Chapter 4

On Structured Overlapping Grids

A major task for calculating the approximate numerical solution of a set of partial differential equations (PDEs) on complex domains, is the problem of mesh or grid generation. Different methods of spatial discretization exist, including Cartesian meshes, unstructured meshes and block structured body-fitted conforming grids. In this chapter, the structured overlapping grids method (which falls within the block structured body-fitted conforming grids classification) is briefly reviewed and discussed in the context of a methodology for the solution and analysis of flows around complex geometries and moving/deforming bodies.

4.1 Approaches to Grid Generation

Grid generation can be defined as the process of breaking up a continuous physical domain into smaller discrete sub-domains, in order to compute the numerical approximate solution of a PDE. The grid generation methods can be classified as structured (body-fitted grids), unstructured (body-fitted grids) or Cartesian (non-body-fitted grids). Hereafter, the different grid generation methods will be briefly surveyed and presented. Since the purpose of this section is to present various techniques to generate grids, the detailed and theoretical derivations for those techniques will be avoided as much as possible.

There is a large body of literature [50, 185, 186] and software packages [137, 162] dealing with structured grid generation. Structured grid methods take their name from the fact that the grid is laid out in a regular repeating pattern called a block. Strictly speaking, in a structured grid the computational domain is selected to be rectangular in shape where the interior grid points are distributed along grid lines, therefore, the grid points can be identified easily with reference to the appropriate grid lines. These types of grids use quadrilateral elements in $2\mathbb{D}$ and hexahedral elements in $3\mathbb{D}$. Algebraic methods, elliptic methods and hyperbolic methods are often employed to generate these grids [84, 85, 185, 186], complex iterative smoothing techniques are also used to align elements with boundaries or physical domains in order to improve the orthogonality and uniformity. Where non-trivial boundaries are required, block structured techniques can be employed which allow the user to break the domain up into several aligned topological blocks, obtaining in this way a multiblock grid. While multiblock grids give the user more freedom in constructing the mesh, the block connection requirements can be restrictive and are often difficult to construct.

There is another block structured grid method which seeks to avoid the problems associated with

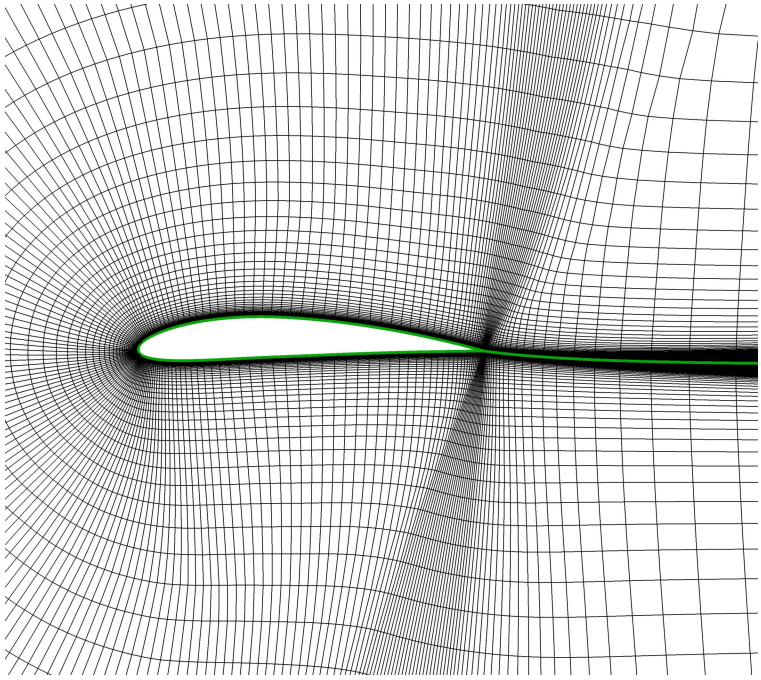

Figure 4.1: *Single-block C-type structured grid around a NACA 4412 airfoil.*

block connections in multiblock grids. Structured overlapping grid methods allow the individual blocks to conform to the physical boundaries, but, different from the multiblock grids, the blocks boundaries not necessary have to be aligned, they are allow to overlap. Sophisticated grid assembly tools are used to compute domain connectivity information and to remove unnecessary grid points. What these methods gain in user convenience, they usually give up in solution accuracy. However, these methods are very efficient when dealing with geometries which would be too daunting a task with conventional block structured methods or when dealing with moving bodies (*e.g.*, helicopters with moving rotor blades and aircraft store separation).

Structured grids enjoy a considerable advantage over other grid methods in that they allow the user a high degree of control. Because the user places control points and edges interactively, she/he has total freedom when positioning the mesh. In addition, hexahedral and quadrilateral elements, which are very efficient at filling space, support a high amount of skewness and stretching before the solution is significantly affected. This allows the user to naturally concentrate points in regions of high gradients in the flowfield and to coarsen the grid away from these areas. Also, because the user interactively lays out the elements, the grid is most often flow aligned, thereby yielding greater accuracy within the solver. Structured flow solvers typically require the lowest amount of memory for a given grid size and execute faster because they are optimized for the structured layout of the grid.

The major drawback of structured grids is the time and expertise required to lay out an optimal grid for a complex geometry. Often this comes down to past user experience and brute force placement of control points and edges is required. Grid generation times are usually measured in days if not weeks.

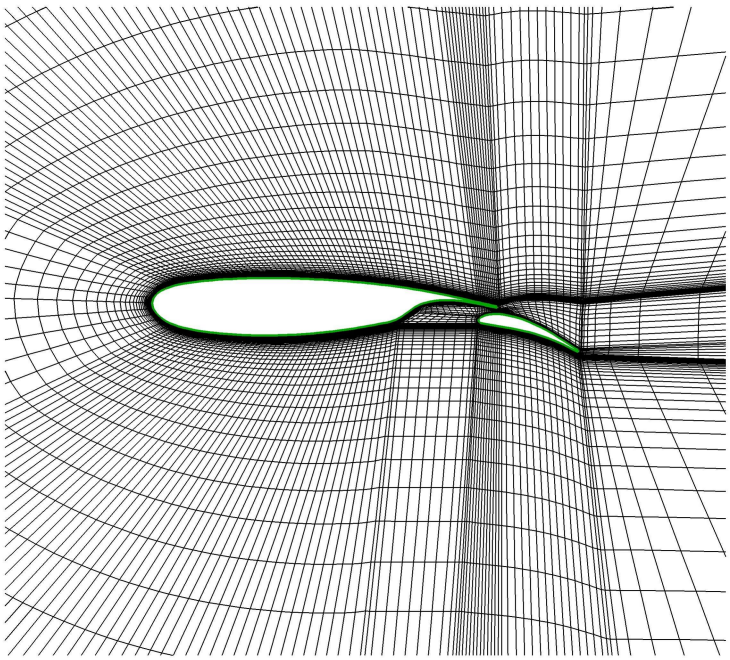

Figure 4.2: *Multi-block structured grid around a NLR 7301 airfoil with flap.*

Unstructured mesh methods [19, 137, 162, 164, 186], on the other hand, use an arbitrary collection of elements to fill the domain. Because the arrangement of elements have no recognizable pattern, the mesh is called unstructured. These types of meshes typically use triangles in $2\mathbb{D}$ and tetrahedrals in $3\mathbb{D}$, although quadrilateral and hexahedral meshes can also be used. As with structured grids, the elements can be stretched and twisted to fit the domain. These methods have the ability to be automated to a very large degree. Given a good CAD model, a good mesher can automatically place triangles on the surfaces and tetrahedrals in the volume with very little input from the user. The automatic meshing algorithm typically involves meshing the boundaries and then either adding elements touching the boundary (advancing front methods) or adding points in the interior and reconnecting the elements (Delaunay methods).

The advantage of unstructured mesh methods is that they are very automated and therefore, require little user time or effort. The user need not worry about laying out block structures or connections. Additionally, unstructured mesh methods are well suited to inexperienced users because they require little user input and will generate a valid mesh under most circumstances. Mesh generation times are usually measured in minutes or hours.

The major drawback of unstructured meshes is the lack of user control when laying out the mesh. Typically any user involvement is limited to the boundaries of the mesh with the mesher automatically filling the interior. Triangle and tetrahedral elements have the problem that they do not stretch or twist well, therefore, the grid is limited to being largely isotropic, *i.e.* all the elements have roughly the same size and shape. This is a major problem when trying to locally refine the mesh, often the entire mesh must be made much finer in order to get the locally desired point

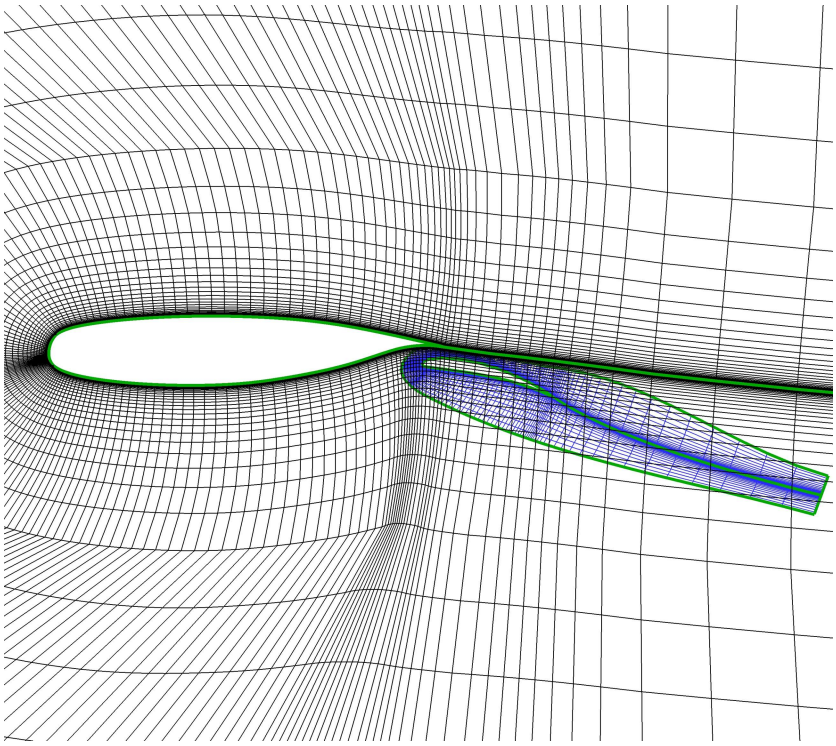

Figure 4.3: *Overlapping structured grid around a NLR 7301 airfoil with flap.*

densities. Unstructured flow solvers typically require more memory and have longer execution times than structured grid solvers on a similar mesh.

