UCLA
COMPUTATIONAL AND APPLIED MATHEMATICS

A New Algorithm for Generating Overlapping Grids

N. Anders Petersson

June 1995

CAM Report 95-31

Department of Mathematics
University of California, Los Angeles
Los Angeles, CA. 90024-1555

A new algorithm for generating overlapping grids

N. Anders Petersson *

June 20, 1995

Abstract

A highly automated general purpose algorithm for constructing overlapping grids is described. The basic feature of the algorithm is its ablility to directly classify most grid points in the overlapping grid; it only needs to iterate on the classification of interpolation points where three or more component grids overlap each other. The most important advantage of the direct classification manifests itself when there are too few grid points in the component grids to form valid interpolation relations. The main parts of the grid will be valid even after the algorithm fails, in which case the algorithm graphically reports all inconsistent grid points to the user, who rather easily can determine why the algorithm failed and modify the component grids to improve the situation.

In order to determine if a grid point can be an interpolation point, the present method uses a fast and memory efficient implementation of the ray-method, which counts the number of intersections between the boundary of the donor grid and a ray that starts at the grid point and ends at infinity.

It will be exemplified that the implementation of the algorithm is very efficient compared to an existing overlapping grid code, and that the present method requires of the order of the total number of grid points to compute the overlapping grid.

1 Introduction

The spatial discretization procedure for numerically solving a partial differential equation (PDE) is not trivial for a generally shaped domain in two or three space dimensions. The most general approach might be to use an unstructured grid together with a finite element or finite volume discretization scheme. Much theory has been developed for the finite element method and almost automatic procedures exist for generating such grids. One drawback of this method lies in the substantial memory overhead induced by storing the connectivity between the grid cells, which is necessary to discretize the PDE. An unstructured PDE solver can also be hard to vectorize because of the added layer of

*Department of Mathematics, UCLA, Los Angeles, CA 90024. This work was supported by Office of Naval Research grant N00014–92–J–1890.

pointers that is necessary to keep track of the neighbors to each cell. To circumvent this overhead and to enable easier vectorization one can instead use a structured grid where the connectivity of the grid cells is known á priori. A variety of structured grid generation techniques are described in Thompson et. al [1]. In its simplest form, one topologically rectangular mesh is deformed smoothly to follow the shape of the computational domain. This technique is limited to cases when the domain is very simple, but can for instance be used to generate a grid around a two-dimensional airfoil. However, as the domain becomes more complicated, the single grid approach becomes less attractive because it is very hard to generate a smooth grid in a complicated domain, especially in three space dimensions.

The structured grid approach can be extended to general domains by subdividing the complicated domain into several simpler sub-domains where each sub-domain is gridded individually by a component grid. In a so called patched grid, the component grids meet at the common sub-division boundaries. Since each component grid is structured, it is straight forward to write an efficient PDE solver for a patched grid. Furthermore, because the component grids meet at common sub-division boundaries, it is not hard to design a conservative discretization scheme on these grids. However, it can be hard to control the smoothness of the combined mesh, especially at sub-division corners. These mesh irregularities can induce stability and accuracy problems in the solution of the PDE.

The global smoothness problems of a patched grid can be overcome by letting the sub-domains overlap, thus relaxing the shape constraints of the sub-domains considerably and removing the requirement of matching sub-division boundaries. An overlapping grid usually consists of a number of curvilinear boundary-fitted grids together with a couple of background grids that often are Cartesian. The boundary-fitted grids resolve the different details close to the boundary of the computational domain and the background grids cover the remaining parts of the domain. Each component grid is a structured curvilinear or Cartesian grid where most of the grid points in each component grid are used to discretize the PDE. The solution value at the grid points on the interior boundaries where two component grids overlap is interpolated from the solution on the overlapping component grid, see figure 1. There can also be grid points that not are used at all, either because they are outside of the computational domain, or in order to reduce the total number of grid points in the overlapping grid. This approach requires more overhead than the patched grid method, since it is now necessary to store the location of the interpolation stencil for each interpolation point. However, the method is still efficient since each component grid is structured. The overlapping grid technique, which also is known as the Chimera overset grid method, can therefore be seen as a compromise between the structured and unstructured discretization techniques. It is more difficult to design a conservative scheme on an overlapping grid than on a patched grid, but if conservation is important, the straight forward interpolation approach can be modified to satisfy that demand [2].

There are three difficulties associated with the construction of an overlapping grid. The most obvious problem is to find the grid points to interpolate from (the donor points) when an interpolation point interpolates from a curvilinear component grid (the donor grid). The second hardship occurs when three or more component grids overlap each

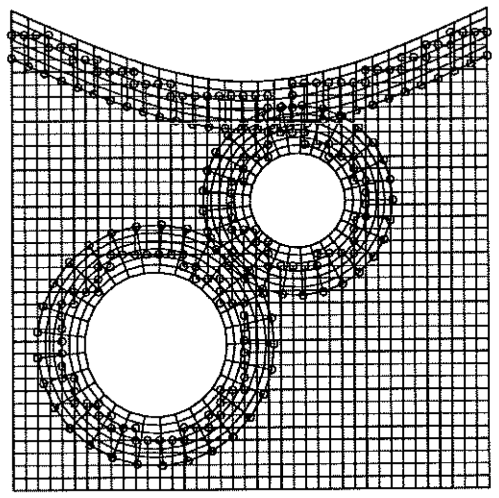

Figure 1: An overlapping grid. The circles indicate interpolation points where the solution value is interpolated from the overlapping component grid.

other and it is decided, for the convenience of the PDE solver, that donor points can not them self be interpolation points. It is then not trivial to select interpolation points and corresponding donor points. The third complication is to design the algorithm to avoid the formation of islands of "orphan points" in the overlapping grid. "Orphan points" are grid points located outside of the computational domain which are surrounded by unused grid points. They are therefore isolated from the rest of the overlapping grid and they have no relevance for the discretization of the PDE.

A number of previous algorithms exist for constructing overlapping grids [3, 4, 5, 6, 7]. The present technique is closest to the Chesshire and Henshaw [6] algorithm, implemented in CMPGRD [8]. While the present method bears many similarities with CMPGRD it also improves on it in important ways. Like CMPGRD, the present method implements a highly automated general purpose algorithm where the user essentially only has to provide the component grids and label the sides of the components that are part of the boundary of the computational domain. However, the present algorithm makes the main classification of all grid points after just one pass through all grid points, and only needs to iterate on the classification of the interpolation points where three or more component grids overlap each other. In contrast, CMPGRD iterates on the classification of all grid points until its algorithm converges, which at least requires two iterations. The present method is therefore faster than CMPGRD.

When three or more component grids overlap each other, it can be hard to predict the number of grid points that is needed in the overlap domain. Creating an overlapping grid is then an iterative process where the component grids are changed by the user until a valid overlapping grid can be formed. Because the present method only iterates on the classification of interpolation points, the main parts of the overlapping grid will be valid even after the algorithm failed. The present method checks for inconsistent grid points after the main classification is completed. If any such points are found, they are reported graphically to the user, who then rather easily can determine why the algorithm failed and modify the component grids to improve the situation. This behavior is maybe the most significant improvement over CMPGRD, which when it fails produces a so called null-grid, where all grid points are removed from the overlapping grid.

