Computational Science and Engineering
(International Master's Program)

Technische Universität München
Department of Informatics

Master's Thesis

Robust and Efficient Barycentric Cell-Interpolation for Volumetric Coupling with preCICE

Boris G. Martin

Computational Science and Engineering
(International Master's Program)

Technische Universität München

Master's Thesis

Robust and Efficient Barycentric Cell-Interpolation for Volumetric Coupling with preCICE

Author:	Boris G. Martin
Examiner:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Assistant advisor:	M.Sc. Frédéric Simonis
Submission Date:	August 5th, 2022

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

August 5th, 2022 Boris G. Martin

Acknowledgments

First and foremost, I would like to thank my supervisor, Frédéric Simonis, for his support throughout this thesis. His helpful advices and recommendations made it a very instructive experience. More generally, I am grateful to all members of the preCICE team for their welcoming attitudes. In particular, I would like to thank David Schneider for his help with ASTE and code reviews and Benjamin Uekermann for his meticulous project management and additional feedback.

Working on preCICE gave me a role in a decade-old, mature yet still expanding academic project. As Isaac Newton once said, we see further because we are standing on the shoulders of giants, and I'm grateful to the generations of students and researchers who brought preCICE to its current state. I hope my contribution will help future students and users feel the same.

Additionally, I would like to express my gratitude to professors Éric Delhez and Pierre Geurts from the University of Liège, whose recommendation letters made my studies at TUM possible.

Last but not least, I am very grateful to my family and friends for their support throughout the last two years. Special thanks go to my friends Régnauld and Guillaume for their advices, feedbacks and more simply their presence.

Abstract

The open-source library preCICE enables the creation of partitioned multi-physics simulations where different solvers run independently and exchange coupling data, such as forces, displacements or heat fluxes.

These different solvers all use their own space discretization of the geometry, which usually leads to non-matching meshes. Coupling these codes requires some form of data mapping of physical quantities from a solver's mesh to the other solver's mesh.

preCICE offers different mapping methods, varying in accuracy, cost, and access to solver internals (such as derivatives or mesh topology). One of them, the nearest-projection mapping uses mesh connectivity to project the points of one mesh into the other mesh and to construct a piecewise linear interpolant. However, this projection method was designed for surface coupling: each mesh used by preCICE represents the boundary of a solver's domain.

preCICE can also be used for volumetric coupling, where data is exchanged on the whole domain and not only at the boundary. In this context, Nearest-Projection Mapping should not be used in its current implementation, as it assumes the mesh represents a boundary and not the domain. This thesis proposes and implement a new data mapping method (called *linear cell interpolation*) adapted for volumetric coupling, using mesh connectivity to build a piecewise linear interpolant over the whole domain instead of a coupling boundary. In 2D, triangles are used, whereas tetrahedra (which were added to preCICE in this context) are used in 3D.

This mapping is mathematically derived then implemented in preCICE. Each point gets mapped to the enclosing triangle or tetrahedra (which is done efficiently with R-Trees), and barycentric coordinates of this point with respect to that cell must be computed. An extensive test suite is written to cover edge cases and validate the implementation, including for solvers running in parallel. Linear cell interpolation mapping is also compared to other mappings using ASTE, the preCICE testing environment, to evaluate accuracy and runtime performance. Finally, a new preCICE tutorial is designed to showcase its use in

coupled simulations.

This new data mapping is more accurate than the cheaper alternatives while significantly cheaper than radial basis function interpolation, the most accurate mapping method currently available. It is expected to become a standard choice for preCICE users solving problems involving volumetric coupling.

Contents

1. Introduction

1.1. Context

Numerical simulations have been used for decades to solve a wide variety of problems such as fluid dynamics, solid mechanics and heat transfer. Most of these problems can be formulated using partial differential equations. In real life, most problems involve different aspects of physics. A heat exchanger involves fluid flow and heat transfer between a solid and a fluid; climate and weather depend on complex interactions between the atmosphere, the ocean, the land and the sun; arteries let the blood flow but deform according to blood pressure, and planes fly because of interactions with the air, but the air flow itself is changed by the presence of a plane. In scientific computing, the discipline of solving these complex, coupled problems is called *multi-physics simulations*.

Naturally, multi-physics simulations are more complex and harder to solve. One direct approach (the *monolithic approach*) is to write specialized software for the specific problem to solve. It is effective and convenient since there is a unique code orchestrating the simulation. However, it lacks generality and can be quite complex to maintain in the long run or to reuse in a different context.

Another approach to multi-physics simulations is the *partitioned approach*. Instead of developing complex software handling everything, we can reuse single-physics solvers and "glue" them together. This brings several challenges: each solver (also called *participant*) uses its own discretization, its own configuration format, its own time stepping, etc. However, if successfully achieved, it can greatly improve flexibility, reusability and time-to-solution. A researcher who needs a simulation with two specific physics can use two specialized codes and couple them without needing to rewrite everything from nothing. We call black-box coupling the usage of partitioned simulations where each solver's knowledge of the other ones is limited to explicitly communicated data.

The preCICE library[5] is a free and open-source library designed to make partitioned simulations easier. It offers a minimally invasive API that allows users to configure coupled simulations with black-box coupling. The library manages communications between the different participants, and is responsible for data mapping and coordinating the coupling. It supports explicit coupling, where solvers exchange data once per time window then move to the next one, and implicit coupling, where a time window can be repeated until convergence (i.e. all participants agree on the exchanged data). In that case, preCICE offers

acceleration methods to help the convergence.
This library has been successfully used for, among others, conjugate heat transfer[24] (i.e. the exchange of heat between a solid and a moving fluid), fluid-structure interaction[10][18] and for coupling GPU ray-tracer to a heat transfer solver[23]. It has been designed with parallelization in mind and has been successfully run on clusters.[22]

1.2. Motivation

All these examples have in common that coupling occurs at a physical interface: when simulating conjugate heat transfer, only the temperature (and heat fluxes) close to the wet surface are exchanged; in fluid-structure interactions, the forces applied by a fluid on a solid are on the boundary, and the fluid solver only needs to know the deformation of the structure, but not the internal state. These problems involve surface coupling.

Sometimes, one must do volumetric coupling, where data is also exchanged in the inside of the domains. One example is the simulation of chemical reactions in a fluid, where reactions occur everywhere depending on the local concentration of substances and can locally impact the temperature or density of the fluid. Magnetohydrodynamics can also be seen as a case of volumetric coupling: a plasma can be simulated as a fluid with additional volume forces coming from electromagnetic fields, which could be computed by a specialized electromagnetics solver.

In its current state, preCICE is not designed for problems involving volumetric coupling. However, it has already been successfully used by researchers in aeroelastics simulations[3], in modelling crystal growth[7], in muscular activation simulations[16] and in nuclear fusion simulations[20]. These results are encouraging, however some improvements to preCICE can still be done. In particular, current data mapping methods were not designed with volumetric coupling in mind.

Among the existing data mapping methods:

- Nearest-neighbor mapping works but is the most inaccurate. However, accuracy can be increased by using gradient data.
- Nearest-projection is designed for surface coupling: points are projected into surface elements (triangles in 3D and edges in 2D) and then data is linearly interpolated. This is accurate when points are close to the surface, but not when the projection distance is long. In volumetric coupling, each point would be mapped after being projected on the *boundary* of the element containing it, which decreases accuracy significantly.
- Radial basis function (RBF) interpolation works in theory but is quite expensive, as a linear problem must be solved every time data must be mapped. This is especially

expensive in the context of volumetric coupling, as the number of vertices involved tends to be much higher than with surface coupling.

While with surface coupling, nearest-projection is often an optimal trade off between cost and accuracy, and RBF interpolation is very accurate while still affordable, neither is suited for volumetric coupling: nearest-projection suffers from a degraded accuracy, and RBF interpolation is simply too expensive.

1.3. Goal and structure of this thesis

The main work of this thesis is the implementation of a new mapping method designed specifically for simulations involving volumetric coupling. It is an adaptation of the nearest-projection method and aims at providing a second-order accurate but inexpensive mapping through linear interpolation using mesh connectivity. This new mapping method is called *linear cell interpolation*.

In the second chapter of this thesis, we introduce the preCICE library with a focus on its data mapping features. In particular, we examine the usability of the currently available methods in the context of volumetric coupling.
The next chapter describes the theoretical background necessary to implement linear cell interpolation mapping.
The fourth chapter describes how preCICE was changed to support this new data mapping method and how it was extensively tested in a wide variety of situations.
The fifth chapter is dedicated to accuracy measurements and runtime analysis of linear cell interpolation mapping to compare it to other data mapping methods.
The sixth chapter describes a 2D example case of coupled simulations involving volumetric coupling, using FEniCS to simulate chemical reactions. The changes to the FEniCS adapter are described and tested to showcase applications of linear cell interpolation mapping to complete simulations. We also discuss the benefits of using it over nearest-neighbor mapping.
Finally, we briefly discuss possible ways to further increase the order of accuracy of data mapping, discuss the reproducibility of the main results and summarize the thesis.
The appendices contain technical details such as the algorithms used to compute linear interpolation in a triangle or a tetrahedron.

2. An overview of preCICE

The preCICE library provides the different components to build a partitioned simulation. Each solver reads the preCICE configuration file (see below) and connects to required participants, then works alone with synchronization points coordinated by preCICE. When required, a participant reads or writes data. An overview of ecosystem is given on figure 2.1. A single-physics solver must be adapted to preCICE, to do the following tasks:

- Initialize preCICE
- Configure the coupling mesh
- Handle the main loop
- Write and/or read data
- Advance the coupling

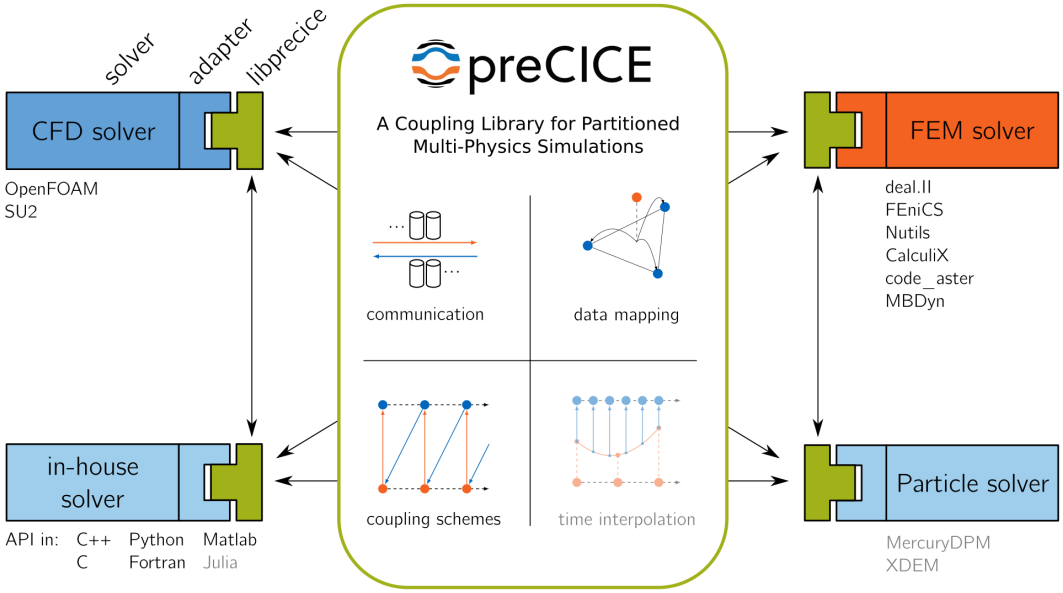


Figure 2.1.: Overview of the preCICE ecosystem, featuring the library and adapted solvers. Picture from the preCICE website.

2.1. Introduction to preCICE

A coupled simulation using preCICE requires first and foremost a *preCICE configuration file*. This XML file describes how many participants are coupled, their communications, how data from a solver must be mapped to other solvers, how they are coupled (in parallel or sequentially ? Using explicit coupling or implicit coupling ?), etc. For illustration, let us consider the tutorial case *perpendicular-flap*[1]. An elastic flap (illustrated on figure 2.2) fixed to the floor of a channel is pushed by a moving fluid.

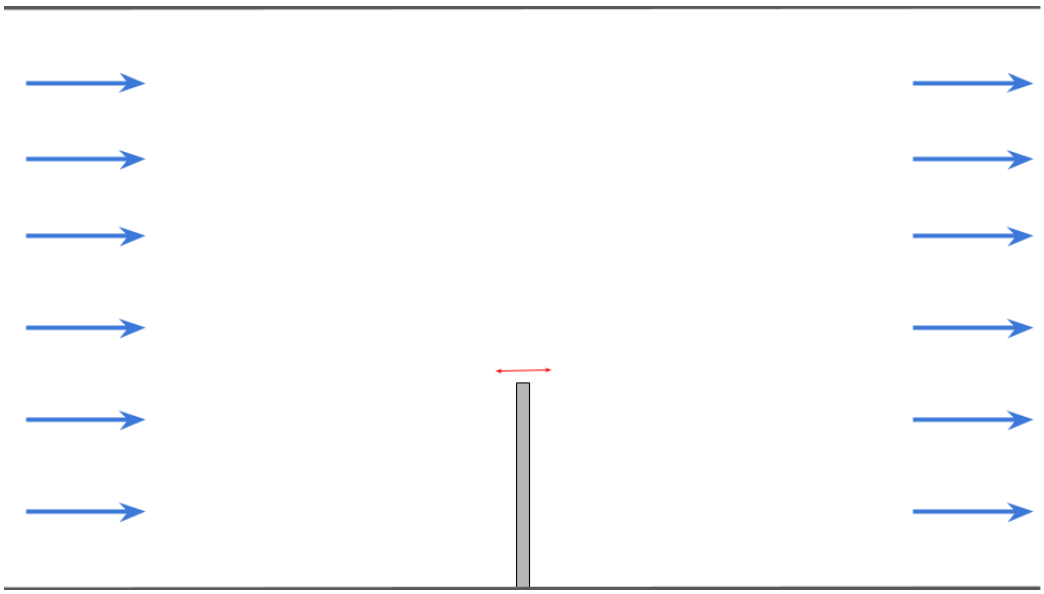


Figure 2.2.: Setup of the perpendicular flap tutorial. (Image from the tutorial documentation)

In this case, the moving fluid applies forces on the flap, and the flap moves and sends its displacement to the fluid solver. The preCICE configuration file is as follows:

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <precice-configuration>
3
4
5   <solver-interface dimensions="2">
6     <data:vector name="Force" />
7     <data:vector name="Displacement" />
```

[1] https://precice.org/tutorials-perpendicular-flap.html

```
8
9       <mesh name="Fluid-Mesh">
10        <use-data name="Force" />
11        <use-data name="Displacement" />
12      </mesh>
13
14      <mesh name="Solid-Mesh">
15        <use-data name="Displacement" />
16        <use-data name="Force" />
17      </mesh>
18
19      <participant name="Fluid">
20        <use-mesh name="Fluid-Mesh" provide="yes" />
21        <use-mesh name="Solid-Mesh" from="Solid" />
22        <write-data name="Force" mesh="Fluid-Mesh" />
23        <read-data name="Displacement" mesh="Fluid-Mesh" />
24        <mapping:rbf-thin-plate-splines
25          direction="write"
26          from="Fluid-Mesh"
27          to="Solid-Mesh"
28          constraint="conservative" />
29        <mapping:rbf-thin-plate-splines
30          direction="read"
31          from="Solid-Mesh"
32          to="Fluid-Mesh"
33          constraint="consistent" />
34      </participant>
35
36      <participant name="Solid">
37        <use-mesh name="Solid-Mesh" provide="yes" />
38        <write-data name="Displacement" mesh="Solid-Mesh" />
39        <read-data name="Force" mesh="Solid-Mesh" />
40        <watch-point mesh="Solid-Mesh" name="Flap-Tip" coordinate="0.0;1" />
41      </participant>
42
43      <m2n:sockets from="Fluid" to="Solid" exchange-directory=".." />
44
45      <coupling-scheme:parallel-implicit>
46        <time-window-size value="0.01" />
47        <max-time value="5" />
48        <participants first="Fluid" second="Solid" />
49        <exchange data="Force" mesh="Solid-Mesh" from="Fluid" to="Solid" />
```

