial temperature T_{I},
- the top boundary solid temperature T_{B},
- the ambient fluid temperature T_{BF},
- the ambient radiance temperature θ_{BR},
- the flux density at the bottom boundary φ,
- the power density throughout the solid phase ψ.

Tests are conducted first without radiation ($\epsilon = 0$, see Fig. C.14, Fig. C.15 and Fig. C.16) and then with radiation (black surfaces, $\epsilon = 1$, see Fig. C.17, Fig. C.18 and Fig. C.19). For each case, the propagation information are stored using a single Monte Carlo run. These propagation information are then used to predict $T(\vec{x}_c, t)$ (and to estimate its uncertainty) when varying the sources values with factors in the $[10^{-2}, 10^{2}]$ range around a fixed reference value for each source: $T_{\mathrm{I}}^{\mathrm{ref}}$, $T_{\mathrm{B}}^{\mathrm{ref}}$, $T_{\mathrm{BF}}^{\mathrm{ref}}$, $\theta_{\mathrm{BR}}^{\mathrm{ref}}$, φ^{ref} and ψ^{ref} (results labeled "Propagator" in the figures). Validation is achieved by comparing with standard Monte Carlo results, labeled "Monte-Carlo" in the figures. The perfect adequacy between the "Propagator" and "Monte-Carlo" results in Figs. C.14, C.15,C.16 C.17, C.18 and C.19 validates the implemented code and the quality of the stored propagation information. Fig. 12 gathers all the computation times, illustrating that the benefits of using the stored propagation information, instead of running the Monte Carlo, is a computation time reduction by a factor 10^3 to 10^4 [22].

In closer details, the following comments can be made:

- "Propagator" (using the propagator) and "Monte-Carlo" results (re-running a Monte Carlo for control) are in perfect agreement, as expected, because "Propagator" is statistically rigorously equivalent to re-running the Monte Carlo (see Fig. C.14 a) and Fig. C.15 a)). However, for the validation runs, new random numbers are used to sample the paths, whereas all "Propagator" simulations are based on the same sampled paths. Therefore, although the errorbars associated to "Propagator" can be fully trusted, the simulations made for various values of the sources are all correlated: typically, there are no statistical fluctuations in the errorbars, and when a simulation result for a given source value happens to be below the reference (within the errorbar, otherwise the validation would have failed), it remains below the reference for all other values of the source.

- "Propagator" (or "Monte-Carlo") and "Deterministic" are in perfect agreement as long as the linearization of heat transfer remains relevant. It is well-known that, exchanged radiative heat flux being proportional to T^4, radiative heat transfer causes non-linear propagation. Moreover, the higher the thermal gradients are, the higher such non-linear effects occur and the larger the bias induced by radiative transfer linearization. Here, without radiation and/or any thermal dependance of the thermal properties, "Propagator" predict the correct temperature for a very wide range of sources values (see Fig. C.14, Fig. C.15 and Fig. C.16). Close to the set of reference values for sources, temperature gradients were purposely chosen as low as suitable for the frame of linear heat transfer. Thus, "Propagator" and "Deterministic" are in good agreement (see Fig. C.17 **b**)). Outside this range of source values, "Propagator" fails to predict the correct temperature field because the linearization of radiation, at the heart of *Stardis*, becomes meaningless (see Fig. C.17 **a**)). Here, two different physical models are solved: *Stardis* linearizes radiation where *COMSOL Multiphysics*® does not. Hence, the gaps observed between the "Propagator" and the "Deterministic" method come from the capacity of a given source to increase thermal gradients, and thus to strengthen non-linearity effects (see Fig. C.17, Fig. C.18 and Fig. C.19). We are presently

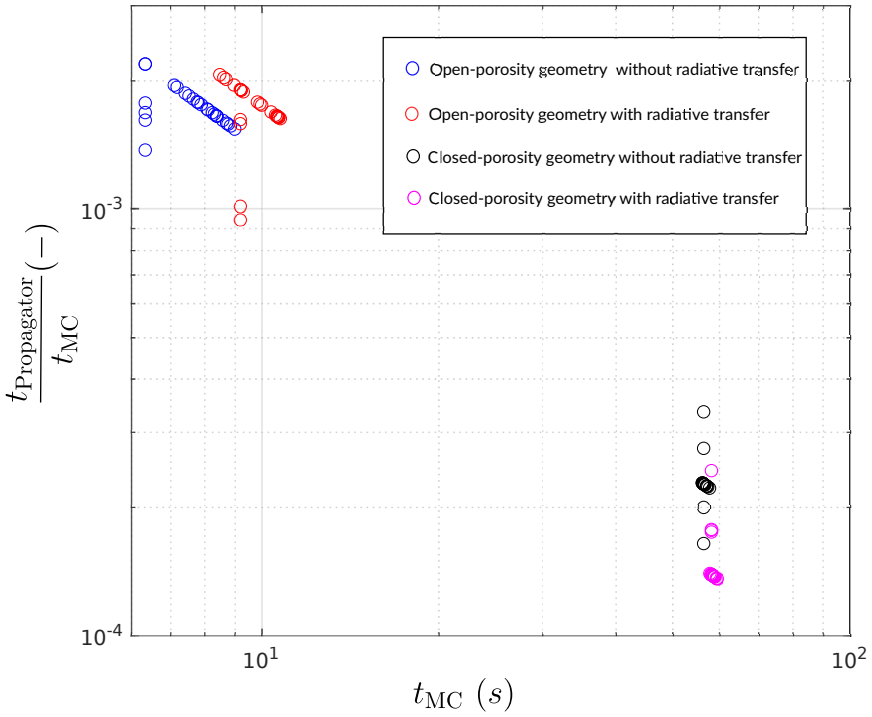

Figure 12: Ratio of computation times $\frac{t_{\mathrm{Propagator}}}{t_{\mathrm{MC}}}$ for the corresponding t_{MC}. $t_{\mathrm{Propagator}}$ is the computation time for the propagator function and t_{MC} is the computation time for the corresponding Monte Carlo computation.

working on this issue, starting from the non-linear Monte Carlo approach given by [6]). The latest available versions of *Stardis* already include a first representation of the T^4 radiation dependance, by use of branching statitsics, but the complete evaluation of this new potential is still under progress. In any case, when nonlinear effects are significant, computing the propagator is meaningless: the descriptions made in the present article are strictly rooted in linear physics and their application will always remain strictly restricted to linear heat transfer.

- There is a notable distinction between open-porosity and closed-porosity as far as computation time is concerned. In the open-porosity case, a thermal path starting at the center of the system can encounter the flow quite rapidly and then the path is stopped because the fluid temperature is known. In the closed-porosity case, a thermal path starting at the center of the system can also encounter the flow inside a fluid cell quite rapidly, but then the flow temperature is unknown and the path is continued until a source is encountered (either the initial condition at any location, or a known temperature at the boundary). These paths are significantly longer and so is the computation time required for their construction. It can even be extremely long if the number of closed cells is increased: Monte Carlo may encounter difficulties when addressing insulating materials with large numbers of closed cells along all directions. However, this difficulty is not reported in "Propagator": even if the computation cost associated to path-sampling in the reference Monte Carlo run is higher in the closed-porosity case, the propagation information stored are similar in the open-porosity and the closed-porosity cases and the computation times associated to the use, by "Propagator", of these propagation information are the same in both cases. The computation benefit of using "Propagator" instead of running a full Monte Carlo is therefore stronger for closed porosities.

8. Going further: Propagation information storage in case of non-uniform and time-dependent sources

When constructing estimates for the propagators, we have split the initial Monte Carlo weight and gathered all the propagation information corresponding to parts of the system where the sources were uniform and constant (see Sec. 5). As was already mentioned, *Stardis* makes uniformity and constancy assumptions for each geometrical part but '`stardis-solver`' does not. And indeed, in the Monte Carlo weight expression of Eq. (41), nothing prevents the initial temperatures, the imposed temperatures at the boundary, the incoming radiance temperature, the surface flux densities imposed at the boundary and the volume power densities imposed inside the solids to be non-uniform and non-constant fields: $\psi\left(\vec{x}_{\gamma,\psi}(k), t_{\gamma,\psi}(k)\right)$, $\varphi\left(\vec{x}_{\gamma,\varphi}(k), t_{\gamma,\varphi}(k)\right)$ and $T_{\gamma,\mathrm{end}}(\vec{x}_{\gamma,\mathrm{end}}, t_{\gamma,\mathrm{end}}, \vec{\omega}_{\gamma,\mathrm{end}}, \mu_{\gamma,\mathrm{end}})$ can hold this information as it is provided, whatever the geometric structure. This means that even running *Stardis* with uniform and constant sources in each part, if we store the locations, times and directions used when constructing Monte Carlo weights, then this information can be later used to virtually re-run the Monte Carlo simulation with non-uniform time-dependent sources.