An important operation in the overlapping grid algorithm is to find out whether a grid point can be an interpolation point. The present method implements a fast and memory conservative algorithm to answer this question, i.e. determine if the grid point is inside or outside of a component grid. The technique is based on counting the number of intersections between the boundary of the component grid and a ray that starts at the grid point and tends to infinity. If the grid point is found to be inside, the algorithm also efficiently finds the enclosing grid cell by using a bisection technique. To compute the weights in the interpolation formula, it is necessary to know where in the enclosing grid cell the interpolation point is located. If a mapping function is available that defines the behavior of the grid in between the grid points, this mapping function can be inverted by a Newton iteration using the location of the enclosing grid cell as initial guess. Otherwise, if only the location of the grid points is known, the mapping function can be approximated locally by Lagrangian interpolation to the required order of accuracy, and the approximate mapping function can be inverted to determine the interpolation weights. Hence, the present method is capable of setting up higher order accurate interpolation relations given only the location of the grid points in the component grids.

The present method avoids the formation of islands of "orphan" points by performing a preparatory marking of the grid points close to component grid corners that are outside of the computational domain. This procedure eliminates the need to iterate on the classification of all grid points, as is done in CMPGRD, but it requires the user to identify the exterior corners.

The overlapping grid algorithm will be described for a vertex centered discretization scheme where the discrete solution values are located at the grid points. While this is not a restriction of the overlapping grid method, it simplifies the presentation of the scheme. We will also assume that the computational domain is two-dimensional; a three-dimensional extension will be described in a forthcoming paper. Henceforth, we consider an overlapping grid with `n_grids` component grids, where component grid k has N_k times M_k grid points with Cartesian coordinates $\mathbf{x}_{i,j}^{(k)}$, $1 \leq i \leq N_k$, $1 \leq j \leq M_k$, $1 \leq k \leq$ `n_grids`. Furthermore, we define grid cell $Q_{i,j}^{(k)}$ to be the quadrilateral with corners $\mathbf{x}_{i,j}^{(k)}$, $\mathbf{x}_{i+1,j}^{(k)}$, $\mathbf{x}_{i+1,j+1}^{(k)}$ and $\mathbf{x}_{i,j+1}^{(k)}$.

The remainder of the paper is organized as follows. We discuss the principles and requirements on an overlapping grid in section 2, which is followed by a presentation of the overlapping grid algorithm in section 3. In section 4, we describe the algorithm for determining if a grid point can be an interpolation point and we also present how to compute the weights in the interpolation formula. We comment on the implementation of the algorithm and show a few examples in section 5.

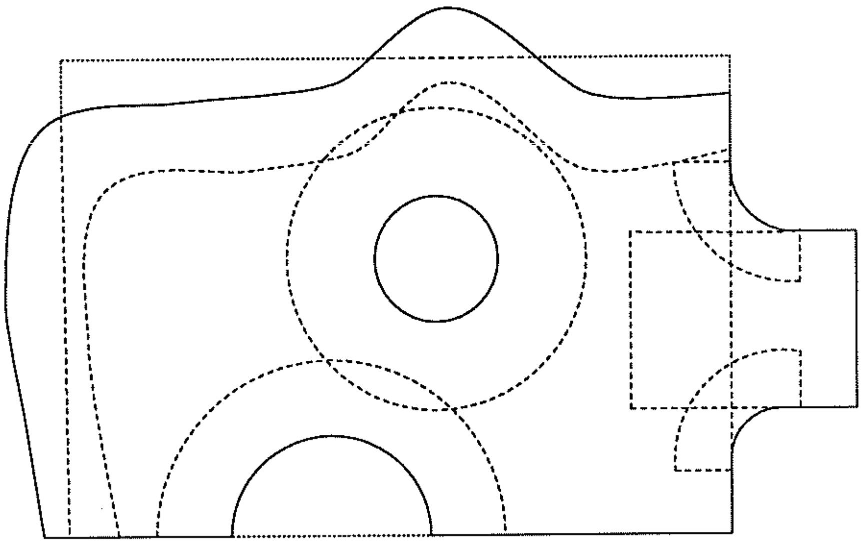

Figure 2: There can be combinations of the three basic boundary types along the same side of one component grid. Here solid lines are physical boundaries, dashed lines represent interpolation boundaries and dotted lines indicate external boundaries.

2 Requirements on an overlapping grid

In addition to the location of the grid points in each component grid, information about the boundaries of each grid must be supplied in order to construct the overlapping grid. There can be three kinds of boundaries in a component grid: physical, interpolating and external boundaries, see figure 2. A physical boundary describes a part of the boundary of the computational domain, an interpolating boundary lies inside another component grid such that the solution value can be interpolated to that boundary, and an external boundary is situated outside of the computational domain. Hence, the grid points between the external boundary and the boundary of the computational domain will not be included in the overlapping grid. Naturally, there can be combinations of these three boundary types along the same side of one component grid.

The present method assumes that the solution value at each interpolation point will be interpolated from a rectangular stencil in the donor grid. The grid points in the donor grid that are involved in the interpolation stencil will be called donor points. To demonstrate the interpolation procedure, we consider one interpolation point with Cartesian coordinate $\mathbf{x}_P = (x_P, y_P)$. In order to assign the correct weights in the interpolation formula, it is necessary to know the behavior of the grid in between the grid points in the donor grid. It is for this purpose convenient to regard each component grid as a mapping from the unit square in the parameter space to the physical domain covered by the component grid; if the mapping is not known, it can be approximated locally by interpolation between the grid points to the required order of accuracy. We will denote the parameter space by $\mathbf{r} = (r, s)$ and the mapping for component grid k by $\mathbf{x} = \mathbf{X}^{(k)}(\mathbf{r})$. In parameter space, each component grid is Cartesian and the grid points are simply $\mathbf{r}_{i,j} = (r_i, s_j)$, where

$r_i = (i-1)/(N_k-1)$, $s_j = (j-1)/(M_k-1)$. The grid points in physical space are therefore $\mathbf{x}^{(k)}_{i,j} = \mathbf{X}^{(k)}(\mathbf{r}_{i,j})$.

By inverting the donor grid's mapping function, we can compute the parameter value $\mathbf{r}_P = (r_P, s_P)$ corresponding to the Cartesian coordinate $\mathbf{x}_P$ of the interpolation point. It is then easy to find the grid point in the donor grid (i_L, j_L) around which the interpolation should be centered. We call this point the interpolation location. Lets denote the solution at grid point (i,j) in the donor grid B by $\phi^{(B)}_{i,j}$. The solution at the interpolation point is assigned the value of the `interp_width` order accurate Lagrangian interpolant:

$$\phi(\mathbf{x}_P) = \sum_{i=i_L-L}^{i_L+H} \sum_{j=j_L-L}^{j_L+H} \alpha_i(r_P)\beta_j(s_P)\phi^{(B)}_{i,j}, \tag{1}$$

where $L = [(\texttt{interp\_width}-1)/2]$ and $H = [\texttt{interp\_width}/2]$. Here $[a]$ denotes the integer part of a. The base functions α_i and β_j are $\texttt{interp\_width}-1$ order polynomials. The functions α_i satisfy $\alpha_i(r_j) = \delta_{ij}$ for $i_L - L \le i, j \le i_L + H$, where δ_{ij} is the Kronecker delta. The functions β_j are defined in a corresponding way.

Depending on the interpolation type, there are two different approaches for doing the interpolation in the overlapping grid; both techniques are supported in the present method. When the interpolation type is implicit, the interpolation formula is allowed to use donor points that them self are interpolation points in the donor grid. This is not allowed when the interpolation type is explicit. Explicit interpolation, which generally leads to slightly wider overlaps between the component grids compared to implicit interpolation, is almost always prefered when a time-dependent problem is solved on the overlapping grid. The solution value at each interpolation point is then independent of the solution values at all other interpolation points. It is therefore possible to update the solution value at the interpolation points after all discretization points have been assigned their value at each time step. Explicit interpolation can also be used for elliptic problems, where it is especially useful in connection with domain decomposition techniques. Implicit interpolation is usually only used when elliptic equations are solved by a direct method like Gaussian elimination. In this case the interpolation relations are additional linear equations that are solved simultaneously to the discretized elliptic equation on each component grid.