```
50        <exchange data="Displacement" mesh="Solid-Mesh" from="Solid" to="Fluid" />
51        <max-iterations value="50" />
52        <relative-convergence-measure limit="5e-3" data="Displacement" mesh="Solid-
53        <relative-convergence-measure limit="5e-3" data="Force" mesh="Solid-Mesh" /
54        <acceleration:IQN-ILS>
55          <data name="Displacement" mesh="Solid-Mesh" />
56          <data name="Force" mesh="Solid-Mesh" />
57          <preconditioner type="residual-sum" />
58          <filter type="QR2" limit="1e-2" />
59          <initial-relaxation value="0.5" />
60          <max-used-iterations value="100" />
61          <time-windows-reused value="15" />
62        </acceleration:IQN-ILS>
63      </coupling-scheme:parallel-implicit>
64    </solver-interface>
65  </precice-configuration>
```

This file tells preCICE that two solvers are running: a fluid solver than reads displacement from a moving structure, and a solid solver reading forces from the fluid solver. They run in parallel, using implicit coupling with quasi-Newton acceleration to converge faster (see [5] for details). This simulation runs for a duration of 5 time units, and participants synchronize every 0.01 time units.

Apart from the preCICE configuration file, a simulation case must be designed for each participant. This is solver-specific and almost unchanged by preCICE, except that calls to the preCICE library must be added to the solver. When using officially supported adapters, which are versions of common solvers modified by preCICE developers or plug-ins, slight modifications must be made, typically in the form of an additional configuration file: each solver must know what to send to preCICE and what to read from it. For instance, with the perpendicular flap tutorial, the solid solver must know which points will receive a force from the fluid (or, from its point of view, from preCICE) and for which points it is necessary to write displacements.

2.2. Data Mapping

In partitioned simulations, each solver has its own mesh, with no *a priori* requirements of being matching, having the same resolution or even using the same type of grid: one can couple a finite volume code with a finite element code, for instance. With preCICE, no solver must implement the required conversion: it is done by preCICE and the solver must simply send data from its own mesh, and read data already converted.

2.2.1. Read and write mappings

Since preCICE is decentralized (each solver calls the library on its own, but there is no process dedicated to running preCICE like a server), data mapping must be done by the solvers. When mapping data from a participant A to a participant B, either A must compute the mapping then send mapped data to B, or B must receive the unmapped data from A and compute the mapping itself. The first approach is called a *write* mapping, whereas the latter is known as a *read* mapping. The difference is negligible when both participants are single-threaded (or multithreaded with shared memory), but when participants run in parallel, additional restrictions apply: all consistent mappings (see below) must be *read* mapping, while all conservative mappings must be *write* mappings. Technical reasons behind this limitation can be found in [22].

2.2.2. Consistent, conservative and scaled-consistent mappings

Depending on the physical nature of the transferred data, different constraints apply. When using continuous data such as displacement or temperature, a consistent mapping must be used: if the input data is constant, the output data must be the same constant, and some form of interpolation is done when the input is not constant.

Most of the time[2], mappings can be represented by linear maps from the input data to the output data. Let x be the input data, y the output data and M a mapping matrix such that $y = Mx$. The consistency requirement means that if $x_j = c$ for all j, then the same holds for y. By definition of matrix multiplication, $y_i = \sum_j M_{ij} x_j = c$. This can only hold if $\sum_j M_{ij} = 1$, i.e. when all rows sum to 1.

On the contrary, some forms of data require a conservative mapping, a mapping that preserves total quantities. The most natural example of data requiring a conservative mapping is forces: the total force applied should not change under transformation; only its distribution can change. If $y = Mx$ and $\sum_i y_i$ must be the same as $\sum_j x_j$, expanding the product yields $\sum_i \sum_j M_{ij} x_j = \sum_j x_j$. If this must be true for all values of x, then $\sum_i M_{ij}$ must be 1 i.e. all columns sum to 1.

A consequence of these algebraic properties is that *the transpose of a consistent mapping from A to B is a conservative mapping from B to A*. Thus, it is usually sufficient to design consistent mappings, and when a conservative mapping is required, build a consistent mapping in the opposite direction then transpose it. Of course, in terms of code, both cases must be implemented, as the mapping is not actually implemented with a matrix.

[2] The notable exception being nearest-neighbor with gradient, where the output depends on both input data and gradient.

Finally, a third type of mapping is the scaled-consistent mapping. It is similar to the consistent mapping, except that the output is afterwards multiplied by a coefficient to ensure preservation of integrals. This requires preCICE to compute the integral in question. In terms of implementation, it is sufficient to use any consistent mapping and rescale afterwards. The typical case a scaled-consistent mapping should be used is the mapping of pressure: it is a continuous field, which excludes a conservative mapping, but the total force applied (the integral of the pressure) should be conserved.

2.2.3. Nearest-neighbor mapping

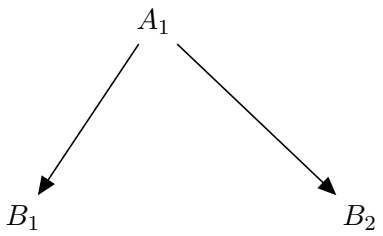
The simplest and cheapest, but least accurate mapping is the nearest-neighbor mapping. In its consistent version, each output vertex takes the value of the nearest vertex from the input mesh. In the conservative version, which is the transposed operation, each input vertex adds its value to the closest output vertex. Section 3.3 describes how to find the nearest vertices efficiently.

Overall, in the consistent case, the interpolant is a piecewise-constant function. It is first order accurate: if the values of the input mesh are samples of a smooth function, the error between this smooth function and the mapped value on an output vertex is $\mathcal{O}(d)$ with d the distance between the output vertex and the closest input vertex. This can be easily seen by truncating the Taylor expansion of the function: if $f(\mathbf{x}+\mathbf{h}) = f(\mathbf{x})+\nabla f(\mathbf{x})\cdot\mathbf{h}+\mathcal{O}(\|\mathbf{h}\|^2)$, then approximating $f(\mathbf{x}+\mathbf{h})$ as $f(\mathbf{x})$ is first order accurate. In particular, this mapping is exact if the vertices on the output mesh are a subset of the vertices of the input mesh. Conversely, a conservative mapping will be exact if the set of input vertices is a subset of the output vertices. More generally, a consistent nearest-neighbor mapping will be accurate when mapping from a fine mesh to a coarser mesh, and a conservative one in the opposite case. A consistent mapping from a coarse mesh to a finer mesh will lead to a "staircase effect", with many close vertices reading the same value. A conservative mapping from a fine mesh to a coarser one will lead to data (e.g. forces) being concentrated on a smaller set of points.

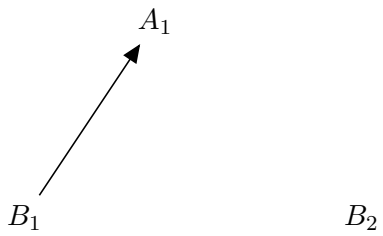
The main advantage of this mapping is its simplicity: it does not need any parameter tuning, any connectivity data nor solving a system of equations. This makes it the easiest to configure and immune to numerical issues like ill-conditioned systems.
Figure 2.3 illustrates different configurations of nearest-neighbors mapping for a mesh A with one vertex, and a mesh B with two vertices.

Using gradient information

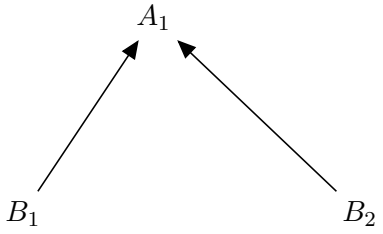
As seen above, the approximation error is $\nabla f(\mathbf{x}) \cdot \mathbf{h} + \mathcal{O}(\|\mathbf{h}\|^2)$. If the gradient is known, as well as the vector from the source point to the output point, then the mapping can become second order at a cheap cost, since the remaining error is simply $\mathcal{O}(\|\mathbf{h}\|^2)$. This



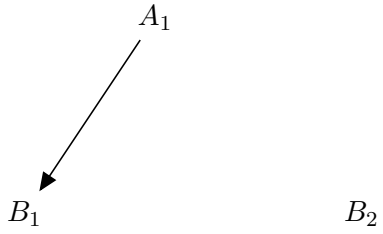
(a) Consistent mapping from A to B. Both output vertices read the same value.



(b) Consistent mapping from B to A. Vertex B_2 is discarded, as B_1 is closer.



(c) Conservative mapping from B to A. The sum of B_1 and B_2 is read. Note how it is the transpose of the consistent mapping from A to B: arrows are reversed.



(d) Conservative mapping from A to B. Note how it is the transpose of the consistent mapping from B to A: arrows are reversed.

Figure 2.3.: Illustration of nearest-neighbor mappings between mesh A and mesh B.

has been recently implemented in preCICE.[2] The main drawback is the need to send the gradient information: depending on the internals of the solver, it cannot always be accurately computed[3]. However, when doable, it greatly improves the nearest-neighbor mapping.

2.2.4. Nearest-projection mapping

The nearest-projection mapping is designed to do linear interpolation in a context of surface coupling. It requires connectivity data from the input mesh (or output mesh in a conservative mapping): vertices must be connected via edges, and in 3D, these edges must be connected to form triangles. Then, in the case of a consistent mapping, for each output vertex, we project it into the nearest triangle or edge and interpolate inside that element. The projection (illustrated on figure 2.4) is required because the output vertex is *a priori* not on the surface of the input mesh, usually because of discretization. Interpolation errors come both from the projection and the linear interpolation. It has been shown[5] to be first-order accurate in the projection distance and second order-accurate on the surface. This means the linear interpolation itself is second-order accurate, but the error if the original point is not on the surface is asymptotically proportional to its distance to the target surface.
Linear interpolation is done through *barycentric coefficients*: the coordinates of a point inside a triangle (or edge) are an affine combination of the coordinates of the vertices of the triangle (or edge), and these same coefficients are used to interpolate by using an affine combination of values on the vertices. For instance, the interpolated value at the center of an edge is always the average of the values at the ends of that edge. See appendix A for more details.

The conservative version of this is similar, except that instead of interpolating with an affine combination of points from the nearest triangle/edge, the data to map is distributed to these points with the barycentric coefficients. Physically speaking, this means that if a force is applied in the middle of an edge, it will be spread on both the vertices making that edge, each end of the edge taking half the input force.

[3] In particular, when solving second order PDEs with finite elements, gradients have discontinuities at the vertices, which makes it impossible to have a proper value. An alternative is to use element centers as vertices in the preCICE mesh to get a continuous gradient.

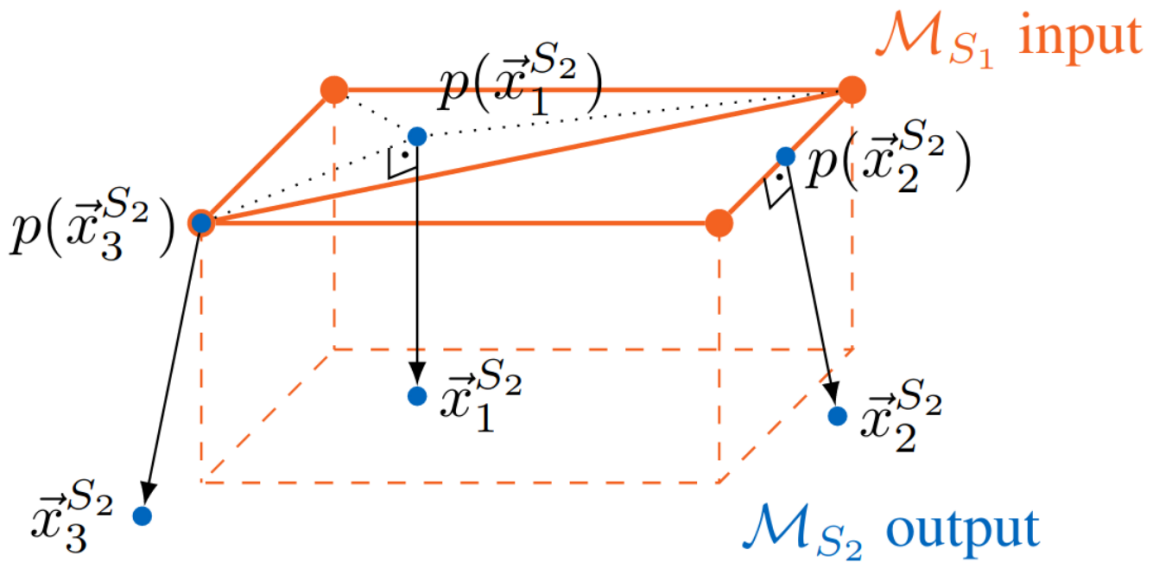


Figure 2.4.: Illustration of nearest-projection in 3D. Points get mapped on triangles, or edges, or vertices. Then, data is linearly interpolated (in case of edge or triangle). From the preCICE v2 reference paper[5].

2.2.5. Radial Basis Functions (RBF) interpolation

Another approach is to build an interpolant that is made of a linear combination of radial basis functions centered on the input vertices. If ϕ is the radial basis function and $\mathbf{x}_i$ the position of the i-th vertex, the interpolant can be written as $u(\mathbf{x}) = \sum_{i=1}^{n} \lambda_n \phi(\|\mathbf{x} - \mathbf{x}_i\|)$ and the output values are computed by sampling this function. Getting the coefficients requires solving a linear system each time data has to be mapped. Parallel efficiency depends on the choice of basis function (with local or global support), as local function yield a sparse system than is easier to solve in parallel. More details, such as the conservative version, the possibility of adding a polynomial term and implementation details can be found in [15].

Overall, this mapping is the most expansive (due to the linear system to solve) and requires careful choice of parameters (function type, shape parameter), but has proven to be very accurate (usually better than second order). Furthermore, it does not require any additional data such as connectivity or gradients.

2.3. Applications to volumetric coupling

The preCICE library was designed with surface coupling in mind, and not all the available mappings are suitable for volumetric coupling. Nearest-projection mapping is not optimal as it was designed assuming surface coupling: preCICE assumes the mesh is a surface (or a curve, in the case of 2D simulations) and tries to project the output mesh into that surface. However, in volumetric coupling, both meshes represent the same volume and no projection is conceptually needed. RBF interpolation does work, but can become extremely expensive when the number of vertices grows, especially when using functions with global support. In the context of volumetric coupling, the number of vertices is expected to be significantly higher than with surface coupling, which can be problematic.
Until now, researchers using preCICE for volumetric coupling used nearest-neighbor mapping, as it was the only usable case. Due to its inaccuracy, volumetric coupling was only doable with reasonable accuracy when both meshes were matching. In theory, nearest-neighbor with gradient should be accurate and generalize easily to volumetric coupling. However, it is a recent addition to preCICE and has not been used in real applications yet. Furthermore, conservative mappings are not supported with this method[4].

The main work of this thesis is to implement a new second-order accurate mapping method tailored for volumetric coupling. It is based on barycentric interpolation, like the nearest-projection mapping, but does not require projection.

[4]When a data mapping consists in a linear map from the input data to the output data, transposing a consistent mapping yields a conservative one. However, nearest-neighbor with gradient uses a linear combination of data *and* derivatives, which can't be transposed. For more details, see [2].

All the methods implemented have different advantages and disadvantages. Nearest-neighbor mapping is the cheapest and simplest, as it does not require additional information or parameters, but the least accurate. Accuracy can be cheaply improved using additional information such as mesh connectivity or gradient data. Another way to increase accuracy is to use RBF interpolation: it is the most accurate method and does not require additional information, but is significantly more expensive and require tweaking some parameters: the user must choose the radial function, a shape parameter and, with compact functions, a support radius. A general overview of these mappings in the consistent case is shown in table 2.1. On this table, we note h the mesh resolution (distance between two neighboring vertices) and δ the distance between a point and its projection, if there is one.

Mapping	Accuracy	Additional requirements	Volumetric coupling?
Nearest-Neighbor	$\mathcal{O}(\|h\|)$	None	Yes
NN + Gradient	$\mathcal{O}(\|h\|^2)$	Gradient data	Yes
Nearest-Projection	$\mathcal{O}(\|h\|^2 + \|\delta\|)$	Mesh connectivity	No[5]
RBF	Varying	RBF choice, linear solver	Too expensive

Table 2.1.: Comparison of available methods in the case of a consistent mapping.

2.4. Parallelization

The preCICE library is designed for high parallelization efficiency[22] and to run efficiently on clusters and supercomputers. We call *intra-solver parallelization* the usage of parallel solvers as participants in a preCICE simulation, in contrast to *inter-solver parallelization*, which is the parallel execution of two or more participants using a parallel coupling scheme. Intra-solver parallelization is achieved using domain decomposition: each process must solve a portion of the problem (i.e. compute values in a subset of the mesh) and synchronize with other processes. Each solver is responsible for partitioning its mesh and ensuring internal synchronization. When preCICE is involved, each process calls the library, and preCICE handles the rest: the solver does not need to be aware of the decomposition of the other participant(s), nor does it even need to know if they run in parallel.
The parallelization approach is decentralized: there is no preCICE 'server' coordinating everything, and each rank of process A only communicates with the processes of participant B that need information from that particular rank.

2.4.1. Parallelization of data mapping

As preCICE is designed for fully parallel simulations, all mapping methods should be designed to be easily parallelizable. To reduce unnecessary communications, each process must receive only the data it needs. For instance, when doing a consistent nearest-neighbor mapping, each output vertex needs to know the value of its corresponding (nearest) input vertex. If the writing participant is also parallel, only one process is writing data for each input vertex. Overall, each writing process will need to send data to some reading process, and each reading process will get data from a few writing processes. With preCICE, the required communications are computed dynamically at simulation setup, which requires temporary constructing the full mesh (by adding the meshes of all subprocesses). Typically, each reading process has to tag all the input vertices it needs. Then, this information is sent back to the writing participant, which then knows what to communicate to whom. A more complete description (taking into account the case of conservative mappings, read and write mappings, global RBFs, ...) can be found in [22]. For this thesis, the main point is that it is the responsibility of a data mapping method to "tell" preCICE what subset of the data is required on a given parallel participant.

2.4.2. Best practices for handling distributed meshes

In a typical parallelized simulation code, each process owns a fraction of the mesh, and has to communicate data (internally) to read necessary data from other ranks, and vice versa. The vertices with communicated data are called *shared vertices*. When using preCICE, vertices must be set up to configure the coupling, but there are different ways to handle shared vertices[1]:

- One can decide to only communicate through *owned* vertices. This makes writing easy, but requires synchronization of read data.
- If shared vertices are communicated too (i.e. they are duplicated in the preCICE mesh), no synchronization is required for reading, but communication is required for writing data[6].
- It is also possible to use a mesh for reading and another for reading, eliminating the need for synchronization.

2.4.3. Additional constraints for connectivity-based mappings

While all these approaches are correct in general, the second one is the best when using connectivity-based mappings: since preCICE only allows the user to set elements connecting vertices from the same rank, duplication is required to get all elements, otherwise there

[6]For a consistent mapping, all solvers must write the same value; with a conservative mapping, one solver must write the value and the others must write zero, otherwise the conserved quantity would be duplicated

will be “holes” close to the border between two ranks. If duplication does not occur, a loss of accuracy will occur around the border between ranks. Figure 2.5 illustrates what can happen without duplication.

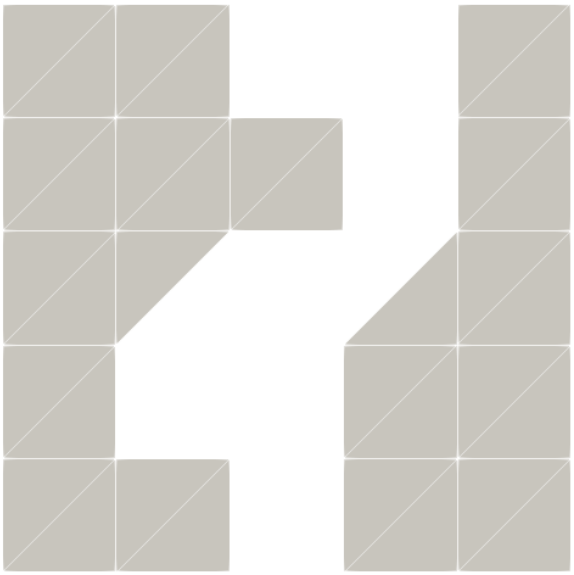


Figure 2.5.: Example of missing connectivity: a triangulation of the unit square with 2 ranks. When no rank owns the 3 vertices of a triangle, this triangle is discarded. This occurs if no duplication is implemented.

2.5. Testing infrastructure of preCICE

preCICE is not an easy library to test[21], as it requires different participants working together and calling the library. Thus, there are always at least two different processes calling preCICE, and when an error happens, finding the responsible is not always trivial. Unit tests (through the Boost library) help checking the correct behavior of individual parts of the library, but integration tests are more complex to write. To tackle this issue, the testing framework has been extended to make the configuration of tests with several participants easier. When defining a preCICE integration test, the developer specifies how many participants run and how many ranks (processes) they own. Then, the test must be run with MPI on 4 ranks, and preCICE handles the distribution of ranks to each participant. This enables testing of coupled runs through one testing executable.

When developing new data mapping methods, unit tests and integration tests are useful to check correctness of an implementation, but deeper testing is required to assess the accuracy of a mapping. Typically, to find the order of convergence of a mapping, we compute the mapping from a coarse mesh to a fine mesh (with a test function sampled on the coarse mesh), then refine the coarse mesh and see how the error decreases. Doing this requires setting up a fake simulation and is time-consuming. To ease this task, preCICE developers created *ASTE* (Artificial Solver Testing Environment), a tool that provides utilities useful for testing mappings: sampling a function on a mesh, map from a mesh to another, compute the difference between the mapped data and the initial function, ... It also offers a *replay mode*, which allows ASTE to mimic a solver by reproducing results from a previous run. For instance, if a user is debugging a coupled simulation between solver A and solver B and thinks B has a bug, he can use ASTE to reproduce the output of A without running it again, which allows him to focus on B.

Finally, a set of test cases (called *tutorials*[7]) is maintained by developers to ensure the correct real-life behavior of preCICE. These cases are actual coupled simulations, involving two (or more) coupled participants. They allow developer to test not only the library itself, but also the various necessary adapters. Some simulations can be run with different solvers (for instance, in a fluid-structure interaction simulation, different solid solvers can be chosen and coupled with the same fluid solver), and similar result increase confidence in the implementation of each adapter. Furthermore, these tutorial cases are often reused as base case by preCICE users.

[7] Publicly available on https://github.com/precice/tutorials.

3. Theoretical aspects

3.1. General concept

The main task of this thesis is the implementation of an alternative to the nearest-projection mapping that can be used in the context of volumetric coupling. This mapping, called *linear cell interpolation* uses connectivity data to build a piecewise linear interpolant (in the consistent case) of the input data. The main difference with nearest-projection mapping is the absence of projection: the interpolant is built on the whole domain instead of being a function lying on the boundary. This means that primitive of a higher dimension must be used: in 2D simulations with surface coupling, the boundary is a curve, represented by vertices connected by edges. In the volumetric case, the whole domain is made of triangles covering the whole computation domain. Similarly, in 3D simulations, surface coupling uses a surface mesh made of triangles, whereas volumetric coupling needs a set of tetrahedra to cover the whole domain.
For points that are not inside a triangle or a tetrahedron, we fall back to nearest-projection. Indeed, the most likely case where this occurs (assuming correct configuration) is the one where the point is close to the domain boundary and both participants have a different resolution for its discretization. Then, the best approximation is to assume the point should be on the boundary.

3.2. Linear interpolation on triangles and tetrahedra

3.2.1. Linear interpolation

When data is available on the 3 vertices of a triangle[1], or the 4 vertices of a tetrahedron, there is a unique linear polynomial that matches these values. If we note by x_i, y_i, z_i the position of the i-th point of the triangle or tetrahedron, and f_i the corresponding value, then:

- In 2D, there is a unique function $f(x, y) = ax + by + c$ such that $f(x_i, y_i) = f_i$ for $i = 1, 2, 3$.
- In 3D, there is a unique function $f(x, y) = ax + by + cz + d$ such that $f(x_i, y_i) = f_i$ for $i = 1, 2, 3, 4$.

[1] Assuming the triangle lies in 2D space, which is the case when doing volumetric coupling. Additional care is needed for a triangle living in 3D space, which happens with nearest-projection mapping.

Solving these equations for a, b, c and d yields coefficients for a linear function over the whole triangle or tetrahedron, depending on the values at the vertices. In practice, we do not want to get the full function for a specific set of values, but rather to compute efficiently the value inside a given point for many input values. Indeed, the data changes over time when performing a simulation but the mesh stays. For this reason, instead of solving for coefficients, we compute *barycentric coordinates*, which represent the linear combination to apply at a given position. For instance, in the center of a triangle, linear interpolation always outputs the average of the 3 nodal values. From a programmer's perspective, it is enough to store that this point takes the value $\frac{f_1+f_2+f_3}{3}$ at each time step. The detailed computations are provided in the appendix A.

3.2.2. Accuracy

By construction, this mapping is exact on linear polynomials. Furthermore, it is a linear map: mapping a linear combination of input functions yields the same result as mapping each input function and then taking the same linear combination. Let f be a smooth enough function, then a Taylor expansion yields $f(\mathbf{x}+\mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot \mathbf{h} + \mathcal{O}(\|\mathbf{h}\|^2)$. The error from linear interpolation comes from the $\mathcal{O}(\|\mathbf{h}\|^2)$ term, as the others do not introduce any error. Thus, the mapping is second order, as the point-wise error is bounded above by the squared distance between the reference point and the target point, which itself is bounded above by the element size.

3.3. Efficient spatial queries on a mesh with R-trees

When implementing data mapping methods relying on connectivity data (i.e. not only on vertices positions, but also on higher dimensional objects like edges or triangles), one often has to answer questions such "what edge is the closest to a given point?", "what triangle (if any) contains a given point?" or "what vertex is the closest to a given point?". All these problems can be easily but inefficiently solved through naive search. For instance, to find the triangle(s) containing a point, it is sufficient to iterate over all triangles and check if the current one contains the requested point. The time complexity of this operation is linear[2]: the number of operations to apply (in the worst case as well as in average) is proportional to the number of triangles. Is there a more efficient way to solve this family of problems?

These problems (relying on *spatial indexing*) are quite general and have numerous applications, such as finding restaurants near one's position or rendering only the necessary objects in a computer game or a computer-generated movie. It has been widely studied and many algorithms exist to solve them. With preCICE, R-trees have been chosen. Detailed description has been made by [12], but a brief introduction is given below to help the reader understand the general idea behind this method.

[2] Linear *per query point*. When both meshes are big, this quickly gets expensive.

3.3.1. Finding a triangle containing a query point X

Let us consider a point X and a set of triangles living in 2D space. Assuming they are not overlapping, there is at most one of them containing X. Can we find it using a more efficient method that naive iteration over all triangles? With naive search, each check only eliminates one triangle from the list of candidates. A better method would be on where a single check can eliminate more than one triangle. A simple but crucial property of this problem is that **if X lies inside a triangle, it also lies inside any region of space containing that triangle**. Conversely, **if there is a region of space that contains a given triangle but does not contain X, then X is not in that triangle**.
A good choice of region would be a region such that:

- Checking if X is inside that region is cheap.
- This region contains many triangles.
- This region should be small: if X is not inside one the contained triangles, it should have a "high probability" of not being in the region.

Clearly, these objectives aren't easily fulfilled simultaneously: the whole 2D space respects the first and second condition but not the last one, whereas the union of some triangles is too costly to check. Some compromises have to be made.

Introducing axis-aligned bounding boxes

A cheap way to eliminate triangles is to use axis-aligned bounding boxes (AABB), which are rectangles (when working in 2D) whose sides are aligned with axes. They can be easily defined by the position of the lower left corner $(x_{\min}, y_{\min})$ and top right corner $(x_{\max}, y_{\max})$. Then, one can easily check whether a point lies inside the box by looking at its coordinates: if X as coordinates (x, y) then it is in the box if and only iff $x_{\min} \leq x \leq x_{\max}$ and $y_{\min} \leq y \leq y_{\max}$.
For a triangle (or any shape), we call *minimum bounding rectangle (MBR)* the smallest AABB that contains it. For polygons, the minimum bounding rectangle can be easily computed by making sure it contains all the vertices. Checking if a point lies in an AABB is cheaper than checking if it lies in a triangle. For this reason, checking first with MBR can avoid expensive computations. A good approach is thus to find all triangles whose MBR contains the query point, and the iterate over them to do the actual triangle check.

Divide and conquer approach

Instead of looking at the MBR of a triangle, one can look at the MBR of a set of triangles (the AABB containing all of them). Then, any point **not** inside this AABB **can not be** inside any of the contained triangles. This powerful idea can eliminate many triangles at once and make search faster. For instance, if we consider a set of N triangles (with N a

large number) and make two groups of $\frac{N}{2}$ triangles, checking if the query point X could eliminate half of the candidates with an unique check, if it is not in one of the two MBRs. Repeating this recursively with a tree structure could yield a logarithmic complexity, significantly better than the linear complexity with a naive search. However, constructing this is in an efficient way is not trivial at all: separating triangles in two arbitrary groups and computing their respective MBRs is very likely to result in overlapping rectangles, which could result in a query point being in both of them. In that case, no group can be eliminated.

Different approaches have been developed to counter this problem. One is to partition space such that the absence of overlap is guaranteed. For instance, an AABB can be split into 4 smaller, similar AABBs with halved dimensions. Doing this[3] recursively leads to quadtrees[8]. However, balance is not guaranteed, as all objects could lie in the same subrectangle.

Another approach, is to use R-trees[12], a tree data structure where indexed objects (triangles with an AABB) are leaf nodes, and where an intermediate nodes contains the AABB enclosing all the child's AABBs. Special care is taken to avoid boxes that are too large (when inserting a node, we try to give it the parent whose MBR would be the less changed) and keeping the tree balanced. While theoretical performance of R-trees in the worst case is disappointing (linear complexity), they perform well on average and are widely used. Many variants exist, such as the R*-tree[4] (used in preCICE) which is slightly slower to build but produces a tree with less overlap, resulting in faster queries.

Figure 3.1 illustrates the connection between bounding boxes and the R-Tree structuring them in a recursive pattern.

[3] This method is usually chosen when indexing vertices and not AABBs, which can cross the boundaries between subboxes.

3.3.2. Finding the nearest vertex to a point X

When implementing nearest-neighbor mapping, one must be able to find the nearest vertex for any input point. Once again, R-trees can be used[13]. Bounding boxes are not necessary for indexing vertices, but group of vertices have a MBR too, which can be used to build a R-tree. Given a point X and a AABB, one can easily compute a lower bound and an upper bound of the distance between any point inside the AABB and X. This can be used to prune branches when traversing a tree.

Let N be the node in a tree where the nearest neighbor of X is searched. For each child node of N one can compute a lower and an upper bound. If the minimum distance in child node C_1 is bigger than the maximum distance in child node C_2, then clearly the nearest vertex is not inside C_1 and there is no reason to look deeper in its own children. An example is shown on figure 3.2, where the nearest neighbor of X is searched for a given R-Tree structure.

3.3.3. Usage in preCICE

The preCICE library already uses R-Tree in nearest-neighbor and nearest-projection mappings. It relies on the Boost library[19] (already used for other aspects of preCICE such as its testing framework) to build the R-Trees and perform queries. Boost provides a very generic implementation that can efficiently be adapted to preCICE objects (vertices, edges, triangles). As tetrahedra are introduced in this thesis, they have to be made compatible with Boost (i.e. we have to tell Boost how to use our tetrahedra type, using C++ templates). Once it is done, R-Tree construction and queries are handled by the library.

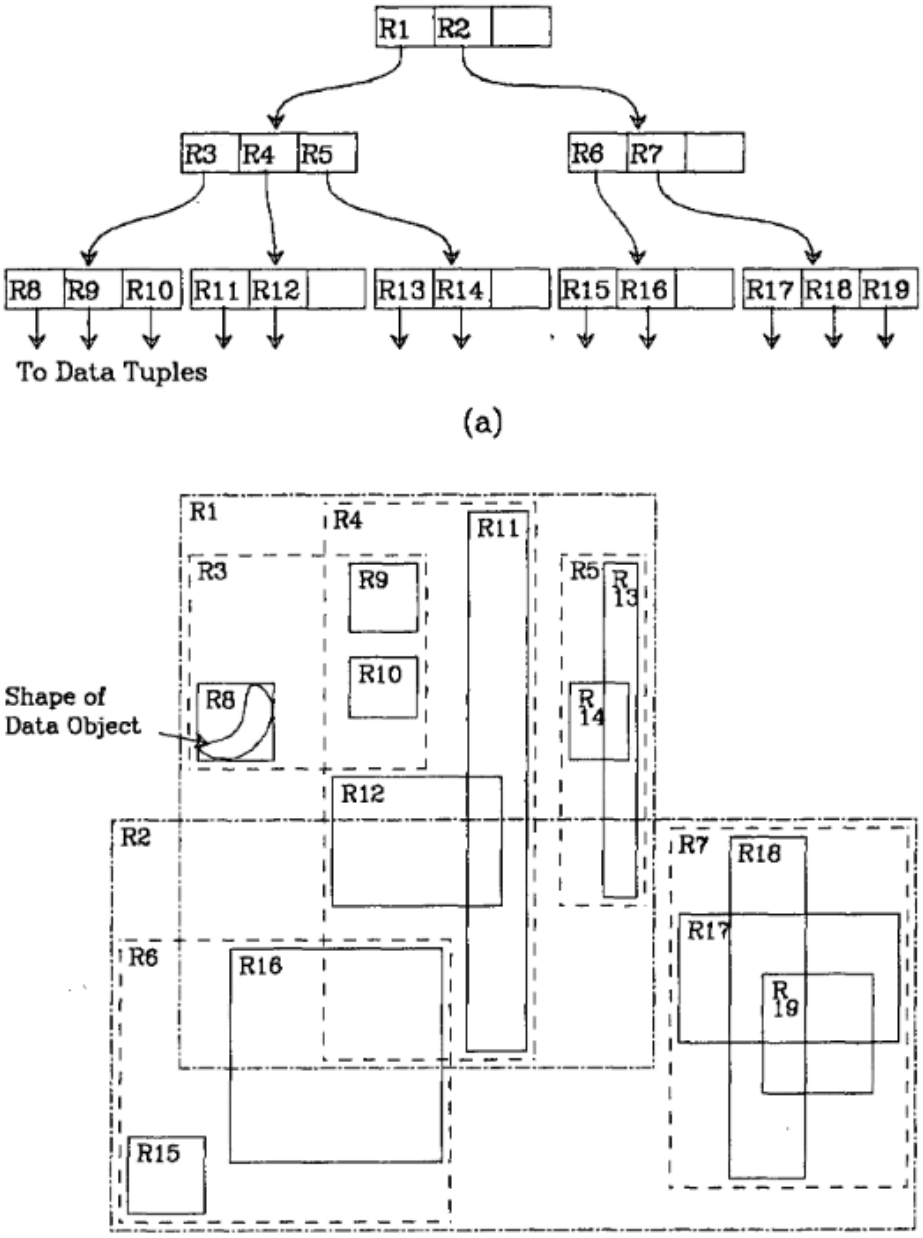


Figure 3.1.: Illustration of a R-Tree with the corresponding boxes. From the original R-Tree paper.[12]

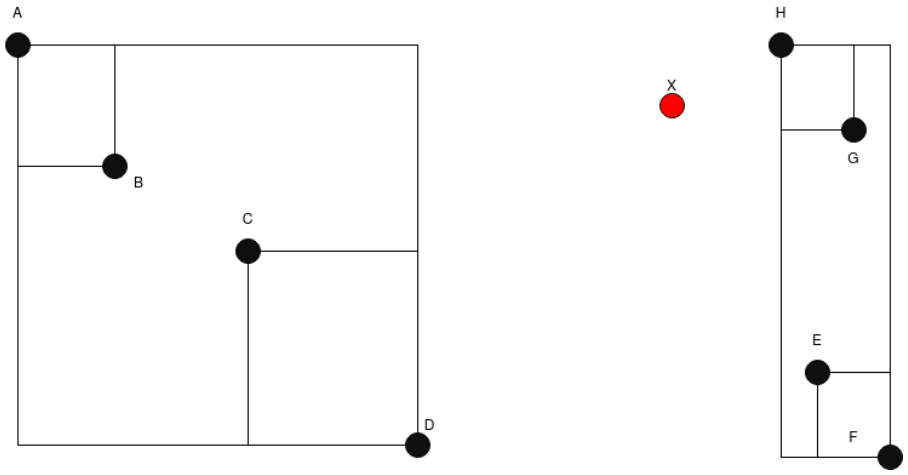


Figure 3.2.: Example tree (root box omitted). Clearly the max distance of the box on the right is bigger than the min distance of the left box, so points A, B, C, D can be discarded.

3.4. Interpretation of linear cell interpolation with the Finite Element Method

Although preCICE does not need to know what kind of discretization (finite differences, finite volumes, finie elements, ...) the participants use, the linear cell interpolation mapping can have a particular meaning when used with some finite element codes.

In finite elements, the discretized solution is fundamentally not a set of point-wise values, but a function in a function space of finite dimension. This function space is a subspace of the (infinite dimensional) function space where the PDE is solved. The subspace used is a function of the mesh and the element type used.

The most common family of finite elements is the family of *Lagrange elements*, where the solution is defined piecewise as a polynomial and degrees of freedom match values at vertices. In the specific case of first order Lagrange element, the function is piecewise linear on a triangle (in 2D) or a tetrahedron (in 3D). This is exactly the same approach as the one used in the linear cell interpolation mapping implemented in this thesis. An interesting consequence is that *if first order Lagrange elements are used, then (consistent) linear cell interpolation is exact*[4]: the linear interpolation used by the mapping is such that the value mapped at output vertices will be the actual value that the solver writing data considers to be correct in its internal discretization.

[4] Assuming connectivity is configured accordingly.

4. Implementation

4.1. Overview

Linear cell interpolation is fundamentally similar to nearest-projection. In terms of C++ code, both can derive from a same base class to reuse the common parts. The steps for a consistent mapping are as follows:

- At initialization, compute the mapping: for each output vertex, find than the corresponding nearest element. Then, compute the barycentric coefficients of the vertex[1] with respect to the vertices of the element and store them.
- Each time the mapping must be computed, assign to each output vertex the affine combination of values from the input mesh with the stored coefficients.

In the conservative case, the input data must be distributed between vertices of the output mesh. The process is transposed: for each input vertex, we find the nearest element of the output mesh and compute barycentric coefficients. However, instead of using these coefficients to compute an affine combination, we use them to express what fraction of the data must be attributed to each output vertex. Note that with a consistent mapping, the input mesh requires connectivity, whereas the output mesh needs it for a conservative mapping. To generalize algorithm description, we call *search space* the one with connectivity and *origin space* the other one.

The difference between linear cell interpolation and nearest-projection lies in the part "find the corresponding nearest element", as different geometric elements are looked for. Furthermore, the computation of barycentric coordinates is different for each type of element. These computations already exist for edges and triangles, but computation of barycentric coordinates for tetrahedra must be added. Furthermore, in the case of triangles, which are used both in 3D simulations with surface coupling and 2D simulations with volumetric coupling, a different algorithm can be used depending on the case, since no projection is needed in 2D.
The different steps necessary to implement this new mapping method are:

- Change preCICE to make it possible to add triangles in 2D. Previously, it was only allowed in 3D

[1] In nearest-projection, we compute the barycentric coefficients of the projection.

- Add the new routines for computing barycentric coordinates.
- Add tetrahedra to preCICE, along with the required API to configure them.
- Add the mapping and update preCICE API to make it possible to use it.
- Write tests for all the above.
- Updating adapters to configure the appropriate connectivity when required.

4.2. Changes related to triangulated 2D meshes

The main change to preCICE was to allow triangles in 2D simulations. Since they already existed in 3D, the underlying architecture existed already. It was required to remove the dimension check in the API (when a user tries to add a triangle in a 2D simulation, preCICE used to throw an error) but also in many other parts such as the mesh communication code, where triangles were communicated only in 3D case. To ensure all the necessary checks were removed, new tests were added. In particular, correct communication behavior was checked through integration tests using linear cell interpolation in different contexts. Participants were either serial or parallel (testing of parallelization used one-to-many and many-to-one configurations), mappings were consistent or conservative, and in serial tests, both *write* and *read* directions were tested[2]. More details about tests can be found in section 4.5.

The next major step was to refactor the nearest-projection mapping to give it a common base class, from which the linear cell interpolation would inherit too. Almost everything is common, except the choice of element on which to interpolate. In terms of code, the method `computeMapping()` must be specialized while the others (most importantly, `mapConsistent()` and `mapConservative()`) are shared. This refactoring helps reduce code duplication and limits the amount of new code to write and test.

To map the mapping (i.e. find the matching element), we rely on existing code once again. For each vertex of the origin space, we look for a triangle containing it. If there isn't, we fall back on nearest-projection as explained in section 3.1. The required code for finding the nearest triangles already existed (from 3D nearest-projection) and it was sufficient to reuse it here.

Finally, a minor change had to be applied to the computation of the barycentric coordinates of a point with respect to a triangle. This function already existed but assumed 3D vectors. The new 2D version (detailed in appendix A) is mathematically similar but works with 2D vectors.

[2] In parallel, only one direction is allowed.

4.3. Changes related to 3D meshes with tetrahedra

Extending the linear cell interpolation to 3D is fundamentally similar but required the use of tetrahedra in the mesh, which did not exist in preCICE until now. Adding these was the first step: the user must be able to set them (from 4 vertices), they must be correctly communicated between participants and properly handled when meshes are repartitioned in parallel runs. Another set of tests were added to check that all these steps were working as intended, and integration tests were crucial to identify missing components[3].

Once tetrahedra are set up on a mesh, the next key task is to find efficiently in which one lies a given vertex from the other participant. To this end, an R-Tree is built (through the Boost library, already used for R-Trees of vertices, edges and triangles) and used to find tetrahedra whose axis-aligned bounding box contains the queried point. Then, iterating over the matches allows us to find which tetrahedron actually contains the point.

Once finding the tetrahedron containing a vertex is doable, the linear cell mapping can be extended to 3D: for each vertex, we find the[4] containing tetrahedron and compute coefficients from it. If no tetrahedron is found, we fall back on nearest-projection again.

4.4. Further changes

Along the linear cell interpolation mapping, a few improvements to preCICE related to volumetric coupling were made:

- When configuring a simulation to export a preCICE mesh as VTK or VTU files, tetrahedra are included. This is a natural consequence of the addition of tetrahedra to the library which helps debugging mesh configuration.
- Watchpoints, where preCICE records the evolution of data on a given point, now interpolate using volumetric primitive if they can. Previously, they interpolated on surface primitives, similarly to nearest-projection mapping. This makes watchpoints inside the domain more accurate, whereas they were initially designed to monitor values at the boundary.
- Scaled-consistent mappings can now be configured[5] to preserve volume integrals instead of surface integrals.

[3] Changes had to be made for communicating between participants, between ranks of a participant, when filtering a mesh to keep only the relevant subset of points, ...

[4] There could be more than one if there is overlap, but it is the responsibility of the user to make sure it does not happen.

[5] By configuring a *'scaled-consistent-volume'* instead of 'scaled-consistent' mapping in the XML configuration file.

All these changes are generalizations of features that already existed but were designed with surface coupling in mind.

4.5. Testing

In software development, proper testing is critical to ensure correct behavior of the code. Automating tests allows developers to make incremental changes without the need to test the code manually (which can be very time-consuming) and with increased confidence they did not break already existing features.
When adding new features, it is a good practice to test them early to spot bugs and edge cases. When implementing linear cell interpolation mapping, tests were written early to cover as many cases as possible. Most of them helped find an oversight or an unexpected edge case.

4.5.1. Unit tests

Unit tests are meant to check the correct behavior of a single unit, usually a function or a method of a class. In this thesis, most new or modified functions were tested this way. A unit test is usually small: one function is called (after the input arguments are set up), and its output is checked. Some examples of tests are the computation of barycentric coordinates of a point with respect to a tetrahedron, simple queries on a small R-Tree and exporting a mesh containing tetrahedra as a VTK file.[6]
The most important tests were those testing the linear cell interpolation mapping function. The following paragraphs describe tests written to check the correct behavior of the mapping. Since they are unit tests, the mapping is tested in isolation: both input and output meshes are provided directly in the test, which allows us to test the mapping without wondering about mesh communication between participants.

On the next page, a sample unit test[7] is shown for illustration. It checks the correct computation of barycentric coordinates in the case where the query point is in the middle of a 2D triangle. Each test is defined by a `BOOST_AUTO_TEST_CASE` macro (from the Boost testing library) and starts with a `PRECICE_TEST` macro that sets up the context and handle parallelization if necessary. Then, computations are done and results are checked through `BOOST_TEST`. Many cases are tested (not only the center, but also points on edges, in the inside but with an asymmetric distribution, on the outside, ...) to ensure that basic requirements are met.

[6] In that case, the VTK file is stored and then checked manually.
[7] The actual test is longer and has been simplified for conciseness.