In algorithmic terms:

- N thermal paths are sampled exactly as in the Monte Carlo algorithm of Sec. 4.1;
- Along each path γ_j, we store the $N_{\varphi,j}$ locations $\vec{x}_{\gamma_j,\varphi}(k)$ and times $t_{\gamma_j,\varphi}(k)$ at which surface flux densities were accessed;
- Similarly we store the $N_{\psi,j}$ locations $\vec{x}_{\gamma_j,\psi}(k)$ and times $t_{\gamma_j,\psi}(k)$ at which volume power densities were accessed;
- The information concerning the end of the path are also stored: $\vec{x}_{\gamma_j,\mathrm{end}}$, $t_{\gamma_j,\mathrm{end}}$, $\vec{\omega}_{\gamma_j,\mathrm{end}}$ and $\mu_{\gamma_j,\mathrm{end}}$;
- When a Monte Carlo estimate is required for any set of non-uniform time-dependent source fields $\tilde{T}_\mathrm{I}$, $\tilde{T}_\mathrm{B}$, $\tilde{T}_\mathrm{BF}$, $\tilde{\theta}_\mathrm{BR}$, $\tilde{\psi}$ and $\tilde{\varphi}$, it is constructed the following way:

$$\tilde{T}(\vec{x},t) \text{ or } \tilde{T}(t) \text{ or } \tilde{\theta}_R(\vec{\omega},\vec{x},t) \approx \tilde{m} = \frac{1}{N}\sum_{j=1}^{N} \tilde{w}_j \tag{53}$$

with

$$\begin{aligned}
\tilde{w}_j = \; & \mathcal{H}(\mu_{\gamma_j,\mathrm{end}} = 0)\; \tilde{T}_\mathrm{I}(\vec{x}_{\gamma_j,\mathrm{end}}) \\
& + \mathcal{H}(\mu_{\gamma_j,\mathrm{end}} = 1)\; \tilde{T}_\mathrm{B}(\vec{x}_{\gamma_j,\mathrm{end}}, t_{\gamma_j,\mathrm{end}}) \\
& + \mathcal{H}(\mu_{\gamma_j,\mathrm{end}} = 2)\; \tilde{T}_\mathrm{BF}(\vec{x}_{\gamma_j,\mathrm{end}}, t_{\gamma_j,\mathrm{end}}) \\
& + \mathcal{H}(\mu_{\gamma_j,\mathrm{end}} = 3)\; \tilde{\theta}_\mathrm{BR}(\vec{x}_{\gamma_j,\mathrm{end}}, t_{\gamma_j,\mathrm{end}}, \vec{\omega}_{\gamma_j,\mathrm{end}}) \\
& + \sum_{k=1}^{N_{\psi,j}} \beta_\psi(\vec{x}_{\gamma_j,\psi}(k))\; \tilde{\psi}\left(\vec{x}_{\gamma_j,\psi}(k), t_{\gamma_j,\psi}(k)\right) \\
& + \sum_{k=1}^{N_{\varphi,j}} \beta_\varphi(\vec{x}_{\gamma_j,\varphi}(k))\; \tilde{\varphi}\left(\vec{x}_{\gamma_j,\varphi}(k), t_{\gamma_j,\varphi}(k)\right)
\end{aligned} \tag{54}$$

For this last strategy, dealing with uncertainties is straightforward. The estimate $\tilde{m}$ in Eq. (53) is finally built exactly as if the Monte Carlo was re-run using new sources. The uncertainty $\tilde{s}$ associated to $\tilde{m}$ is therefore the same as for any Monte Carlo simulation (using an estimate of the standard deviation of the Monte Carlo weights) as detailed in Eq. (25).

In terms of code changes, storing unmerged flux and power terms allows code simplification, as a significant part of the merged-storage version of the code is about retrieving the term to which the current contribution needs to be merged. On the other hand, allowing all types of non-uniform and time-dependent sources is still at the cost of storing more propagation since information terms data structures have to store the term's location (time and space). The data structures and functions that implement the unmerged storage of sources' contributions are available in Appendix B.

9. Conclusion

One of the strengths of Monte Carlo approaches is the ease with which information can be stored, during one run, and then used offline to learn about the physics, preserving all geometric features. The simplest example is the storing of the paths themselves (or of a large enough fraction of the paths). For coupled heat transfer, displaying a selection of the paths and analyzing how they visit the system, both in time and space, switching from one heat transfer mode to the other, is indeed a very practical way to learn how the sources are viewed from one location, how their impact is delayed by the inertial parts of the system, and therefore how a design can be adjusted to achieve a given objective. In the present paper, we have left aside these details about the paths themselves. We focused on the act of quantifying the propagation and not on the analysis of the physical phenomena and the coupling processes responsible for the propagation. But these two practices, computing the propagators and visualizing the propagation processes, are worth being considered sideways in all engineering contexts requiring a close understanding of heat transfer physics at the system scale. Therefore, in addition to the functionalities of *Stardis* described in Sec. 6, a set of post-treatment tools have also been designed to help visualizing and analyzing thermal paths throughout large scale geometries [19].

For automated engineering practices, e.g. inversion, optimization or command algorithms, analyzing the paths is useless; all is needed is the addressed quantity as a function of the sources: the tools described in the present paper are therefore self-sufficient. However there are numerous questions of direct interest to thermal engineers that cannot be addressed this way. These are all the dependencies on parameters that cannot be considered as sources (in the general sense provided by Green's theory). Typically the dependence on emissivities, convective exchange coefficients, conductivities or capacities rises more complex questions. If only sensitivities were required, i.e. the derivative with respect to each parameter, then the general theory of sensitivity evaluation in Monte Carlo algorithms could be used [23], but we would not build the complete dependence (the function) as we did here with the sources. Addressing the complete non-linear dependence on other parameters than sources is not theoretically unfeasible: it has notably been achieved in the field of radiative transfer under the literature name of "Symbolic Monte Carlo" [24, 25, 26, 27] and we have started to work on extending these symbolic techniques to coupled heat transfer, with the objective of implementing them inside `stardis-solver` [28, 29].

By far more difficult would be the question of addressing the dependence on geometrical parameters. Here also, some attempts have already been made in the field of radiative transfer, but to the best of our knowledge and although large impacts could be expected in terms of applications, there is no report available of any attempt to go beyond the computation of derivatives (geometrical sensitivities). Constructing a thermal heat transfer observable as a symbolic function of a geometric parameter is a question that has not yet been addressed.

Another difficult point associated to strong applicative concerns is the withdraw of the linearization of radiative transfer. This linearization is at the heart of present Monte Carlo approaches to coupled heat transfer. There are convincing perspectives as far as handling non-linearities in the Monte Carlo framework is concerned [6], and some of the corresponding propositions could be used to avoid the linearization of radiation, but then the overall coupled physical problem would be non-linear and the concept of propagation could not be used anymore. All our present proposition would then have to be revisited.

Finally, we have highlighted the fact that storing propagators allows the design of fast external solvers: adressing the same observable for other sources without reruning the Monte Carlo. But another quite significant usage is uncertainty propagation. As the model is linear, any statistical distribution of source-uncertainties is propagated without transformation and the propagator tells us how (exactly the same way it computes the observabale when changing the sources). Typically, if a given source has a normal distribution, the technique described in the present paper will provide the center of the resulting normal distribution of the observable and computing its standard deviation will require nothing more than the square of the propagator. If the Monte Carlo run is very accurate, there will be no need for any consideration of the Monte Carlo uncertainties when studying such source-uncertainty propagations. Of course, if the source uncertainty and the Monte Carlo uncertainty are of the same order of magnitude, then further attention will have to be devoted to the resulting statistical correlations, typically when several sources are uncertain, possibly correlated, their associated propagators being themselves correlated because they were computed using the

same Monte Carlo run. But again, all the required information is in the data already stored at the Monte Carlo weight level when running *Stardis* and only a post treatment is required when such further statistics are required.

Acknowledgements

This project has received funding from the "Investissement d'avenir" program of the National Agency for Research of the French state under award number "ANR-10-LBX-22-01-SOLSTICE" and was supported by the ANR HIGH-TUNE, grant ANR-16-CE01-0010 and ANR MCG-RAD, grant ANR-18-CE46-0012 and from the Occitanie Region (Projet CLE-2016 ED-Star).

References

[1] S. Heinrich, Monte carlo complexity of global solution of integral equations, Journal of Complexity 14 (2) (1998) 151–175. doi:https://doi.org/10.1006/jcom.1998.0471.
URL https://www.sciencedirect.com/science/article/pii/S0885064X9890471X

[2] R. Farnoosh, M. Ebrahimi, Monte carlo method for solving fredholm integral equations of the second kind, Applied Mathematics and Computation 195 (1) (2008) 309–315. doi:https://doi.org/10.1016/j.amc.2007.04.097.
URL https://www.sciencedirect.com/science/article/pii/S0096300307005541

[3] N. Villefranque, R. Fournier, F. Couvreux, S. Blanco, C. Cornet, V. Eymet, V. Forest, J.-M. Tregan, A path-tracing monte carlo library for 3-d radiative transfer in highly resolved cloudy atmospheres, Journal of Advances in Modeling Earth Systems 11 (8) (2019) 2449–2473. doi:https://doi.org/10.1029/2018MS001602.
URL https://arxiv.org/abs/1902.01137

[4] N. Villefranque, F. Hourdin, L. d'Alençon, S. Blanco, O. Boucher, C. Caliot, C. Coustet, J. Dauchet, M. E. Hafi, V. Eymet, O. Farges, V. Forest, R. Fournier, J. Gautrais, V. Masson, B. Piaud, R. Schoetter, The “teapot in a city”: A paradigm shift in urban climate modeling, Science Advances 8 (27) (2022) eabp8934. arXiv:https://www.science.org/doi/pdf/10.1126/sciadv.abp8934, doi:10.1126/sciadv.abp8934.
URL https://www.science.org/doi/abs/10.1126/sciadv.abp8934

[5] L. Ibarrart, S. Blanco, C. Caliot, J. Dauchet, S. Eibner, M. El-Hafi, O. Farges, V. Forest, R. Fournier, J. Gautrais, R. Konduru, L. Penazzi, J.-M. Tregan, T. Vourc'h, D. Yaacoub, Advection, diffusion and linear transport in a single path-sampling Monte-Carlo algorithm : getting insensitive to geometrical refinement, working paper or preprint (Oct. 2022).
URL https://hal.archives-ouvertes.fr/hal-03818899

[6] J. Dauchet, J.-J. Bezian, S. Blanco, C. Caliot, J. Charon, C. Coustet, M. El Hafi, V. Eymet, O. Farges, V. Forest, R. Fournier, M. Galtier, J. Gautrais, A. Khuong, L. Pelissier, B. Piaud, M. Roger, G. Terrée, S. Weitz, Addressing nonlinearities in Monte Carlo, Scientific Reports 8 (1) (Dec. 2018). doi:10.1038/s41598-018-31574-4.