We proceed by discussing the proper requirements to put on discretization and interpolation points in an overlapping grid.

2.1 Discretization points

The width of the discretization stencil is governed by the order of accuracy and the order of the PDE. It is easy to realize that in the general case, different discretization widths must be allowed for in the discretization formula depending on if the grid point is separated away from all boundaries, close to a boundary, or close to a corner. We denote the different widths as follows:

`disc_width` The width of the discretization formula away from boundaries. The overlapping grid will be constructed for a centered discretization stencil, so we assume that $\texttt{disc\_width} \ge 3$ is odd.

`normal_width`, `tangent_width` The width of the discretization formula in the normal and tangential directions relative to the boundary, on or close to one physical boundary. No preference is given to a particular direction in the grid, so `tangent_width` is assumed to be odd. We assume that `tangent_width` $\leq$ `disc_width` so that all boundary points that are at least (`disc_width` $- 1)/2$ points away from a corner can be discretized by the boundary formula.

`corner_width` The width of the discretization formula on or close to a physical corner, i.e. a corner where two physical boundaries from the same component grid meet.

Depending on whether a grid point is separated away from boundaries, close to a boundary, or close to a corner, the definition of a valid discretization point falls into one of the following three categories:

Interior point. A grid point (i^*, j^*) is a valid interior point if all of the following grid points (i, j) exist and either are discretization points or interpolation points:

$$i^* - \frac{\texttt{disc\_width} - 1}{2} \leq i \leq i^* + \frac{\texttt{disc\_width} - 1}{2},$$
$$j^* - \frac{\texttt{disc\_width} - 1}{2} \leq j \leq j^* + \frac{\texttt{disc\_width} - 1}{2}.$$

Boundary point. This rule is used only if a grid point (i^*, j^*) is on or so close to *one* physical boundary that some of the grid points in the definition of an interior point do not exist. For example, let $i^* \leq (\texttt{disc\_width} - 1)/2$. The grid point (i^*, j^*) is then a valid boundary point if all of the following grid points (i, j) exist and either are discretization points or interpolation points:

$$1 \leq i \leq \texttt{normal\_width},$$
$$j^* - \frac{\texttt{tangent\_width} - 1}{2} \leq j \leq j^* + \frac{\texttt{tangent\_width} - 1}{2}.$$

Corner point. This rule is used only if a grid point (i^*, j^*) is on or so close to *two* physical boundaries that the point is neither an interior point, nor a boundary point. For example, this happens when $i^* \leq (\texttt{disc\_width} - 1)/2$ and $j^* \leq (\texttt{disc\_width} - 1)/2$. The point is then a valid corner point if all of the following grid points (i, j) exist and are either discretization or interpolation points:

$$1 \leq i \leq \texttt{corner\_width},$$
$$1 \leq j \leq \texttt{corner\_width}.$$

As a consequence of these definitions, there can not be a discretization point on an interpolating boundary.

2.2 Interpolation points

The width of the interpolation stencil, `interp_width` ≥ 2, is chosen based on the order of accuracy, the type of the PDE (elliptic, parabolic, hyperbolic, etc.), and by the behavior of the overlap when the grid size decreases. For example, if a second order elliptic equation is discretized to second order accuracy, it is necessary to use third order (bi-quadratic) interpolation if the width of the overlap domain is proportional to the grid size. However, second order (bi-linear) interpolation suffices if the width of the overlap domain is independent of the grid size. For more details see [6].

We define a grid point (i,j) in grid A to be a valid interpolation point if it is inside a donor grid B and the interpolation location is valid. A grid point (i_L, j_L) is a valid interpolation location if all of the following donor grid points (i_d, j_d) in the donor grid B exist and either are discretization points or interpolation points:

$$i_L - \left\lceil \frac{\texttt{interp\_width} - 1}{2} \right\rceil \leq i_d \leq i_L + \left\lceil \frac{\texttt{interp\_width}}{2} \right\rceil,$$
$$j_L - \left\lceil \frac{\texttt{interp\_width} - 1}{2} \right\rceil \leq j_d \leq j_L + \left\lceil \frac{\texttt{interp\_width}}{2} \right\rceil.$$

Note that when `interp_width` is even, the interpolation location $(i_L, j_L) = (p,q)$ is defined by the position of the grid cell $Q^{(B)}_{p,q}$ that contains $\mathbf{x}^{(A)}_{i,j}$. But when `interp_width` is odd, the interpolation location is the closest grid point to $\mathbf{x}^{(A)}_{i,j}$ in the cell $Q^{(B)}_{p,q}$. The interpolation location is modified to allow for a less centered interpolation formula with the same interpolation width when the enclosing grid cell $Q^{(B)}_{p,q}$ is so close to a physical boundary in grid B that the centered formula can not be used. Because we do not want the overlap between two grids to be arbitrarily small, we do not allow the interpolation stencil to be non-centered close to an interpolating boundary in grid B. When `interp_width` $= 2$, we must also require the parameter value of the interpolation point $\mathbf{r}_P$ to be separated by at least half a grid cell from all interpolating boundaries in grid B.

When the same physical boundary is represented by the sides of two or more component grids, a small mismatch can sometimes occur between the boundary descriptions in the overlap region. The mismatch is especially likely to happen on curved boundaries that are not very well resolved by the grid on the boundary, cf. figure 3. Under these circumstances it is necessary to allow the interpolation points on the physical boundary to be slightly outside of the donor grid. Hence, it is necessary to label the sides of the component grids that represent the same physical boundary to tell the algorithm when this type of mismatch is allowed. The complications related to boundary interpolation are discussed further in section 4.2.

3 Classifying the grid points

The overlapping grid algorithm classifies each grid point in each component grid into one of three categories:

1. Discretization point, where the partial differential equation or the boundary conditions are discretized.

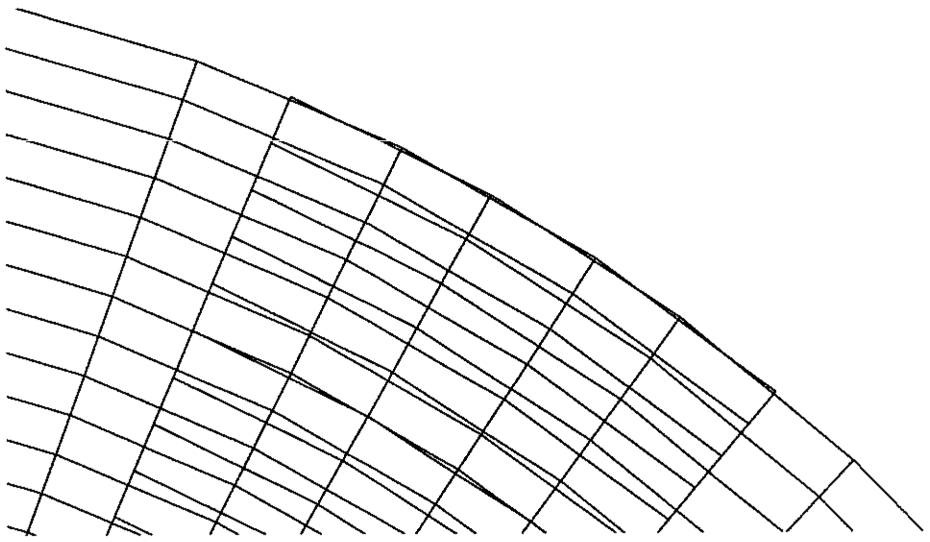

Figure 3: A mismatch between the representations of a physical boundary can lead to failure of the overlapping grid algorithm unless interpolation points on the boundary are allowed to be slightly outside of the donor grid.