```
BOOST_AUTO_TEST_CASE(BarycenterTriangle2D)
{
  PRECICE_TEST(1_rank);
  using Eigen::Vector2d;
  using Eigen::Vector3d;
  using precice::testing::equals;
  Vector2d a(0.0, 0.0);
  Vector2d b(0.0, 1.0);
  Vector2d c(1.0, 0.0);

  {
    Vector2d center = (a + b + c) / 3;
    Vector3d coords(1.0 / 3.0, 1.0 / 3.0, 1.0 / 3.0);
    auto     ret = calcBarycentricCoordsForTriangle(a, b, c, center);
    BOOST_TEST(ret.sum() == 1.0);
    BOOST_TEST(equals(ret, coords));
  }

}
```

Consistent mapping in 2D

A single triangle filling half the unit square (coordinates $(0,0), (1,0)$ and $(0,1)$) is created as input mesh, along with the relevant edges. Each vertex gets a different value. We check that the mapping produces the expected result on different points:

- The triangle center must take the average value of the 3 data points.
- Evaluating the mapping at a vertex should yield the exact value of this point.
- When a point is outside the triangle but close to an edge, linear interpolation of its projection on this edge should occur.
- Points that are outside the triangle and can't be projected on an edge should take the value of the nearest vertex.[8]

Conservative mapping in 2D

For this test, the same triangle as before is created, but this time as an output mesh. Different points (which can be interpreted as the position where some forces are applied) are

[8] This is actually not true now, as the behavior of nearest-projection is incorrect in some cases. A fix is under investigation but has not been approved yet at the time of submission of this thesis. See https://github.com/precice/precice/issues/1304.

used as input mesh. We check that each point of the triangle receives the correct share of each force:

- A load in the center should be spread evenly.
- When the position matches a vertex, it should receive the entire load.
- Applying a force along an edge should spread it between the 2 corresponding vertices, with a ratio depending linearly on its position along the edge.

Furthermore, the overall force (the sum of all data) should be conserved.

Consistent mapping in 3D

To test the 3D mapping, we proceed similarly with one tetrahedron. To have a detailed testing of the fall-back mechanism, only one of the 4 relevant triangles is added, as well as one of the 6 edges. The following cases are checked:

- The center takes the average value
- All vertices take their exact respective value.
- A point close to the center of the configured triangle but outside the tetrahedron gets projected into the triangle.
- A point outside the tetrahedron and close to triangles that weren't set up gets mapped to the nearest vertex.[9]
- A point close to the edge gets projected to it.

Conservative mapping in 3D

Similarly to the previous test, we set up a tetrahedron (without edges nor triangles this time) on the output mesh and apply different loads on it. We check the correct distribution of:

- A load applied in the center
- Loads on a vertex
- Loads along an edge
- Loads on a triangle

Furthermore, the conservation of the sum is checked, as it was in the 2D conservative.

[9] Again, this is not yet true due to the behavior of nearest-projection and will be fixed in the future.

4.5.2. Serial integration tests

Integration test do not test a single part of the library, but check these parts work as expected together. For testing linear cell interpolation, instead of creating two meshes, mapping and checking the result, we set up an actual partitioned simulation with preCICE with two participants. Each participant sets up its mesh and connect to the other one, and the configuration is done using the XML configuration file. These tests are also handled by Boost, and preCICE macros are used to configure the communications between participants, as well as their names. Then, the test defines behavior of both participants. Following is the typical structure of a test to check the behavior of a consistent mapping:

```
BOOST_AUTO_TEST_CASE(OneTriangleRead)
{
  PRECICE_TEST("SolverOne"_on(1_rank), "SolverTwo"_on(1_rank));
  // context object is provided by the macro
  precice::SolverInterface interface(context.name, context.config(),
                                     context.rank, context.size);

  if (context.isNamed("SolverOne")) {
    // Configure mesh
    // Initialize then write data
    double dt = interface.initialize();

    interface.writeBlockScalarData(/*Add arguments*/);

    interface.advance(dt);
    interface.finalize();
  } else { // SolverTwo
    // Configure mesh
    // Initialize then read data
    double dt = interface.initialize();

    interface.readBlockScalarData(/*Add arguments*/);

    BOOST_TEST{readData == expectedData};