[7] C. Supplis, J. Dauchet, V. Gattepaille, F. Gros, T. Vourc'h, J.-F. Cornet, Radiative analysis of luminescence in photoreactive systems: Application to photosensitizers for solar fuel production, PLOS ONE 16 (7) (2021) 1–38. doi:10.1371/journal.pone.0255002.
URL https://doi.org/10.1371/journal.pone.0255002

[8] M. E. Muller, Some Continuous Monte Carlo Methods for the Dirichlet Problem, The Annals of Mathematical Statistics 27 (3) (1956) 569 – 589. doi:10.1214/aoms/1177728169.
URL https://doi.org/10.1214/aoms/1177728169

[9] M. Mascagni, C.-O. Hwang, ϵ-shell error analysis for "walk on spheres" algorithms, Mathematics and Computers in Simulation 63 (2) (2003) 93–104. doi:https://doi.org/10.1016/S0378-4754(03)00038-7.
URL https://www.sciencedirect.com/science/article/pii/S0378475403000387

[10] C.-O. Hwang, S. Hong, J. Kim, Off-centered "walk-on-spheres" (wos) algorithm, Journal of Computational Physics 303 (2015) 331–335. doi:https://doi.org/10.1016/j.jcp.2015.10.002.
URL https://www.sciencedirect.com/science/article/pii/S0021999115006646

[11] M. Galtier, S. Blanco, C. Caliot, C. Coustet, J. Dauchet, M. El-Hafi, V. Eymet, R. A. Fournier, J. Gautrais, A. Khuong, B. Piaud, G. Terrée, Integral formulation of null-collision Monte Carlo algorithms, Journal of Quantitative Spectroscopy and Radiative Transfer 125 (2013) 57–68. doi:10.1016/j.jqsrt.2013.04.001.
URL https://hal.archives-ouvertes.fr/hal-01688110

[12] M. Kac, On distributions of certain wiener functionals, Transactions of the American Mathematical Society 65 (1949) 1–13.

[13] EDSTAR, Thermal simulator training tool (2019).
URL https://www.starwest.ups-tlse.fr/fr/training/tool/therm/

[14] EDSTAR, Introduction scenario for the thermal simulater (2019).
URL https://www.starwest.ups-tlse.fr/fr/training/scenario/tool/therm/

[15] J.-M. Tregan, J.-L. Amestoy, M. Bati, J.-J. Bézian, S. Blanco, L. Brunel, C. Caliot, J. Charon, J.-F. Cornet, C. Coustet, L. D'alençon, J. Dauchet, S. Dutour, S. Eibner, M. El-Hafi, V. Eymet, O. Farges, V. Forest, R. Fournier, M. Galtier, V. Gattepaille, J. Gautrais, Z. He, F. Hourdin, L. Ibarrart, J.-L. Joly, P. Lapeyre, P. Lavieille, M.-H. Lecureux, J. Lluc, M. Miscevic, N. C. Mourtaday, Y. NYFFENEGGER-PERE, L. P. Pelissier, L. Penazzi, B. Piaud, C. Rodrigues-Viguier, G. Roques, M. Roger, T. Saez, G. Terrée, N. Villefranque, T. Vourc'h, D. Yaacoub, Coupling radiative, conductive and convective heat-transfers in a single Monte Carlo algorithm: a general theoretical framework

for linear situations, working paper or preprint (Oct. 2022).
URL https://hal.archives-ouvertes.fr/hal-03819157

[16] B. Lapeyre, E. Pardoux, R. Sentis, Méthodes de Monte-Carlo pour les équations de transport et de diffusion, Mathématiques et Applications, Springer Berlin Heidelberg, 1997.
URL https://books.google.fr/books?id=qgIqQgAACAAJ

[17] D. Knuth, Literate Programming, The Computer Journal 27 (1984) 97–111.

[18] V. Eymet, F. Vincent, B. Piaud, C. Coustet, R. Fournier, S. Blanco, L. Ibarrart, J.-M. Tregan, P. Lavieille, C. Caliot, M. El-Hafi, J.-J. Bézian, R. Bouchie, M. Galtier, M. Roger, J. Dauchet, O. Farges, C. Péniguel, I. Rupp, G. Eymet, Synthèse d'images infrarouges sans calcul préalable du champ de température, in: SFT 2019 - 27ème Congrès Français de Thermique, Nantes, France, 2019, pp. 153–160.
URL https://hal.archives-ouvertes.fr/hal-02419604

[19] Méso-Star, Stardis (2021).
URL https://www.meso-star.com/projects/stardis/starter-pack.html

[20] Comsol multiphysics®, v. 5.6. (2020).
URL http://www.comsol.com

[21] M. Sans, O. Farges, V. Schick, C. Moyne, G. Parent, Modeling the Flash Method by using a Conducto-Radiative Monte-Carlo Method: Application to Porous Media, in: Proceeding of Proceedings of the 9th International Symposium on Radiative Transfer, RAD-19, Begellhouse, Athens, Greece, 2019, pp. 319–326. doi:10.1615/RAD-19.390.

[22] L. Penazzi, S. Blanco, C. Caliot, C. Coustet, M. El-Hafi, R. Fournier, J. Gautrais, M. Sans, Transfer function estimation with SMC method for combined heat transfer: insensitivity to detail refinement of complex geometries, in: CHT-21 ICHMT - International Symposium on Advances in Computational Heat Transfer, Rio de Janeiro (online), Brazil, 2021, pp. 383–386.
URL https://hal-mines-albi.archives-ouvertes.fr/hal-03374353

[23] P. Lapeyre, S. Blanco, C. Caliot, J. Dauchet, M. El Hafi, R. Fournier, O. Farges, M. Roger, Monte-Carlo and domain-deformation sensitivities, in: Proceeding of Proceedings of the 9th International Symposium on Radiative Transfer, RAD-19, Begellhouse, Athens, Greece, 2019, pp. 213–220. doi:10.1615/RAD-19.260.

[24] W. L. Dunn, Inverse Monte Carlo analysis, Journal of Computational Physics 41 (1) (1981) 154–166.

[25] W. L. Dunn, J. K. Shultis, Monte Carlo methods for design and analysis of radiation detectors, Radiation Physics and Chemistry 78 (10) (2009) 852–858. doi:10.1016/j.radphyschem.2009.04.030.

[26] M. Galtier, M. Roger, F. André, A. Delmas, A symbolic approach for the identification of radiative properties, Journal of Quantitative Spectroscopy and Radiative Transfer 196 (2017) 130–141. doi:10.1016/j.jqsrt.2017.03.026.

[27] Y. Maanane, M. Roger, A. Delmas, M. Galtier, F. André, Symbolic Monte Carlo method applied to the identification of radiative properties of a heterogeneous material, Journal of Quantitative Spectroscopy and Radiative Transfer 249 (2020) 107019. doi:10.1016/j.jqsrt.2020.107019.

[28] L. Penazzi, S. Blanco, C. Caliot, C. Coustet, M. El-Hafi, R. A. Fournier, M. Galtier, L. Ibarrart, M. Roger, Toward the use of Symbolic Monte Carlo for Conduction-Radiation Coupling in Complex Geometries, in: RAD-19 - 9th International Symposium on Radiative Transfer, Begellhouse, Athens, Greece, 2019, p. 8 p. doi:10.1615/RAD-19.380.
URL https://hal-mines-albi.archives-ouvertes.fr/hal-02265075

[29] M. Sans, S. Blanco, C. Caliot, M. El-Hafi, O. Farges, R. A. Fournier, L. Penazzi, Méthode de Monte-Carlo Symbolique pour la caractérisation des propriétés thermiques : application à la méthode flash, in: SFT 2021 - 29 ème congrès Franças de Thermique, Belfort (online), France, 2021, pp. 293–300. doi:10.25855/SFT2021-076.
URL https://hal-mines-albi.archives-ouvertes.fr/hal-03260534

Appendix A. Source code chunks implementing merged storage

Listing 1: struct sdis_green_function and related types

```
struct sdis_green_function {
  struct htable_medium media;
  struct htable_interf interfaces;
  struct darray_green_path paths; /* List of paths used to estimate the green */

  size_t npaths_valid;
  size_t npaths_invalid;

  struct accum realisation_time; /* Time per realisation */

  struct ssp_rng_type rng_type;
  FILE* rng_state;

  ref_T ref;
  struct sdis_scene* scn;
};
```

Listing 2: struct green_path

```
struct green_path {
  double elapsed_time;
  struct darray_flux_term flux_terms; /* List of flux terms */
  struct darray_power_term power_terms; /* List of volumic power terms */
  union {
    struct sdis_rwalk_vertex vertex;
    struct sdis_interface_fragment fragment;
  } limit;
  unsigned limit_id; /* Identifier of the limit medium/interface */
  enum sdis_green_path_end_type end_type;

  /* Indices of the last accessed medium/interface. Used to speed up the access
   * to the medium/interface. */
  uint16_t ilast_medium;
  uint16_t ilast_interf;
};

struct power_term {
double term;
unsigned id; /* Identifier of the medium */
};

struct flux_term {
double term;
unsigned id; /* Identifier of the interface */
enum sdis_side side;
};
```