While both structured and unstructured approaches have enjoyed reasonable success in their application to real world problems, neither method has offered a truly fully automatic method for discretizing the domain around arbitrarily complex geometries. One reason for this stems from the fact that both techniques are body-fitted, *i.e.* cells neighboring the body must conform to the surface. This implies that the connectivity of the computational mesh is intimately linked to the body's geometry and topology. As a result, the surface mesh is subject to conflicting requirements of resolving both the local geometry and the expected flow variation.

Cartesian methods, as the name suggests, use a regular underlying Cartesian non-body-fitted grid. Solid objects are carved out from the interior of the mesh, leaving a set of irregularly shaped cells along the surface boundary. Early work with Cartesian grids used a stair-cased representation of the boundary. In contrast, modern Cartesian grids allow planar surface approximations at walls, and some even retain sub-cell descriptions of the boundary within the body-intersected cells. Obviously, this additional complexity places a greater burden on the flow solver, and recent research has focused on developing numerical methods to accurately integrate along the surface boundaries of a Cartesian grid. Since most of the volume mesh is completely regular, highly efficient and accurate flow solvers can be used. All the overhead for the geometric complexity is at the boundary, where the Cartesian cells are cut by the body.

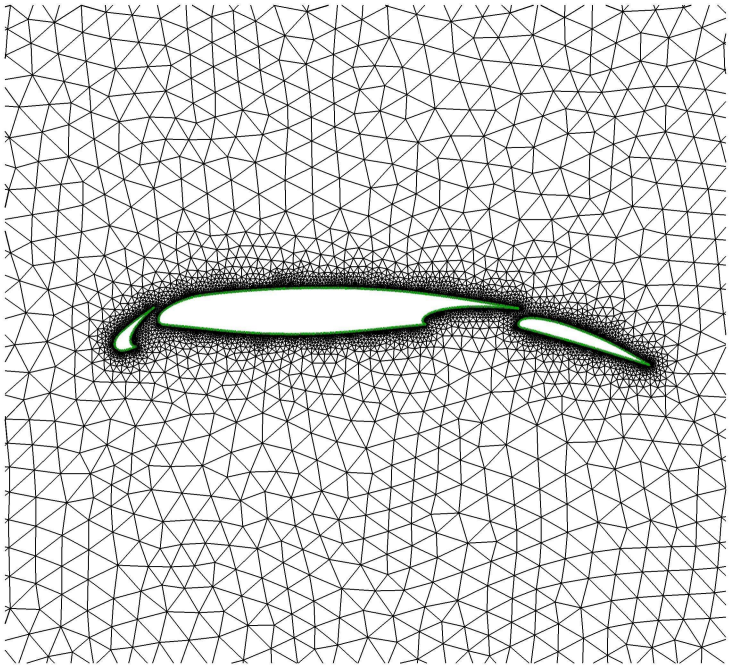

Figure 4.4: *Unstructured mesh around a NHLP-2D three element airfoil.*

Although Cartesian grid methods date back to the 1970s, it was only with the advent of adaptive mesh refinement (AMR) that their use became practical [1, 18]. Without some provision for grid refinement, Cartesian grids would lack the ability to efficiently resolve fluid and geometry features of various sizes and scales. This resolution is readily incorporated into structured meshes via grid point clustering. Many algorithms for automatic Cartesian grid refinement have, however, been developed in the last decade, largely alleviating this shortcoming. A fairly extensive literature on the flow solvers developed for Cartesian grids with embedded adaptation is available, for a more thorough discussion of Cartesian mesh topics refer to [1, 123, 137, 162].

The last decade has witnessed a resurgence of interest in Cartesian mesh methods. In contrast to body-fitted structured or unstructured methods, Cartesian grids are inherently non-body-fitted. This characteristic promotes extensive automation, dramatically eases the burden of surface preparation, and greatly simplifies the re-analysis processes when the topology of a configuration changes. By taking advantage of these important characteristics, well-designed Cartesian approaches virtually eliminate the difficulty of grid generation for complex configurations. Typically, meshes with millions of cells can be generated in minutes on a modest workstations [1, 2].

The most serious current drawback of Cartesian grids is that their use is restricted to inviscid or low Reynolds number flows. An area of active research is their coupling to prismatic grids or other methods for incorporating boundary layer zoning into the Cartesian grid framework [2].

The approach discussed in this chapter, using overlapping grids, may be viewed as a combination of Cartesian grids and structured grids methods. Body-fitted conforming structured grids are used in order to achieve high-quality representations of near-body boundaries. At the same

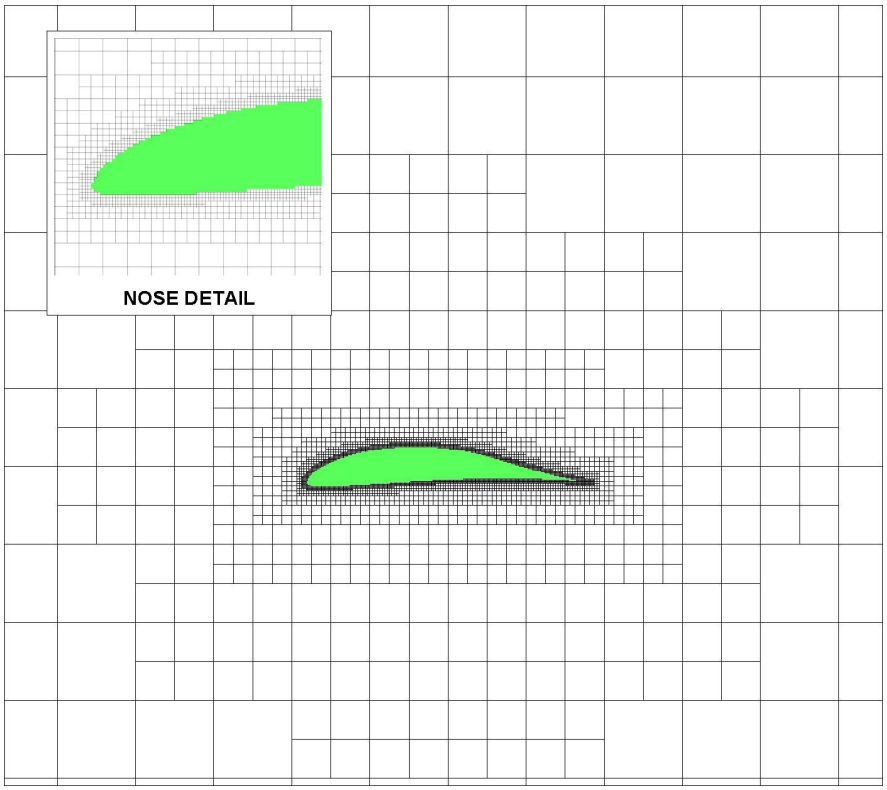

Figure 4.5: *Cartesian grid around a Drela DAE11 low Reynolds number airfoil.*

time, the majority of grid points in an overlapping grid system tend to belong to Cartesian grids (off-body) so that the numerical and computational efficiencies inherent with such grids can be exploited. In table 4.1, some of the advantages and disadvantages of some of the currently used grid generation methods are listed.

4.2 Overview and Historical Background of the Structured Overlapping Grids Method

The overlapping grids method, also known as overset composite grids or Chimera grids (named like this after the composite monster of Greek mythology), provides a flexible and efficient spatial discretization method for numerically solving a PDE on a general $1\mathbb{D}$, $2\mathbb{D}$ or $3\mathbb{D}$ domain.