2. Interpolation point, where the solution is interpolated from another component grid.
3. Exterior point. A grid point that neither is a discretization point, nor an interpolation point.

For each interpolation point, the overlapping grid algorithm also determines the donor grid, the interpolation location, and the weights in the interpolation formula. The classification of each grid point (i, j) in each component grid k in the overlapping grid is described by the `flag` array according to:

$$\texttt{flag}(i,j,k) = \begin{cases} k, & \text{Discretization point,} \\ q, & \text{Interpolation point, interpolating from grid } q, \\ 0, & \text{Exterior point.} \end{cases}$$

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 k has priority k. 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.

We proceed by describing the six steps that constitute the classification algorithm: initialization, boundary preparation, main classification, explicit interpolation, consistency check, and trimming.

Initialization. All grid points are initialized to be discretization points:

$$\texttt{flag}(i,j,k) = k, \quad 1 \le i \le N_k, \quad 1 \le j \le M_k, \quad 1 \le k \le \texttt{n\_grids}.$$

Boundary preparation. To be able to classify all grid points away from the boundaries in just one iteration, it is necessary to do some preparatory classification of the boundary points before the main classification step can be executed. To make sure that all grid points outside of the physical domain will be flagged as exterior points during the main classification step, we go through all grid points on physical boundaries in each component grid to find all grid cells in other component grids that contain the grid point on the physical boundary. The grid points in these cells that are outside of the component grid in question are flagged as exterior if they are not them self situated on a physical boundary in their own component grid.

In order to prevent the formation of "orphan point" islands, the user is required to identify each component grid corner that is outside of the computational domain. Starting at each exterior corner, it is first checked if the corner is inside of any other component grid and a list of these exterior component grids is formed. Note that this list is often empty. Because the boundary of the computational domain is always discretized by boundary conforming component grids, a grid point can only be inside of the computational domain if it is inside of a component grid not in the exterior grid list. We therefore proceed along the two boundaries that meet at the exterior corner and mark every grid point that is outside of all component grids not in the exterior grid list. The marking is terminated at the first non-exterior grid point. We then repeat that procedure along each grid line that starts at an exterior boundary point. Hence, this step marks all exterior points close to the exterior corner. While we do not attempt to prove that the above procedure always leads to an overlapping grid free from "orphan points", our experience from many different overlapping grids indicates that its works well in practice.

To correctly identify interpolation points on a mixed interpolating / physical boundary during the main classification algorithm, we go through all grid points on those boundaries and mark the grid points that are sufficiently far inside of another component grid as interpolation boundary points. By sufficiently far inside, we here mean so far inside of the donor grid that the interpolation location is valid. To ensure explicit interpolation when two mixed interpolating / physical boundaries intersect, the interpolation location is only valid if it is an additional $(\texttt{disc\_width} - 1)/2$ grid points away from the other mixed boundary. The grid points on the mixed boundary that were not interpolation boundary points are marked as physical boundary points.

Main classification. To more easily present the details of the main classification algorithm, we introduce three functions that implement the rules for discretization and interpolation points that were described in section 2.

`interp_from_higher`$(\mathbf{x}, k)$ Check if a grid point with Cartesian coordinate $\mathbf{x}$ can be an interpolation point that interpolates from a donor grid with higher priority than k. Return the priority of the highest such donor grid, or if $\mathbf{x}$ could not interpolate from a higher grid, return zero.

```
for  $k = \texttt{n\_grids}, \texttt{n\_grids} - 1, \ldots, 1$
    for  $i = 1, 2, \ldots, N_k$
        for  $j = 1, 2, \ldots, M_k$

/* Do not alter the points that were classified as exterior in the previous step. */
            if flag$(i, j, k) \neq 0$ then
              interpolee = interp_from_higher($\mathbf{x}_{i,j}^{(k)}, k$);
              if  interpolee > 0 then
                flag$(i, j, k)$ = interpolee;
              else if discretization_point$(i, j, k)$ then
                flag$(i, j, k)$ = $k$;
              else
                flag$(i, j, k)$ = interp_from_lower($\mathbf{x}_{i,j}^{(k)}, k$);
              end if
            end if

        end for
    end for
end for
```

Figure 4: Main classification algorithm in pseudo-C-code.

`interp_from_lower(x, k)` As above, but restrict the search to donor grids with priority less than k.

`discretization_point(i, j, k)` Return TRUE if the grid point (i, j) in grid k is a valid discretization point, otherwise return FALSE.

Pseudo-C-code for the main classification algorithm is presented in figure 4. The algorithm starts with the grid with the highest priority and proceeds in decreasing priority order such that fewer restrictions are put on grids with higher priority. For each grid point we first investigate if it can interpolate from a grid with higher priority. If this is not possible, we find out if it is a valid discretization point, and if it is not, we instead check if it can interpolate from a lower grid. Finally, if not even this is possible, the grid point is flagged to be exterior.

It is essential that the grid points in each grid are traversed in a sequential manner so that information about the classification of the neighboring grid points is efficiently propagated through the grid. We have found a line ordering to be adequate, as is indicated in the pseudo-C-code. After the main classification, the overlapping grid will in general have implicit interpolation with unnecessarily many interpolation points in the overlap regions. These deficiencies will be taken care of in the steps to follow.

Explicit interpolation. The next step is to eliminate any implicit interpolations, so this step is disregarded if the interpolation type is implicit. We present the algorithm in pseudo-C-code in figure 5. As in the main classification step, we start with the highest grid and proceed in decreasing priority order such that fewer restrictions are enforced on the higher grids. For each interpolation point, we determine if the interpolation is implicit or explicit by checking if any of the donor points are interpolation points in the donor grid. If the interpolation is explicit, we proceed to next interpolation point. Otherwise, we first try to reclassify the donor points that are interpolation points in the donor grid. If all these donor points can be reclassified to be discretization points in the donor grid, the interpolation point becomes explicit, and we proceed to next interpolation point. However, if it is not possible to reclassify all of them, the interpolation point is flagged as an implicit point. The algorithm proceeds by reclassifying the implicit interpolation point. To be consistent with the initial classification strategy, we first investigate if the interpolation point can interpolate from a donor grid with lower priority than the present donor grid, but with higher priority than that of the interpolation point's grid. If this fails, we check if it can be a discretization point. And if this is not the case, we instead attempt at having the interpolation point interpolate from a donor grid with lower priority than that of the interpolation point's grid. If not even this is possible, we flag the implicit interpolation point as an exterior point. Observe that if the interpolation point was assigned to interpolate from a new donor grid, it is necessary to reiterate and check if the new donor points cause the interpolation to be implicit.