    interface.advance(dt);
    interface.finalize();
  }
}
```

Consistent mapping in 2D

A 2-participants simulation with serial-explicit, unidirectional coupling is configured, with a unique time window. This means that participant 1 sends data once and reads nothing, while participant 2 waits for this data and performs one step before finalizing. A consistent linear cell interpolation mapping is chosen in the configuration file, either as a *read* or *write* mapping. Participant 1 sets up a triangle (with edges), initializes preCICE and writes some data. On the other side, participant 2 defines some vertices, reads data and checked that the interpolation is correct. Many more tests are performed: is the size of the mesh matching its expected value? Does preCICE correctly tell the solver to finalize after one step?

On the first prototype, this test passed with a *write* mapping but not with a *read* mapping, which was surprising since both options usually give the same result in serial simulations. After investigations, it turned out the problem was that triangles were not communicated correctly, as preCICE used to assume no triangle existed in 2D simulations and only sent vertices and edges. Writing this integration highlighted the need for an update of the communication code of preCICE. Such a problem could not be detected by the unit tests of the mapping alone. This integration test was thus updated to check (in the case of *read* mappings) that participant 2 received the full mesh, including the triangle. This helps to maintain preCICE: if in the future a developer changes the communication code and breaks the communication of triangles, this test would fail with an error saying a triangle is missing, instead of simply reporting a wrong mapping output.

Conservative mapping in 2D

This test is similar to the consistent one, except that (like in the unit tests), the first participant writes data on its own mesh, whereas the second defines a triangle and reads the data. Again, we check that the correct distribution occurs with either a *write* mapping or a *read* mapping. However, if communications were to break, the *read* mapping would be the one working instead of the *write* mapping, as the connectivity is part of the mesh of participant 2.

Consistent and conservative mappings in 3D

Serial integration tests in 3D are quite similar, except that they involve a tetrahedron instead of a triangle. These are the first tests involving the new API function `setMeshTetrahedron`, so particular care is given to its correct behavior. Furthermore, this test also confirms, if it works for both *read* and *write* mappings, that tetrahedra are communicated correctly.

4.5.3. Parallel integration tests

Since preCICE is designed to work in parallel (with potentially each participant having many ranks) it is crucial to test the correct behavior of data mapping in parallel. In particular, handling of tetrahedra in parallel must be tested thoroughly, as many mesh transformations occur. Indeed, if we consider a consistent *read* mapping with both participants using intra-solver parallelization, the following steps occur [10]:

- Primary rank of participant 1 gathers all sub-meshes to create the global mesh. Internal communication is needed, as well as mesh "summation" to combine all parts.
- This global mesh is sent to the primary rank of participant 2; which requires inter-solver communication.
- The primary rank broadcasts the global mesh to all other ranks; which again requires internal communication.
- Each rank *filters* the mesh to keep only the parts needed to compute mapped data on its owned vertices.

All these steps (communication from a rank to another, broadcast, mesh combination and mesh filtering) were adapted to take in account tetrahedra. Writing parallel integration tests helped to ensure all these steps were working as intended with respect to tetrahedra. Debugging these tests was crucial to find the missing components of tetrahedra implementation in preCICE.

While preCICE can run "many-to-many" simulations, parallel tests were split into "one-to-many" and "many-to-one" to isolate problems. They were written in 2D and 3D and in the consistent and conservative variants, which yields a total of 8 test cases. All the consistent mappings are *read* mappings and the conservative mappings are *write* mappings, as only these combinations can be used in parallel[22].

Configuration of parallel tests is similar to the serial case, except that the preCICE macro configures more than one rank. For a one-to-two test:
`PRECICE_TEST("SolverOne"_on(1_rank), "SolverTwo"_on(2_ranks));`.
This macro defines a context object, and the rank number (local to a participant) can be accessed as `context.rank`, and the participant's number of ranks as `context.size`.

Consistent mappings

In 2D, a unit square made of 2 triangles is used as input mesh. A 1-to-2 (1 rank on first participant and 2 on the second one) test checks that all ranks of participant 2 can read the

[10] In a typical case, as some steps are configuration dependent. See [22] for details.

correct data, and a 2-to-1 test ensures that the correct data is written if each rank of the first participant sets up one triangle. In that case, vertices have to be duplicated, and the aggregated input mesh has 2 triangles made of 6 vertices. The interpolant is continuous if duplicated vertices take the same value, which is true in the test but must be enforced by the user (through synchronization) in real applications.

Similar tests are performed in 3D with a unit cube partitioned in 6 tetrahedra. A 3-to-1 test (where each rank of participant 1 owns 2 tetrahedra) performed to check the correct aggregation of tetrahedra, and a 1-to-3 test ensures that different ranks can read correctly in parallel.

Conservative mappings

Once again, a unit square or unit cube is used, but this time as output mesh. A 2-to-1 test in 2D and a 3-to-1 test in 3D help check the correct summation in conservative mappings: the final read value is the sum of the written values by different input ranks. Then, a 1-to-2 and 1-to-3 test (in 2D and 3D respectively) showcase the use of partitioned meshes with connectivity. In this case, it is the user responsibility to compute sums over ranks to get the correct value on duplicated vertices. This behavior is illustrated on figure 4.1. Finally, as for all conservative tests, the total sum must be preserved.

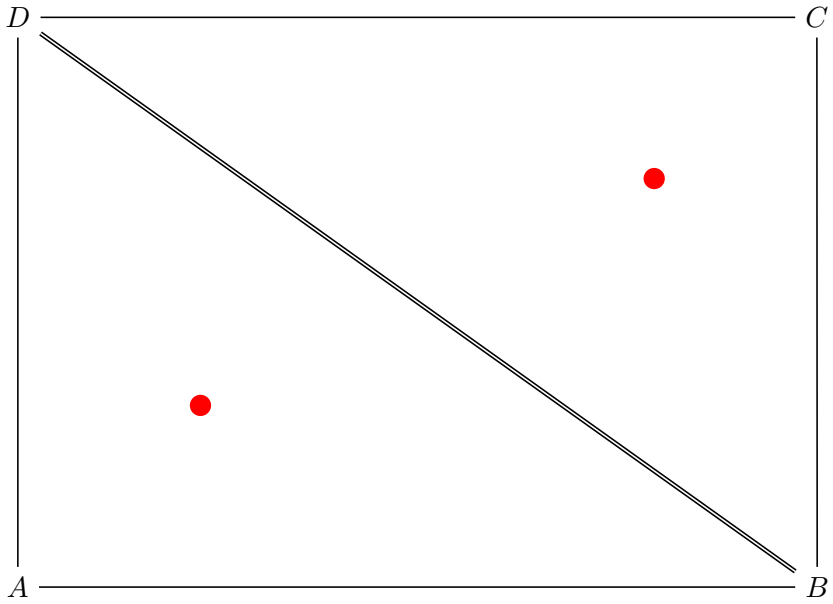


Figure 4.1.: Illustration of a conservative mapping with duplicated vertices. If each red dot represents a vertex from the input mesh, both versions of D and B will receive a part of the conserved data. It is the user (coupled solver) responsibility to use internal communications to get the total values.

5. Accuracy & runtime analysis of linear cell interpolation

5.1. Overview

In theory, linear cell interpolation should be second order accurate, faster than RBF mappings and moderately smaller than nearest neighbor mapping. It is also expected to have similar performance to nearest-neighbor with gradient. To check these hypotheses, tests are conducted using ASTE (which was upgraded to set up tetrahedra if required), a set of scripts designed to test preCICE. Different mappings are tested on meshes with different resolutions. Accuracy, time and memory consumption are measured for all the relevant combinations. In 2D, triangular meshes of the unit square are used, and in 3D, the unit cube is meshed with tetrahedra. These meshes are generated with gmsh[11], using a mesh generation script that was added to ASTE during this thesis.

Experiments were limited to serial runs. In terms of accuracy, parallelization should not provide different results unless connectivity holes appear at the boundary between two ranks (which is the case in the current state of ASTE). In terms of performance, benchmarking has been done for other mappings in [22] and linear cell interpolation should overall behave similarly to nearest-projection.

Only consistent mapping are tested, as they can be studied with common metrics for interpolations. It is harder to quantify the accuracy of a conservative mapping. In practice, methods with a good behavior in consistent mappings yield good conservative mappings.

As expected, linear cell interpolation is exact for a global linear polynomial, in the sense that the mapping error exists only from round off. Let us study the behavior with more complex input.

5.2. Convergence analysis for a smooth function

To study the general convergence, we start with a standard function to test interpolation: the Franke function[9]. It is a 2D function defined on the unit square as a combination of Gaussian with two peaks and a dip. The exact definition[1] is given by equation 5.1:

[1] Taken from the ASTE source code.

$$\begin{aligned} f(x,y) =& 0.75e^{\left(-\frac{(9x-2)^2}{4}-\frac{(9y-2)^2}{4}\right)} + 0.75e^{\left(-\frac{(9x+1)^2}{49}-\frac{9y+1}{10}\right)} \\ &+0.5e^{\left(-\frac{(9x-7)^2}{4}-\frac{(9y-3)^2}{4}\right)} - 0.2e^{\left(-(9x-4)^2-(9y-7)^2\right)}. \end{aligned} \tag{5.1}$$

Since it is a combination of functions in the form $e^{p(x)}$ with p a polynomial, it is infinitely differentiable. Furthermore, it does not have many oscillations. This makes it an idealized test case, good to confirm theoretical orders of convergence but not necessarily representative of real life data.

Another studied function is the Rosenbrock function, defined in equation 5.2. Finally, in 3D, analog functions will be used. The 3D Franke Function is given by equation 5.3 and the 3D Rosenbrock by equation 5.4.

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2, \tag{5.2}$$

$$\begin{aligned} f(x,y,z) =& 0.75e^{\left(-\frac{(9x-2)^2}{4}-\frac{(9y-2)^2}{4}-\frac{(9z-2)^2}{4}\right)} + 0.75e^{\left(-\frac{(9x+1)^2}{49}-\frac{9y+1}{10}-\frac{9z+1}{10}\right)} \\ &+0.5e^{\left(-\frac{(9x-7)^2}{4}-\frac{(9y-3)^2}{4}-\frac{(9y-5)^2}{4}\right)} - 0.2e^{\left(-(9x-4)^2-(9y-7)^2-(9z-5)^2\right)}, \end{aligned} \tag{5.3}$$

$$f(x,y,z) = 100(y-x^2)^2 + (x-1)^2 + 100(z-y^2)^2 + (y-1)^2. \tag{5.4}$$

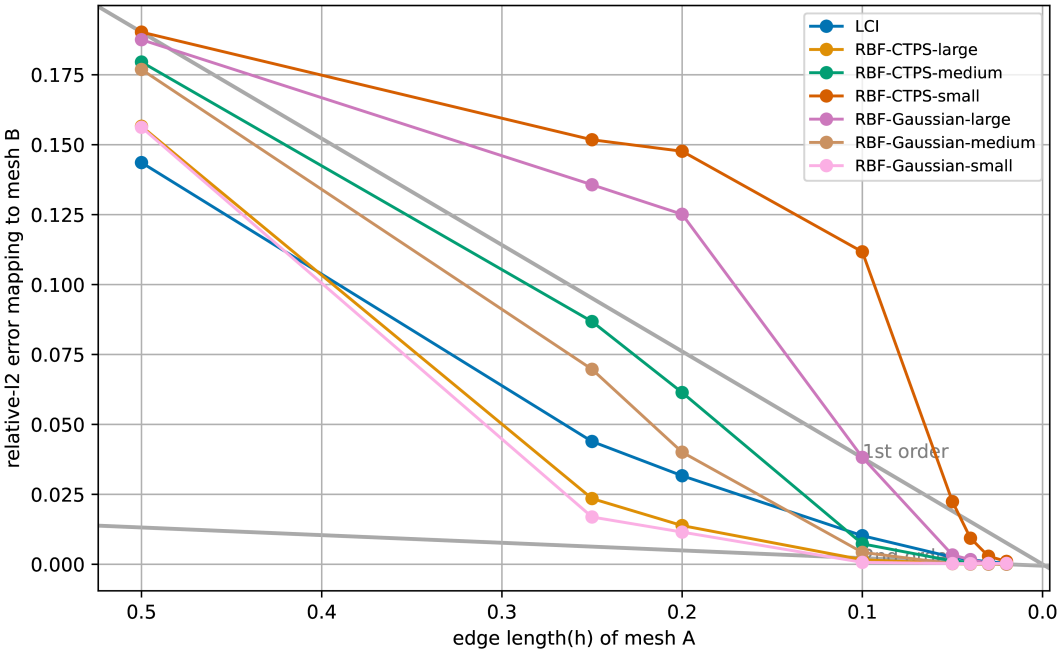
5.2.1. Accuracy compared to RBF mapping

Linear cell interpolation is first compared to some RBF mappings. There are many functions available in preCICE, but for simplicity only Gaussian RBFs (with global support) and compact thin plate splines (compact support) were used. Gaussian were used with 3 shape parameters[2]: $20, 10$ and 5. More extreme values led to divergence for some mesh resolutions. A too large value leads to very sharp RBFs, while a small value leads to almost flat functions, leading to an ill-conditioned system: in the limit case of an almost zero shape parameter, each function is 1 everywhere and the mapping cannot be computed.
Compact thin plate splines were used with support radii of $0.1, 0.3$ and 0.7. This basis function is bell-shaped with a value of 1 at the origin and 0 outside the support radius. It goes from 1 to 0 monotonically in a way that makes the function two times continuously differentiable.
In the following plots, the Gaussian RBFs are labeled 'Gaussian-large', 'Gaussian-medium'

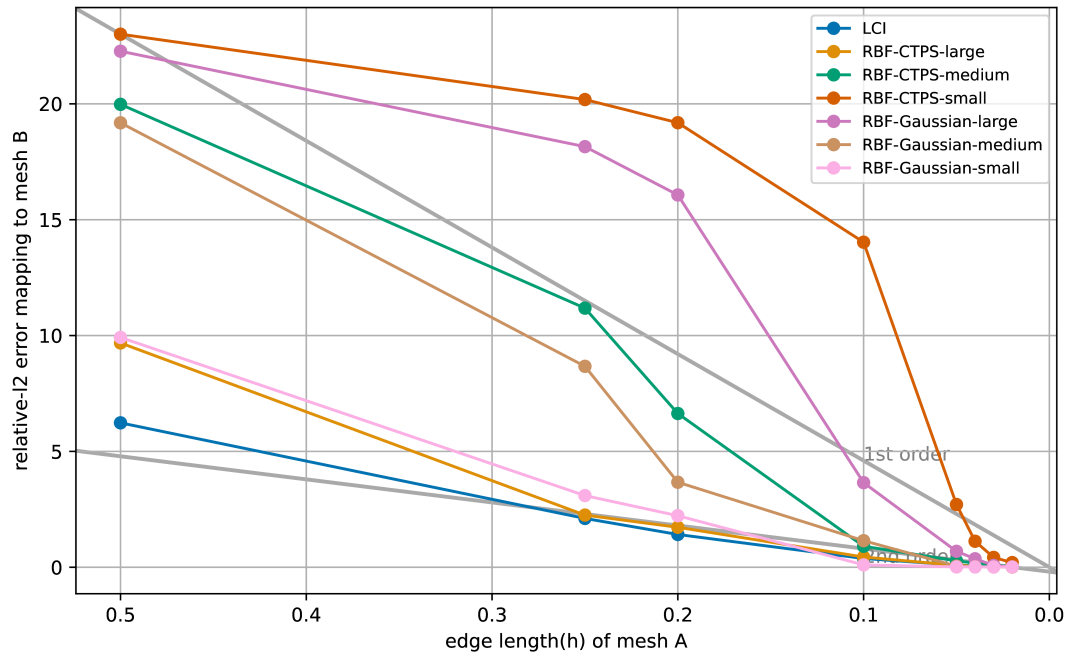
[2]The shape parameter is a in the function $f(r) = e^{-(ar)^2}$.

and 'Gaussian-small' depending on their shape parameters. Similarly, compact thin plate splines are referred as 'CTPS-large', 'CTPS-medium' and 'CTPS-small' according to their support radius. Note that the name can be misleading for the Gaussians: a large shape parameters actually translates as a very sharp peak.
A good choice of shape parameter and/or support radius is crucial to get optimal results. In this sense, linear cell interpolation has the advantage of not requiring such parameters. Figure 5.1 shows the root-mean-square relative error plotted for different mesh resolutions of a unit square. It is a logarithmic plot with reversed x-axis: points on the right represent finer meshes. In log space, a uniform order of convergence will be represented as a straight line[3]: a line for 1st and 2nd order accuracies is drawn for comparison. Similarly, figure 5.2 show the convergence of the 3D versions of these functions on the unit cube.
Overall, we can see that linear cell interpolation has an average performance compared to RBFs. For a given mesh resolution, there is always at least one RBF mapping that outperforms linear cell interpolation, and some that are worse. However, no RBF mapping with a given parametrization is consistently better than linear cell interpolation. This highlights the issue of parameter tuning: a fine-tuning can always provide better accuracy, but linear cell interpolation is robust and gives consistently good results.
The order of convergence of linear cell interpolation is clearly 2 in 2D but more chaotic in 3D. Note that 3D measurements did not use meshes as refined as 2D meshes for performance reasons: a mesh of a given resolution has a higher vertex count in 3D than in 2D. Since the order of convergence is asymptotic for small enough elements, this could explain the chaotic 3D behavior.

[3] If $f(x) = Cx^n$, then $\log f = \log C + n \log x$: all parallel lines represent the same order with different constant factor.

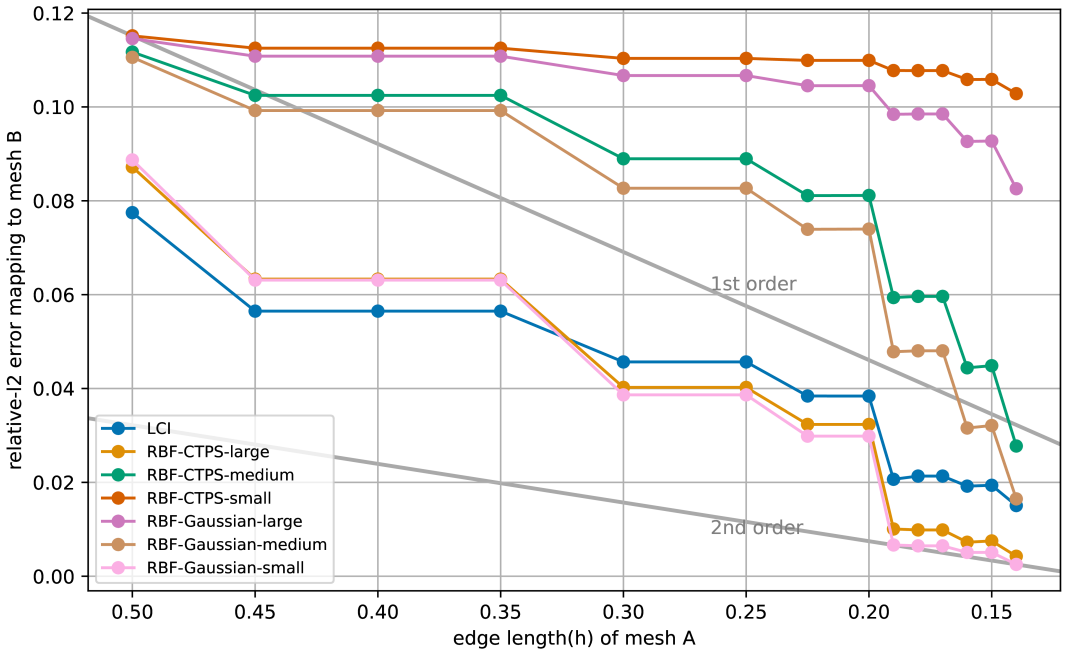


(a) Franke 2D

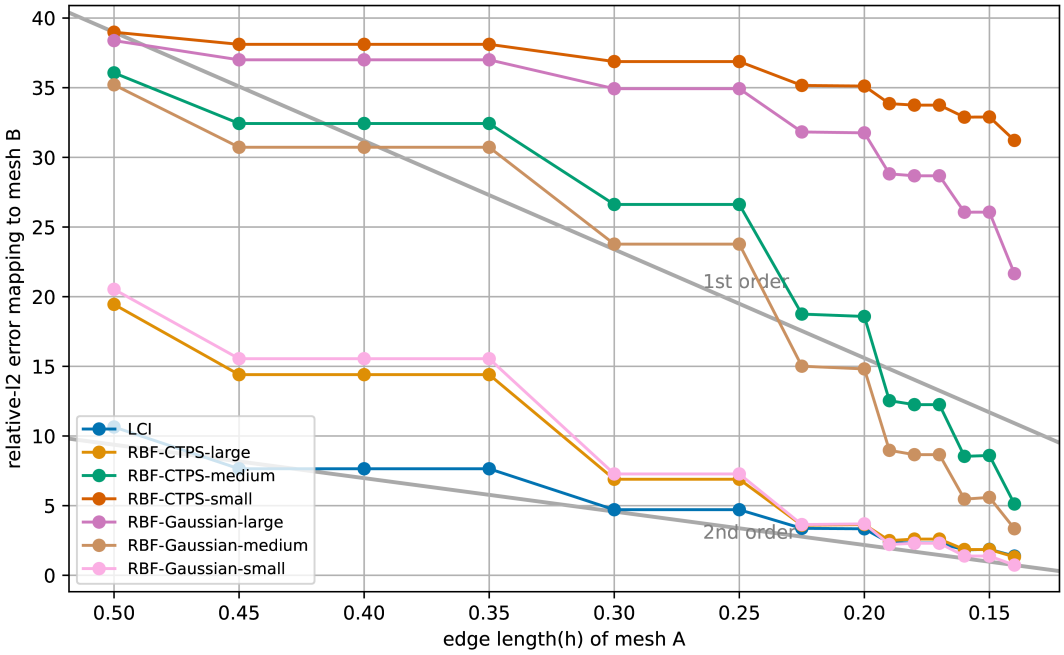


(b) Rosenbrock 2D

Figure 5.1.: Analysis of RBF and LCI behavior in 2D



(a) Franke 3D



(b) Rosenbrock 3D

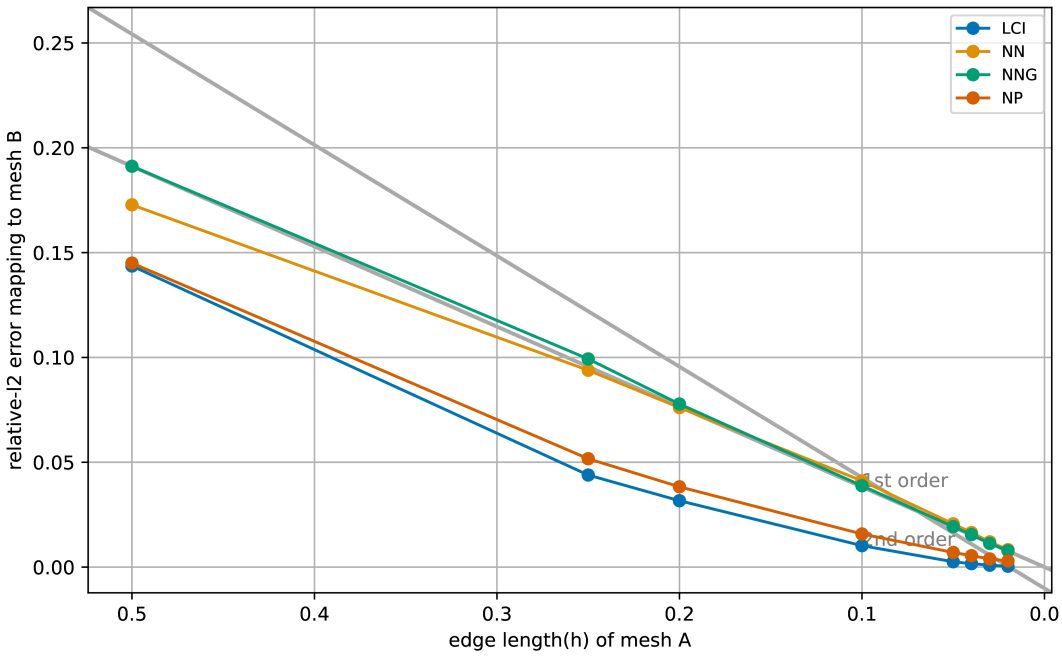
Figure 5.2.: Analysis of RBF and LCI behavior in 3D

5.2.2. Accuracy compared to other mappings

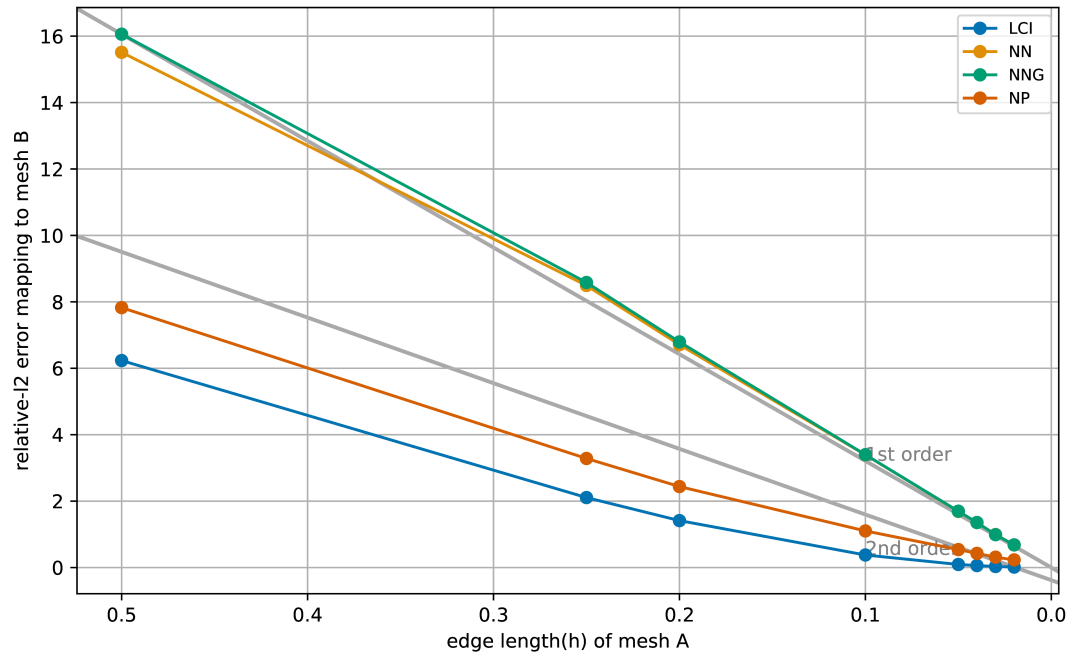
Linear cell interpolation is tested against nearest-neighbor, nearest-projection and nearest-neighbor with gradient mappings. Similar plots as above but with these mappings are shown in figures 5.3 and 5.4, for 2D and 3D respectively. Clearly, linear cell interpolation should be more accurate than nearest-neighbor and nearest-projection. In a context of volumetric coupling, the nearest-projection mapping should be inaccurate (although still better than nearest-neighbor) as all vertices get projected to the closest edge instead of being interpolated using the relevant triangle. As mentioned earlier, nearest-projection is second order accurate in general but first order accurate with respect to the projection distance, which will most of the time be non-zero.

On the other hand, nearest-neighbor with gradient should give similar performance to linear cell interpolation, as both are supposed to be second order accurate.
As expected, linear cell interpolation clearly and consistently outperforms nearest-neighbor and nearest-projection. However, nearest-projection is still reasonably good, although mostly 1st order accurate. A bigger surprise is the behavior of nearest-neighbor with gradients, which is only slightly better than nearest-neighbor. This is contradictory to theoretical expectations and experimental findings in [2], although these experiments were in a context of surface coupling. No satisfactory explanation has been found to this bad behavior.

Once again, orders of convergence are clearer in 2D. However, the ordering of mappings in terms of accuracy is constant, and linear cell interpolation always outperforms all the non-RBF mappings.

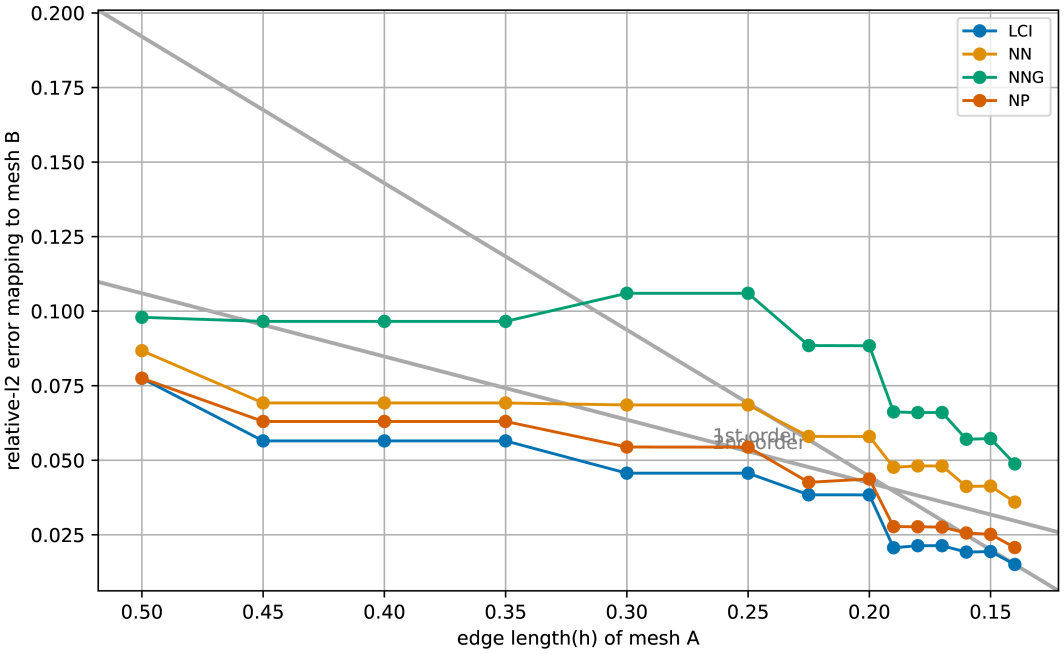


(a) Franke 2D

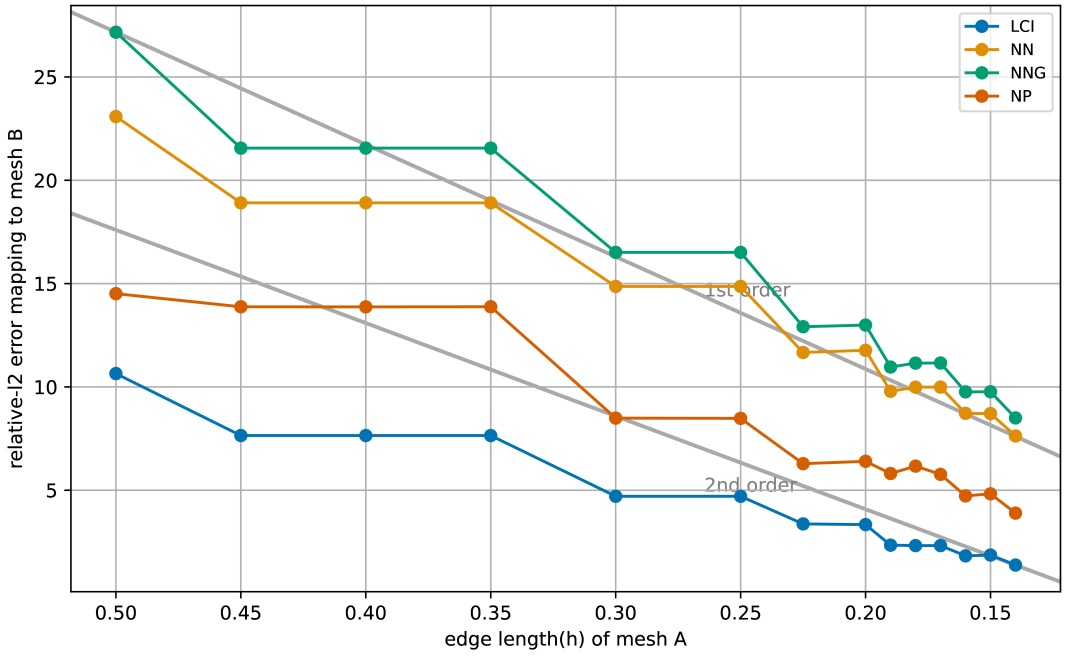


(b) Rosenbrock 2D

Figure 5.3.: Analysis of NN, NNG, NP and LCI behavior in 2D



(a) Franke 3D



(b) Rosenbrock 3D

Figure 5.4.: Analysis of NN, NNG, NP and LCI behavior in 3D

5.3. Runtime analysis

Each mapping method has a different behavior in terms of computational time and memory consumption. Time is spent at initialization of the mapping, once at the beginning of a simulation (the `computeMapping()` method of the mapping), and when mapping data (`mapConsistent()` and `mapConservative()`), once per iteration of a time window. The latter is overall more important, as it runs many times over a simulation.

Non-RBF mappings have a complicated initialization, as it implies finding a nearest vertex, edge, triangle or tetrahedron. Once it is done, the mapping itself is straightforward:

- With nearest-neighbor, each output vertex copies the value of its corresponding input vertex.
- When gradients are used, the same applies, then gradient terms ($\nabla f \cdot \mathbf{h}$ with $\mathbf{h}$ known, fixed h for each vertex) are added.
- With nearest-projection and linear cell interpolation, the same affine combination is used at each iteration.

In terms of memory, the consumption should be at most linear, as the amount of data to store per vertex is bounded above.
With RBF mappings, both initialization and mapping are complex: at initialization, the matrix of the problem must be built (either a dense or sparse matrix depending on whether the RBF has compact support or not), and at mapping time, the linear system must be solved[4]. This is why RBF is the most expensive mapping method: it is the only method that is expensive at each iteration.

In terms of memory consumption, RBFs with global support should be expected to have at most a quadratic consumption, as a dense matrix must be stored. Compact support RBFs lead to sparse matrices which should require less memory.
Let us now compare theory and practice with some measurements. For simplicity, only Franke function will be used.

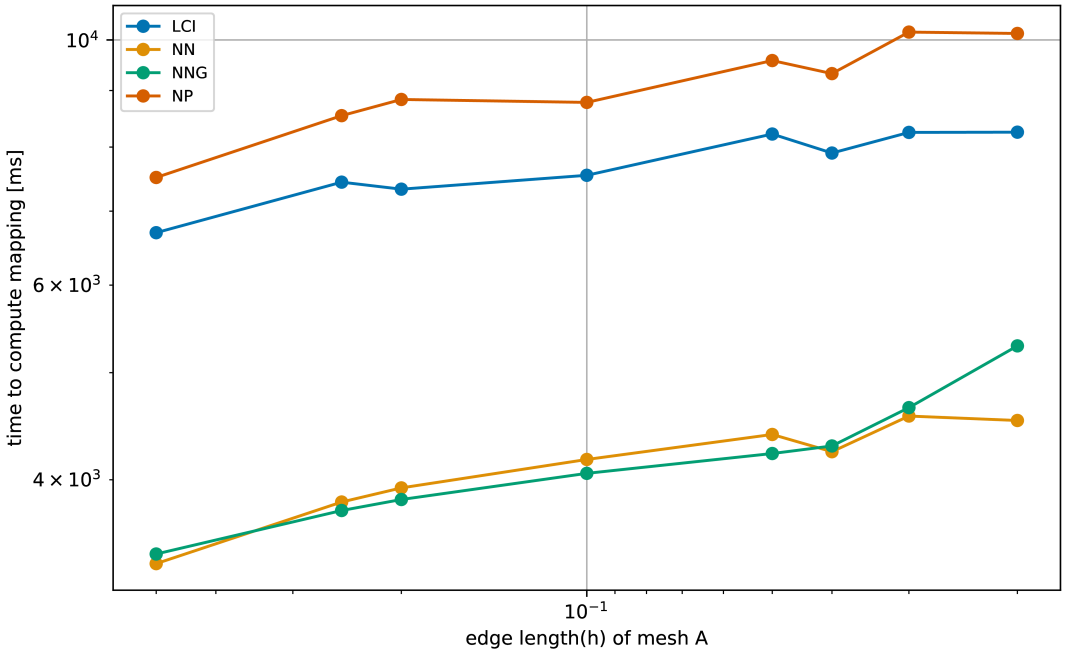
5.3.1. Time needed to compute the mapping

Figure 5.5 shows the time spent computing the mapping with Franke function in 2D, and figure 5.6 the equivalent in 3D. On the first plot, we can see that nearest-projection and linear cell interpolation take more time than the nearest-neighbor (with or without gradient), approximately by a factor of 2. This could be explained by the need to look not just for the nearest vertex but for some of the nearest primitives (in nearest-projection) or all the enclosing primitives (linear cell interpolation), as well as the need to compute barycentric

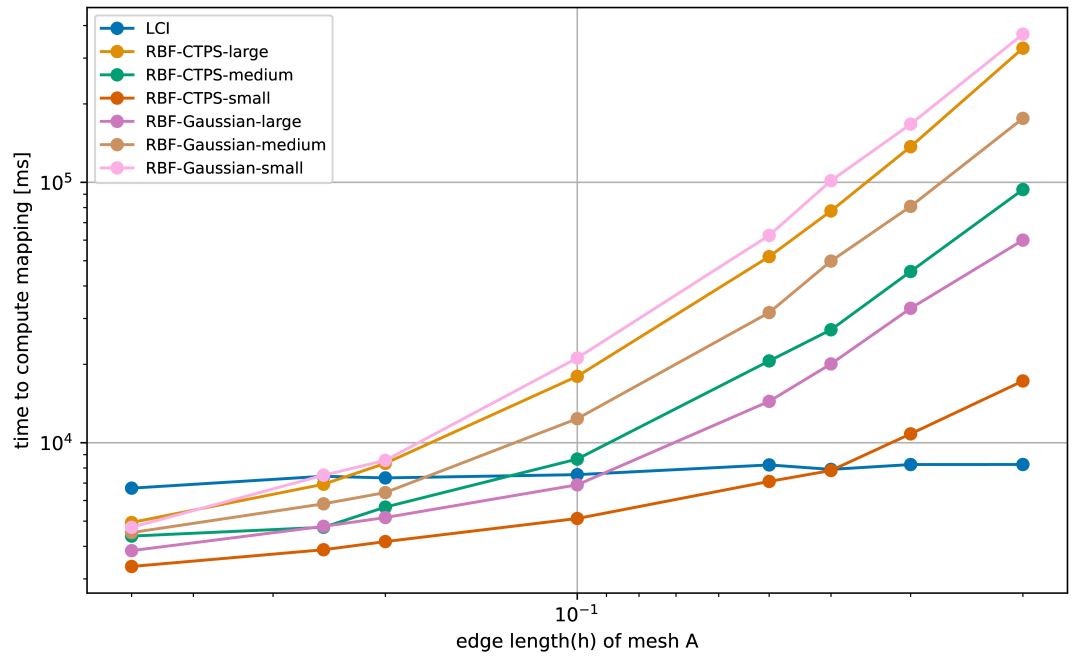
[4]For global support matrices, a QR decomposition is used to speed up the process.

coordinates afterwards. Overall, scalability is very good: fine meshes, despite having a hundred times more vertices, seem to only take at most twice the time. This is much better than worst case scenarios: R-Tree queries have a $n \log n$ worst case complexity, and doing it for all output vertices could get expensive. Note that since the output mesh has many more vertices than the input mesh, iterating over all these could provide most of the overhead, explaining the quasi-constant complexity.

On the RBF side, scalability is worse, as the matrix must be built. In particular, Gaussian RBFs (which have global support and require dense matrices) perform worse than CTPS, which have local support and use sparse matrices. It is interesting to note that with the latter, a higher support radius translates into a slower build, as each vertex is influenced by more neighboring vertices than with a small radius. Similarly, wide Gaussian also take more time, although the reason is less obvious.

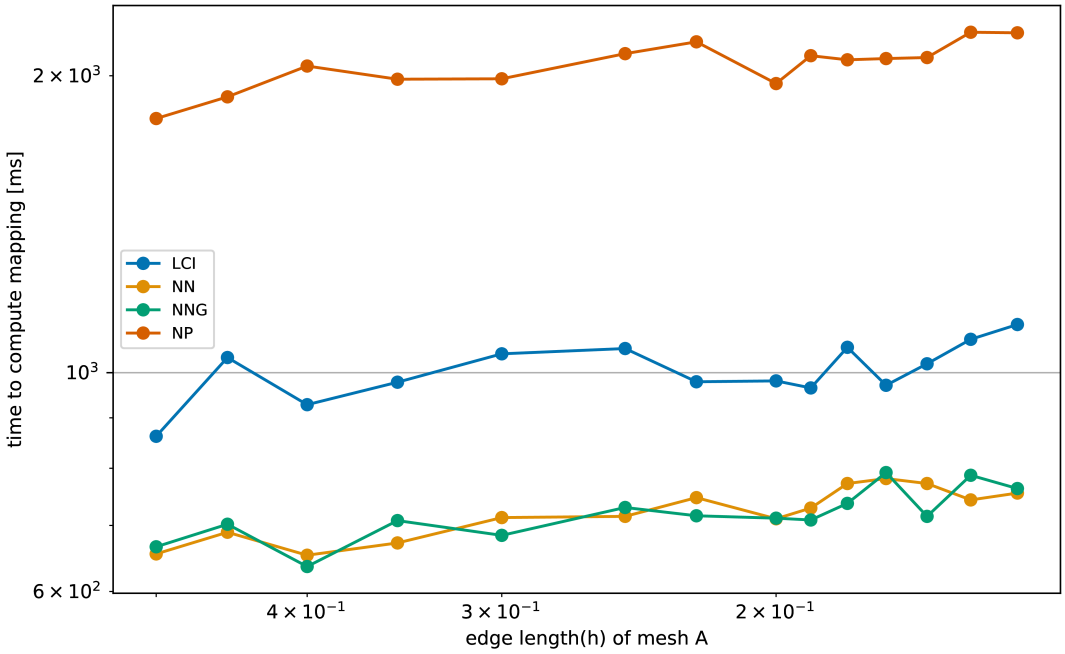


(a) Non-RBF mappings

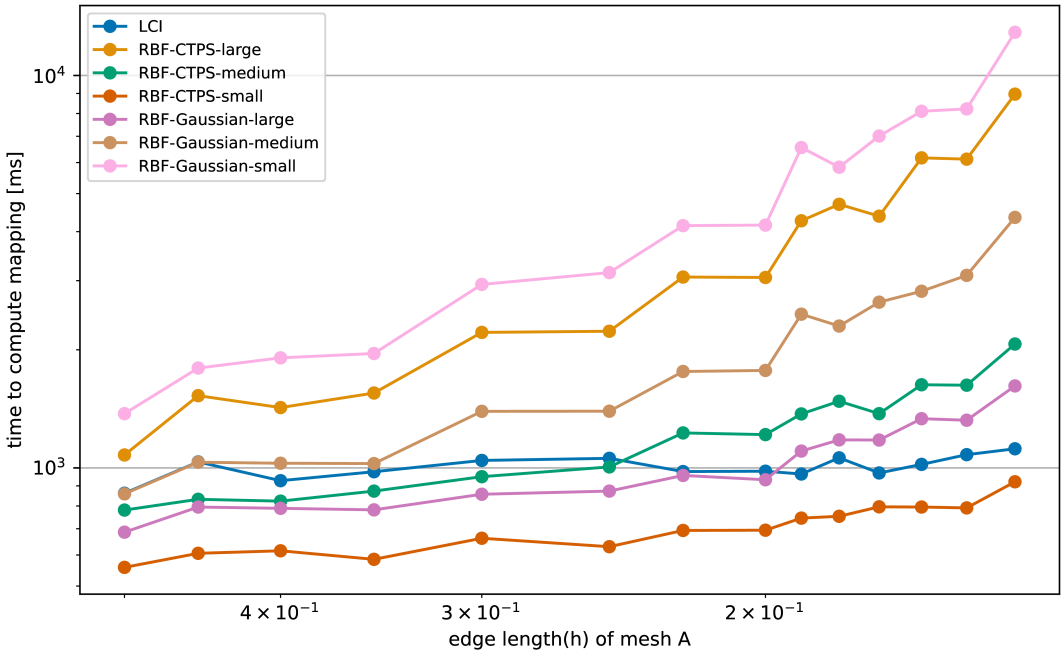


(b) RBF mappings

Figure 5.5.: Compute time analysis of Franke function in 2D.



(a) Non-RBF mappings



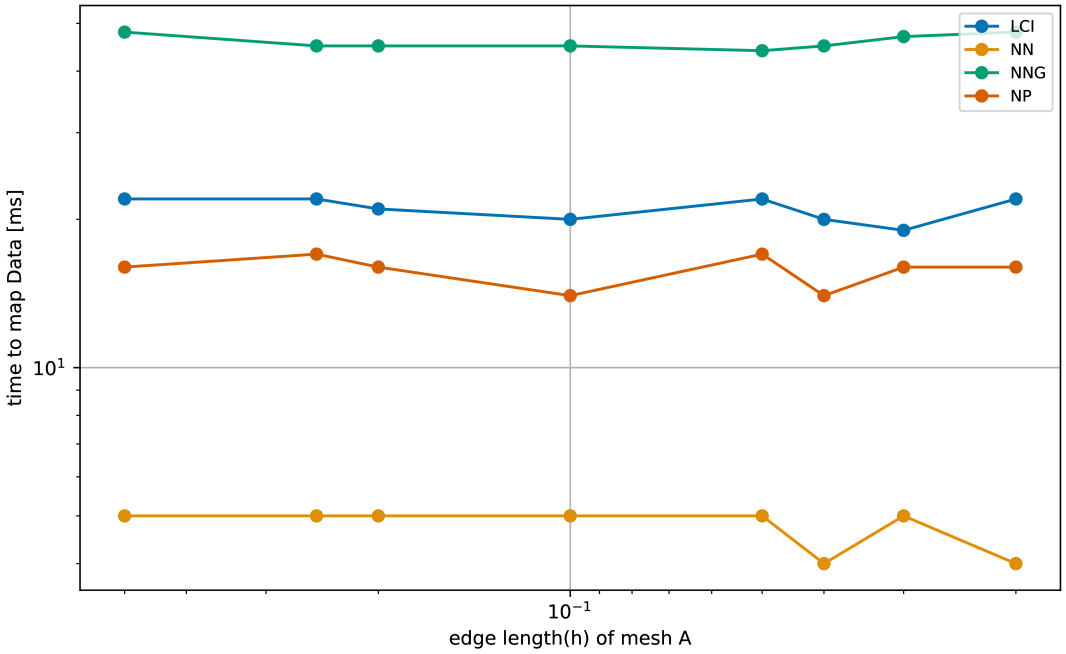
(b) RBF mappings

Figure 5.6.: Compute time analysis of Franke function in 3D.

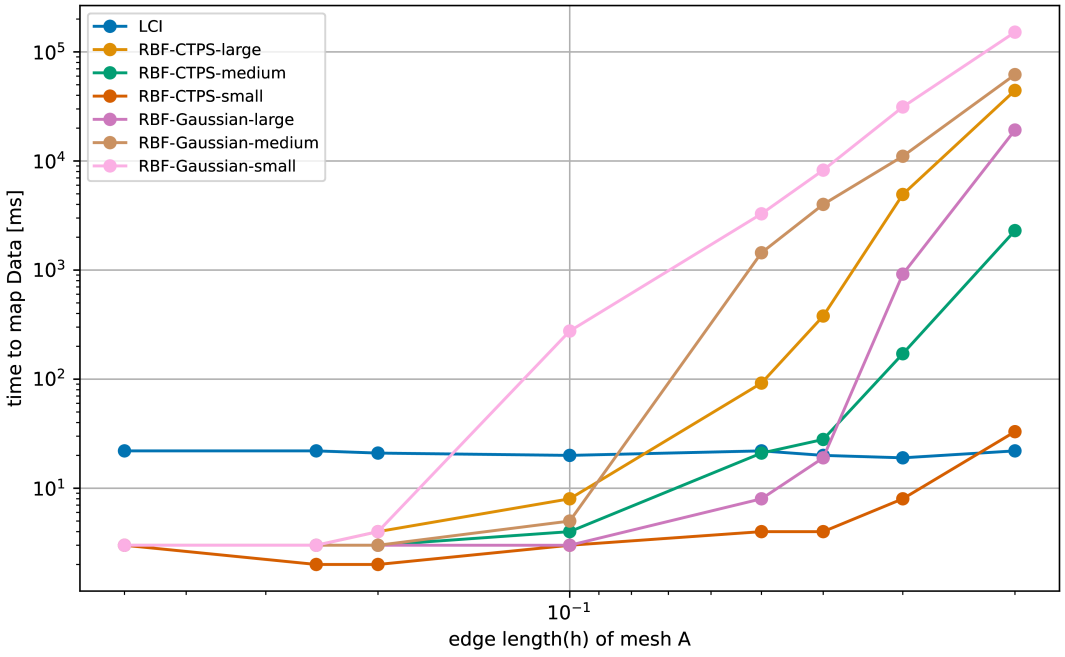
5.3.2. Time needed to map data

Mapping time is show on figures 5.7 and 5.8. Note that more extensive measurements have been done in 2D. As expected, non-RBF mappings are quite fast once the setup is done. Barycentric mappings are slower than nearest-neighbor as they must compute the interpolation as an affine combination instead of simply copying values, and nearest-neighbor with gradient requires computing the effect of the gradient. Overall, the difference is small enough to justify using better mappings than nearest-neighbor.

On the RBF side, things get uglier. While they are surprisingly fast on small meshes, scalability is quite bad, especially with Gaussian RBFs, as dense matrices eventually become too expensive. When using sparse matrices, an appropriate support radius has a big impact, as the conditioning of the matrix impacts the performance of the iterative solver used.

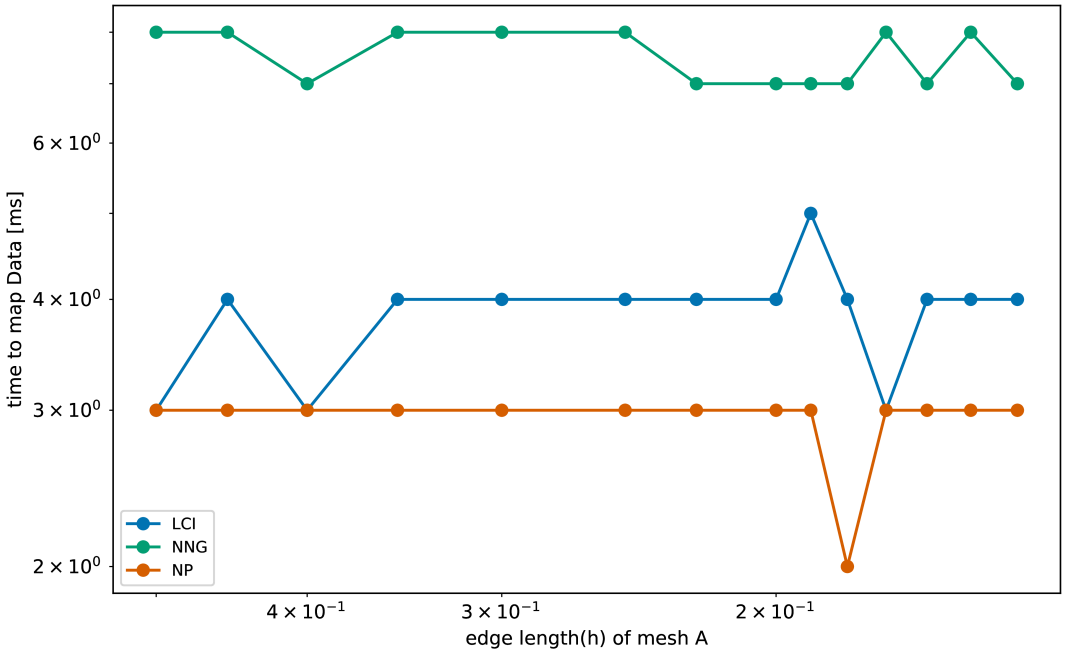