The structured overlapping grids method consists in generating a set of body-fitted conforming structured components grids that completely cover the physical domain that is being modeled and overlap where they meet [141] (see figure 4.6). Reducing in this way a single, complex domain into a series of smaller, potentially simpler ones. The governing PDEs are solved separately on each component grid and domain connectivity is obtained through proper interpolation in the overlapping areas. The geometry of the components of the domain can be defined individually and hence the grids around them can be generated separately. Body-fitted conforming grids are

Gridding method	Advantages	Disadvantages
Cartesian	Small memory and CPU requirements which results in very fast flow solvers; easiness in the grid generation process; does not requires much user experience; easy to parallelize; user grid generation process is almost automatic, it requires little user time and effort	Difficulties in resolving boundary layers; difficulties in resolving complex boundaries
Single Structured	Small memory and CPU requirements which results in fast flow solvers; very well fitted for viscous flow computations; can handle very efficiently trivial geometries, accurate representation of boundaries	Restricted to simple geometries; requires user experience; not amenable to local refinement schemes, user grid generation process can be time consuming
Block Structured	Small memory and CPU requirements which results in fast flow solvers; very well fitted for viscous flow computations; can efficiently handle complex geometries; provides natural domain decomposition, accurate representation of boundaries	Requires a lot of user experience; user grid generation process is very time consuming, not amenable to local refinement schemes; requires block connectivity information
Unstructured	Easiness in grid generation process; they do not require a lot of user input or experience; user grid generation process is almost automatic, it requires little user time and effort; can efficiently handle very complex geometries; allows arbitrary use of element types; allows adaptivity and local refinement, accurate representation of boundaries	Large memory and CPU requirements which results in slower flow solver; not suited for viscous flow computations
Overlapping	Small memory and CPU requirements which results in fast flow solvers; very well fitted for viscous flow computations; can efficiently handle complex and moving geometries; eases grid generation burden for very complex geometries or when working in collaborative environments; provides natural domain decomposition, accurate representation of boundaries	Requires a lot of user experience; user grid generation process can be very time consuming; non-conservative interpolation issues in overlapping area; often needs separate tools to handle domain connectivity information

Table 4.1: *Some of the currently used grid generation methods.*

used near the components boundaries while one or more background Cartesian grids are used to handle the rest of the domain, all without any constraints on the grid boundaries as long as overlap exists between adjacent grids. While originally developed as a means to address complex geometries [128, 139], the overlapping grids method have also been employed to simulate multiple bodies in relative motion [121, 149] and to resolve fine-scale flow features through the use of adaptive mesh refinement (AMR) [73]. The use of structured grids, together with the use of optimized discretizations for large regions typically covered by Cartesian grids and the intrinsic domain decomposition nature of the overlapping grids methodology, leads to an efficient method in both computer time and computer memory, highly scalable to parallel computing platforms [149]. Finally, the overlapping grids method is advantageous for performing simulations in a production environment, where component grids for a complex system may be developed concurrently by different team members, libraries of grids of common grid components may be developed for reusability, and small changes may be quickly incorporated into a grid system by modifying only the impacted component grids and not the entire grid system [13].

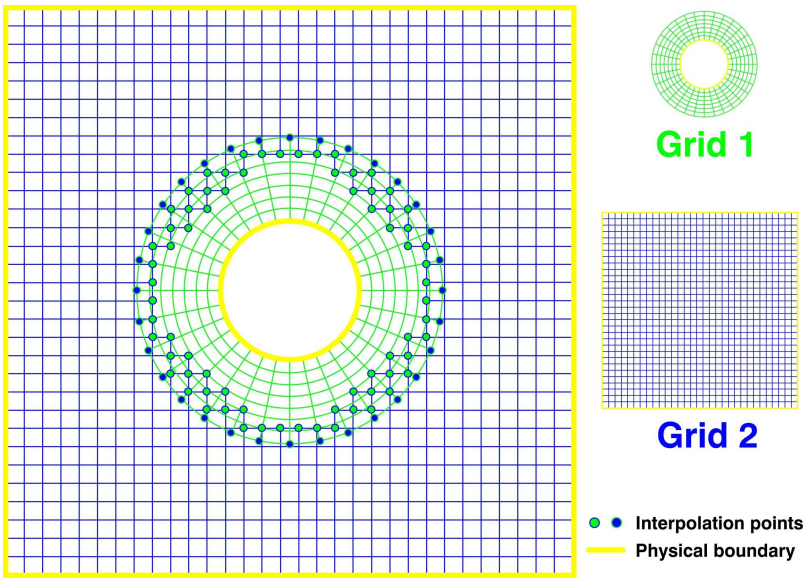

Figure 4.6: *Simple overlapping grid system in physical space $\mathcal{P}$.*

In many ways, the overlapping grids method is similar to the so-called patched or block structured approach. What differentiates overlapping grids from multiblock grids is that alignment constraints set in multiblock grids are relaxed. Overlapping grids, are only required to overlap so that no part of the computational domain is left uncovered. Clearly, the discretization becomes more complicated at overlap boundaries, but the flexibility of having smooth overlapping grids seems to be worthwhile.

The overlapping grids method has been in use for some time. Apparently, the first use of overlapping grids was described by Volvov in 1966 [203, 204], who considered approximations to Poisson's equation on regions with corners. The method was further developed and promoted by Starius and Kreiss. Starius, in 1977, looked at the convergence of elliptic problems on two overlapping meshes using the Schwarz alternating procedure [174]. In a later paper ([175]), he considered

the numerical solution of hyperbolic problems on overlapping grids. In [175], the stability of the Lax-Wendroff method was shown for a model problem on a one-dimensional overlapping grid. Moreover, he solved the shallow water equations in a two-dimensional basin, showing that despite the overlap the mass was conserved to within a few percent. Later on, a method for the construction of composite meshes and the solution of hyperbolic PDEs was described by Kreiss [104]. Also of interest is the work of Berger [17], where she indicates how to obtain conservative difference approximations at grid interfaces. However, despite the fact that the method had been around for a while, it was first introduced into the CFD community about two decades ago by Steger *et al.* [176] and Benek *et al.* [15]; and it has been further developed by Meakin and Suhs [122], Chesshire and Henshaw [37] and Noack *et al.* [133, 130, 131, 132]. It is now recognized as an attractive approach for treating problems with complex geometries and moving boundaries. The solution process discussed in this dissertation uses a grid system that discretizes the problem domain by using separately generated but overlapping body-fitted conforming structured grids; however, the use of unstructured meshes have also been considered by Togashi *et al.* [190] and Wang and Kannan [208]. In industry and academia, the overlapping grids method has been used to solve a wide variety of problems in fields such as: aerodynamics [139, 149], rotor dynamics [41, 172], combustion [26], fluid-structure interaction [56, 179], reactive flow with detonations [73, 77], incompressible flows [78], biological flows [100], non-Newtonian flows [52] and flows with deforming boundaries [51, 144], to name a few. For example, a Reynolds-averaged Navier-Stokes calculations for a prototype Martian rotorcraft vehicle was carried out successfully using the overlapping grids method [41], in this study, solutions for hovering rotor performance at Mars experimental flow conditions were produced for a series of collective pitch angles, in a moving grid system of about 10 millions grids points. Also, the unsteady viscous flow around a V-22 tiltrotor helicopter in high-speed forward flight was solved using moving overlapping grids [121].

The maturation process for overlapping grids generation tools is ongoing and is an area of active research. Historically, users of the overlapping grids method have used grid generation tools designed for block structured grids to generate required component grids to be used in the overlapping grid system. It is just recently, that grid generation tools that exploit the flexibility inherent to overlapping grids have been efficiently implemented and coupled with flow solvers. Table 4.2 lists some of the codes that are currently available for assembling overlapping grid systems. But in general, the major distinguishing features between these different approaches to overlapping grids generation lie in the grid construction algorithm, the manner of performing interpolation, the data structures, amount of user data input and the details of implementation.

In this dissertation, the Ogen[1] grid generator [70], is used to assemble the overlapping grid system. In Ogen, the user may first generate the component grids that describe the geometry or may import the component grids into it in a readable format. The overlapping grid then is constructed. This latter step consists of determining how the different component grids interpolate from each other, removing grid points from holes in the domain and eliminating unnecessary grid points in regions of excess overlap.

[1] https://computation.llnl.gov/casc/Overture/

Code	Comments
DiRTlib [131, 132]	Donor interpolation Receptor Transaction library. It is a solver neutral library designed to provide the required capability for using overlapping grids in any general flow solver. It is designed with the idea of minimizing the number of modifications required in the flow solver. It is not freely available.
SUGGAR [130, 131]	Structured, Unstructured, Generalized overset Grid AssembleR. It is used to build domain connectivity information for the wide range of grid topologies and solver formulations in codes that use DiRTlib. It can handle moving bodies simulations. It is not freely available.
Chimera Grid Tools [33, 34]	Chimera Grid Tools (CGT) is a software package containing a variety of tools for generating overlapping grids for solving complex configuration problems. It is part of the OVERFLOW-D general purpose Navier-Stokes solver. It can handle moving bodies simulations. It is not freely available.
BEGGAR [10, 116]	It is a flow solution environment, specially designed for store separation problems. It provides automated grid assembly. It is specially targeted to production work environments. It can handle moving bodies simulations and 6-DOF rigid body motion. It is not freely available.
Ogen [70, 74]	It is part of the Overture framework, which is a collection of C++ libraries for solving PDEs on overlapping grids. Provides tools for structured grid generation and overlapping grids assembly. It can handle moving bodies simulations and adaptive mesh refinement on moving bodies. It is freely available for research and academic purposes.

Table 4.2: *Some of the codes that are currently available for assembling overlapping grid systems.*

4.3 Problem Formulation

Let us suppose we want to solve some PDE on a domain $\mathcal{D}$ in $\mathbb{N}$ space dimensions ($\mathbb{N}$=1, 2 or 3), using overlapping grids. Then, an overlapping grid system $\mathbb{G}$ of the domain $\mathcal{D}$, consists of a set of $\mathcal{N}$ structured component grids $\mathcal{G}_g$,

$$\mathbb{G} = \{\mathcal{G}_g\}, \qquad g = 1, 2, ..., \mathcal{N}$$

that entirely cover the domain $\mathcal{D}$ and overlap where the component grids $\mathcal{G}_g$ meet. Each component grid is a logically rectangular structured grid in $\mathbb{N}$ space dimensions and is defined by a smooth mapping $\mathbf{M}_g$ from the computational space $\mathcal{C} = \mathcal{C}(\xi, \eta, \zeta, \tau)$ (the unit interval for $1\mathbb{D}$ applications, unit square for $2\mathbb{D}$ applications and an equivalent hexahedral domain for $3\mathbb{D}$ applications) to the physical space $\mathcal{P} = \mathcal{P}(x, y, z, t)$, such that

$$\mathcal{P} = \mathbf{M}_g(\mathcal{C}), \qquad \mathcal{C} \in [0, 1]^{\mathbb{N}}, \qquad \mathcal{P} \in \Re^{\mathbb{N}}$$

Here $\mathcal{P}$ is equal to $\mathbf{x} = (x, y, z)$ for $\mathbb{N} = 3$ and contains all the coordinates in physical space and $\mathcal{C}$ is equal to $\mathbf{r} = (\xi, \eta, \zeta)$ for $\mathbb{N} = 3$ and contains the logically uniform array in computational space. Variables defined on a component grid, such as the coordinates of the grid points, are stored in rectangular arrays. For example, grid vertices are represented as the array

$$\mathcal{P}^g_{\mathbf{i}} : \text{grid vertices}, \qquad \mathbf{i} = (i_1, \ldots, i_{\mathbb{N}}), \qquad i_\alpha = 0, \ldots, N^g_\alpha, \qquad \alpha = 1, \ldots, \mathbb{N}$$

where N^g_α is the number of grid points in the i_α-*coordinate* direction. In the case of Cartesian grids, the grid vertex information and other mapping information is not stored, which results in a considerable savings in memory use.