Consistency check. At this point, it is appropriate to check if the classification of the grid points is self-consistent, i.e. if all discretization and interpolation points satisfy the necessary requirements. For example, the overlapping grid might not be self-consistent where two very coarse grids overlap each other. The self-consistency step will mark all points that do not satisfy the appropriate rule and report these points to the user of the code, so that he can see where the algorithm was unsuccessful. When invalid grid points are encountered, the user should check that the following properties of the component grids are satisfied:

- All parts of the physical domain must be covered by component grids.
- There must be sufficiently many grid points in the component grids for the interpolation relation to be formed.
- The boundary condition information must correctly correspond to physical, interpolation and exterior boundaries (or a mixture of the three) of the computational domain.
- The sides of component grids that share the same physical boundary must be labeled consistently to allow for boundary interpolation.
- All exterior corners must been identified to avoid the formation of islands of "orphan points".

```
for  $k$ = n_grids, n_grids − 1, …, 1

    for this_interp_point = each interpolation point in grid $k$
        donor_grid = donor grid for this_interp_point;
        do
            new_donor = FALSE; implicit = FALSE;
            for $(i,j)$ = index of each donor point for this_interp_point

/* Check if this interpolation is implicit, i.e. if the donor point is an interpolation point. */
                if flag($i,j$, donor_grid) $\neq$ donor_grid then

/* Check if the donor point can be a discretization point instead. */
                    if (not implicit) and discretization_point($i,j$, donor_grid) then
                        flag($i,j$, donor_grid) = donor_grid;
                    else
                        implicit = TRUE;
                    end if
                end if
            end for

/* The interpolation point is still implicit. */
            if implicit then
                $(i,j)$ = index of this_interp_point;

/* Try to find a donor grid with a lower priority. */
                donor_grid = interp_from_lower($x^{(k)}_{i,j}$, donor_grid);

/* Try to reclassify the interpolation point $(i,j)$ as a discretization point. */
                if donor_grid $< k$ and discretization_point($i,j,k$) then
                    flag($i,j,k$) = $k$;
                else
                    flag($i,j,k$) = donor_grid;
                    new_donor = (flag($i,j,k$) $\neq 0$);
                end if
            end if

/* Check the new donor points for implicit interpolation. */
        while (new_donor);
    end for

end for
```

Figure 5: Pseudo-C-code for removing implicit interpolation points.

If the overlapping grid is found to be self-consistent we proceed to next step, where all unnecessary interpolation points in the overlap regions are trimmed away.

Trimming. The philosophy behind the trimming step is to minimize the total number of grid points in the overlap domain. We will therefore aim at reclassifying interpolation points into exterior points. The trimming algorithm employed here is very similar to the trimming step in the method by Chesshire and Henshaw [6].

When the interpolation type is implicit, we must before the trimming step inspect each interpolation point in each grid to check if its donor grid has lower priority than the priority of the interpolation point's grid. In that case we mark the corresponding donor points to make sure that they are not removed during the trimming algorithm. When the interpolation type is explicit, all points can be regarded as unmarked, since the trimming step only reclassifies interpolation points and we know that all donor points are discretization points when the interpolation type is explicit.

To better describe the trimming algorithm, we introduce two functions that will be used in the following pseudo-code.

`needed_by_disc`(i,j,k) Return TRUE if the interpolation point (i,j) in grid k is needed by a discretization point in grid k. Otherwise return FALSE. There are three kinds of discretization points, so it is necessary to check if the point (i,j) is needed by any interior, boundary, or corner point. However, a grid point that is more than `corner_width` points away from the nearest physical corner and more than `normal_width` points away from the nearest physical boundary can only be needed by an interior discretization point. In that case, the interpolation point (i,j) is needed if at least one of the following points (i_d, j_d) is a discretization point:

$$i - \frac{\texttt{disc\_width} - 1}{2} \le i_d \le i + \frac{\texttt{disc\_width} - 1}{2},$$
$$j - \frac{\texttt{disc\_width} - 1}{2} \le j_d \le j + \frac{\texttt{disc\_width} - 1}{2}.$$

`needed_by_interp`(i,j,k) Return TRUE if the interpolation point (i,j) in the grid k is a donor point for any interpolation point in another grid. Otherwise return FALSE. Note that when the interpolation type is explicit, interpolation points can not be donor points for other interpolation points in different component grids.

The trimming algorithm, cf. figure 6, starts at the component grid with the lowest priority and proceeds to the component grid with the highest priority. The list of interpolation points for each grid is traversed three times. During the first examination, each unmarked interpolation point is examined. If the present interpolation point is neither needed by a discretization point, nor another interpolation point, the interpolation point is removed and the grid point is reclassified to be an exterior point. The interpolation points in the present grid are then traversed again. This time we attempt at reclassifying all unmarked interpolation points into valid discretization points. During the third and last inspection of the interpolation points, which only is done when the interpolation type is implicit, we mark all donor points if the donor grid has higher priority than the present grid. The purpose of the marking is to make sure that the donor points are not removed when the grids with higher priority are trimmed. The marking is not necessary if the interpolation type is explicit, because in that case, the donor points can not be interpolation points.

The trimming step completes the overlapping grid algorithm. We remark that it is easy to change the trimming algorithm to produce a grid where the size of the overlap is essentially independent of the grid size. To achieve this, we would only need to do the first two sub-steps of the trimming algorithm in opposite order, i.e. first reclassified as many interpolation points as possible to be discretization points, and thereafter remove the interpolation points that were not needed.

4 Determining if a grid point can be an interpolation point

A key operation in the overlapping grid algorithm is to determine if a grid point in component grid A with Cartesian coordinates $\mathbf{x}_P^{(A)}$ can be an interpolation point. To answer this question we must be able to tell if the point lies inside another component grid B and in that case find the parameter value $\mathbf{r}_P^{(B)} : X^{(B)}(\mathbf{r}_P^{(B)}) = \mathbf{x}_P^{(A)}$, such that the donor points can be identified and the weights in the interpolation formula (1) can be computed.

Initially, the minimum and maximum Cartesian coordinates $\mathbf{x}_{min}^{(k)}$ and $\mathbf{x}_{max}^{(k)}$ are computed for every component grid. Hence, a point $\mathbf{x}_P^{(A)}$ can only be inside of component grid B if it is inside of the corresponding bounding box. Depending on the type of grid, different strategies are used to determine if a point which is inside of the bounding box is inside of the component grid. For simple mappings like those of a Cartesian or an annular grid, it is possible to invert the mapping analytically. For other more complicated mappings, like when a grid is grown out from a spline curve in the direction of the normal of the curve, only the forward mapping function $X^{(B)}(\mathbf{r})$ is known. And in the most general case, the mapping function is only known at the grid points, i.e. we only know $\mathbf{x}_{i,j}^{(B)}$.

When the inverse of the mapping is known, we determine if a point $\mathbf{x}_P^{(A)}$ is inside of component grid B by first computing the corresponding parameter value $\mathbf{r}_P^{(B)}$ and then checking if it is inside the unit square in the parameter plane: $0 \leq r_P^{(B)} \leq 1$, $0 \leq s_P^{(B)} \leq 1$. If the width of the interpolation stencil, `interp_width`, is odd, the interpolation location is the closest grid point to $\mathbf{r}_P^{(B)}$: $i_L = 1 + [r_P^{(B)}(N_B - 1) + 0.5]$, $j_L = 1 + [s_P^{(B)}(M_B - 1) + 0.5]$. And if `interp_width` is even, the interpolation location equals the index of the enclosing grid cell: $i_L = 1 + [r_P^{(B)}(N_B - 1)]$, $j_L = 1 + [s_P^{(B)}(M_B - 1)]$. The interpolation location is adjusted when (i_L, j_L) is so close to a physical boundary that some of the points in the interpolation formula are outside of the component grid. However, the point $\mathbf{x}_P^{(A)}$ is considered to be an invalid interpolation point if the interpolation location is that close to an interpolation or external boundary, or if $\mathbf{r}_P^{(B)}$ is closer than half a grid cell from any interpolation or external boundary.

```
for  $k = 1, 2, \ldots,$ n_grids

    for this_interp_point = each interpolation point in grid $k$
        $(i, j)$ = index of this_interp_point;
        if interp_type = explicit or not marked$(i, j, k)$ then
          if not needed_by_disc$(i, j, k)$ and not needed_by_interp$(i, j, k)$ then
            flag$(i, j, k) = 0$;
          end if
        end if
    end for

    for $(i, j)$ = index of each interpolation point in grid $k$
        if discretization_point$(i, j, k)$ then
          flag$(i, j, k) = k$;
        end if
    end for

    if interp_type = implicit then
      for this_interp_point = each interpolation point in grid $k$
          if donor_grid(this_interp_point) $> k$ then
            for $(i, j)$ = index of each donor point for this_interp_point;
                marked$(i, j,$ donor_grid$)$ = TRUE;
            end for
          end if
      end for
    end if

end for
```

Figure 6: Pseudo-C-code for the trimming algorithm.