Listing 3: green functions to store path data

```
res_T
green_path_set_limit_interface_fragment
  (struct green_path_handle* handle,
   struct sdis_interface* interf,
   const struct sdis_interface_fragment* frag,
   const double elapsed_time)
{
  res_T res = RES_OK;
  ASSERT(handle && interf && frag);
  ASSERT(handle->path->end_type == SDIS_GREEN_PATH_END_TYPES_COUNT__);
  res = ensure_interface_registration(handle->green, interf);
  if(res != RES_OK) return res;
```

```
824     handle->path->elapsed_time = elapsed_time;
825     handle->path->limit.fragment = *frag;
826     handle->path->limit_id = interface_get_id(interf);
827     handle->path->end_type = SDIS_GREEN_PATH_END_AT_INTERFACE;
828     return RES_OK;
829   }
830
831   res_T
832   green_path_set_limit_vertex
833     (struct green_path_handle* handle,
834      struct sdis_medium* mdm,
835      const struct sdis_rwalk_vertex* vert,
836      const double elapsed_time)
837   {
838     res_T res = RES_OK;
839     ASSERT(handle && mdm && vert);
840     ASSERT(handle->path->end_type == SDIS_GREEN_PATH_END_TYPES_COUNT__);
841     res = ensure_medium_registration(handle->green, mdm);
842     if(res != RES_OK) return res;
843     handle->path->elapsed_time = elapsed_time;
844     handle->path->limit.vertex = *vert;
845     handle->path->limit_id = medium_get_id(mdm);
846     handle->path->end_type = SDIS_GREEN_PATH_END_IN_VOLUME;
847     return RES_OK;
848   }
849
850   res_T
851   green_path_set_limit_radiative
852     (struct green_path_handle* handle,
853      const double elapsed_time)
854   {
855     ASSERT(handle);
856     ASSERT(handle->path->end_type == SDIS_GREEN_PATH_END_TYPES_COUNT__);
857     handle->path->elapsed_time = elapsed_time;
858     handle->path->end_type = SDIS_GREEN_PATH_END_RADIATIVE;
859     return RES_OK;
860   }
861
862   res_T
863   green_path_add_power_term
864     (struct green_path_handle* handle,
865      struct sdis_medium* mdm,
866      const struct sdis_rwalk_vertex* vtx,
867      const double val)
868   {
869     struct green_path* path;
870     struct power_term* terms;
871     size_t nterms;
872     size_t iterm;
873     unsigned id;
874     res_T res = RES_OK;
875     ASSERT(handle && mdm && vtx);
876
877     /* Unused position and time: the current implementation of the green function
878      * assumes that the power is constant in space and time per medium. */
879     (void)vtx;
880
881     res = ensure_medium_registration(handle->green, mdm);
882     if(res != RES_OK) goto error;
883
884     path = handle->path;
885     terms = darray_power_term_data_get(&path->power_terms);
886     nterms = darray_power_term_size_get(&path->power_terms);
887     id = medium_get_id(mdm);
888     iterm = SIZE_MAX;
```

```
889
890     /* Early find */
891     if(path->ilast_medium < nterms && terms[path->ilast_medium].id == id) {
892       iterm = path->ilast_medium;
893     } else {
894       /* Linear search of the submitted medium */
895       FOR_EACH(iterm, 0, nterms) if(terms[iterm].id == id) break;
896     }
897
898     /* Add the power term to the path wrt the submitted medium */
899     if(iterm < nterms) {
900       terms[iterm].term += val;
901     } else {
902       struct power_term term = POWER_TERM_NULL__;
903       term.term = val;
904       term.id = id;
905       res = darray_power_term_push_back(&handle->path->power_terms, &term);
906       if(res != RES_OK) goto error;
907     }
908
909     /* Register the slot into which the last accessed medium lies */
910     CHK(iterm < UINT16_MAX);
911     path->ilast_medium = (uint16_t)iterm;
912
913   exit:
914     return res;
915   error:
916     goto exit;
917   }
918
919   res_T
920   green_path_add_flux_term
921     (struct green_path_handle* handle,
922      struct sdis_interface* interf,
923      const struct sdis_interface_fragment* frag,
924      const double val)
925   {
926     struct green_path* path;
927     struct flux_term* terms;
928     size_t nterms;
929     size_t iterm;
930     unsigned id;
931     res_T res = RES_OK;
932     ASSERT(handle && interf && frag && val >= 0);
933
934     res = ensure_interface_registration(handle->green, interf);
935     if(res != RES_OK) goto error;
936
937     path = handle->path;
938     terms = darray_flux_term_data_get(&path->flux_terms);
939     nterms = darray_flux_term_size_get(&path->flux_terms);
940     id = interface_get_id(interf);
941     iterm = SIZE_MAX;
942
943     /* Early find */
944     if(path->ilast_interf < nterms
945     && terms[path->ilast_interf].id == id
946     && terms[path->ilast_interf].side == frag->side) {
947       iterm = path->ilast_interf;
948     } else {
949       /* Linear search of the submitted interface */
950       FOR_EACH(iterm, 0, nterms) {
951         if(terms[iterm].id == id && terms[iterm].side == frag->side) {
952           break;
953         }
```

```
954       }
955     }
956
957     /* Add the flux term to the path wrt the submitted interface */
958     if(iterm < nterms) {
959       terms[iterm].term += val;
960     } else {
961       struct flux_term term = FLUX_TERM_NULL__;
962       term.term = val;
963       term.id = id;
964       term.side = frag->side;
965       res = darray_flux_term_push_back(&handle->path->flux_terms, &term);
966       if(res != RES_OK) goto error;
967     }
968
969     /* Register the slot into which the last accessed interface lies */
970     CHK(iterm < UINT16_MAX);
971     path->ilast_interf = (uint16_t)iterm;
972
973   exit:
974     return res;
975   error:
976     goto exit;
977   }
```

Listing 4: green_function_solve_path

```
978   static res_T
979   green_function_solve_path
980     (struct sdis_green_function* green,
981      const size_t ipath,
982      double* weight)
983   {
984     const struct power_term* power_terms = NULL;
985     const struct flux_term* flux_terms = NULL;
986     const struct green_path* path = NULL;
987     const struct sdis_medium* medium = NULL;
988     const struct sdis_interface* interf = NULL;
989     struct sdis_scene* scn = NULL;
990     struct sdis_rwalk_vertex vtx = SDIS_RWALK_VERTEX_NULL;
991     struct sdis_interface_fragment frag = SDIS_INTERFACE_FRAGMENT_NULL;
992     double power;
993     double flux;
994     double end_temperature;
995     size_t i, n;
996     res_T res = RES_OK;
997     ASSERT(green && ipath < darray_green_path_size_get(&green->paths) && weight);
998
999     path = darray_green_path_cdata_get(&green->paths) + ipath;
1000    if(path->end_type == SDIS_GREEN_PATH_END_ERROR) { /* Rejected path */
1001      res = RES_BAD_OP;
1002      goto error;
1003    }
1004
1005    /* Compute medium power terms */
1006    power = 0;
1007    n = darray_power_term_size_get(&path->power_terms);
1008    power_terms = darray_power_term_cdata_get(&path->power_terms);
1009    FOR_EACH(i, 0, n) {
1010      vtx.time = INF;
1011      medium = green_function_fetch_medium(green, power_terms[i].id);
1012      power += power_terms[i].term * solid_get_volumic_power(medium, &vtx);
1013    }
1014
1015    /* Compute interface fluxes */
```

```
1016    flux = 0;
1017    n = darray_flux_term_size_get(&path->flux_terms);
1018    flux_terms = darray_flux_term_cdata_get(&path->flux_terms);
1019    FOR_EACH(i, 0, n) {
1020      frag.time = INF;
1021      frag.side = flux_terms[i].side;
1022      interf = green_function_fetch_interf(green, flux_terms[i].id);
1023      flux += flux_terms[i].term * interface_side_get_flux(interf, &frag);
1024    }
1025
1026    /* Compute path's end temperature */
1027    switch(path->end_type) {
1028      case SDIS_GREEN_PATH_END_AT_INTERFACE:
1029        interf = green_function_fetch_interf(green, path->limit_id);
1030        frag = path->limit.fragment;
1031        end_temperature = interface_side_get_temperature(interf, &frag);
1032        break;
1033      case SDIS_GREEN_PATH_END_IN_VOLUME:
1034        medium = green_function_fetch_medium(green, path->limit_id);
1035        vtx = path->limit.vertex;
1036        end_temperature = medium_get_temperature(medium, &vtx);
1037        break;
1038      case SDIS_GREEN_PATH_END_RADIATIVE:
1039        SDIS(green_function_get_scene(green, &scn));
1040        SDIS(scene_get_ambient_radiative_temperature(scn, &end_temperature));
1041        if(end_temperature < 0) { /* Cannot have it negative if used */
1042          res = RES_BAD_ARG;
1043          goto error;
1044        }
1045        break;
1046      default: FATAL("Unreachable code.\n"); break;
1047    }
1048
1049    /* Compute the path weight */
1050    *weight = power + flux + end_temperature;
1051
1052  exit:
1053    return res;
1054  error:
1055    goto exit;
1056  }
```

Listing 5: green_function_solve

```
1057  res_T
1058  sdis_green_function_solve
1059    (struct sdis_green_function* green,
1060     struct sdis_estimator** out_estimator)
1061  {
1062    struct sdis_estimator* estimator = NULL;
1063    size_t npaths;
1064    size_t ipath;
1065    size_t N = 0; /* #realisations */
1066    double accum = 0;
1067    double accum2 = 0;
1068    res_T res = RES_OK;
1069
1070    if(!green || !out_estimator) {
1071      res = RES_BAD_ARG;
1072      goto error;
1073    }
1074
1075    npaths = darray_green_path_size_get(&green->paths);
1076
1077    /* Create the estimator */
```

```
1078    res = estimator_create(green->scn->dev, SDIS_ESTIMATOR_TEMPERATURE, &estimator);
1079    if(res != RES_OK) goto error;
1080
1081    /* Solve the green function */
1082    FOR_EACH(ipath, 0, npaths) {
1083      double w;
1084
1085      res = green_function_solve_path(green, ipath, &w);
1086      if(res == RES_BAD_OP) continue;
1087      if(res != RES_OK) goto error;
1088
1089      accum += w;
1090      accum2 += w*w;
1091      ++N;
1092    }
1093
1094    /* Setup the estimated temperature */
1095    estimator_setup_realisations_count(estimator, npaths, N);
1096    estimator_setup_temperature(estimator, accum, accum2);
1097    estimator_setup_realisation_time
1098      (estimator, green->realisation_time.sum, green->realisation_time.sum2);
1099
1100  exit:
1101    if(out_estimator) *out_estimator = estimator;
1102    return res;
1103  error:
1104    if(estimator) {
1105      SDIS(estimator_ref_put(estimator));
1106      estimator = NULL;
1107    }
1108    goto exit;
1109  }
```

Appendix B. Source code chunks implementing unmerged storage

Listing 6: unmerged terms structs

```
1111  struct unmerged_power_term {
1112    double term;
1113    unsigned id; /* Identifier of the medium */
1114    struct sdis_rwalk_vertex vertex; /* location of the term */
1115  };
1116
1117  struct unmerged_flux_term {
1118    double term;
1119    unsigned id; /* Identifier of the interface */
1120    struct sdis_interface_fragment fragment; /* location of the term */
1121  };
```