Figure 4.7 shows a simple overlapping grid system consisting of two component grids, an annular boundary fitted grid and a background Cartesian grid. The top view shows the overlapping grid system in physical space $\mathcal{P}$ while the bottom view shows each grid in computational space $\mathcal{C}$. In this example the annular component grid cuts a hole in the Cartesian background grid, so that the latter grid has a number of unused points (chimera hole). These unused points are tagged and no computation is performed there. They are either outside of the computational domain $\mathcal{C}$, or are eliminated to make the total number of grid points in the overlapping grid system $\mathbb{G}$ smaller. The other points on the component grids are marked as either discretization points or interpolation points. The discretization points are those where the discretization of the governing equations or boundary conditions is applied and the interpolation points provide domain connectivity by interpolating their solution values back-and-forth between the different overlapping grids.

The classification of points on a component grid into discretization, interpolation and unused points and as well the computation of all the metrics and Jacobians used when forming discrete approximations is done by Ogen [70], which is highly optimized to treat overlapping grids and moving overlapping grids. Ogen takes as input a set of overlapping component grids along with a classification of the boundaries of each grid as a physical boundary, an interpolation boundary or a periodic boundary. This boundary information is held in a generic array $\text{flag}_g(\beta, \alpha)$, where $\beta = 1$ or 2 denotes the boundary side and $\alpha = 1, \ldots, \mathbb{N}$ is the i_α-*coordinate* direction, *i.e.*,

$$\text{flag}_g(\beta, \alpha) = \begin{cases} > 0 & \text{physical boundary} \\ = 0 & \text{interpolation boundary} \\ < 0 & \text{periodic boundary} \end{cases}$$

and

$$\begin{aligned} &\text{flag}_g(1,1) = \text{left side} \\ &\text{flag}_g(2,1) = \text{right side} \\ &\text{flag}_g(1,2) = \text{bottom side} \\ &\text{flag}_g(2,2) = \text{top side} \\ &\text{flag}_g(1,3) = \text{front side (for } 3\mathbb{D} \text{ domains)} \\ &\text{flag}_g(2,3) = \text{back side (for } 3\mathbb{D} \text{ domains)} \end{aligned}$$

Physical boundaries are discretization points. Interpolation boundaries are non-physical boundaries where the grid generator will attempt to interpolate the points from other components grids.

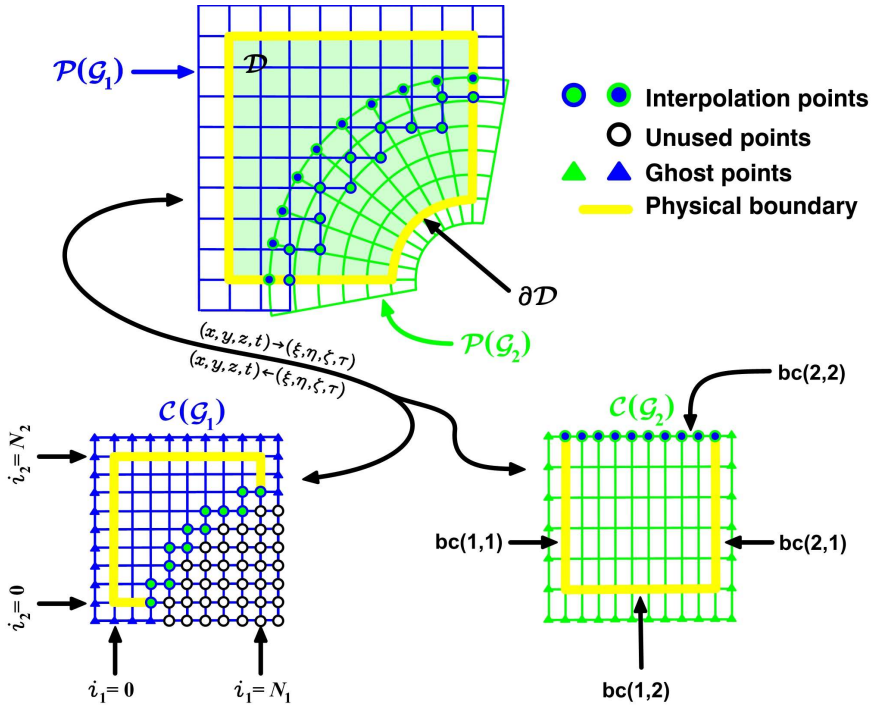

Figure 4.7: *Simple overlapping grid system consisting of two component grids $\mathcal{G}_g$. An annular boundary fitted grid ($\mathcal{G}_2$) and a background Cartesian grid ($\mathcal{G}_1$). The top view shows the overlapping grid in physical space $\mathcal{P}$ while the bottom view shows each grid in computational space $\mathcal{C}$.*

A periodic boundary can either be a branch cut (as on an annulus) or it can indicate a periodic domain. Unused points are determined by Ogen using physical boundaries to mark points exterior to the domain following a hole-cutting algorithm [37, 70, 141]. The remaining interior points are classified as either discretization points or interpolation points.

To determine which component grid to prefer when there are two or more grids that overlap each other, the component grids are ordered with respect to their priority such that grid $\mathcal{G}_g$ has priority g. When there is a choice which grid points to use in the overlap domain, the basic strategy of the overlapping grid algorithm is to prefer grid points from component grids with higher priority.

When adaptive mesh refinement (AMR) is used on an overlapping grid system, new refinement grids are added where an error in the numerical solution is estimated to be large. The approach used by Ogen [70], follows the pioneering work of Berger and Oliger [18], but with some modifications for moving grids [73]. The refinement grids are added to each component grid and are aligned with the computational space $\mathcal{C}$. The refinement grids are arranged in a hierarchical way, with the base grids belonging to a refinement level $r_{level} = 0$, the next finer grids belonging to $r_{level} = 1$, and so on. The grids on refinement level r_{level} are a factor r_{factor} finer than the grids on level $r_{level} - 1$. An AMR regridding procedure is performed every r_{regrid} time steps, where r_{regrid} is typically equal to $2 \times r_{factor}$. This procedure begins with the computation of an error estimate based on the current solution. Once the error estimate is obtained, grid points are flagged if the error is larger than a tolerance. A new set of refinement grids is generated to

cover all flagged points, and the solution is transferred from the old grid hierarchy to the new one. Since the regridding procedure takes place at a fixed time, it is effectively decoupled from moving grids cases.

In computational space $\mathcal{C}$, solution values at interpolation points are generally determined by a tensor-product polynomial interpolation scheme [76]. A simple interpolation method is to directly transfer the flow variables from the donor cell or point to the receptor cell or point. With little additional work, however, a more accurate higher-order interpolation scheme can be used, *e.g.*, a polynomial interpolation scheme. Hence, it is clear that higher-order interpolation schemes can be used with no major difficulties, but a larger overlapping region with more stencil points will be needed. In this dissertation, solution values at interpolation points are determined by using a non-conservative Lagrange interpolation scheme, whose interpolation stencil is fully compatible with the stencil of the numerical scheme discussed in the following chapter. More sophisticated interpolation algorithms that maintain conservation are presented by Chesshire and Henshaw [38], Wang [206] and Zheng and Liou [217, 218].

Consider the situation depicted in figure 4.8, in which a point $\mathcal{P}_{\mathbf{i}}^{\mathcal{G}_2}$ (where $\mathbf{i} = (i_1, i_2), i_\alpha = 0, \ldots, N_\alpha^g, \alpha = 1, 2$ and N_α^g is the number of grid points in the i_α-*coordinate* direction) is to be interpolated from component grid $\mathcal{G}_1$. Ogen supplies the $\mathcal{C}_{\mathbf{i}}^{\mathcal{G}_1}$ coordinates in computational space $\mathcal{C}$ ($\mathbf{r} = (r_1, r_2)$) of point $\mathcal{P}_{\mathbf{i}}^{\mathcal{G}_2}$, such that $\mathcal{C}_{\mathbf{i}}^{\mathcal{G}_1} = \mathbf{M}_{\mathcal{G}_1}^{-1}(\mathcal{P}_{\mathbf{i}}^{\mathcal{G}_2})$. Therefore it is only necessary to know how to interpolate a point from a rectangular grid. A list of the overlapping grid system $\mathbb{G}$ domain connectivity information such as interpolation points, the donor grid from which they interpolate, the location of the interpolation point in the computational space $\mathcal{C}$ of the donor grid and so on, is provided and kept by Ogen [70]. In particular, if grid $\mathcal{G}_g$ has n_{ip}^g interpolation points, then for each $n = 1, 2, \ldots, n_{\text{ip}}^g$

$$
\begin{aligned}
\mathbf{ip} &= \text{ip}_n^g && \text{(interpolation point } n \text{ on grid } g\text{)} \\
dg &= \text{dg}_n^g && \text{(donor grid for interpolation point ip}_n^g\text{)} \\
iw &= \text{iw}_n^g && \text{(interpolation width of the overlap area of grid } g\text{)} \\
\mathbf{r} &= \text{dgm}_n^g && \text{(donor grid location of interpolation point ip}_n^g\text{, } \mathbf{r} = \mathbf{M}_{dg}^{-1}(\mathcal{P}_{\mathbf{i}}^g)\text{)} \\
\mathbf{j} &= \text{dgs}_n^g && \text{(lower left corner of the donor grid stencil of ip}_n^g\text{)}
\end{aligned}
$$

denote the interpolation information associated with the interpolation point. The width of the interpolation stencil iw is chosen based on the order of accuracy, the type of PDE (elliptic, parabolic, hyperbolic, mixed), and by the behavior of the overlap when the grid size decreases; see [37] for details. In the following, we will assume that $iw \geqslant 2$.