When only the forward mapping is known, we can in principle apply Newton's method to iteratively invert the mapping to find the parameter value $\mathbf{r}_P^{(B)}$. However, Newton's method only converges if the initial guess is sufficiently close to the solution. Furthermore, if $\mathbf{x}_P^{(A)}$ is outside of grid B, it is not always meaningful to invert the mapping because it can be undefined or singular outside of the unit square in the parameter space. The situation is similar when we only know the mapping at the grid points, but in that case we also need to approximate the mapping locally by interpolation to determine the parameter value $\mathbf{r}_P^{(B)}$. For these reasons, a different approach is needed to determine if the point $\mathbf{x}_P^{(A)}$ is inside of grid B when the inverse of the mapping is not known. Furthermore, if the point is inside the grid, a method must be devised to approximately find where the point is in the grid to give Newton's method a good initial guess or to decide where the mapping should be interpolated locally.

4.1 Locating the enclosing grid cell

An exhaustive search through all grid cells in the donor grid could obviously be used to find the closest cell to $\mathbf{x}_P^{(A)}$. However, this becomes prohibitively expensive for fine grids, because it requires $\mathcal{O}(N_B M_B)$ operations. Another possibility which was employed by Meakin [7] is to cover each component by a fine Cartesian "help grid" and á priori invert the mapping at all grid points in the "help grid". These point values are then used to invert the mapping for interpolation points which lie in between the grid points in the "help grid". This approach requires a significant memory overhead, but is efficient when some component grids translate and rotate relative to the rest of the component grids. It is then possible to use the same "help grids" for computing the overlapping grids corresponding to each position of the moving component grids.

In the present method, we use a less memory requiring technique which still is much more efficient than searching through all grid points. The method is based on a well-known lemma from computational geometry [9].

Lemma 1 *A point $\mathbf{x}_P \in \Re^2$ is inside a bounded domain $\Omega \subset \Re^2$ with continuous boundary $\partial\Omega$ if and only if a ray, which starts at $\mathbf{x}_P$ and ends at infinity, intersects $\partial\Omega$ an odd number of times.*

Remark: The lemma also holds in three space dimensions.

For simplicity, we will in the remainder of this section drop the grid indices A and B. We define a grid line $L_{a,b}^{c,d}$, where either $a = b$ or $c = d$, to be the polygon that connects the grid points $\mathbf{x}_{i,j}$ with $a \le i \le b$ and $c \le j \le d$. Let the ray be horizontal with Cartesian coordinates $\mathbf{x}_R = (x_R, y_R)$ satisfying

$$x_R \le x_P, \quad y_R \equiv y_P.$$

It is easy to count the number of intersections between the ray and a grid line $L_{a,b}^{c,c}$ or $L_{a,a}^{c,d}$. The method will be demonstrated for the constant-j line $L_{a,b}^{c,c}$. We will separately check each straight line between the grid points $\mathbf{x}_{i,c}$ and $\mathbf{x}_{i+1,c}$ for $i = a, a+1, \ldots, b-1$.

Some care must be taken to avoid counting an intersection twice if the y-coordinate of the ray coincides with that of one grid point. We therefore say that no intersection occurs if

$$\min(y_{i,c}, y_{i+1,c}) \geq y_P \tag{2}$$

or

$$\max(y_{i,c}, y_{i+1,c}) < y_P \tag{3}$$

However, if $\min(y_{i,c}, y_{i+1,c}) < y_p \leq \max(y_{i,c}, y_{i+1,c})$, a necessary condition for intersection is

$$\min(x_{i,c}, x_{i+1,c}) < x_P, \tag{4}$$

and a sufficient condition for intersection is

$$\max(x_{i,c}, x_{i+1,c}) < x_P. \tag{5}$$

In the borderline case, when the necessary but not the sufficient condition for intersection is satisfied, the x-coordinate for the straight line at $y = y_P$ must be computed:

$$x^* = x_{i,c} + \frac{x_{i+1,c} - x_{i,c}}{y_{i+1,c} - y_{i,c}}(y_P - y_{i,c}). \tag{6}$$

Intersection occurs if

$$x^* < x_P.$$

Note that in most cases, only the two inequalities (2) and (3) need to be checked to exclude the possibility for an intersection. And if y_P is found to be in the right interval, most cases are resolved by checking the additional two inequalities (4) and (5). It is only in rare borderline cases when the intersection formula (6) must be evaluated. There are $b-a$ straight lines along the grid line $L^{c,c}_{a,b}$, so the number of intersections can be determined in $\mathcal{O}(b-a)$ operations.

It can now be verified if a point $\mathbf{x}_P$ is inside the donor grid by applying lemma 1 to the polygon consisting of the four grid lines $L^{1,1}_{1,N}$, $L^{M,M}_{1,N}$, $L^{1,M}_{1,1}$, and $L^{1,M}_{N,N}$. If the total number of intersections between the ray and the four grid lines is odd, the point $\mathbf{x}_P$ is inside the donor grid, otherwise it is outside.

It is not necessary to traverse through all grid points on the boundary to count the number of intersections with the ray. The number of operations can be greatly reduced by subdividing each side of each component grid in a binary tree structure before the overlapping grid algorithm is started, cf. figure 7. To count the number of intersections with the ray, it is only necessary to count the number of intersections with the sub-levels whose bounding box is intersected by the ray. We found by experiments that the highest efficiency occured when each branch of the tree was subdivided recursively until it contained less than 5 grid points. The operational count is as follows. The least expensive case occurs when the ray does not intersect the top level bounding box. This case only requires $\mathcal{O}(1)$ operations. When the ray intersects the top level bounding box, it is likely to only intersect one of the bounding boxes on each sub-level. There are $\mathcal{O}(\log_2 N)$ levels on a boundary with N grid points, so this case requires of the order $\mathcal{O}(\log_2 N)$ operations. In the worst case scenario, which is rather unlikely to happen, the boundary oscillates wildly