Listing 7: green functions to store unmerged path data

```
1122  res_T
1123  green_path_add_power_term
1124    (struct green_path_handle* handle,
1125     struct sdis_medium* mdm,
1126     const struct sdis_rwalk_vertex* vtx,
1127     const double val)
1128  {
1129    struct green_path* path;
1130    struct unmerged_power_term* terms;
1131    struct power_term term = POWER_TERM_NULL__;
1132    size_t nterms;
1133    unsigned id;
1134    res_T res = RES_OK;
1135    ASSERT(handle && mdm && vtx);
```

```
1136
1137    res = ensure_medium_registration(handle->green, mdm);
1138    if(res != RES_OK) goto error;
1139
1140    path = handle->path;
1141    terms = darray_power_term_data_get(&path->power_terms);
1142    nterms = darray_power_term_size_get(&path->power_terms);
1143    id = medium_get_id(mdm);
1144
1145    /* store term */
1146    term.term = val;
1147    term.id = id;
1148    term.vertex = *vtx;
1149    res = darray_power_term_push_back(&handle->path->power_terms, &term);
1150    if(res != RES_OK) goto error;
1151
1152  exit:
1153    return res;
1154  error:
1155    goto exit;
1156  }
1157
1158  res_T
1159  green_path_add_flux_term
1160    (struct green_path_handle* handle,
1161     struct sdis_interface* interf,
1162     const struct sdis_interface_fragment* frag,
1163     const double val)
1164  {
1165    struct green_path* path;
1166    struct flux_term* terms;
1167    struct unmerged_flux_term term = FLUX_TERM_NULL__;
1168    size_t nterms;
1169    unsigned id;
1170    res_T res = RES_OK;
1171    ASSERT(handle && interf && frag && val >= 0);
1172
1173    res = ensure_interface_registration(handle->green, interf);
1174    if(res != RES_OK) goto error;
1175
1176    path = handle->path;
1177    terms = darray_flux_term_data_get(&path->flux_terms);
1178    nterms = darray_flux_term_size_get(&path->flux_terms);
1179    id = interface_get_id(interf);
1180
1181    /* store term */
1182    term.term = val;
1183    term.id = id;
1184    term.fragment = *frag;
1185    res = darray_flux_term_push_back(&handle->path->flux_terms, &term);
1186    if(res != RES_OK) goto error;
1187
1188  exit:
1189    return res;
1190  error:
1191    goto exit;
1192  }
```

Listing 8: green_function_solve_path for unmerged terms

```
1193  static res_T
1194  green_function_solve_path
1195    (struct sdis_green_function* green,
1196     const size_t ipath,
1197     double* weight)
```

```
1198 {
1199   const struct power_term* power_terms = NULL;
1200   const struct flux_term* flux_terms = NULL;
1201   const struct green_path* path = NULL;
1202   const struct sdis_medium* medium = NULL;
1203   const struct sdis_interface* interf = NULL;
1204   struct sdis_scene* scn = NULL;
1205   double power;
1206   double flux;
1207   double end_temperature;
1208   size_t i, n;
1209   res_T res = RES_OK;
1210   ASSERT(green && ipath < darray_green_path_size_get(&green->paths) && weight);
1211
1212   path = darray_green_path_cdata_get(&green->paths) + ipath;
1213   if(path->end_type == SDIS_GREEN_PATH_END_ERROR) { /* Rejected path */
1214     res = RES_BAD_OP;
1215     goto error;
1216   }
1217
1218   /* Compute medium power terms */
1219   power = 0;
1220   n = darray_power_term_size_get(&path->power_terms);
1221   power_terms = darray_power_term_cdata_get(&path->power_terms);
1222   FOR_EACH(i, 0, n) {
1223     medium = green_function_fetch_medium(green, power_terms[i].id);
1224     power += power_terms[i].term
1225       * solid_get_volumic_power(medium, &power_terms[i].vertex);
1226   }
1227
1228   /* Compute interface fluxes */
1229   flux = 0;
1230   n = darray_flux_term_size_get(&path->flux_terms);
1231   flux_terms = darray_flux_term_cdata_get(&path->flux_terms);
1232   FOR_EACH(i, 0, n) {
1233     interf = green_function_fetch_interf(green, flux_terms[i].id);
1234     flux += flux_terms[i].term
1235       * interface_side_get_flux(interf, &flux_terms[i].fragment);
1236   }
1237
1238   /* Compute path's end temperature */
1239   switch(path->end_type) {
1240     case SDIS_GREEN_PATH_END_AT_INTERFACE:
1241       interf = green_function_fetch_interf(green, path->limit_id);
1242       end_temperature =
1243         interface_side_get_temperature(interf, &path->limit.fragment);
1244       break;
1245     case SDIS_GREEN_PATH_END_IN_VOLUME:
1246       medium = green_function_fetch_medium(green, path->limit_id);
1247       end_temperature = medium_get_temperature(medium, &path->limit.vertex);
1248       break;
1249     case SDIS_GREEN_PATH_END_RADIATIVE:
1250       SDIS(green_function_get_scene(green, &scn));
1251       SDIS(scene_get_ambient_radiative_temperature(scn, &end_temperature));
1252       if(end_temperature < 0) { /* Cannot have it negative if used */
1253         res = RES_BAD_ARG;
1254         goto error;
1255       }
1256       break;
1257     default: FATAL("Unreachable code.\n"); break;
1258   }
1259
1260   /* Compute the path weight */
1261   *weight = power + flux + end_temperature;
1262
```

```
1263  exit:
1264    return res;
1265  error:
1266    goto exit;
1267  }
```

Appendix C. Figures

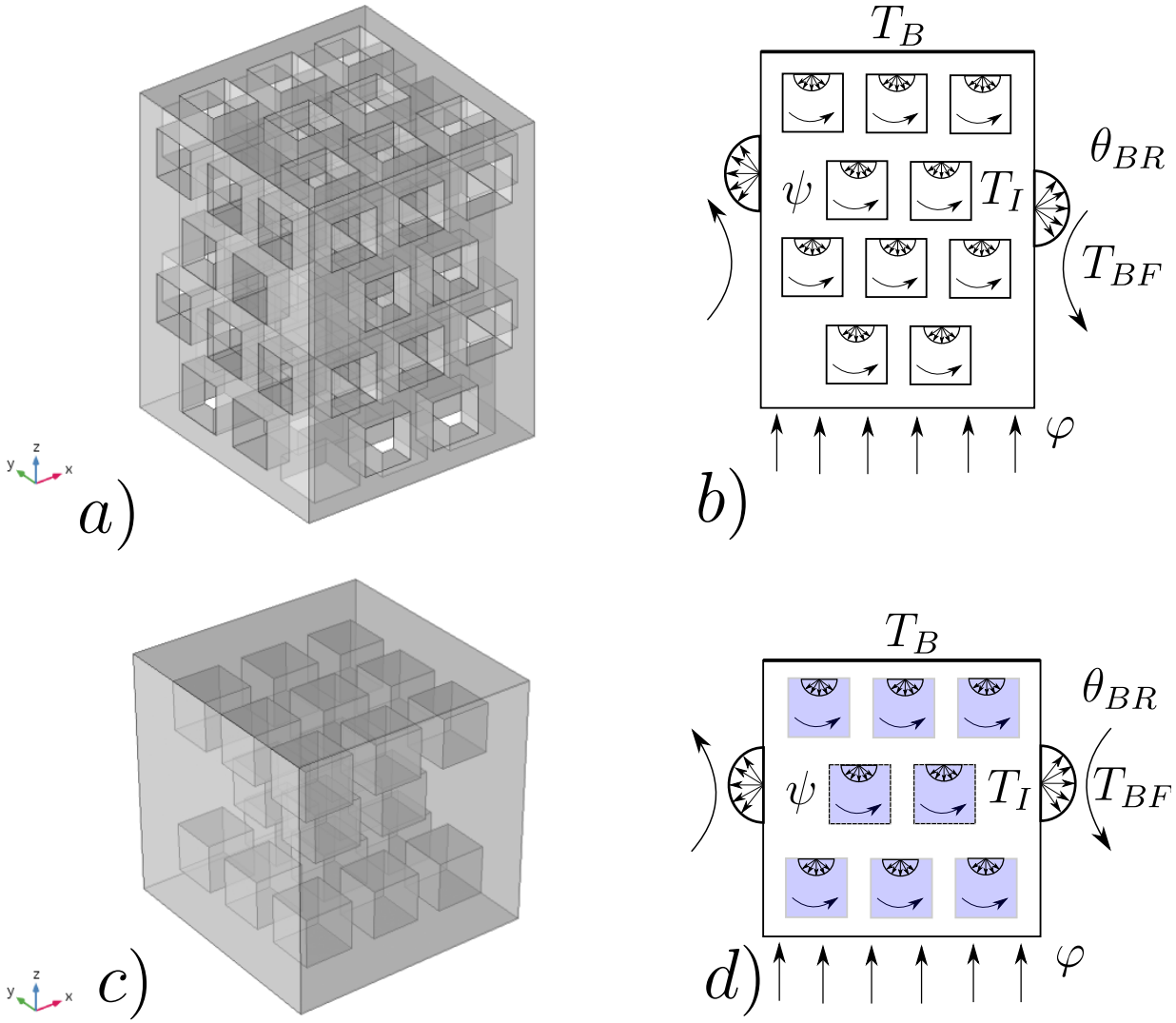

Figure C.13: Schemes of the two benchmark configurations. **a)** Geometry with open cavities; **b)** Different sources applied to this first benchmark for the physical problem. - **c)** Geometry with enclosed cavities, **d)** Sources applied the second benchmark for the physical problem.