Then, the two dimensional interpolation formula of any quantity ϕ on a component grid $\mathcal{G}_g$ is given by standard Lagrange interpolation,

$$
\phi_{\mathbf{ip}}^g = \sum_{m_1=0}^{iw-1} \sum_{m_2=0}^{iw-1} \beta_{\mathbf{m}} \phi_{\mathbf{j}+\mathbf{m}}^{dg} \tag{4.1}
$$

where

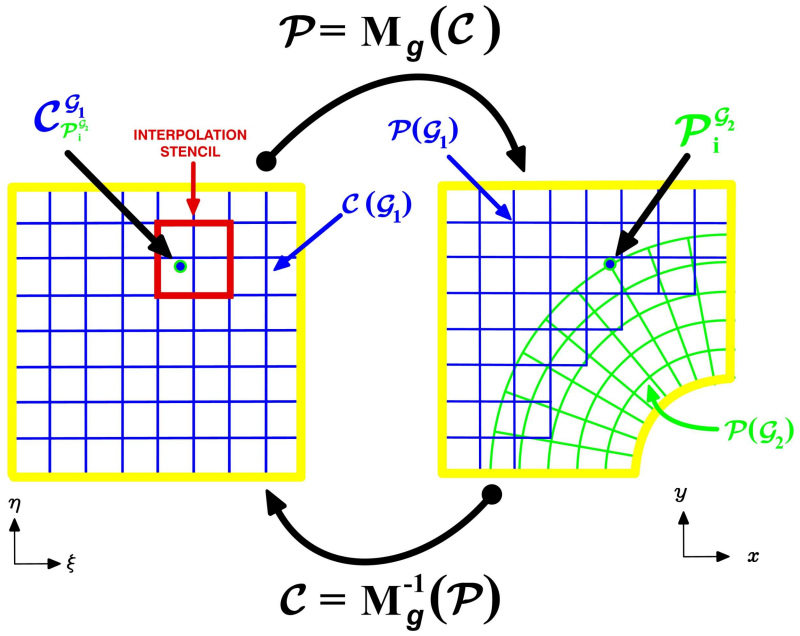

Figure 4.8: *Interpolation scheme for overlapping grids. The interpolation is performed in computational space $\mathcal{C}$.*

$$\beta_{\mathbf{m}} = \mathcal{L}^{iw}_{m_1}(\tilde{r}_1)\,\mathcal{L}^{iw}_{m_2}(\tilde{r}_2)\,, \qquad \tilde{r}_\alpha = (r_\alpha - j_\alpha)\,\Delta r_\alpha$$

Here $\mathbf{m} = (m_1, m_2)$, $\Delta r_\alpha = 1/N^g_\alpha$, and the Lagrange polynomials $\mathcal{L}^{iw}_\mu$ are defined in the usual way as

$$\mathcal{L}^{iw}_\mu(r) = \frac{\prod^{iw-1}_{j=0, j\neq\mu}(r-j)}{\prod^{iw-1}_{j=0, j\neq\mu}(\mu-j)}$$

Extension of eq. 4.1 to three dimensional cases is straightforward. The above approach to interpolation not only permits an easy way of implementing arbitrary order interpolation schemes (*e.g.*, bi-linear, bi-quadratic, tri-linear, Lagrange, spline cubic), but also makes the interpolation step less prone to error [37].

There are two different ways to interpolate in an overlapping grid [143] (figure 4.9). When the interpolation type is implicit, the solution values at the interpolation points are coupled, because they interpolate from both discretization and interpolation points in the donor grid *dg*. This makes the required overlap smaller compared to when explicit interpolation is used, since in that case only discretization points are allowed to be donor or interpolee points. When implicit interpolation is used, a small system of equations must be solved to obtain the solution at the interpolation points in terms of the values at other points, hence it becomes necessary to solve a linear system of equations in order to update the solution values at all interpolation points after each time step for time-dependent PDEs. The advantage of implicit interpolation is that the

amount of overlap is less and thus there are fewer grid points. Explicit interpolation is sometimes preferred when a time-dependent problem is solved on the overlapping grid, because it simplifies the solution procedure, but with the shortcoming of having more overlap between the grids.

Finally, when building an overlapping grid system $\mathbb{G}$, where we want the overlap to be as small and centred as possible, the following practical issues regarding to the type of interpolation must be considered. When the interpolation is implicit, the component grids must overlap one another by at least half a grid cell (for second or third order interpolation). Furthermore, the required amount of overlap is independent of the number of components grids that overlap. The situation is different for explicit interpolation. For instance, if the discretization and interpolation stencils are three points wide in each grid direction, the amount of overlap must exceed one and a half grid cells where the two component grids overlap each other. Also, the overlap must be up to three grid cells wide, close to where more than two component grids overlap each other. For further details and a complete demonstration, the interested reader should refer to [37, 141, 143].

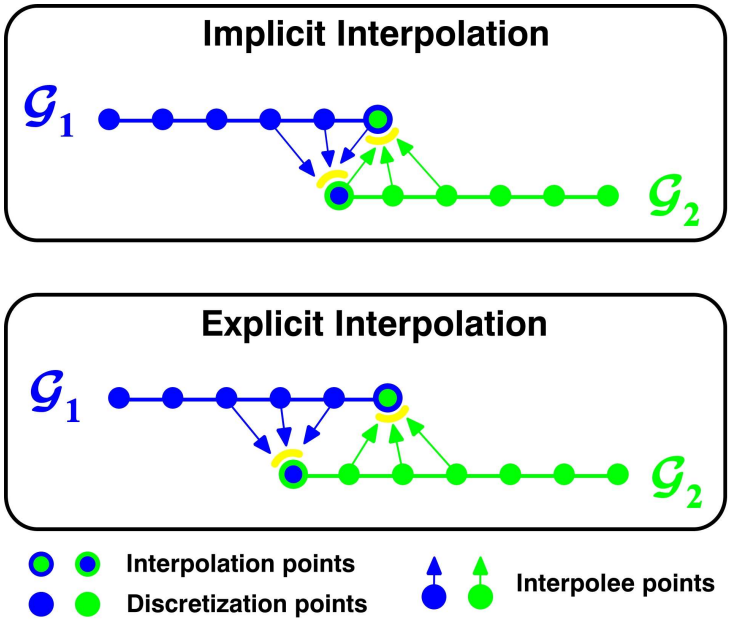

Figure 4.9: *Explicit and implicit interpolation for a one-dimensional overlapping grid.*

In addition, component grids are usually created with one or more rows of auxiliary ghost cells around the boundary of each component grid $\mathcal{G}_g$, these ghost cells are used to facilitate the discretization of boundary conditions. In the present dissertation, two rows of ghost cells are used in order to be compatible with the stencil of the numerical method and the order of the interpolation scheme. Clearly, the number of rows of ghost cells depends of the size of the numerical stencil, the interpolation width *iw* and the order of the interpolation scheme.

4.3.1 Extension of the Overlapping Grids Method to Moving Boundaries Problems

The presence of moving bodies changes the relative position of the overlapping grids continuously during the flow simulation. As the component grid (around a moving body) traverses through the computational domain, overlapping connectivity information, such as interpolation stencils and unused points regions (Chimera holes), is recomputed. The automation of hole cutting and interpolation stencils computation, makes the present methodology a powerful tool for the simulation of flows with one or multiple moving bodies, since the grids do not have to be regenerated as the solution evolves. Only Chimera holes and the interpolation stencils used to provide domain connectivity are recomputed at each time step, an operation which can be performed very efficiently. In general, the motion of the component grids and/or boundaries may be an user defined time dependent function, may obey the Newton-Euler equations for the case of rigid body motion or may be the boundary nodes displacement in response to the stresses exerted by the fluid pressure for the case of fluid structure interaction problems (FSI).