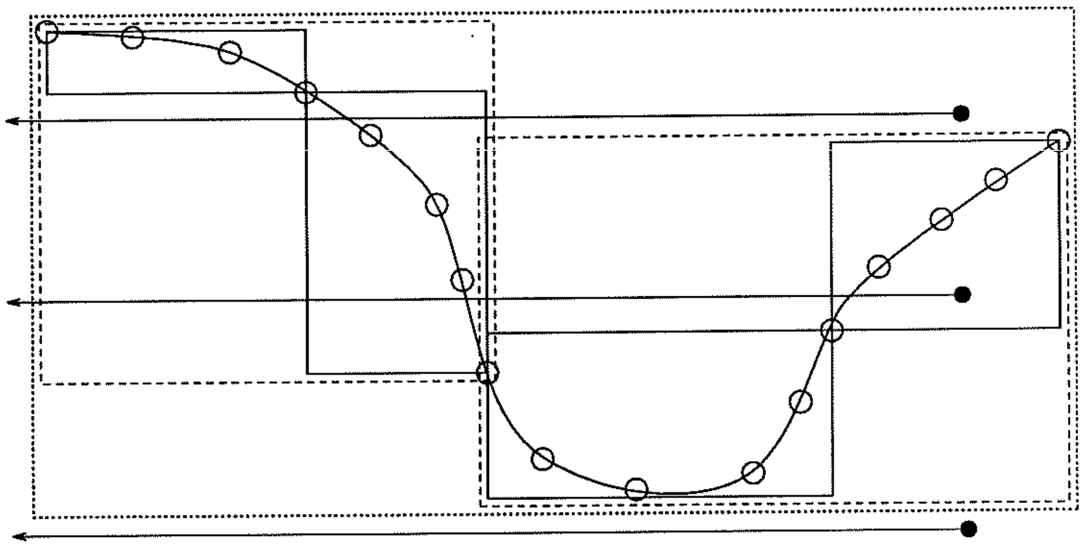

Figure 7: The boundary of each component grid is subdivided in a binary tree structure to speed up the counting of intersections between the ray and the boundary. Circles indicate grid points on the boundary and filled circles mark different locations of the grid point $\mathbf{x}_P$. Also shown are the bounding boxes corresponding to three levels in the binary search tree.

and the ray intersects every sub-division of the boundary. This leads to an operation count of $\mathcal{O}(N)$. The total number of operations for determining if a point is inside a component grid follows by summing the effort in counting the number of intersections with the four bounding grid lines. If $\mathbf{x}_P$ is inside the bounding box of the component grid, the ray must intersect at least one of the top level bounding boxes. By the above argument, the algorithm is most likely to require $\mathcal{O}(\log_2 N + \log_2 M)$ operations, but can in rare difficult cases take up to $\mathcal{O}(N + M)$ operations.

If $\mathbf{x}_P$ is found to be inside the donor grid, we proceed by checking in which grid cell $Q_{i,j}$ it is located. This is to supply a good initial guess for Newton's method when the forward mapping is known or to determine where the mapping needs to be approximated locally when only the location of the grid points is known. For this purpose, we use a bisection algorithm. In the general case, we know that $\mathbf{x}_P$ is inside the grid bounded by the four grid lines $L^{1,1}_{1,N}$, $L^{M,M}_{1,N}$, $L^{1,M}_{1,1}$, and $L^{1,M}_{N,N}$. We now subdivide the grid along the grid lines $i = [(1+N)/2]$ and $j = [(1+M)/2]$, which results in four sub-grids. By counting the number of intersections between the ray starting at $\mathbf{x}_P$ and the boundaries of the sub-grids, we can determine in which sub-grid $\mathbf{x}_P$ is located. We then repeat the procedure recursively until only one sub-grid containing only one grid cell remains. This determines in which grid cell $Q_{i,j}$ the point $\mathbf{x}_P$ is located.

Because the grid points are classified sequentially in the overlapping grid algorithm, information about the previous grid point is often available. In this case, the efficiency of the algorithm for locating the enclosing grid cell can be substantially improved. Instead of starting the bisection at the boundary of the donor grid, we then begin by growing

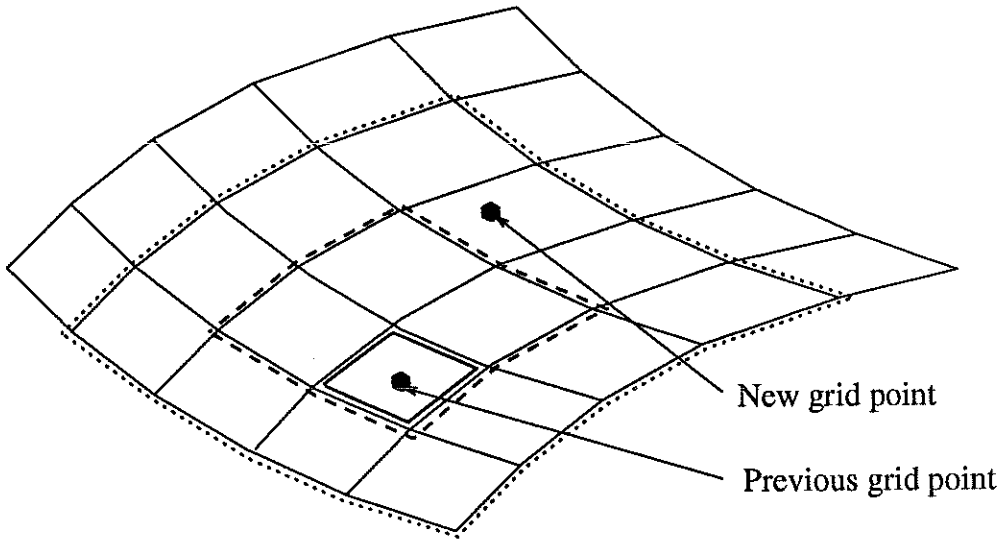

Figure 8: The sub-grid is grown by a factor two around the previous grid point until the new grid point is enclosed by the sub-grid. The sub-grid is then shrunken until the enclosing grid cell for the new grid point is located.

a sub-grid around the grid cell that enclosed the previous grid point, see figure 8. We grow the size of the sub-grid by a factor two until the new point is enclosed. The previous bisection technique is then applied to shrink the sub-grid down to locate the new enclosing grid cell. Naturally, the new grid point can sometimes be outside of the donor grid. To incorporate this case into the algorithm, we limit the growth to a 8×8 sub-grid. If the new grid point is outside of that sub-grid, we treat it as a grid point without an initial guess.

When there is an initial guess for the enclosing grid cell, the new enclosing grid cell can be located in $\mathcal{O}(1)$ operations. In the absence of an initial guess, approximately $\log_2(\max(N, M))$ subdivisions of the grid are required to locate the grid cell $Q_{i,j}$. And $\mathcal{O}((N+M)/2^q)$ operations are necessary to proceed from subdivision q to $q+1$ because we only compute subdivision information for the boundaries of the component grid. Hence, the operational count becomes $\mathcal{O}(N + M)$ in the absence of an initial guess.

If the forward mapping is known, Newton's method can be applied to find the parameter coordinate $\mathbf{r}_P$ corresponding to $\mathbf{x}_P$, which is needed by the interpolation formula (1), by taking the initial guess to be the parameter value at the center of the enclosing grid cell $Q_{i,j}$: $r = (i - 0.5)/(N_B - 1)$, $s = (j - 0.5)/(M_B - 1)$. The interpolation location is then found in the same way as in the case when the inverse of the mapping is known.

When only the location of the grid points is known, it is necessary to approximate the mapping locally by interpolation before the parameter coordinate $\mathbf{r}_P$ can be computed. It is consistent to approximate the mapping by a Lagrangian interpolation formula of the same width as when the solution value is interpolated because both interpolations lead to errors that are of the same order of accuracy. When `interp_width` is even, the enclosing grid cell uniquely determines the location of the interpolation stencil. However, this is not the case when `interp_width` is odd. We resolve this redundancy by increasing the width of the interpolation stencil for the mapping by one to always make it even. This is harmless, since it only makes the interpolation of the mapping more accurate. Hence, the interpolation formula is always centered around the enclosing grid cell unless a component grid boundary forces the interpolation stencil to be skewed. Once the location of the interpolation stencil has been determined, the Lagrangian interpolant is inverted by a Newton iteration to approximate the parameter value $\mathbf{r}_P$ corresponding to $\mathbf{x}_P$.