(a) Non-RBF mappings

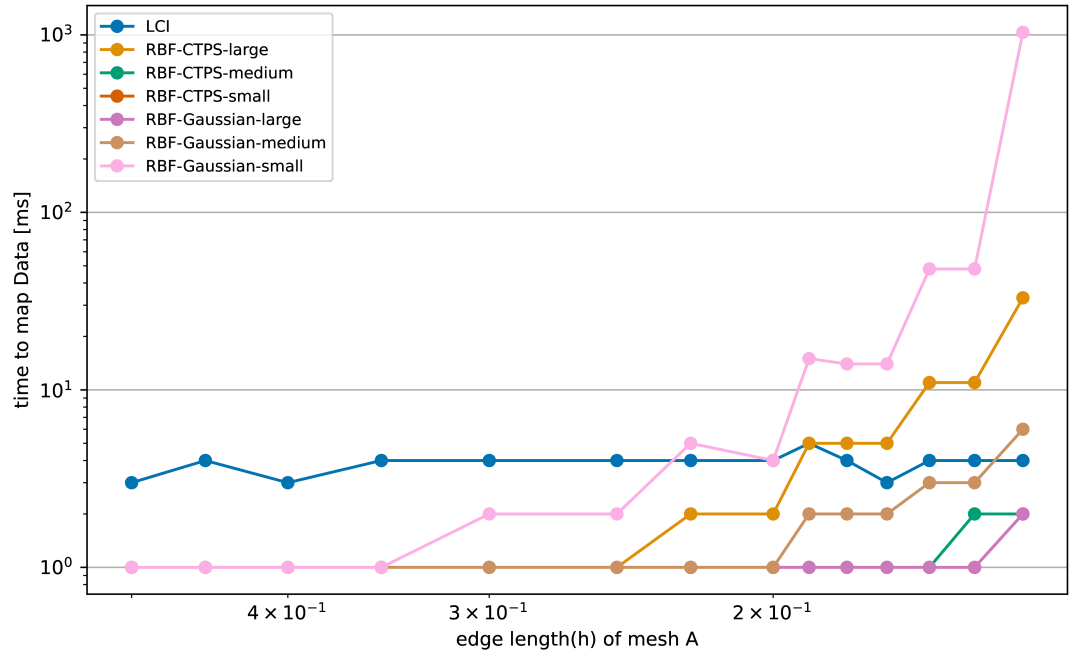


(b) RBF mappings

Figure 5.7.: Mapping time analysis of Franke function in 2D.



(a) Non-RBF mappings



(b) RBF mappings

Figure 5.8.: Mapping time analysis of Franke function in 3D.

5.3.3. Memory consumption

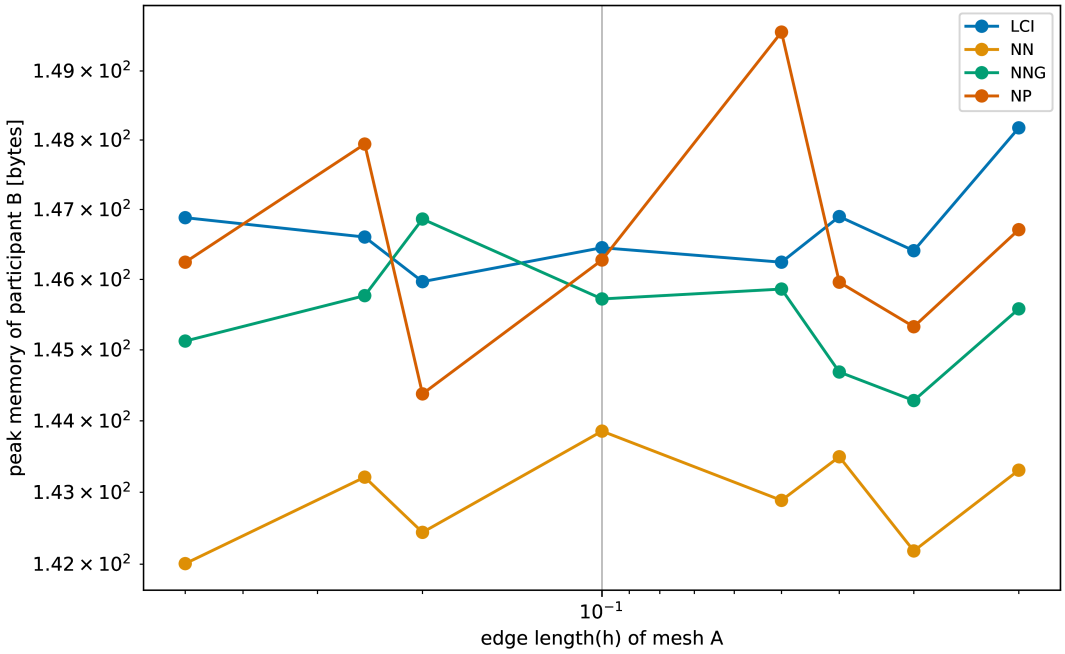
The mapping tester script measures the memory consumption of a run (where a fake simulation with only one time window). It is difficult to see what part of the memory comes from the mappings and what part comes from the rest. Figures 5.9 and 5.10 show the measured quantities. For non-RBF mappings, plots are very chaotic because of the scale, but the data is actually almost constant. This indicates that storing interpolations in connectivity elements is marginal in the total memory consumption. However, the effect is not null. In 2D simulations, linear cell interpolation seems to consume 3-4% more memory than nearest-neighbor. Overall, this should not be a big concern. On the other hand, RBF mappings consume much more memory, especially for finer meshes. As expected, storing dense matrices is more expensive (although the impact of the shape parameter is harder to explain). With compact thin plate splines, the consumption is more moderate, especially with smaller support radii.

5.4. Conclusion

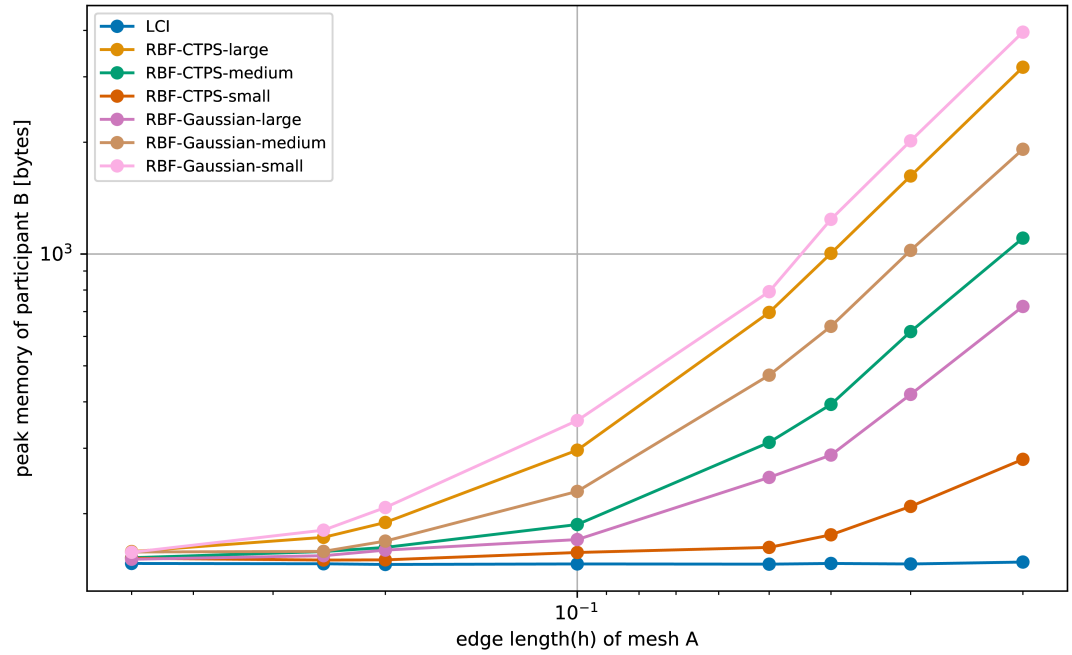
As expected, linear cell interpolation provides robust second order interpolation at a reasonable computational price. It clearly outperforms nearest-neighbor and nearest-projection (as long as it is used in a context of volumetric coupling), and is only bested by some appropriate choices of RBF interpolation. However, RBF interpolation is significantly more expensive than other methods, especially with volumetric coupling. This cost is probably prohibitive in actual simulations. It is worth noting that for non-RBF mappings, most of the computation cost lies at initialization, while RBF is expensive every time data must be mapped.

The only surprise here is the poor performance of nearest-neighbor with gradient, which was expected to be approximately as accurate as linear cell interpolation but behaves poorly in most tested cases. Note that even if nearest-neighbor with gradient had the expected accuracy (or if an error in its implementation explaining the bad results is found later), linear cell interpolation has two advantages: it can be used as a conservative mapping, and it is an actual interpolation, meaning that mapped data will always be bounded by the highest and lowest values of the input data. This is not true when using gradients, which can sometimes overshoot. This can be a serious problem when this leads to invalid data such as negative temperature[2].

Overall, when connectivity data is available, linear cell interpolation should become the default mapping choice when volumetric coupling is involved. It is clearly superior to non-RBF alternatives and much cheaper. RBFs should only be used if the increased accuracy is crucial or if the vertex count is low enough to make it affordable.

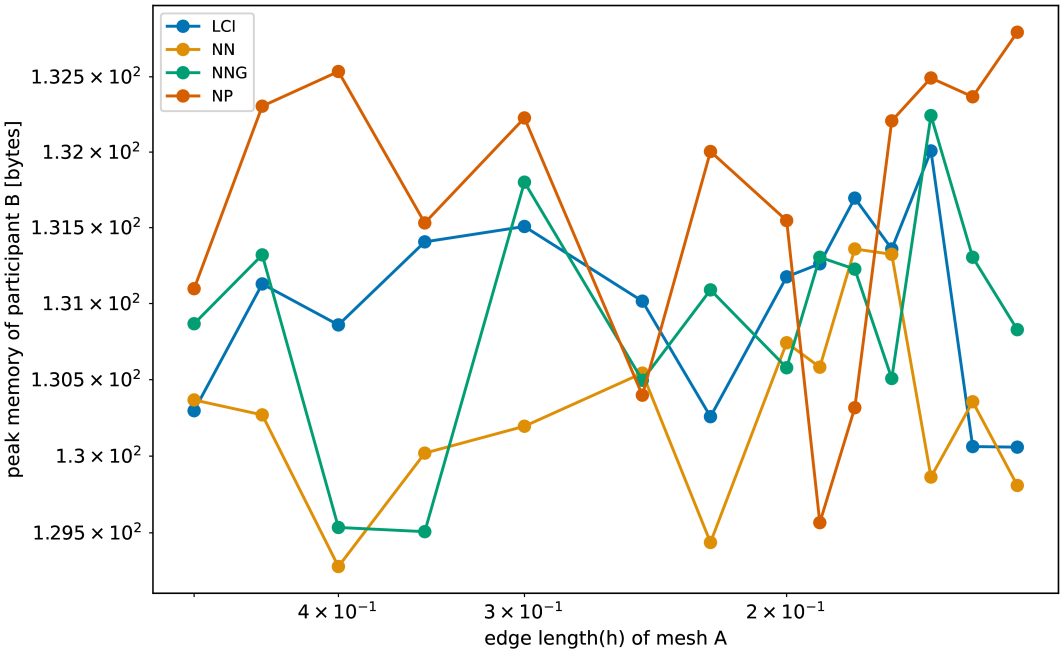


(a) Non-RBF mappings

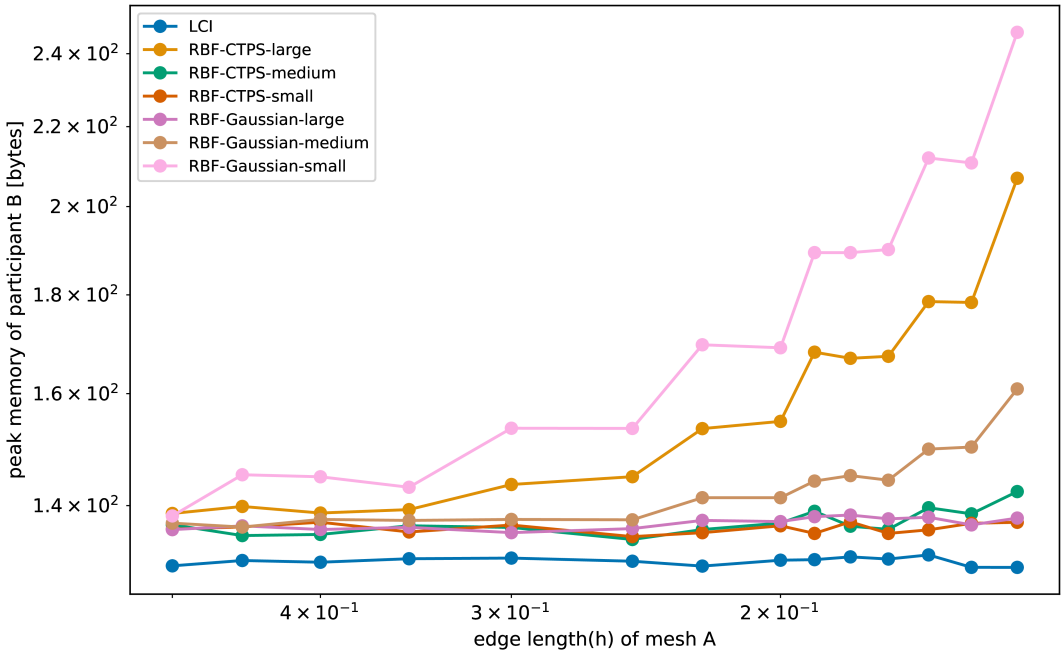


(b) RBF mappings

Figure 5.9.: Memory analysis of Franke function in 2D.



(a) Non-RBF mappings



(b) RBF mappings

Figure 5.10.: Memory analysis of Franke function in 3D.

6. Example case: chemical reactions in a channel flow

6.1. Introduction

One example volumetric coupling occurs in multiphysics is the simulation of chemical reactions in a fluid. The fluid acts as a solvent and different chemical reagents are dissolved in it. In a static fluid, these reagents diffuse through the fluid, and can potentially react. If additionally, the fluid is moving, these reagents (as well as the reaction products) move along, getting mixed. The equations describing the evolution of the concentration of each substance over time and space are called *reaction-diffusion-advection equations*[6], as they take in account these 3 phenomena. The advection term in these equations come from solving the Navier-Stokes equations, which makes the system coupled. If the Navier-Stokes equations can be solved without knowing the concentrations, the coupling is unidirectional, but it can become bidirectional if the concentrations influence the flow, for instance by impacting the density of the fluid or generating heat (e.g. through an exothermic reaction) that produces a buoyancy force.

A partitioned implementation comes quite naturally: the simulation can be split between a fluid simulation and a chemical simulation, where the chemical simulation reads the fluid velocity field, and if necessary, the fluid solver reads data from the chemical solver such as heat generation.

6.2. Case description

To showcase volumetric coupling with preCICE, the reaction-diffusion-advection example from the FEniCS tutorial[14] is reproduced. This tutorial consists of a flow in a channel (containing a cylinder, as described on figure 6.1[1]) as well as a chemical reaction $A+B \rightarrow C$ occurring inside it. A source of A is located in the top-left corner and a source of B in the bottom-left corner. Through diffusion and advection, these two reactants come in contact and produce some C. The coupling is unidirectional: the concentration of A, B and C does not impact the flow. In that tutorial, the fluid flow is solved entirely on a first simulation, then the chemical reaction case loads the results and does the chemical simulation. This

[1] From http://www.mathematik.tu-dortmund.de/~featflow/en/benchmarks/cfdbenchmarking/flow/dfg_benchmark2_re100.html.

case is adapted to preCICE by using a serial explicit coupling scheme: a fluid solver solves a time step[2] then sends its velocity fluid to the chemical solver which computes a step in the chemical reactions.
The original approach has the advantage that the fluid flow is computed once, which means that many chemical simulations can be run with a unique flow simulation if needed. However, preCICE is more flexible as it allows us to use different meshes for both participants, or even different solvers. In that case, a good data mapping can improve the accuracy. The newly implemented linear cell interpolation mapping will be compared to the less accurate nearest-neighbor mapping.

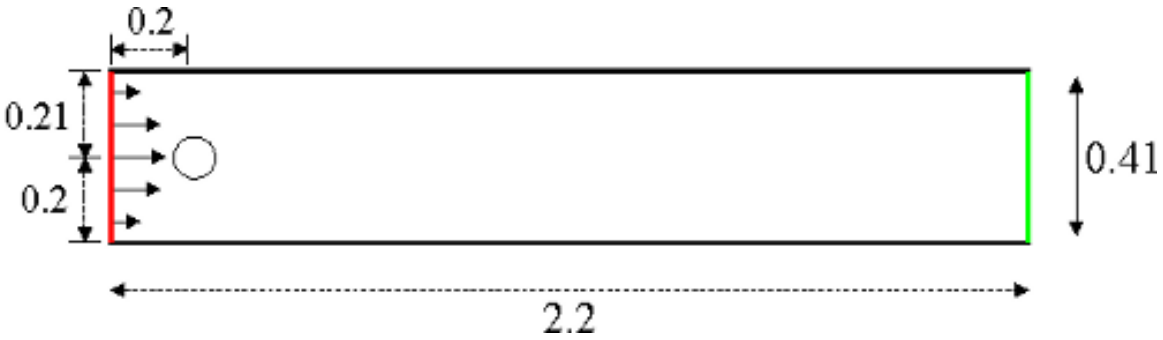


Figure 6.1.: Geometry of the case. Notice the small asymmetry of the cylinder's position. From http://www.featflow.de/en/benchmarks/cfdbenchmarking/flow/dfg_benchmark2_re100.html

6.3. Description of FEniCS and its adapter

FEniCS[14] is a set of open-source C++ and Python libraries designed to solve partial differential equations with finite elements. It has a high level API to describe meshes (which can also be imported from dedicated software such as gmsh[11]), function spaces over them and variational problems. However, it is the responsibility of the user to define the time stepping. This high level approach is optimal for prototyping: the user can solve complex PDEs easily in a very limited number of lines of code. Furthermore, it comes with efficient linear and non-linear solvers and built-in parallelization. All these features make FEniCS an optimal choice to experiment with our own differential equations, whereas some specialized libraries focus on specific PDEs (e.g. fluid solvers solving the Navier-Stokes equations).

The FEniCS adapter for preCICE (*FEniCS-preCICE*[17]) provides utility functions to couple FEniCS to preCICE. It helps for tasks such as reading/writing data between FEniCS and

[2]Depending on configuration, the fluid solver can *subcycle* and solves many steps, whereas the chemical solver uses larger steps.

preCICE or setting up the preCICE mesh from the one defined in FEniCS. In particular, it provides an interpolation mechanism to convert data from preCICE (a set of point values) into a FEniCS *Expression* object, which is a continuous function used to express boundary conditions or source terms independently of the mesh or type of element used. This mechanism uses RBF interpolation, but must not to be confused with the one potentially used by preCICE.

The adapter was designed with surface coupling in mind, but due to the intrinsic flexibility of FEniCS, using it for volumetric coupling is feasible without major changes. However, the RBF interpolation mechanism becomes quite expensive when the coupling domain includes many points, which happens quickly when doing volumetric coupling.

Finally, it is worth noting that the adapter was designed with 2D simulations in mind. It has been upgraded with partial support for 3D, but some features such as the usage of *Expressions* made with RBF interpolation are not available in 3D. This means it is not possible to read continuous boundary conditions (temperature, displacement, ...). However, it is possible to read conservative quantities (which do not rely on expressions).

6.4. Changes to the FEniCS adapter

Before this thesis, the adapter used to configure connectivity by setting mesh edges. Since surface coupling was assumed, no triangle was configured[3]. To make a good use of linear cell interpolation, the connectivity code was upgraded to set triangles.
The FEniCS adapter does not duplicate shared vertices; consequently, connectivity contains holes when a participant runs in parallel, as described in section 2.4.3.

The RBF interpolation mechanism to convert read data into a FEniCS *Expression* was significantly slowing down the data mapping. While the adapter was not changed in this aspect, a workaround found was to write simulations such that we compute this expression then sample it on the mesh to get a *Function* object. This did not avoid the RBF computation, but avoided repeatedly evaluating the expression and had a noticeable effect on timing.

6.5. Equations and discretization

The fluid solver solves the Navier-Stokes equations (equations 6.1 and 6.2, with $\mathbf{u}$ the velocity, p the pressure, ρ the density, σ the stress tensor and $\mathbf{f}$ the body forces, set to zero in this case) for an incompressible, Newtonian fluid. Each time step is computed in 3 parts, to ensure the zero divergence of the velocity fluid. First, velocity at the next time step

[3] Triangles are also needed in 3D surface coupling but are not configured either: it is another aspect of the limited 3D support.

is estimated with an explicit scheme and current pressure. Then, a pressure correction is computed (by solving a Poisson equation) such that the final velocity (whose computation is the third step) computed using this new pressure has a zero divergence. Each of these steps involves solving a linear problem, which is moderately expensive. The velocity field is then sent to preCICE.

$$\rho(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u}) = \nabla \cdot \sigma(\mathbf{u}, p) + \mathbf{f} \tag{6.1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{6.2}$$

The details of the conversion to weak form and the form of the stress tensor can be found in the FEniCS tutorial book. The velocity field is computed on a grid of quadratic triangular elements, while the pressure is on a mesh made of linear triangular elements. The actual triangles are, geometrically speaking, the same.

As for boundary conditions, a parabolic profile is forced as inflow velocity and the outflow has zero pressure. The other borders (including the cylinder inside the domain) are non-slip walls.
The chemical solver solves the system of reaction-diffusion-advection: each reagent is diffused, transported by the fluid flow, and involved in reactions. An implicit time stepping is used to ensure stability. However, the equations are not linear (the reaction rate being proportional to the product of concentrations in A and B) and the non-linear solver of FEniCS is used. This makes a step significantly more expensive than a fluid step. The system of equations is described by equations 6.3, 6.4 and 6.5:

$$\frac{\partial [A]}{\partial t} + \mathbf{u} \cdot \nabla [A] = \nabla \cdot (\epsilon \nabla [A]) + f_A - K[A][B], \tag{6.3}$$

$$\frac{\partial [B]}{\partial t} + \mathbf{u} \cdot \nabla [B] = \nabla \cdot (\epsilon \nabla [B]) + f_B - K[A][B], \tag{6.4}$$

$$\frac{\partial [C]}{\partial t} + \mathbf{u} \cdot \nabla [C] = \nabla \cdot (\epsilon \nabla [B]) + K[A][B] - K_d[C]. \tag{6.5}$$

The quantities f_A and f_B are the generating terms (constant over the source region and zero elsewhere), ϵ is the diffusivity (assumed to be the same for the 3 substances for simplicity), K is the reaction rate and K_d is the decay rate. Homogeneous von Neumann boundary conditions are applied on all the boundaries.

6.5.1. Time steps

The simulation is run with time windows of duration 0.01, which is also the time step of the chemical solver. On the other hand, the fluid solver uses smaller steps (0.001) that are

required for stability reasons: the Courant number must be smaller than 1.
This means that the fluid solver is *subcycling*: fluid velocity data is sent to preCICE every 10 steps, and the chemical solver then does one large time step covering the same duration. Subcycling is a built-in functionality of preCICE, which can tell the solver whether it is necessary to write data.
The same time stepping is used in the reference case from the FEniCS tutorial. In that case, the subcycling is not handled by preCICE, but FEniCS provides functionality to read a time series at any point in time, using linear interpolation if needed.

6.6. Results

To evaluate the relevance of linear cell interpolation mapping, three kinds of simulations are run:

- A reference solution, with matching meshes.
- A simulation with a coarser fluid mesh, with nearest-neighbor mapping.
- A simulation with a coarser fluid mesh and linear cell interpolation mapping.

When using non-matching meshes, the coarser one is always the fluid one. Since the coupling is unidirectional and the mapping is a consistent reading of velocity, this is the most interesting case: accuracy is lost when mapping from coarse to fine, whereas the other direction is less prone to accuracy losses[4].

Clearly, the reference solution will be the most accurate one. The most interesting question is whether using linear cell interpolation mapping instead of nearest-neighbor mapping has a significant effect on accuracy.

The reference mesh have a resolution of 64 and coarse meshes have a resolution of 32. A resolution n means that the mesh generator tries to generate meshes with a size smaller than D/n with D the diameter of the geometry, which is the domain length in this case. Figure 6.2 shows both meshes and the produced velocity field at time $t = 1.20$.

Note that the case is defined using adimensional lengths and time units. All plots will thus have adimensional data.
The FEniCS tutorial case was also run, but yielded result that were indistinguishable from the simulation with preCICE and matching meshes.

[4] As a reminder, a fine to coarse consistent mapping is exact if the vertices of the fine mesh are a superset of those of the coarse mesh.

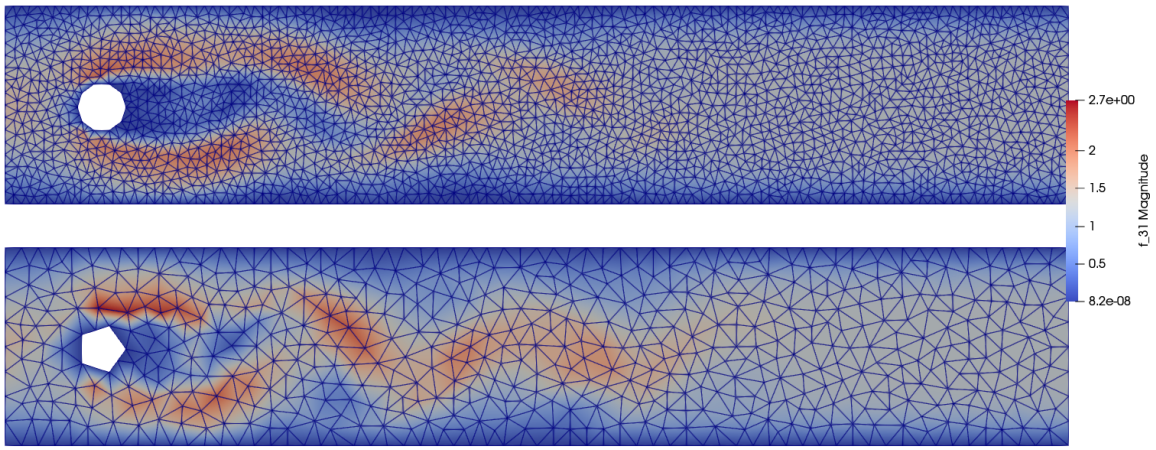


Figure 6.2.: Velocity fields produced by the fine and coarse meshes, before mapping. ($t = 1.20$)

6.6.1. Comparison of read velocity fields

Figure 6.3 show the velocity field magnitude as seen by the chemical solver (i.e. after mapping) at time 1.20. One can see that the coarse mesh with linear cell interpolation yields a smoother field and avoid a staircase effect. However, the coarse fluid simulation has a different physical behavior from the fluid one, and no interpolation scheme can fix this issue. Linear cell interpolation produces a better approximation of what the fluid solver thinks the velocity is, but cannot improve the solver itself.

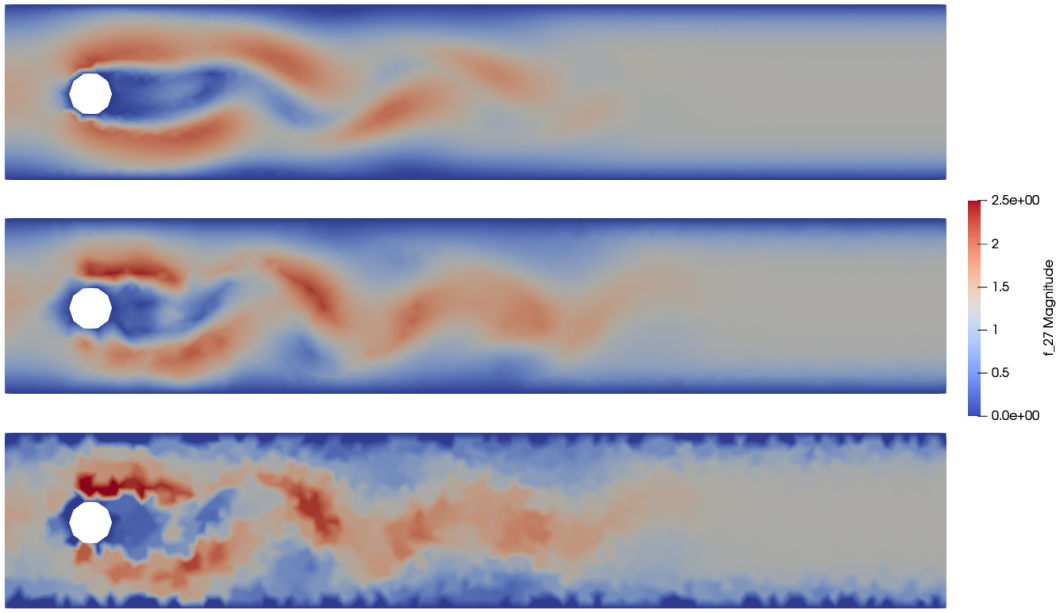


Figure 6.3.: Velocity fields from reference, LCI mapping and NN mapping at time $t = 1.20$.

6.6.2. Comparison of product concentration

Figures 6.4, 6.5 and 6.6 show the concentration of substances A, B and C. For A and B, all simulations give a similar result, although coarser fluid meshes seem to yield higher concentrations at the sources. More generally, using the coarse fluid mesh (with either linear cell interpolation or nearest-neighbor mapping) seems to yield slightly higher concentrations everywhere.
Higher concentrations in A or B could be explained by lower reaction rates or by discretization errors. Looking at figure 6.6 shows that concentrations in C are also higher with the coarser mesh, and the effect is stronger than with A and B. Discretization errors are thus the most likely explanation.
It is worth mentioning that using finite elements do not provide guarantees on conservation of quantities. A finite volume solver would probably have yielded more consistent results, where the total amount of matter is not dependent on advection velocity (except for the matter leaving the channel).

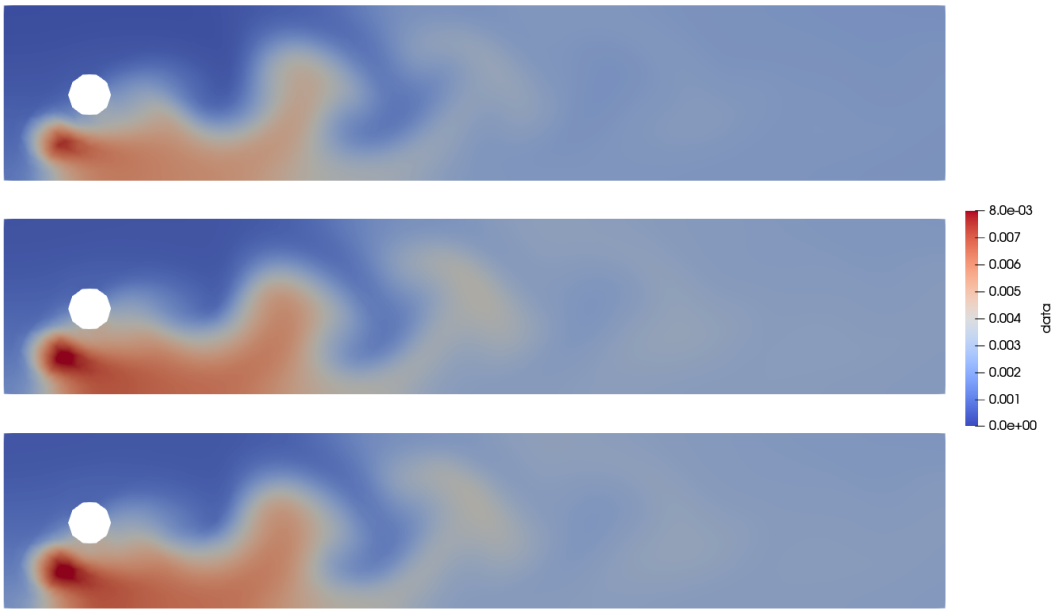


Figure 6.4.: Concentration of A at end time $t = 5$.

A similar measurement that can be used to compare results is the *total* amount of A, B and C. This can be done easily using FEniCS native support for computing integrals, each total quantity being the integral over the domain of the corresponding concentration. Figures 6.7, 6.8, 6.9 show the evolution over time of each integral. We can then easily see that the coarser mesh overestimates all quantities, but using linear cell interpolation slightly mitigates the problem.

Figure 6.5.: Concentration of B at end time $t = 5$.

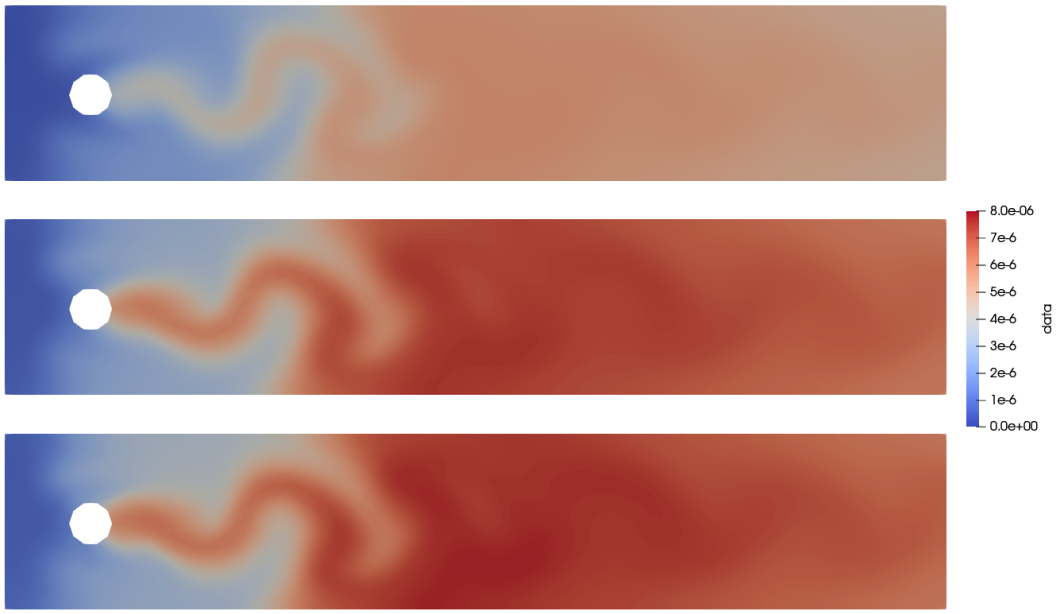


Figure 6.6.: Concentration of C at end time $t = 5$.

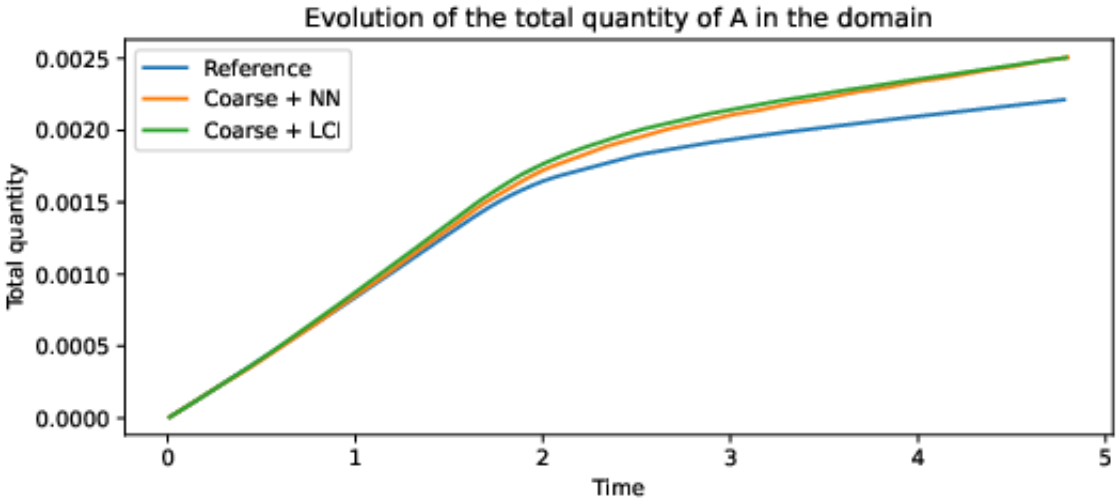


Figure 6.7.: Total quantity of A over time.

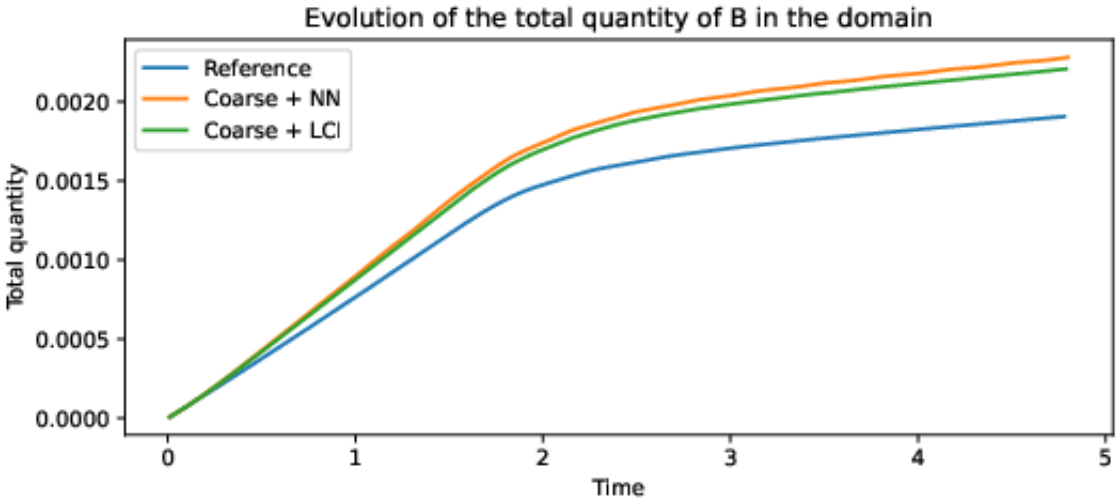


Figure 6.8.: Total quantity of B over time.

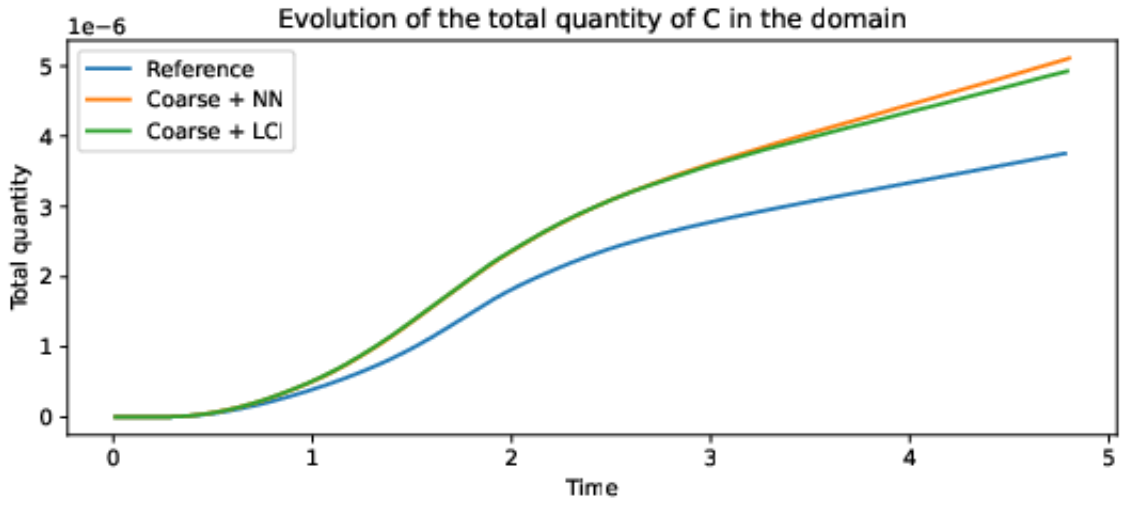


Figure 6.9.: Total quantity of C over time.

6.6.3. Conclusion

Linear cell interpolation successfully improves the accuracy of the mapping. However, the loss of accuracy from using a coarser mesh is dominant and the effect of an improved mapping on the final result is limited. This conclusion is specific to this case and does not necessarily generalize to other equations, other geometries or other mesh resolution combinations. However, some tests were run with a smaller resolution difference than here (instead of a coarse mesh with resolution 32, 50 was tried) and it yielded similar results.
This test case validates the update of the FEniCS adapter in terms of connectivity.

6.7. Relevance of preCICE in this simulation

As seen when using matching meshes, partitioned simulations using preCICE can reproduce the same result as FEniCS used alone, with the benefit of additional flexibility but at the cost of some overhead. Using preCICE allows us to extend or modify the case easily: one of the participants could be replaced (e.g. the fluid simulation could be run with software specialized in fluid simulations such as OpenFOAM) without changing anything in the other solver. It is also very easy to make the coupling bidirectional, like if we want the chemical reactions to generate heat that impacts the flow. This bidirectional could be explicit coupling or implicit coupling.
Finally, using preCICE makes it easy to use non-matching meshes. In that case, a good data mapping method is needed to preserve accuracy. In the studied case, the accuracy

gain from using linear cell interpolation instead of nearest-neighbor interpolation gave a limited, but non-zero, gain in accuracy.

7. Results and conclusion

7.1. Towards higher order interpolation?

Moving from piecewise constant interpolation to linear interpolation (using linear cell interpolation mapping or nearest-projection mapping instead of nearest-neighbor mapping) allows us to have a second-order accurate mapping through connectivity information. Second order accuracy can also be reached using nearest-neighbor mapping with gradient data. Alternatively, a higher accuracy can be reach using radial basis function interpolation, at a significant cost in terms of computations (solving linear systems) and complexity: the basis function as well as a shape parameter or support radius must be chosen carefully.

Could a higher accuracy be reached without RBF interpolation? The following subsections outline future possibilities for higher order mappings, as well as their respective drawbacks.