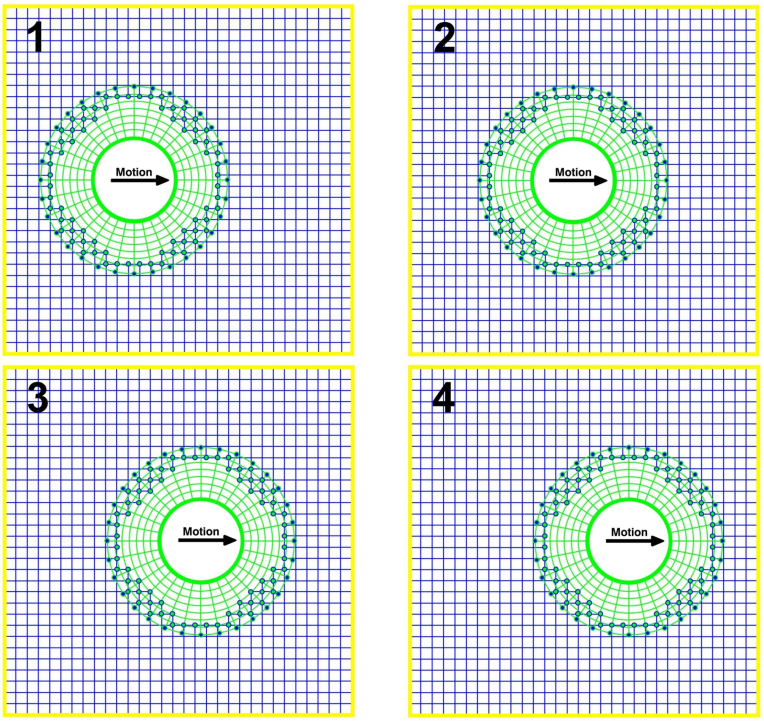

Figure 4.10: *Moving overlapping grid. The new overlapping grid system* $\mathbb{G}$ *interpolation stencils and chimera holes are determined by Ogen at each time step.*

When a component grid changes position during a moving grid computation, the overlapping grid generator Ogen [70] is called at each time step in order to update the interpolation stencils and Chimera holes. The component grids themselves do not have to be recomputed unless they deform in shape. An optimized algorithm is used to determine the new points classification for each grid. The algorithm only considers component grids affected by the moving boundary and a new classification of points begins by assuming that the actual structure is similar to that of

the grids at the previous time. After identifying interpolation points that are no longer valid, a local search is made for new candidates. In the event that the local search algorithm fails in completing the new classification of points, then the general overlapping grid algorithm is used (global search). Without any automation, such problems require the user to provide a priori all the input that would enable hole cutting and interpolation stencil identification for any given configuration of the overlapping grid, a cumbersome and time consuming operation.

Finally, as the problem of moving boundaries is a transient problem, the governing equations are intrinsically time dependent, thus they must be modified in order to handle the unsteady nature of the moving boundaries problem. The time derivative of any quantity $\phi = \phi(x, y, z, t)$ at a fixed point of the physical space $\mathcal{P}$ is related to its time-derivative at a fixed point of the computational space $\mathcal{C}$ by the equation

$$\left(\frac{\partial \phi}{\partial t}\right)_{\mathcal{P}} = \left(\frac{\partial \phi}{\partial t}\right)_{\mathcal{C}} - \dot{\mathbf{G}} \cdot \nabla \phi \tag{4.2}$$

with $\nabla\phi$ evaluated in the physical space $\mathcal{P}$. In eq. 4.2, $\dot{\mathbf{G}}$ is the rate of change of position of a given set of grid points $\mathcal{P}^g_{\mathbf{i}}$ in the physical space and can be called the grid points velocity or grid velocity and is equal to

$$\dot{\mathbf{G}} = \left(\frac{\partial \mathcal{P}^g_{\mathbf{i}}}{\partial t}\right)_{\mathcal{C}} \tag{4.3}$$

In eq. 4.2 and eq. 4.3, $t_{\mathcal{P}} = t$ and $t_{\mathcal{C}} = \tau$ and from eq. 3.27 we know that $t = \tau$. Thus,

$$\left.\frac{\partial \phi}{\partial t}\right|_{\mathcal{P}} = \left.\frac{\partial \phi}{\partial t}\right|_{\mathcal{C}} - \dot{\mathbf{G}} \cdot \nabla \phi \quad \text{where} \quad \dot{\mathbf{G}} = \left.\frac{\partial \mathcal{P}^g_{\mathbf{i}}}{\partial t}\right|_{\mathcal{C}} \tag{4.4}$$

By replacing eq. 4.4 into the respective governing equations, they are now expressed in a reference frame moving with the component grid. It is important to mention that the new governing equations expressed in the moving reference frame, must be accompanied by the proper boundary conditions. For a moving body with a corresponding moving no-slip wall, only one constraint may be applied and this corresponds to the velocity on the wall

$$\mathbf{u}\left(\mathcal{P}^g_{\mathbf{i}}|_{wall}, t\right) = \dot{\mathbf{G}}, \quad \text{where} \quad \mathcal{P}^g_{\mathbf{i}}|_{wall} \in \partial\mathcal{D}_{wall}\left(t\right) \tag{4.5}$$

4.3.2 Time Stepping Algorithm

In figure 4.11, the pseudo C++ code for the basic time-stepping algorithm for moving overlapping grids with AMR regridding used by Ogen is presented [74]. In the algorithm, $\mathbf{Q}^n_i$ denotes the numerical solution of a system of PDEs on a domain represented by an overlapping grid system $\mathbb{G}$. The input of the algorithm is an overlapping grid system $\mathbb{G} = \mathcal{G}^n_g$ generated by Ogen and the final time t_{final} over which the equations are to be integrated (for the purpose of the present discussion the precise governing equations involved or their numerical approximation are not of interest). The algorithm begins with the specification of the initial conditions according to the class **Set_initial_Conditions** and the possibly creation of an initial AMR hierarchy of grids according to an AMR algorithm contained in the **if-end** *clause*. The AMR steps involve estimating the error, regridding to better resolve the solution and interpolation of the solution from the old overlapping grid, including its hierarchy of refined grids, to a new one. These steps are repeated

until either the error tolerance is met or until the maximum number of refinement levels have been added.

Once the initial solution and initial AMR grid hierarchy have been determined, the discrete solution is advanced in time. At the top of the **while** *loop*, the solution, $\mathbf{Q}_i^n$, and grid, $\mathcal{G}_g^n$, are known at the current time t. Before advancing the grid and solution to the next time level, the algorithm checks to see whether the AMR grids need to be regenerated, then an AMR regridding procedure is performed every n_{regrid} steps.

```
1  PDE_To_Solve(𝔾, t_final)
2  {
3      t = 0;    n = 0;    𝔾 = 𝒢_g^n;
4      Q_i^n = Set_Initial_Conditions(𝒢_g^n);
5      while  t < t_final
6          if(n mod r_regrid = 0)                                   /* rebuild the AMR grids */
7              e_i = Apply_Error_Estimator(𝒢_g^n, Q_i^n);
8              𝒢_g^* = regrid(𝒢_g^n, e_i);
9              Q_i^* = Interpolate_To_New_Grid(Q_i^n, 𝒢_g^n, 𝒢_g^*);
10             𝒢_g^n = 𝒢_g^*;    Q_i^n = Q_i^*;
11         end
12         Δt = Compute_TimeStep(𝒢_g^n, Q_i^n);                     /* compute overlapping grids timestep */
13         𝒢_g^{n+1} = Move_Components_Grids(𝒢_g^n, Q_i^n);          /* predict the new grid state */
14         Update_Overlapping_Information(𝒢_g^{n+1});               /* update overlapping grid connectivity */
15         Q_i^{n+1} = Advanced_TimeStep(𝒢_g^n, 𝒢_g^{n+1}, Q_i^n, Δt);   /* advance the solution */
16         interpolate(𝒢_g^{n+1}, Q_i^{n+1});                       /* interpolate overlapping grid points */
17         Apply_Boundary_Conditions(𝒢_g^n, 𝒢_g^{n+1}, Q_i^n, Q_i^{n+1}, t + Δt);
18         t = t + Δt;    𝒢_g^n = 𝒢_g^{n+1};    n = n + 1;
19     end
20 }
```

Figure 4.11: *Pseudo C++ Code for the basic time stepping algorithm for overlapping grids.*

The first step in the main time-stepping loop moves the grids one time step according to the class **Move_Components_Grids**. The motion of the component grids $\mathcal{G}_g$ and/or boundaries may be an user defined time dependent function, may be determined by the Newton-Euler equations of motion (*i.e.*, rigid body motion) or may be the response to the stresses exerted on the boundaries by the fluid pressure (*i.e.* fluid structure interaction). Then, the new grid position $\mathcal{G}_g^{n+1}$ is determined at $t+\Delta t$. After the grids have moved, the overlapping grid generator is called to update the overlapping grid connectivity information for $\mathcal{G}_g^{n+1}$ (class **Update_Overlapping_Information**) which includes the new classification of points as discretization, unused, or interpolation points. The numerical solution may now be advanced to the next time level according to the current state $\mathbf{Q}_i^n$ and grids $\mathcal{G}_g^n$ and $\mathcal{G}_g^{n+1}$, as indicated by the class **Advanced_Time_Step**. After the solution is advanced at all discretization points, the **interpolate** class is called to update the solution on overlapping grid interpolation points and on the interpolation points on the refinement grids. The boundary conditions are then applied (class **Apply_Boundary_Conditions**) so that the solution $\mathbf{Q}_i^{n+1}$ is now specified at all interior, boundary and ghost points.

4.4 Overlapping Grids Assembling Algorithm

In this section, the algorithm for assembling overlapping grids is outlined and illustrated. In general, the method consist of three major steps. The first step is simply the geometry definition and component grids generation. The second step consist in detecting all hole points outside of the computational domain, and the third step consist in finding the grid points to interpolate

from (donor or interpolee points) for all interpolation points on the fringe of the hole (receptor or interpolation points). As we are only interested in illustrating the assembling algorithm, the detailed and theoretical definitions will be avoided as much as possible. For a detailed explanation of the algorithm see for example [37, 70, 143, 141, 149].