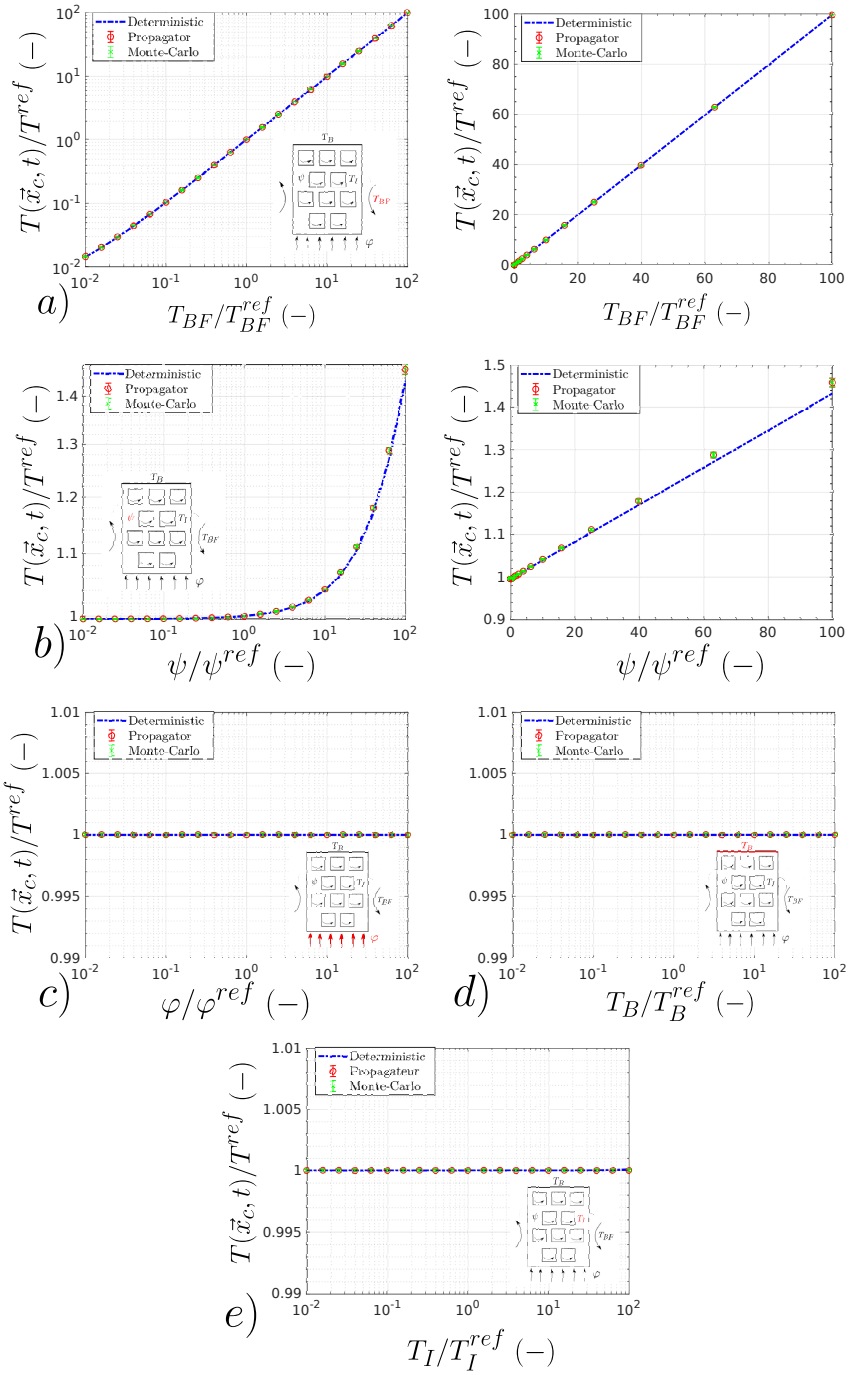

Figure C.14: Open-porosity geometry without radiative transfer : **a)** Ambient fluid temperature **b)** Power density **c)** Flux density **d)** Solid boundary temperature **e)** Initial temperature. Volume and surface of the geometry are noted V and S and $L = 4V/S$ ($L = 1m$) is retained as the characteristic size. The probe location $\vec{x}_c = (0.5, 0.5, 0.5)$ (at the center of the solid). The probe time estimation is $t* = \frac{\lambda t}{\rho c L^2 = 0.89}$ ($t = 1 \times 10^6$). The fluid reference temperature $T_{\mathrm{BF}}^{\mathrm{ref}} = 505\mathrm{K}$. The reference physical parameters are $\frac{T_{\mathrm{I}}^{\mathrm{ref}} - T_{\mathrm{BF}}^{\mathrm{ref}}}{T_{\mathrm{BF}}^{\mathrm{ref}}} = -0.01$ (reference initial temperature $T_{\mathrm{I}}^{\mathrm{ref}} = 500\mathrm{K}$), $\frac{T_{\mathrm{B}}^{\mathrm{ref}} - T_{\mathrm{BF}}^{\mathrm{ref}}}{T_{\mathrm{BF}}^{\mathrm{ref}}} = +0.01$ (known reference boundary temperature $T_{\mathrm{B}}^{\mathrm{ref}} = 510\mathrm{K}$). The convective heat transfer coefficient $h = 10\ W.m^{-2}.K^{-1}$ and the thermal conductivity $\lambda = 1\ W.m^{-1}.K^{-1}$ leading to $Bi = \frac{hL}{\lambda} = 10.68$ ($h = 10\ W.m^{-2}.K^{-1}$). For propagator function, initial calculation uses a dimensionless numerical step $\frac{\delta}{L} = 0.05$ and $N = 10^4$. The reference volume power value $\Psi^{\mathrm{ref}} = 20 W.m^{-3}$. The reference density flux $\varphi^{\mathrm{ref}} = 2000 W.m^{-2}$.

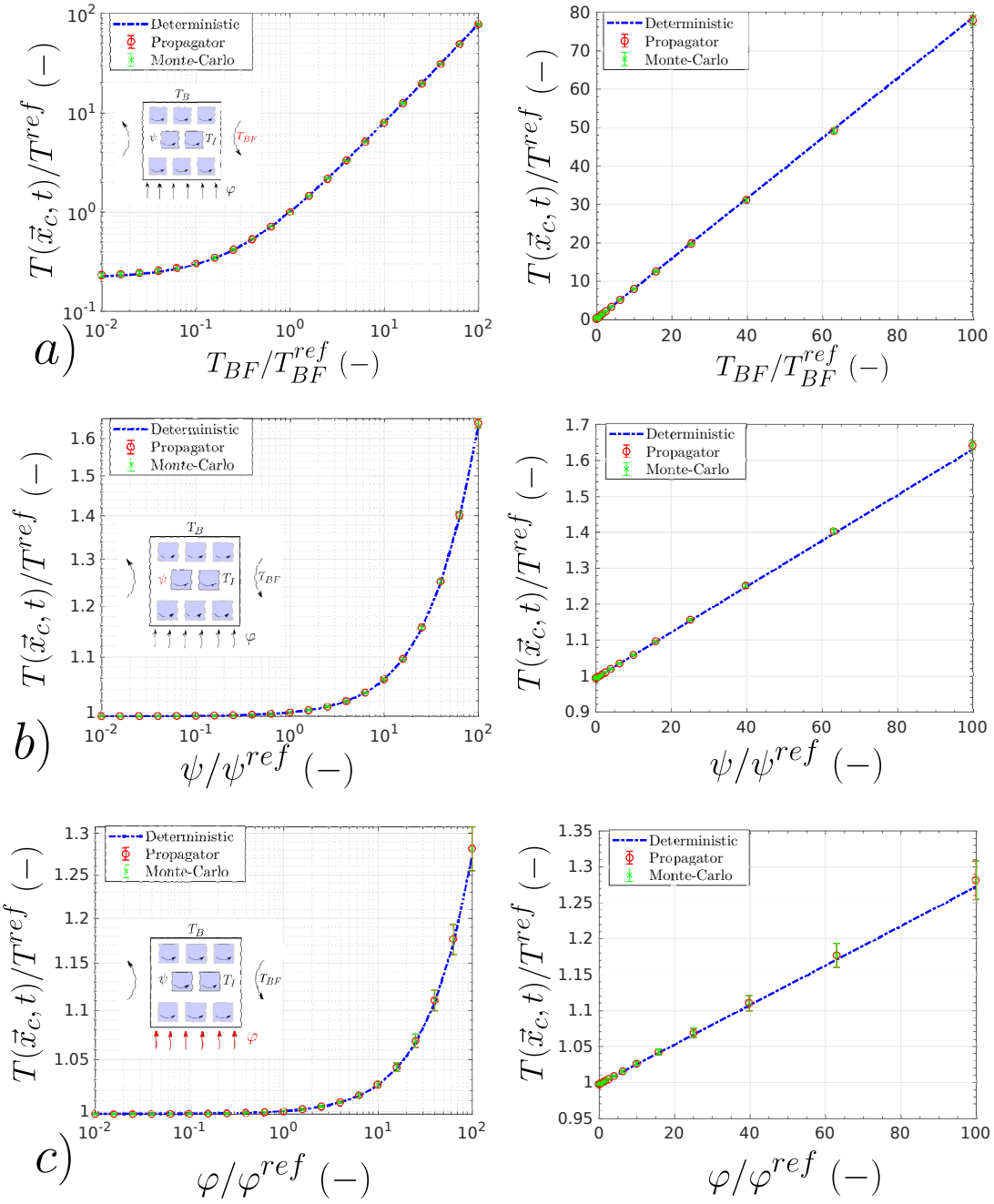

Figure C.15: Closed-porosity geometry without radiative transfer : **a)** Ambient fluid temperature **b)** Power density **c)** Flux density **d)** Solid boundary temperature **e)** Initial temperature. Volume and surface of the geometry are noted V and S and $L = 4V/S$ ($L = 2.4358m$) is retained as the characteristic size. The probe location $\vec{x}_c = (0.5, 0.5, 0.5)$ (at the center of the solid). The probe time estimation is $t* = \frac{\Delta t}{\rho c L^2 = 0.169}$ ($t = 1 \times 10^7$). The fluid reference temperature $T_{\mathrm{BF}}^{\mathrm{ref}} = 505\mathrm{K}$. The reference physical parameters are $\frac{T_{\mathrm{I}}^{\mathrm{ref}} - T_{\mathrm{BF}}^{\mathrm{ref}}}{T_{\mathrm{BF}}^{\mathrm{ref}}} = -0.01$ (reference initial temperature $T_{\mathrm{I}}^{\mathrm{ref}} = 500\mathrm{K}$), $\frac{T_{\mathrm{B}}^{ref} - T_{\mathrm{BF}}^{\mathrm{ref}}}{T_{\mathrm{BF}}^{\mathrm{ref}}} = +0.01$ (known reference boundary temperature $T_{\mathrm{B}}^{\mathrm{ref}} = 510\mathrm{K}$). The convective heat transfer coefficient is expressed as $h = 10\ W.m^{-2}.K^{-1}$, this is leading to $Bi = \frac{hL}{\lambda} = 24.358$ with $\lambda = 1\ W.m^{-1}.K^{-1}$. For propagator function, initial calculation uses a dimensionless numerical step $\frac{\delta}{L} = 0.05$ and $N = 10^4$. The reference volume power value $\Psi^{\mathrm{ref}} = 1W.m^{-3}$. The reference density flux $\varphi^{\mathrm{ref}} = 5W.m^{-2}$.