7.1.1. Third order accuracy with quadratic triangular or tetrahedral elements

If a linear polynomial yields second order accurate mapping, it makes sense to imagine a quadratic polynomial could yield a third order accurate mapping. This can once again be proven using Taylor series: if $\mathcal{H}$ is the Hessian matrix of a sufficiently regular function f, then it can be expanded as described by equation 7.1.

$$f(\mathbf{x}+\mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot \mathbf{h} + \frac{1}{2}\,\mathbf{h} \cdot \mathcal{H}(\mathbf{x}) \cdot \mathbf{h} + \mathcal{O}(\|\mathbf{h}\|^3) \tag{7.1}$$

If we build an interpolation method that is exact for second order polynomials and is linear, then third order accuracy can be achieved. One way to do this would be to use quadratic triangular and tetrahedral elements, which are common in finite element codes. A quadratic triangle has data on 6 points (typically the 3 vertices and the 3 edge centers), which provide 6 interpolation conditions to satisfy. A 2D complete quadratic polynomial is of the form $f(x,y) = c_1x^2 + c_2y^2 + c_3xy + c_4x + c_5y + c_6$ with 6 parameters. Thus, assuming a non-degenerate triangle, there is a unique quadratic polynomial that interpolates the given data. Similarly, in 3D, there are 4 vertices and 6 edges, whereas a quadratic polynomial has 10 parameters.

Adding these to preCICE should work, but add extra complexity: the user would have to choose between setting regular triangles or quadratic triangles (or tetrahedra) and the

library would have to keep track of both kind of elements. Furthermore, setup by the user could be quite complex unless the coupled code is a finite element code using this kind of elements. This does not fit well with the philosophy of preCICE to be as solver-agnostic as possible.

7.1.2. Third order accuracy using both gradients and connectivity

So far, second order accuracy has been reached using either connectivity or gradient data. When both are available, could we combine these to get a better result? One approach would be to use variations of the gradient to approximate the Hessian matrix. If this approximation is good enough, a truncated Taylor series with second order term could be used as interpolant. Let us first consider a 1D analogy: let a and b be two points where data and gradient data are known. Using $f(a), f(b), f'(a)$ and $f'(b)$, how should we approximate $f(x)$ for x between a and b?

Using Taylor series, we can expand $f(x)$ as described in equation 7.2, for some ξ between x and a.

$$f(x) = f(a) + (x-a)f'(a) + \frac{1}{2}(x-a)^2 f''(a) + \frac{1}{6}(x-a)^3 f'''(\xi) \tag{7.2}$$

If $f''(a)$ was known, we could have a third order accurate approximation (the error would be $\mathcal{O}(|h|^3$ for $h = |b-a|$, as long as x is between a and b). Using gradient data, we can build an approximation from a first order Taylor approximation (equation 7.3 for some ζ between a and b). Plugging this approximation of the second derivative into equation 7.2 (noting $\frac{f'(b)-f'(a)}{b-a} = f''_{\text{approx}}(a)$) yields equation 7.4.

$$f'(b) - f'(a) = (b-a)f''(a) + \frac{1}{2}(b-a)^2 f'''(\zeta) \tag{7.3}$$

$$f(x) = f(a)+(x-a)f'(a)+\frac{1}{2}(x-a)^2 f''_{\text{approx}}(a)-\frac{1}{4}(b-a)(x-a)^2 f'''(\zeta)+\frac{1}{6}(x-a)^3 f'''(\xi) \tag{7.4}$$

The last two terms are third-order, and all the others can be computed: using the difference of gradient to approximate the second derivative yields a third order accurate mapping method.

To generalize this to 2D or 3D, we do not need a second derivative but a Hessian matrix. In 2D, let us consider a triangle ABC with known gradient information. Then, if we truncate third order terms in the expansion of the gradients, we get equations 7.5 and 7.6.

$$\nabla f(\mathbf{b}) - \nabla f(\mathbf{a}) \approx \mathcal{H}(\mathbf{a})(\mathbf{b}-\mathbf{a}) \tag{7.5}$$

$$\nabla f(\mathbf{c}) - \nabla f(\mathbf{a}) \approx \mathcal{H}(\mathbf{a})(\mathbf{c} - \mathbf{a}) \tag{7.6}$$

Using both these equations, we can determine a unique (approximate) Hessian matrix by solving two linear systems. Let D be the matrix whose first column is $(\mathbf{b} - \mathbf{a})$ and second column is $(\mathbf{c} - \mathbf{a})$, and G the matrix whose columns are $\nabla f(\mathbf{b}) - \nabla f(\mathbf{a})$ and $\nabla f(\mathbf{c}) - \nabla f(\mathbf{a})$. Then, the equality $\mathcal{H}(a)D = G$ holds. Transposing yields $D^T\mathcal{H}^T = G^T$, which can be solved for $\mathcal{H}^T$. A similar approach can be used in 3D on a tetrahedron ABCD, and all the 2x2 matrices will become 3x3 matrices.

Once the Hessian is approximated, the function can be approximated inside the triangle or tetrahedron as $f(\mathbf{x}) = f(\mathbf{a}) + \nabla f(\mathbf{x}) \cdot (\mathbf{x} - \mathbf{a}) + \frac{1}{2}\ (\mathbf{x} - \mathbf{a}) \cdot \mathcal{H}(\mathbf{x}) \cdot (\mathbf{x} - \mathbf{a})$.

This method has not been implemented in preCICE during this thesis. However, unlike the approach with quadratic elements, it relies on already existing features (gradients and regular triangles/tetrahedra). Should it ever be implemented, no major API change would be required.

A major drawback of this method is that it is not an interpolation: if the sampled function is not quadratic, there is no guarantee that the reconstructed function will match the input data at vertices other than the one used as starting point: if we approximate from A, values at B and C (as well as D for a tetrahedron) could be wrong. The mapping is still exact for quadratic data (including linear and constant data), which makes it third order accurate as needed.

7.1.3. Direct mesh access

The best way to map data from a mesh to another one could be to let one of the solver do it by itself instead of relying on preCICE. Typically, with finite element codes using more complex elements (high order triangles, quads, elements with continuous derivatives, non-conforming elements, ...), it can be appropriate to simply evaluate the function at the requested points[1].

This is currently part of the experimental API of preCICE: a solver can request access to the mesh and compute the mapping.

While potentially the most accurate method, it has the obvious drawback of being very invasive, going against the black-box coupling coupling philosophy of preCICE. Implementing this requires specialized code for each solver. Furthermore, parallelization can be complex: the solver is responsible for sharing the workload, and the requirement of using *read-consistent* and *write-conservative* can cause trouble. For instance, parallel consistent

[1] At least for consistent mappings. Conservative mappings might be harder to handle

mapping would be *read* mappings, which means the participant reading data must do the mapping. This complicates the sampling of finite element functions, as the type of element and the connectivity would be known by the other participant.

7.2. Reproducibility of the results

The preCICE ecosystem (library, language bindings, adapters, tutorial cases), is entirely open-source and hosted on GitHub (https://github.com/precice). All the changes to the ecosystem with detailed history can be accessed there.

7.2.1. Version of preCICE

This thesis was started shortly after preCICE 2.3.0 was released, and version 2.4.0 was released during its redaction. Linear cell interpolation mapping (as well as all the related changes, such as the addition of tetrahedra) is currently only available in the *develop* branch of the library (which contains unreleased changes made after the release of 2.4.0), but will become an official feature of preCICE in future versions. Version 2.5.0 is planned for release around the time of submission of this thesis. It will contain the main parts of this thesis (linear cell interpolation mapping and support for tetrahedra) but volumetric scaled-consistent mapping is still under review[2] and will probably be available only in version 3.0.0. The library can be found on https://github.com/precice/precice/.

7.2.2. State of ASTE

ASTE is currently under active development and has no fixed versioning scheme. In this thesis, it has been updated to support tetrahedra from VTK files, which was necessary to test 3D linear cell interpolation mapping. Measurements were done using the *develop* branch at its state in the end of July 2022[3]. Currently, ASTE must be built with the `SUPPORT_NN_GRADIENT_MAPPING_AND_TETRA` option enabled, otherwise a version compatible with preCICE 2.4.0 (without tetrahedra) would get built instead. This requirement should be removed in the future, as tetrahedra become part of the official API.

7.2.3. Test cases used with ASTE

Results in chapter 5 used the *mapping-tester* script from ASTE. The configuration files used to produce the presented data as well as instructions to reproduce it can be found at https://github.com/boris-martin/lci-aste-tests/.

[2] https://github.com/precice/precice/pull/1387
[3] https://github.com/precice/aste/tree/000572b42d900fdf23657a5900c324b4e8190e0d

7.2.4. State of the FEniCS adapter

To run the chemical reactions in a channel flow test case, the FEniCS adapter had to be modified to configure triangles over the coupling interface. This change has not been merged into the official version of the adapter at the time of submission. To run the version of the adapter used in this thesis, the branch in the relevant pull request[4] should be used.

7.2.5. Addition of the chemical reaction case to the list of tutorials

At the time of submission of this thesis, the chemical reactions in a channel flow test case is currently under review before joining the list of official preCICE tutorials. Until it gets approved, the pull request[5] can be used to run this case.

7.3. Conclusion

In this thesis, we developed a new data mapping method for the preCICE library, called *linear cell interpolation*. This method uses connectivity information (triangles and tetrahedra configured on the preCICE mesh) to do linear interpolation (for consistent mappings) over the coupling domain. It is similar to the *nearest-projection* mapping that already existed in preCICE, except that it was designed for volumetric coupling instead of surface coupling.

This mapping works in both 2D and 3D, and the library was changed to allow triangles in 2D (they were only allowed in 3D before this thesis) and to add support for tetrahedra. The code was upgraded to communicate this element correctly between participants, but also between ranks of participants running in parallel. It was also necessary to index tetrahedra efficiently, which was done with R-Trees.

An extensive test suite was written to validate the implementation and avoid edge cases. Correct behavior was tested in both 2D and 3D for consistent and conservative mappings, running in serial or in parallel. Serial cases were tested for both *read* and *write* mapping directions, and parallel cases (which have to be *read-consistent* or *write-conservative*) were tested with either participant being parallel (one to many or many to one). Unit tests were also written to check correctness of smaller portions of code: R-Tree indexing, interpolation in a given tetrahedron or triangle, ...

The mapping was tested on a more realistic scale using ASTE (which was upgraded to support tetrahedra) to check the accuracy of the mapping (by comparing different mesh resolutions), expected to be second order accurate. A runtime analysis was also performed

[4] https://github.com/precice/fenics-adapter/pull/152
[5] https://github.com/precice/tutorials/pull/278

to evaluate timing and memory consumption. It was found that linear cell interpolation clearly outperforms nearest-neighbor and nearest-projection while remaining cheap. RBF interpolation can still outperform it, but at a higher computational cost and only if the right parameters are chosen. Nearest-neighbor with gradient was expected to yield similar results to linear cell interpolation, but actually performed worse than anticipated.

The mapping was also tested in the context of an actual simulation involving volumetric coupling. This case used FEniCS (whose adapter was upgraded to configure the required connectivity) to simulate chemical reactions occurring in a moving fluid. The simulation was split between a fluid simulation and a chemical solution, where the latter has to read the fluid velocity to compute the advection of chemical species. Partitioning allows us to uses non-matching meshes: simulations with a coarser mesh on the fluid side were run with either nearest-neighbor mapping or linear cell interpolation mapping, to evaluate how an improved mapping could improve the simulation accuracy. It was found that the effect of a better mapping (which yielded a smoother interpolation) was not as high as hoped: a better mapping is not always enough to mitigate the effects of a coarser mesh. This result is probably dependent on the case and not general.

Alongside this new data mapping, preCICE was also modified to make it more adapted to volumetric coupling. Scaled-consistent mappings, which used to preserve surface integrals, can now be configured to preserve volume integrals instead. Similarly, watch points, which monitored data at a defined location, now use volumetric interpolation instead of surface interpolation when volume connectivity is given. This means that watch points inside the coupling region are more accurate than they used to.

Finally, approaches towards more accurate mappings were discussed, as they have theoretical advantages but practical issues.

Overall, the capacities of preCICE to simulate problems involving volumetric coupling has been extended: linear cell interpolation offers a very good trade off between cost and accuracy and will probably become the default mapping choice of users. Furthermore, an additional tutorial will help users configure these simulations, and other minor functionalities have been upgraded to work with volumetric coupling as well as for surface coupling.

Appendix

A. Numerical computation of barycentric coordinates

The nearest-projection mapping and the newly added volume cell interpolation mapping require the computation of weights for linear interpolation on a simplex (line segment, triangle or tetrahedron). This appendix describes how the computation of these coefficients is implemented withing preCICE, as well as the mathemical derivation.
Some of the formulas derived below (interpolation on an edge and on a triangle in 3D space) were already implemented in preCICE, whereas the ones for triangle in 2D space and tetrahedra were added in the context of this thesis.

A.1. General concept

In dimension d, a simplex is made of $d+1$ points. Any point of the space can be described as an affine combination of these points. For instance, in a coordinate system whose origin is O, a triangle ABC and a point U, there exists a unique tuple (w_A, w_B, w_C) of real numbers such that $\overrightarrow{OU} = w_A\overrightarrow{OA} + w_B\overrightarrow{OB} + w_C\overrightarrow{OC}$ and $w_A + w_B + w_C = 1$. The point U is inside the triangle if and only if all these weights are positive.

When doing linear interpolation on a simplex, we look for a first order polynomial (e.g. for a triangle a function of the form $ax + by + c$), with $d + 1$ coefficients, such that this function interpolates correctly the d+1 vertices. This system is solvable if the simplex is not degenerate.
Instead of solving a system to find these coefficients, we can use barycentric coordinates: if the triangle mentioned before takes values f_A, f_B, f_C on the vertices, then the linear interpolation at point U can be shown to output the quantity $w_A f_A + w_B f_B + w_C f_C$ on point U, and similarly for line segments and tetrahedra. This is particularly useful in preCICE, since many interpolations with different values but at the same positions get computed. One must only compute the barycentric coordinates once (per vertex on the output mesh) then simply compute the linear combination of values each time a mapping is required.

A.2. Line segment

Let AB be the edge where interpolation is required, U the query point and P the projection of U on the line. Interpolation will be done using the barycentric coordinates of the

projection P. The vector $\overrightarrow{AP}$ can be constructed by starting from A and moving towards B with (signed) length $\frac{\overrightarrow{AU}\cdot\overrightarrow{AB}}{\|\overrightarrow{AB}\|}$. The barycentric coordinate of P with respect to B is the ratio of this quantity and the length of the edge: $\frac{\overrightarrow{AU}\cdot\overrightarrow{AB}}{\|\overrightarrow{AB}\|^2}$, which can be simply computed as $w_B = \frac{\overrightarrow{AU}\cdot\overrightarrow{AB}}{\overrightarrow{AB}\cdot\overrightarrow{AB}}$. Then w_B is simply $1 - w_A$.

A.3. Triangle

With triangles, barycentric coordinates represent fraction of the total area of the triangle. The barycentric coordinate of U with respect to A in the triangle ABC, if U is inside the triangle, is the area of UBC divided by the area of ABC, etc. When U is outside the triangle (but still on the same plane), a generalization ca be done using signed areas.

To prove this, we recall that the area of any triangle ABC can be computed used cross-product: the area is $\frac{1}{2}\|\overrightarrow{AB} \times \overrightarrow{BC}\|$. This suggests the following formula for barycentric coordinate of a point U with respect to A:

$$w_A = \frac{\|\overrightarrow{UB} \times \overrightarrow{UC}\|}{\|\overrightarrow{AB} \times \overrightarrow{BC}\|}.$$

Clearly, this formula works when U is one the vertices: if U is A, this ratio is 1, and if U is either B or C, the numeraotr becomes zero, which is correct. To generalize with points outside of the triangle, we use "signed areas" instead, and replace the norms by multiplication with the normal $\overrightarrow{n}$ of the triangle:

$$w_A = \frac{(\overrightarrow{UB} \times \overrightarrow{UC}) \cdot \overrightarrow{n}}{(\overrightarrow{AB} \times \overrightarrow{BC}) \cdot \overrightarrow{n}}.$$

This allows us to get negative weights. Let us now prove this formula. Let (w_A, w_B, w_C) be the barycentric coordinates of U. Then:

$$\begin{aligned}
\overrightarrow{UB} &= \overrightarrow{OB} + \overrightarrow{UO} \\
&= \overrightarrow{OB} + (w_A\overrightarrow{AO} + w_B\overrightarrow{BO} + w_C\overrightarrow{CO}) \\
&= (1 - w_B)\overrightarrow{OB} + w_A\overrightarrow{AO} + w_C\overrightarrow{CO} \\
&= (w_A + w_C)\overrightarrow{OB} + w_A\overrightarrow{AO} + w_C\overrightarrow{CO} \\
&= w_A\overrightarrow{AB} + w_C\overrightarrow{CB},
\end{aligned}$$

where we used the fact that the sum of weights is 1. Similarly, one can show that $\overrightarrow{UC} = w_A\overrightarrow{AC} + w_B\overrightarrow{BC}$. Plugging this into the numerator of the formula, we get

$$\begin{aligned}\overrightarrow{UB} \times \overrightarrow{UC} &= (w_A\overrightarrow{AB} + w_C\overrightarrow{CB}) \times (w_A\overrightarrow{AC} + w_B\overrightarrow{BC}) \\ &= w_A w_A(\overrightarrow{AB} \times \overrightarrow{AC}) + w_A w_B(\overrightarrow{AB} \times \overrightarrow{BC}) + w_C w_A(\overrightarrow{CB} \times \overrightarrow{AC}) + w_B w_A(\overrightarrow{CB} \times \overrightarrow{BC}) \\ &= w_A \left(w_A(\overrightarrow{AB} \times \overrightarrow{AC}) + w_B(\overrightarrow{AB} \times \overrightarrow{BC}) + w_C(\overrightarrow{CB} \times \overrightarrow{AC}) \right) \\ &= w_A\overrightarrow{AB} \times \overrightarrow{AC},\end{aligned}$$

since $\overrightarrow{CB} \times \overrightarrow{BC} = \overrightarrow{0}$, $\overrightarrow{AB} \times \overrightarrow{AC} = \overrightarrow{AB} \times \overrightarrow{BC} = \overrightarrow{CB} \times \overrightarrow{AC}$ and $w_A + w_B + w_C = 1$. This proves the formula given outputs the correct barycentric coordinates for any point U in the plane. A similar formula applies for w_B, then w_C can be deduced since the sum is 1.

A.3.1. Implementation for triangles in 2D

When working in a 2D context, formulas derived in 3D can be applied assuming everything is in the plane $z = 0$. The normal vector of the triangle is the z basis vector, and cross producs output a vector in this direction. In terms of code, cross-product can be implemented as a function of two vectors that outputs a scalar. A direct application of the formula above can be done while omitting the multiplication by the normal vector:

```
Vector2d ab, ac, ub, uc, ua;
ab = b - a;
ac = c - a;
ub = b - u;
uc = c - u;
ua = a - u;

auto twiceArea = crossProduct2D(ab, ac);
PRECICE_ASSERT(twiceArea != 0, "It seems a degenerate triangle was sent.");
scaleFactor = 1.0 / twiceArea;

barycentricCoords(0) = crossProduct2D(ub, uc) * scaleFactor;
barycentricCoords(1) = crossProduct2D(uc, ua) * scaleFactor;
barycentricCoords(2) = 1 - barycentricCoords(0) - barycentricCoords(1);
```

A.3.2. Implementation in 3D

When working in 3D, a projection step is needed: point U does not necessary lie on the plane. We first construct a projection P then compute the coefficients of the projected point. Thus, the coefficient w_A is

$$w_A = \frac{(\overrightarrow{PB} \times \overrightarrow{PC}) \cdot \overrightarrow{n}}{(\overrightarrow{AB} \times \overrightarrow{BC}) \cdot \overrightarrow{n}}.$$

However, there is no need to explicitly construct P. The point U can be used with correct results, since

$$\begin{aligned}
\overrightarrow{PB} \times \overrightarrow{PC} &= (\overrightarrow{PU} + \overrightarrow{UB}) \times (\overrightarrow{PU} + \overrightarrow{UC}) \\
&= \overrightarrow{PU} \times \overrightarrow{PU} + \overrightarrow{PU} \times \overrightarrow{UC} + \overrightarrow{UB} \times \overrightarrow{PU} + \overrightarrow{UB} \times \overrightarrow{UC} \\
&= \overrightarrow{PU} \times \overrightarrow{UC} + \overrightarrow{UB} \times \overrightarrow{PU} + \overrightarrow{UB} \times \overrightarrow{UC}.
\end{aligned}$$

When multiplying by the normal vector, which is parallel to $\overrightarrow{PU}$, we can eliminate extra terms and see that

$$(\overrightarrow{PB} \times \overrightarrow{PC}) \cdot \overrightarrow{n} = (\overrightarrow{UB} \times \overrightarrow{UC}) \cdot \overrightarrow{n}.$$

This means that the usual formula can be applied with U as if it were in the plane, which reduces the number of computations needed. Furthermore, the normal vector can be substituted by $\overrightarrow{AB} \times \overrightarrow{BC}$, which is the same up to a constant, but this constant appears on both the numerator and denominator. In terms of code, this gives:

```
Vector3d ab, ac, au, n;

ab          = b - a;
ac          = c - a;
n           = ab.cross(ac);
auto nDotN = n.dot(n);
PRECICE_ASSERT(nDotN != 0, "It seems a degenerate triangle was sent.");
scaleFactor = 1.0 / nDotN;

// varying per point
au = u - a;

barycentricCoords(2) = n.dot(ab.cross(au)) * scaleFactor;
barycentricCoords(1) = n.dot(au.cross(ac)) * scaleFactor;
barycentricCoords(0) = 1 - barycentricCoords(1) - barycentricCoords(2);
```

A.4. Tetrahedra

Similarly to the case of triangles, the barycentric coordinates of a point U in a tetrahedron $ABCD$ can be seen as fractions of volume. Once again, this princple leads to a formula that generalizes well with negative weights for a point outside of the tetrahedron. The volume of a tetrahedron is a third of the product of the base and the height, i.e. the product of the area of a triangle and the distance between this triangle and the fourth point. Since the cross product of two edges of a triangle produces a perpendicular vector whose magnitude is twice the area of the triangle, we can express the (signed) volume as $\frac{1}{6}(\overrightarrow{AB} \times \overrightarrow{BC}) \cdot \overrightarrow{AD}$. Sign is a function of the ordering of nodes. Getting the correct sign does not really matter, since we compute ratios of volumes. If the convention is consistent, an error on the sign would cancel itself.

Using this, we conjecture that the barycentric coefficient with respect to A is given by

$$w_A = \frac{(\overrightarrow{UB} \times \overrightarrow{BC}) \cdot \overrightarrow{UD}}{(\overrightarrow{AB} \times \overrightarrow{BC}) \cdot \overrightarrow{AD}}$$

and similarly for other vertices. To prove it, we expand $\overrightarrow{UB}$ as $w_A\overrightarrow{AB} + w_C\overrightarrow{CB} + w_D\overrightarrow{DB}$ with a similar reasoning as the one used with triangles. Similarly, $\overrightarrow{UD}$ is $w_A\overrightarrow{AD} + w_B\overrightarrow{BD} + w_C\overrightarrow{CD}$. Using this, one can easily compute

$$\begin{aligned}\overrightarrow{UB} \times \overrightarrow{BC} &= w_A(\overrightarrow{AB} \times \overrightarrow{BC}) + w_C(\overrightarrow{CB} \times \overrightarrow{BC}) + w_D(\overrightarrow{DB} \times \overrightarrow{BC}) \\ &= w_A(\overrightarrow{AB} \times \overrightarrow{BC}) + w_D(\overrightarrow{DB} \times \overrightarrow{BC})\end{aligned}$$

with the central term vanishing since it is a cross-product of parallel vectors. Then, dot product with $\overrightarrow{UD}$ yields

$$\begin{aligned}(\overrightarrow{UB} \times \overrightarrow{BC}) \cdot \overrightarrow{UD} &= \Big(w_A(\overrightarrow{AB} \times \overrightarrow{BC}) + w_D(\overrightarrow{DB} \times \overrightarrow{BC})\Big) \cdot (w_A\overrightarrow{AD} + w_B\overrightarrow{BD} + w_C\overrightarrow{CD}) \\ &= w_Aw_A(\overrightarrow{AB} \times \overrightarrow{BC}) \cdot \overrightarrow{AD} + w_Aw_B(\overrightarrow{AB} \times \overrightarrow{BC}) \cdot \overrightarrow{BD} + w_Aw_C(\overrightarrow{AB} \times \overrightarrow{BC}) \cdot \overrightarrow{CD} \\ &+ w_Dw_A(\overrightarrow{DB} \times \overrightarrow{BC}) \cdot AD + w_Dw_B(\overrightarrow{DB} \times \overrightarrow{BC}) \cdot \overrightarrow{BD} + w_Dw_C(\overrightarrow{DB} \times \overrightarrow{BC}) \cdot \overrightarrow{CD}.\end{aligned}$$

The last two terms vanish, since the cross-products yield vectors parallel to the normal of triangle BCD, which means the dot-product with a vector inside this triangle is zero. The 4 remaining triple products are all the same:

- $(\overrightarrow{AB} \times \overrightarrow{BC}) \cdot \overrightarrow{BD} = (\overrightarrow{AB} \times \overrightarrow{BC}) \cdot \overrightarrow{AD}$. This can be shown by expanding $\overrightarrow{BD}$ as $\overrightarrow{BA} + \overrightarrow{AD}$, with $\overrightarrow{BA}$ orthogonal to the cross-product.
- Similarly, $(\overrightarrow{AB} \times \overrightarrow{BC}) \cdot \overrightarrow{CD} = (\overrightarrow{AB} \times \overrightarrow{BC}) \cdot \overrightarrow{AD}$.
- $(\overrightarrow{DB} \times \overrightarrow{BC}) \cdot \overrightarrow{AD} = (\overrightarrow{AB} \times \overrightarrow{BC}) \cdot \overrightarrow{AD}$. It can be proven by expanding $\overrightarrow{DB}$ into $\overrightarrow{DA} + \overrightarrow{AD}$.

Using this and the sum of weights being 1, we can show that $(\overrightarrow{UB} \times \overrightarrow{BC}) \cdot \overrightarrow{UD} = w_A\Big((\overrightarrow{AB} \times \overrightarrow{BC}) \cdot \overrightarrow{AD}\Big)$, which proves the formula.

In terms of code, barycentric coordinates of a point with respect to a tetrahedron can be implemented like this:

```
Vector4d barycentricCoords;

// Varying per point
Vector3d au = u - a;
Vector3d du = u - d;
Vector3d cu = u - c;

// Necessary to compute triangles
Vector3d ab = b - a;
Vector3d ac = c - a;
Vector3d ad = d - a;
Vector3d bc = c - b;

// Triangles' normal vectors
Vector3d abc = ab.cross(bc);
Vector3d abd = ab.cross(-ad);
Vector3d acd = ac.cross(ad);

// Volume is signed
auto volume = abc.dot(ad);

barycentricCoords(3) = abc.dot(au) / volume;
barycentricCoords(2) = abd.dot(du) / volume;
barycentricCoords(1) = acd.dot(cu) / volume;
barycentricCoords(0) = 1 -
                       barycentricCoords(3) -
                       barycentricCoords(2) -
                       barycentricCoords(1);
```

Glossary

black-box coupling is a coupling between two or more solvers where each solver has no knowledge of the internals of the others, and only interacts with its output. 3, 47, 59

conservative mapping is a mapping that preserves total quantities. It is appropriate for extensive quantities such as forces. 10, 59

consistent mapping is a mapping such that, if the data to map is constant on the input mesh, the output is guaranteed to be that constant for all points of the output mesh. In general, consistent mappings are implemented as an interpolation. 10, 59

data mapping is the conversion of data from one mesh to another. It often takes the form of interpolation, with consistent mapping, but also includes conservative mapping where data must be "distributed" between output points. 3

explicit coupling is a coupling where each data is exchanged once per time window, and each solver performs time steps only once, without iterations prescribed by preCICE. 3, 8

implicit coupling is a coupling where a global system of equations must be solved over the whole system. In black-box coupling, this is solved through iterative methods, since no participant has a complete knowledge of the system to solve . 3, 8, 9

partial differential equations are differential equations involving functions of more than one variable, often time and space. 3, 60

scaled-consistent mapping is a consistent mapping where the output is multiplied by a corrective coefficient to preserve integrals. Typical example is pressure: a conservative mapping is inappropriate, unlike with forces, but the total force, which is the integral of the pressure, must be conserved. A consistent mapping can change this integral due to interpolation errors. 11

surface coupling is a coupling where data is exchanged at an interface, which is part of the boundary of each solver. Exchanged data is usually used afterwards as boundary conditions. When doing 2D simulations, the interface is a curve. 4, 13, 14, 21, 30, 49

volumetric coupling is a coupling where data is exchanged over all the simulation domain, and different solvers work on overlapping domains. Exchanged data is usually used as a source term in the solved partial differential equations. 4, 14, 21, 29, 49

Bibliography

[1] preCICE documentation - Dealing with distributed meshes. http://precice.org/couple-your-code-distributed-meshes.html. Accessed: 2022-06-12.

[2] Boshra Ariguib. Second-order projection-based mapping methods for coupled multi-physics simulations. Bachelor's thesis, Institute for Parallel and Distributed Systems, University of Stuttgart, Apr. 2022.

[3] Nitish Arya. Volume coupling using precice for an aeroacoustic simulation. https://www.youtube.com/watch?v=TrZD9SnG2Tsy, Feb 2020. preCICE Workshop 2020.

[4] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. *SIGMOD Rec.*, 19(2):322–331, may 1990.

[5] G Chourdakis, K Davis, B Rodenberg, M Schulte, F Simonis, B Uekermann, G Abrams, HJ Bungartz, L Cheung Yau, I Desai, K Eder, R Hertrich, F Lindner, A Rusch, D Sashko, D Schneider, A Totounferoush, D Volland, P Vollmer, and OZ Koseomur. precice v2: A sustainable and user-friendly coupling library [version 1; peer review: 2 approved]. *Open Research Europe*, 2(51), 2022.

[6] Jean Clairambault. *Reaction-Diffusion-Advection Equation*, pages 1817–1817. Springer New York, New York, NY, 2013.

[7] Arved Enders-Seidlitz. Development of a coupled heat/gas flow model for crystal growth. https://www.youtube.com/watch?v=8ivCDyz2FlI, Feb 2022. preCICE Workshop 2022.

[8] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, Mar 1974.

[9] Richard Franke. A critical comparison of some methods for interpolation of scattered data, 1979-03.

[10] Bernhard Gatzhammer. *Efficient and Flexible Partitioned Simulation of Fluid-Structure Interactions*. Dissertation, Technische Universität München, München, 2014.

[11] Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[12] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, jun 1984.

[13] Joseph Kuan, Paul Lewis, and So Nh. Fast knearest neighbour search for r-tree family, 1997.

[14] Hans Petter Langtangen and Anders Logg. *Solving PDEs in Python*. Springer, 2017.

[15] Florian Lindner. *Data transfer in partitioned multi-physics simulations: interpolation & communication*. PhD thesis, Institute for Parallel and Distributed Systems, University of Stuttgart, 2019.

[16] B Maier, N Emamy, A Krämer, and M Mehl. Highly parallel multi-physics simulation of muscular activation and EMG. pages 610–621, 2019.

[17] Benjamin Rodenberg, Ishaan Desai, Richard Hertrich, Alexander Jaust, and Benjamin Uekermann. FEniCS-preCICE: Coupling FEniCS to other simulation software. *SoftwareX*, 16:100807, 2021.

[18] Julian Schließus and Louis Gagnon. Data for: Create a Fluid-Structure Simulation Framework for Cycloidal Rotors, 2022.

[19] Boris Schling. *The Boost C++ Libraries*. XML Press, 2011.

[20] Arpan Sicar. Volumetric coupling of openfoam for multi-physics simulations of fusion reactor blankets. https://www.youtube.com/watch?v=_NKCGny-cJE, Feb 2022. preCICE Workshop 2022.

[21] Frédéric Simonis, Gerasimos Chourdakis, and Benjamin Uekermann. Overcoming complexity in testing multiphysics coupling software. https://bssw.io/blog_posts/overcoming-complexity-in-testing-multiphysics-coupling-software, Feb 2022.

[22] Benjamin Walter Uekermann. *Partitioned Fluid-Structure Interaction on Massively Parallel Systems*. Dissertation, Technische Universität München, München, 2016.

[23] Dominik Volland. Coupling TherMoS with preCICE. Master's thesis, Technical University of Munich, Jul 2019.

[24] Lucia Cheung Yau. Conjugate heat transfer with the multiphysics coupling library preCICE. Master's thesis, Department of Informatics, Technical University of Munich, Dec. 2016.