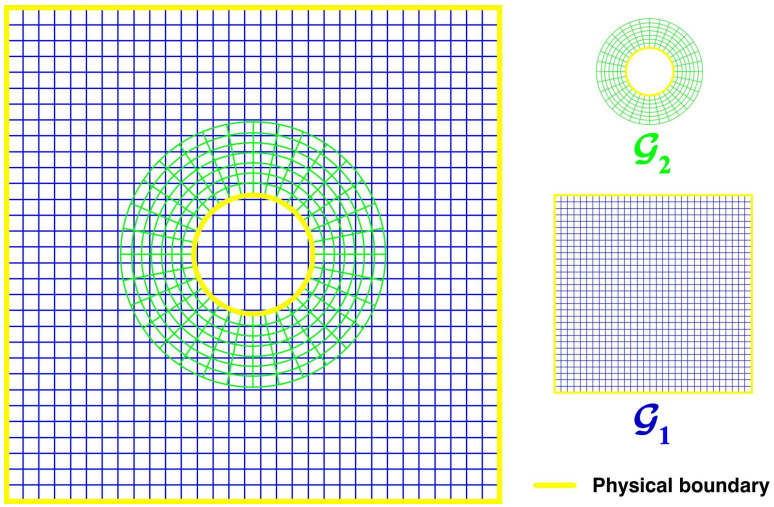

Figure 4.12: *Left. Initial overlapping grids system* $\mathbb{G}$*. Right. Individual component grids* $\mathcal{G}_g$*.*

We now proceed to describe the major steps and its corresponding substeps. First, the algorithm must start with a set of component grids $\mathcal{G}_g$ and a set of boundary conditions and constraints (figure 4.12), this step can be seen as mainly CAD and grid generation work.

The second step consist in marking hole boundaries and removing exterior or unused points (figure 4.13). This step can be seen as a two-substeps process. Firstly, for each physical boundary we find points on other component grids $\mathcal{G}_g$ that are near to and inside or outside of the physical boundary. After this substep, the holes in the grid will be bounded by a boundary of exterior or unused points next to a boundary of interpolation points. Next, all remaining points within the hole are marked as exterior or unused points. These points can be now easily swept out since the hole cutting algorithm ensures that all holes are bounded by interpolation points. At this stage, we have created a Chimera hole.

The third step consist in finding and classifying all valid interpolations points. This step starts with the highest grid priority and proceed in decreasing priority order such that fewer restrictions are enforced on the grids with higher priority. Here, the points on the physical boundaries and interpolation boundaries are collected into a list of interpolation points. Then, we proceed to classify these points by using improper interpolation [70]. A point is said to interpolate in an improper way from another grid if it simply lies within that grid. Since all the points in the list lie within the domain they must interpolate from some other grid or else there is something wrong (figure 4.14). Next, we proceed to find all the proper interpolation points. In this step, the points belonging to the list of improper interpolation points are classified as proper interpolation points or discretization points. A point of a grid is said to be a proper interpolation point if the appropriate stencil of points exist on the donor grid and consists of the correct types of points

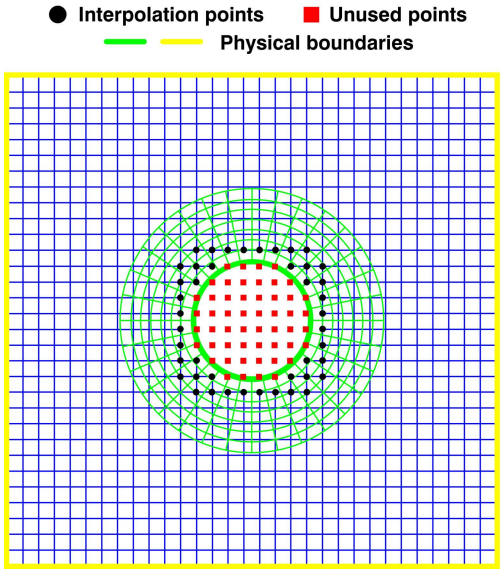

Figure 4.13: *Overlapping grid system $\mathbb{G}$ after cutting holes and removing all exterior or unused points. The hole cutting algorithm generates a barrier of unused points and interpolation points that bounds the entire hole region.*

for the implicit or explicit interpolation. We also attempt to interpolate the discretization points on each grid from grids of higher priority (figure 4.15). At this step, we should have a valid overlapping grid system $\mathbb{G}$.

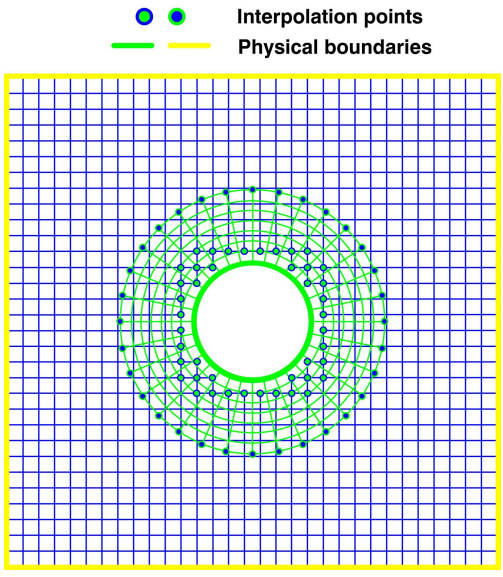

Figure 4.14: *Overlapping grid system $\mathbb{G}$ after marking points on the physical boundaries (stairstep boundary) and interpolation boundaries.*

At this stage, an additional step known as trimming (figure 4.16), is performed. Basically, this step consist in reducing the amount of overlap. Here, any interpolation point that is not needed is removed from the computation according to an user defined criteria or minimum overlap requirement. Interpolation points that are needed but can just as well be used as discretization points are turned into discretization points.

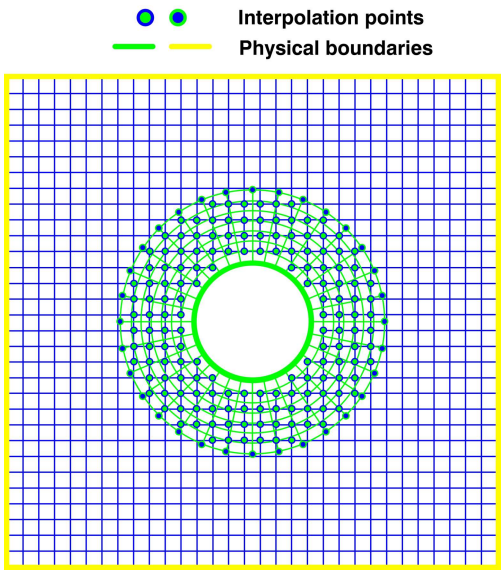

Figure 4.15: *Overlapping grid system $\mathbb{G}$ after marking all proper interpolation.*

By now, the overlapping grid system $\mathbb{G}$ has been assembled and optimized for the minimum or desired overlap. The last step simply consist in a consistency check, where we check if the classification of the grids points is consistent and if all discretization and interpolation points satisfy the necessary requirements. It is worth to mention that this consistency check is recursively done after each one of the previous steps. This check will mark all points that fail to satisfy the requirements and the output may be used as a reference for troubleshooting the overlapping grid system generated by Ogen [70].

In the above algorithm, one of the most important operations which is recursively performed is the task of inverting a component grid mapping $\mathbf{M}_g$ corresponding to a general component grid $\mathcal{G}_g$ in order to find a donor or interpolee point for an interpolation point $\mathcal{P}_{\mathbf{i}}^{g}$. Hence, it is essential to perform this operation as quickly as possible. Using figure 4.8 as a reference, the algorithm used by Ogen is as follows [37, 70, 141, 143],

1. Before attempting to invert the mapping, we first must check if $\mathcal{P}_{\mathbf{i}}^{\mathcal{G}_2}$ lies inside the rectangle $\Omega^{\mathcal{G}_1}$ that bounds the donor component grid $\mathcal{G}_1$. If the point does not lies inside $\Omega^{\mathcal{G}_1}$, then it cannot be interpolated and we are done. Else, if the point $\mathcal{P}_{\mathbf{i}}^{\mathcal{G}_2}$ is inside $\Omega^{\mathcal{G}_1}$, the second problem is to generate a sufficiently good initial guess for Newton's method, which is used to determine the $\mathcal{C}_{\mathbf{i}}^{\mathcal{G}_1}$ coordinates of the point $\mathcal{P}_{\mathbf{i}}^{\mathcal{G}_2}$.

2. If $\mathcal{P}_{\mathbf{i}}^{\mathcal{G}_2}$ lies within the rectangle $\Omega^{\mathcal{G}_1}$, we need a good initial guess for Newton's method. In order to get a good initial guess for this step, we do an optimized local search through all

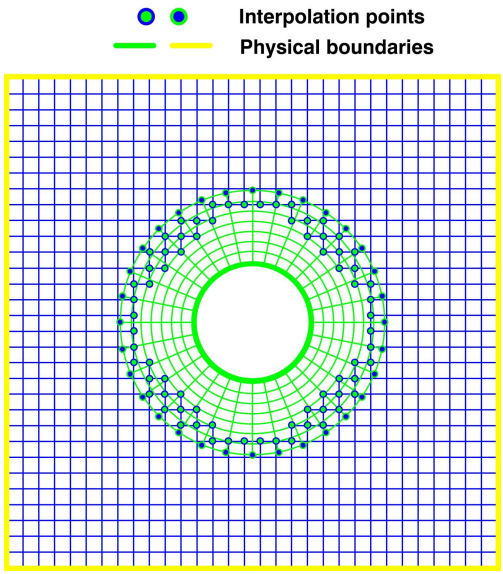

Figure 4.16: *Final overlapping grid system $\mathbb{G}$ after removing excess of interpolation points*

grid points in order to find the closest grid point to $\mathcal{P}_{\mathbf{i}}^{\mathcal{G}_2}$ on the donor component grid $\mathcal{G}_1$. As soon as the closest point to $\mathcal{P}_{\mathbf{i}}^{\mathcal{G}_2}$ is determined, we proceed to the next step.