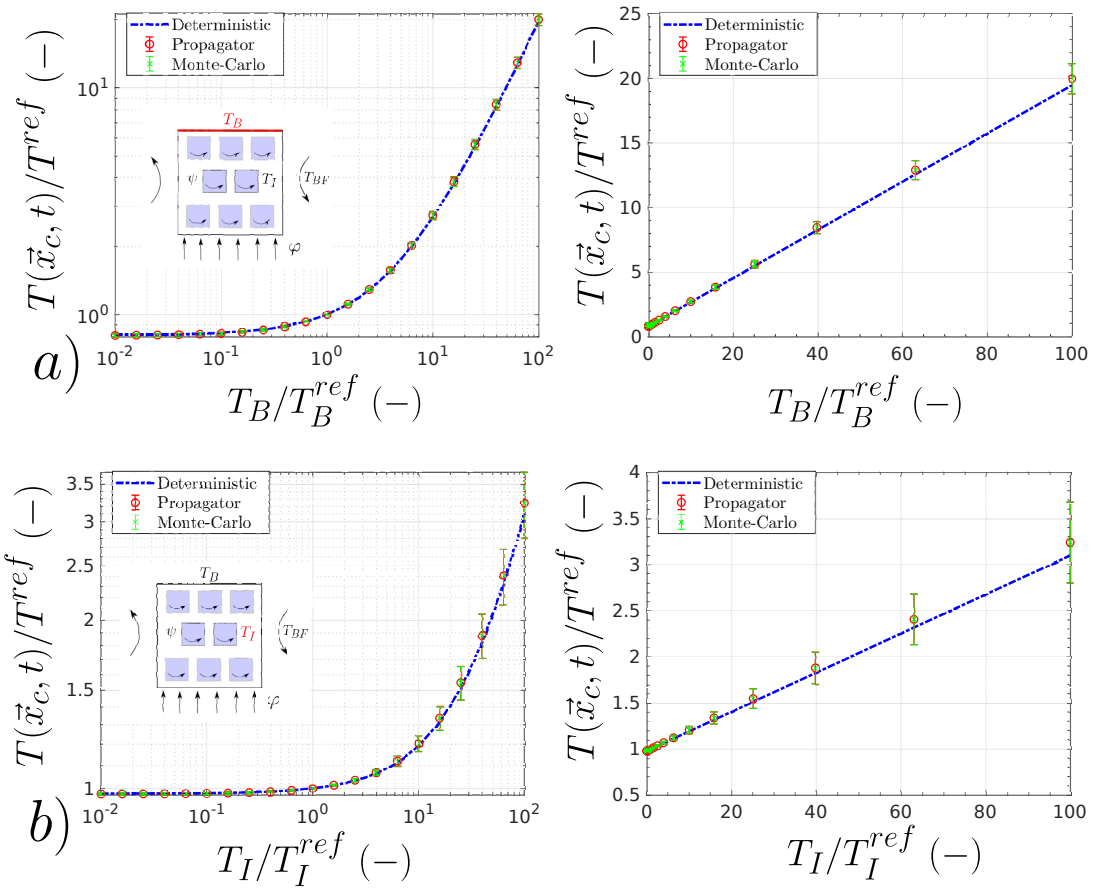

Figure C.16: Closed-porosity geometry without radiative transfer : **a)** ambient fluid temperature **b)** power density **c)** flux density **d)** solid boundary temperature **e)** initial temperature.

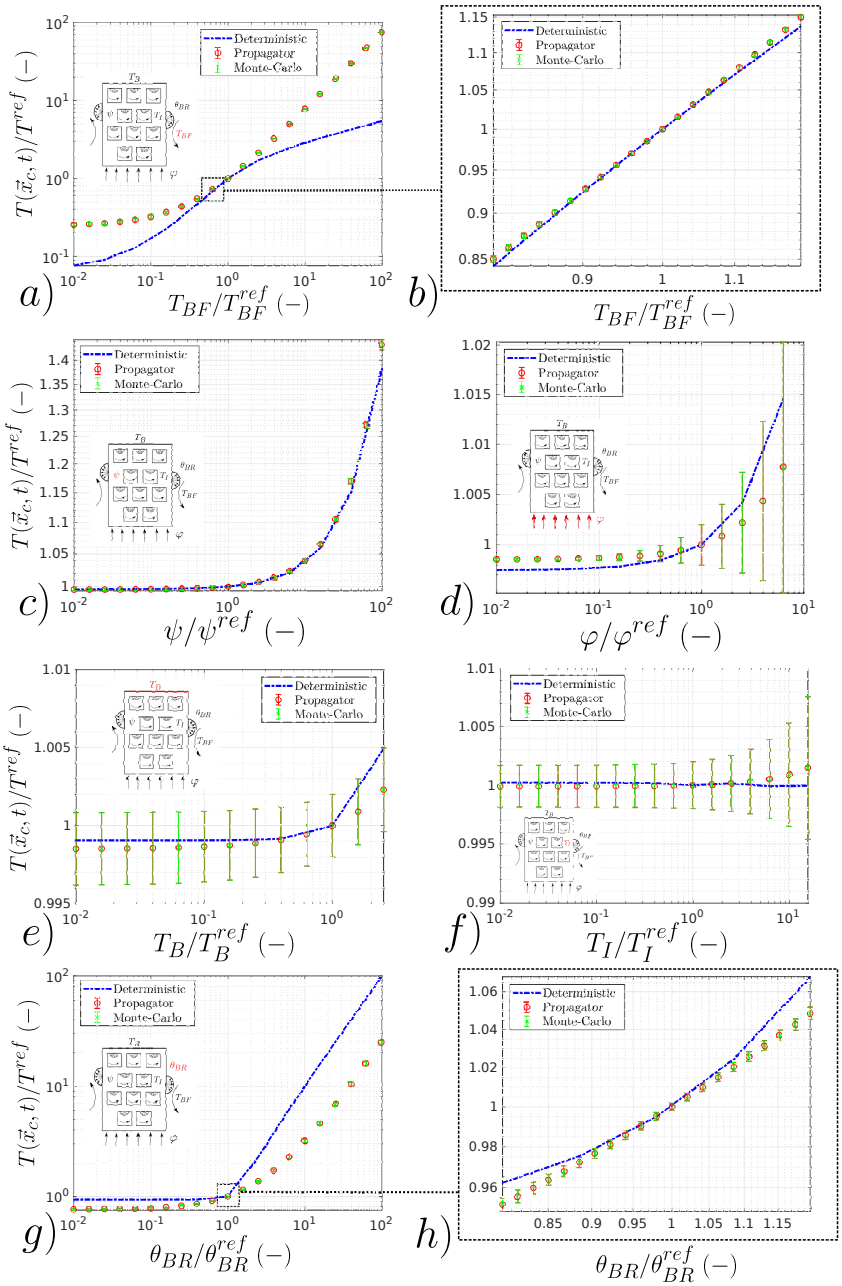

Figure C.17: open-porosity geometry with radiative transfer : **a-b)** ambient fluid temperature **c)** power density **d)** flux density **e)** solid boundary temperature **f)** initial temperature **g-h)** ambient radiant temperature. Volume and surface of the geometry are noted V and S and $L = 4V/S$ ($L = 1m$) is retained as the characteristic size. The probe location $\vec{x}_c = (0.5, 0.5, 0.5)$ (at the center of the solid). The probe time estimation is $t* = \frac{\lambda t}{\rho c L^2 = 0.89}$ ($t = 1 \times 10^6$). The fluid reference temperature $T_{\mathrm{BF}}^{\mathrm{ref}} = 505\mathrm{K}$. The reference physical parameters are $\frac{T_{\mathrm{I}}^{\mathrm{ref}} - \theta_{\mathrm{BR}}^{\mathrm{ref}}}{\theta_{\mathrm{BR}}^{\mathrm{ref}}} = -0.01$ (reference initial temperature $T_{\mathrm{I}}^{\mathrm{ref}} = 500\mathrm{K}$ and ambient radiative temperature $\theta_{\mathrm{BR}}^{\mathrm{ref}} = 505\mathrm{K}$), $\frac{T_{\mathrm{B}}^{\mathrm{ref}} - \theta_{\mathrm{BR}}^{\mathrm{ref}}}{\theta_{\mathrm{BR}}^{\mathrm{ref}}} = +0.01$ (known reference boundary temperature $T_{\mathrm{B}}^{\mathrm{ref}} = 510\mathrm{K}$). The radiative transfer coefficient is expressed as $h_R = 4\epsilon\sigma T_{\mathrm{ref}}^3 = 29.21$ with the emissivity epsilon $\epsilon = 1$, the Stefan-Boltzmann constant $\sigma = 5.6703 \times 10^{-8}\ J.s^{-1}.m^{-2}.K^{-4}$ and the reference temperature $T_{\mathrm{ref}} = 305\mathrm{K}$. This is leading to $Bi_R = \frac{h_R L}{\lambda} = 31.21$, with $\lambda = 1\ W.m^{-1}.K^{-1}$ and $Bi = \frac{hL}{\lambda} = 10.68$ ($h = 10\ W.m^{-2}.K^{-1}$). For propagator function, initial calculation uses a dimensionless numerical step $\frac{\delta}{L} = 0.05$ and $N = 10^4$. The reference volume power value $\Psi^{\mathrm{ref}} = 20 W.m^{-3}$. The reference density flux $\varphi^{\mathrm{ref}} = 2000 W.m^{-2}$.