3. Now we can determine the $\mathcal{C}_{\mathbf{i}}^{\mathcal{G}_1}$ coordinates of the point $\mathcal{P}_{\mathbf{i}}^{\mathcal{G}_2}$ by inverting the component grid $\mathcal{G}_1$ transformation $\mathbf{M}_g$ with a Newton iteration, using the closest point found in the previous step as an initial guess. As soon as $\mathcal{C}_{\mathbf{i}}^{\mathcal{G}_1}$ of $\mathcal{P}_{\mathbf{i}}^{\mathcal{G}_2}$ has been found ($\mathcal{C}_{\mathbf{i}}^{\mathcal{G}_1} = \mathbf{M}_{\mathcal{G}_1}^{-1}(\mathcal{P}_{\mathbf{i}}^{\mathcal{G}_2})$), it is trivial to determine the enclosing grid cell and the interpolation stencil according to the interpolation scheme.

We emphasize the above point, since the construction of a composite grid requires some computation and it is important to do certain tasks efficiently, so that the turnaround time for the grid assembly algorithm is small (specially for moving grids where the domain connectivity information have to be recomputed at each time step).

4.5 Discretization on Overlapping Grids

On an overlapping grid system $\mathbb{G}$, the solution of a PDE can be seen as the solution of the transformed PDE on a set of unit domains in computational space $\mathcal{C}$. The governing PDE is transformed from physical space $\mathcal{P}$ to computational space $\mathcal{C}$ by replacing the Cartesian derivatives by their equivalent in the transformed computational space $\mathcal{C}$ (see Chapter 3, Section 3). Hereafter, we present a simple example of how to discretize a model one-dimensional PDE on a structured overlapping grid system $\mathbb{G}$. For the sake of simplicity, the following example considers Cartesian coordinates, nevertheless, the discretization on the unit interval in the transformed computational space $\mathcal{C}$ is straightforward. Let us consider the boundary value problem (BVP) for the one-dimensional Poisson equation,

$$\begin{aligned} u_{xx} &= f, \qquad x \in \mathcal{P}\,[0,1]^{\mathbb{N}}, \qquad \mathcal{P} \in \Re^{\mathbb{N}}, \qquad \mathbb{N} = 1 \\ u\,(0) &= g_0, \qquad \text{(Dirichlet boundary condition)} \\ u_x\,(1) &= g_1, \qquad \text{(Neumann boundary condition)} \end{aligned} \tag{4.6}$$

eq. 4.6 is to be discretized on the overlapping grid system $\mathbb{G}$ shown in figure 4.17, by using standard finite differences. A second-order discretization to this problem is

$$\begin{aligned} \mathbf{U}_0 &= g_0\,(\text{Dirichlet } bc \text{ on } \mathcal{G}_1) \\ \frac{\mathbf{U}_{i-1} - 2\mathbf{U}_i + \mathbf{U}_{i+1}}{h^2_{\mathcal{G}_1}} &= f(\mathcal{P}_i^{\mathcal{G}_1}), \quad i = 1, 2, \ldots, N_1 - 1, \quad (dp \text{ on } \mathcal{G}_1) \\ \mathbf{U}_{N_1} - (\alpha_0\mathbf{V}_0 + \alpha_1\mathbf{V}_1 + \alpha_2\mathbf{V}_2) &= 0, \quad (ip \text{ on } \mathcal{G}_1) \\ \mathbf{V}_0 - (\beta_0\mathbf{U}_{N_1-2} + \beta_1\mathbf{U}_{N_1-1} + \beta_2\mathbf{U}_{N_1}) &= 0, \quad (ip \text{ on } \mathcal{G}_2) \\ \frac{\mathbf{V}_{i-1} - 2\mathbf{V}_i + \mathbf{V}_{i+1}}{h^2_{\mathcal{G}_2}} &= f(\mathcal{P}_i^{\mathcal{G}_2}), \quad i = 1, 2, \ldots, N_2, \quad (dp \text{ on } \mathcal{G}_2) \\ \frac{\mathbf{V}_{N_2+1} - \mathbf{V}_{N_2-1}}{2h_{\mathcal{G}_2}} &= g_1 \quad (\text{Neumann } bc \text{ on } \mathcal{G}_2) \end{aligned} \tag{4.7}$$

where *bc* stands for boundary conditions, *dp* stands for discretization points, *ip* stands for interpolations points and $i = i_\alpha$-*coordinate* direction with $\alpha = 1$. As the problem is solved in one-dimension, the notation $variable_\alpha$-*coordinate* direction will be dropped for all index variables.

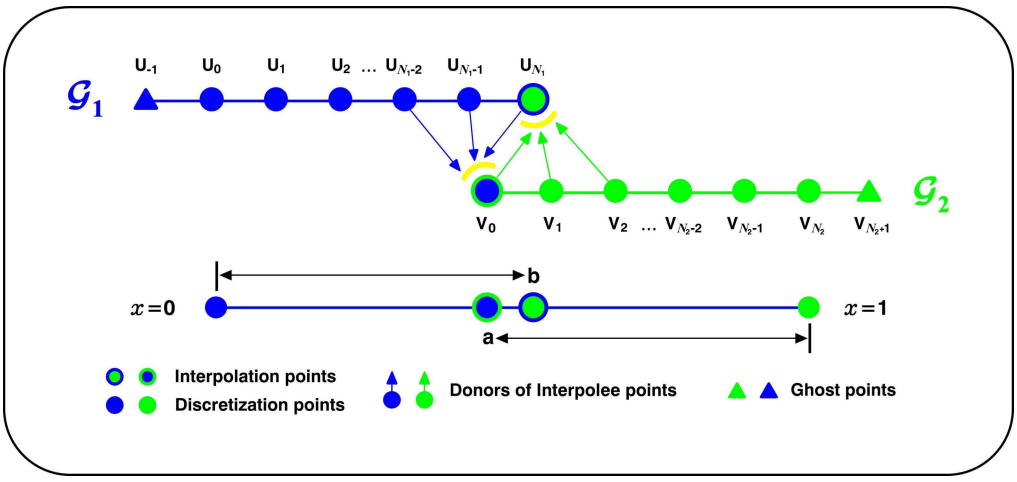

Figure 4.17: *Overlapping grid discretization in one dimension.*

In eq. 4.7, $\mathbf{U}_i$ is the approximate solution on grid $\mathcal{G}_1$ on the subinterval $[0, b]$, with $\mathcal{P}_i^{\mathcal{G}_1} = ih_{\mathcal{G}_1}$, $h_{\mathcal{G}_1} = b/N_1$ and N_1 the number of nodes on grid $\mathcal{G}_1$. Respectively, $\mathbf{V}_i$ is the approximate solution on grid $\mathcal{G}_2$ on the subinterval $[a, 1]$, with $\mathcal{P}_i^{\mathcal{G}_2} = ih_{\mathcal{G}_2}$, $h_{\mathcal{G}_2} = (1-a)/N_2$ and N_2 the number of nodes on grid $\mathcal{G}_2$. The interpolation weights α_m and β_n are chosen appropriately according to

the interpolation scheme (see section 4.3). For this example, we assumed implicit interpolation and an interpolation width iw equal to 3, hence the interpolation stencil is equal to $j+m$, where j is the lower left corner of the donor grid dg interpolation stencil and $0 < m \leqslant iw - 1$. The Neumann boundary condition at $i = N_2$ on grid $\mathcal{G}_2$ is implemented by adding a ghost point $\mathbf{V}_{N_2+1}$; then, by using centred finite difference approximations, we obtain the value of the ghost point which is considered to be defined by the Neumann boundary condition and is equal to $\mathbf{V}_{N_2+1} = 2h_{\mathcal{G}_2}g_1 + \mathbf{V}_{N_2-1}$. At this point, the system of equations eq. 4.7 can be solved by using any direct or iterative solution method, obtaining in this way the approximate solution of eq. 4.6 on the discrete overlapping domain $\mathcal{P}_{\mathbf{i}}^{\mathbb{G}} \in (\mathcal{P}_i^{\mathcal{G}_1} \cup \mathcal{P}_i^{\mathcal{G}_2})$.

4.6 Comments on Overlapping Grids

The main advantage of the overlapping grids scheme is that individual body-fitted conforming structured grids can be created separately for each component defining the overall geometry and then superimposed to form one complete grid system that covers the entire physical domain, all without any constraints on the grid boundaries as long as overlap exists between adjacent grids. This allows complex geometries to be treated more easily, theoretically reducing the time and effort to generate a grid; also, as several overlapping grids are used, only the Chimera holes and the interpolation stencils are recomputed as the solution evolves in time when dealing with moving boundaries.

Of course, the main disadvantage of the overlapping grid scheme is that the algorithm is far more complex than a single-block or multi-block structured grid algorithm due to the use of multiple structured overlapping grids. Drawbacks to using overlapping grids include having to interpolate in a non-conservative way data points along overlapping zones, which in practice rarely seems to be an issue if good standard practices are followed [32]. In addition, the data structure bookkeeping can be especially complex if more than two grids overlap one another. Additional to the flow solver, class libraries are needed to interconnect the overlapping grids, create proper hole regions (Chimera holes), define hole boundaries, and determine the interpolation stencils for properly transmitting information among overlapping grids. Though this is perhaps a bit complex and time consuming, the computational efficiency and convenience that is gained by using such a scheme when dealing with complex or moving geometries makes the method worthwhile.

There are two additional practical complications related to generating an overlapping grid. First, it can be hard to judge *a priori* if the component grids overlap each another sufficiently. Second, the user can make a mistake when labeling the boundaries of the component grids, which can lead to an inconsistent definition of the overlapping grid system. Creating an overlapping grid is therefore sometimes an iterative process which requires much user experience and care, and where the component grids are changed by the user until a valid overlapping grid can be formed.

Finally, the use of overlapping grids does not eliminate the planning stage in grid generation process. Care must be taken to ensure that the grid distributions in the overlapping grid system are not drastically different. Radical differences between overlapping grids can lead to poor interpolation results. Poor interpolation can in turn lead to increased computational time, due to poor solution convergence.