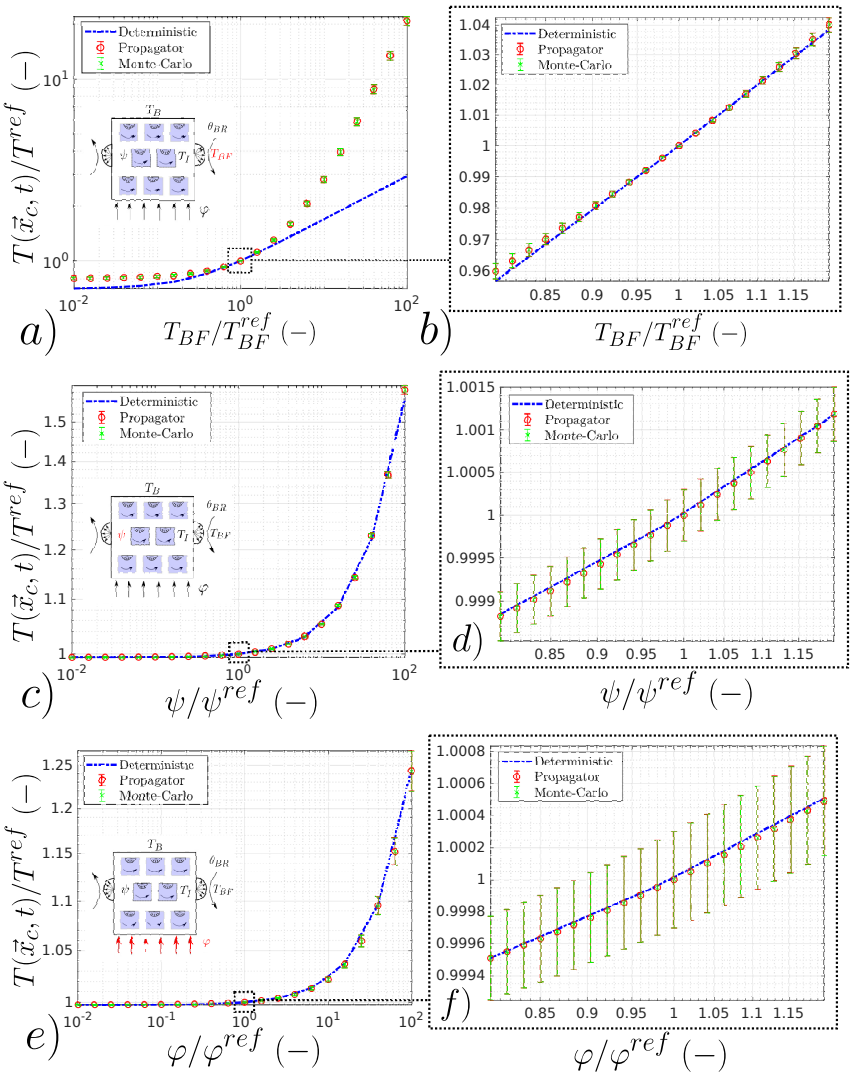

Figure C.18: closed-porosity geometry with radiative transfer : **a-b)** ambient fluid temperature **c-d)** power density **e-f)** flux density. Volume and surface of the geometry are noted V and S and $L = 4V/S$ ($L = 1m$) is retained as the characteristic size. The probe location $\vec{x}_c = (0.5, 0.5, 0.5)$ (at the center of the solid). The probe time estimation is $t* = \frac{\lambda t}{\rho c L^2 = 2.4358}$ ($t = 1 \times 10^6$). The fluid reference temperature $T_{\mathrm{BF}}^{\mathrm{ref}} = 505\mathrm{K}$. The reference physical parameters are $\frac{T_{\mathrm{I}}^{\mathrm{ref}} - \theta_{\mathrm{BR}}^{\mathrm{ref}}}{\theta_{\mathrm{BR}}^{\mathrm{ref}}} = -0.01$ (reference initial temperature $T_{\mathrm{I}}^{\mathrm{ref}} = 500\mathrm{K}$ and ambient radiative temperature $\theta_{\mathrm{BR}}^{\mathrm{ref}} = 505\mathrm{K}$), $\frac{T_{\mathrm{B}}^{\mathrm{ref}} - \theta_{\mathrm{BR}}^{\mathrm{ref}}}{\theta_{\mathrm{BR}}^{\mathrm{ref}}} = +0.01$ (known reference boundary temperature $T_{\mathrm{B}}^{\mathrm{ref}} = 510\mathrm{K}$). The radiative transfer coefficient is expressed as $h_R = 4\epsilon\sigma T_{\mathrm{ref}}^3 = 29.21$ with the emissivity epsilon $\epsilon = 1$, the Stefan-Boltzmann constant $\sigma = 5.6703 \times 10^{-8}\ J.s^{-1}.m^{-2}.K^{-4}$ and the reference temperature $T_{\mathrm{ref}} = 305\mathrm{K}$. This is leading to $Bi_R = \frac{h_R L}{\lambda} = 76.017$, with $\lambda = 1\ W.m^{-1}.K^{-1}$ and $Bi = \frac{hL}{\lambda} = 24.358$ ($h = 10\ W.m^{-2}.K^{-1}$). For propagator function, initial calculation uses a dimensionless numerical step $\frac{\delta}{L} = 0.05$ and $N = 10^4$. The reference volume power value $\Psi^{\mathrm{ref}} = 1W.m^{-3}$. The reference density flux $\varphi^{\mathrm{ref}} = 5W.m^{-2}$.

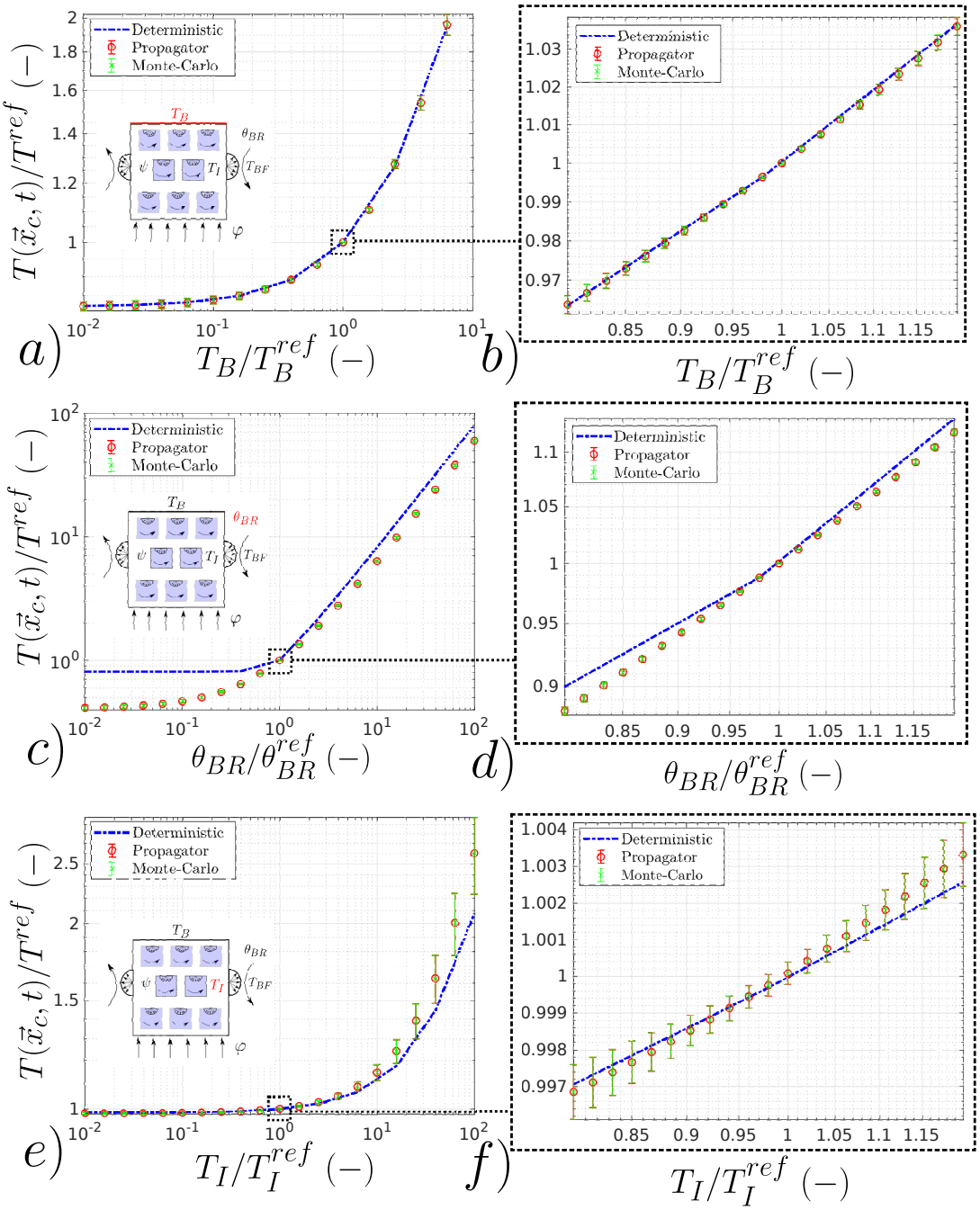

Figure C.19: Closed-porosity geometry with radiative transfer : **a-b)** solid boundary temperature **c-d)** ambient radiant temperature **e-f)** initial temperature.