4.2 Boundary interpolation

Even when the polygonal approximation of the grid boundary indicates that the point $\mathbf{x}_P$ is inside of the grid, it is still possible that the Newton iteration converges to a point in parameter space that is slightly outside of the unit square. The reason for this is that when the boundary is concave, the polygonal approximation is always slightly outside of the boundary. The opposite phenomena happens when the boundary is convex; then the polygonal approximation is always inside of the real boundary, so points that are very close to the boundary, but inside the unit square in parameter space, will be classified as being outside of the grid when the polygonal approximation of the boundary is used. These matters are only of importance when the boundary is physical and when $\mathbf{x}_P$ is a grid point on a physical boundary, because according to the above rules, $\mathbf{x}_P$ is not allowed to interpolate from points on an interpolating boundary. And when $\mathbf{x}_P$ is not on a physical boundary, it is not crucial for the success of the algorithm that it becomes an interpolation point, because there are other grid points in that component grid that can relive $\mathbf{x}_P$ from its duties as an interpolation point.

However, when the boundary in question is physical and when the point $\mathbf{x}_P$ is on a physical boundary, it is vital for the success of the algorithm that $\mathbf{x}_P$ is allowed to interpolate from the boundary points in the overlapping component grid, even though the grid point is slightly outside of that donor grid according to its mapping function. This situation is illustrated in figure 3. Observe that this problem can also occur when the inverse of the mapping is known, because there can still be a small mismatch between the representations of the physical boundary in the overlap domain. To signal the importance of this matter to the algorithm, the user must assign the the same `curve_label` to the sides of the component grids that represent the same physical boundary.

A tolerance ϵ is provided to the algorithm to indicate how far a grid point can be outside of a boundary with matching `curve_label` and still be considered to be inside for the purpose of boundary interpolation. To investigate if a point satisfies this criterion, we locate the grid point on the boundary that is closest to $\mathbf{x}_P$. The distance between $\mathbf{x}_P$ and that boundary grid point is decomposed into the normal and tangential components relative to the boundary. If the normal distance is less than ϵ and the tangential projection of $\mathbf{x}_P$ onto the boundary is between the endpoints of the boundary, we consider the point $\mathbf{x}_P$ to be sufficiently close to the boundary for boundary interpolation. Newton's method is then applied to determine the parameter value $\mathbf{r}_P$ in the same way as before.

The above boundary interpolation procedure will not perform well when the grid is very fine in the direction normal to the boundary and the grid cells have a large aspect ratio, which is common when a boundary layer behavior is expected in the solution. In fluid mechanics applications, the grid cells at the boundary can have an aspect ratio of 1000. It is then possible that physical boundary points will interpolate from several cells inside of the boundary in the donor grid. This problem was addressed by Parks et. al [10]. They suggest that the interpolation points on the physical boundary should be moved to coincide with the description of the boundary of the donor grid. That component grid should then be re-generated using the improved surface description, or alternatively, all grid points along the grid line that starts at the physical boundary point and extends into the computational domain should be moved the same distance to correct the mismatch. Hence, by this approach the difficulty is moved from the overlapping grid algorithm to the component grid generator.

5 Examples

The overlapping grid algorithm has been implemented together with a geometry modeling package and a number of grid generators in an interactive code called Xcog (eXtendible Composite Overlapping Grid-generator). Xcog's user interface provides a set of powerful commands for defining the geometry, the component grids, the stretching functions, and the overlapping grid. The user has complete freedom in which order the different elements of the overlapping grid are designed and all previously designed elements can easily be modified. The code is written in ANSI-C and uses Xlib for graphical output. This platform exists on any modern UNIX based workstation, which enables the same source code and almost the same configuration procedure to be used on any such machine.

The code consists of a main program and four modules that are responsible for the geometry, the component grids, the stretching functions and the overlapping grid algorithm. The main program provides basic tasks such as reading and writing files and transferring information between the four modules. Each module is designed in an object oriented fashion, so in addition to having a good builtin functionality, it is straight forward to extend their capabilities. For example, if the user needs a stretching function that is not available, it is only necessary to add that specific stretching type to the stretching module. The code is written such that the new stretching type becomes available to the other modules once the modified stretching module has been linked to the program. The same principle applies to the curves in the geometry module, the grids in the component grid module and to the input of the overlapping grid algorithm.

Readers who are interested in obtaining a copy of Xcog should contact the author.

We proceed by making an overlapping grid to demonstrate how the different parts of the algorithm work. We consider the grid in figure 9, which is designed for a second order accurate discretization of a second order PDE with explicit interpolation, `disc_width` $= 3$ and `interp_width` $= 2$. The computational domain is probably not realistic, but it was chosen to demonstrate as many features as possible in one example. A closeup of the right part of the grid is shown in figure 10 together with the boundary conditions and the `curve_label` flags. In Xcog, a positive, zero, or negative boundary condition indicates if

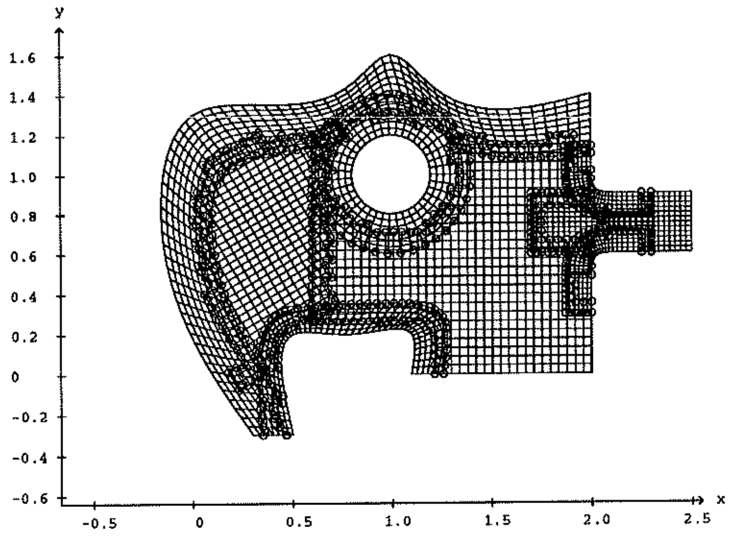

Figure 9: An overlapping grid consisting of eight component grids. The interpolation points are marked with circles.

the component grid boundary is physical, interpolating, or mixed physical / interpolating, respectively. The same `curve_label` must be given to both the upper and lower part of the vertical boundary because both parts are represented by the same side of the Cartesian base-grid on the left.

To show how the algorithm behaves when there are too few grid points to form a valid overlapping grid, we coarsen the two Cartesian grids in the middle of the domain. The result can be seen in figure 11. From this graphical output, it is not difficult to see that the Cartesian grids in fact are too coarse to form valid interpolation relations. Observe that the overlap between the component grids is larger than in figure 9 because the consistency of the grid points is tested before the overlap is trimmed.

The lower left corner of the Cartesian grid on the right side of the circle is exterior to the computational domain, and was marked as exterior to construct the overlapping grid in figure 9. If one attempts to construct the overlapping grid without giving the algorithm the information about the exterior corner, one arrives at an overlapping grid with one orphan point, cf. figure 12.

It is interesting to compare the CPU-time requirements of Xcog and CMPGRD. As a test case, we generated the overlapping grid shown in figure 13 for three different grid refinements. We show the intermediate grid in the figure. The coarsest grid had half as many grid points in each direction of each component and the finest grid had twice as many grid points in each direction of each component. All three overlapping grids were constructed for explicit interpolation with `disc_width` = 3 and `interp_width` = 2. To make the comparison as fair as possible, we did not use the built in geometry modeling or grid generation capabilities of either Xcog or CMPGRD, but rather read in the location of the grid points in PLOT3D format and only use the two codes to compute the overlap

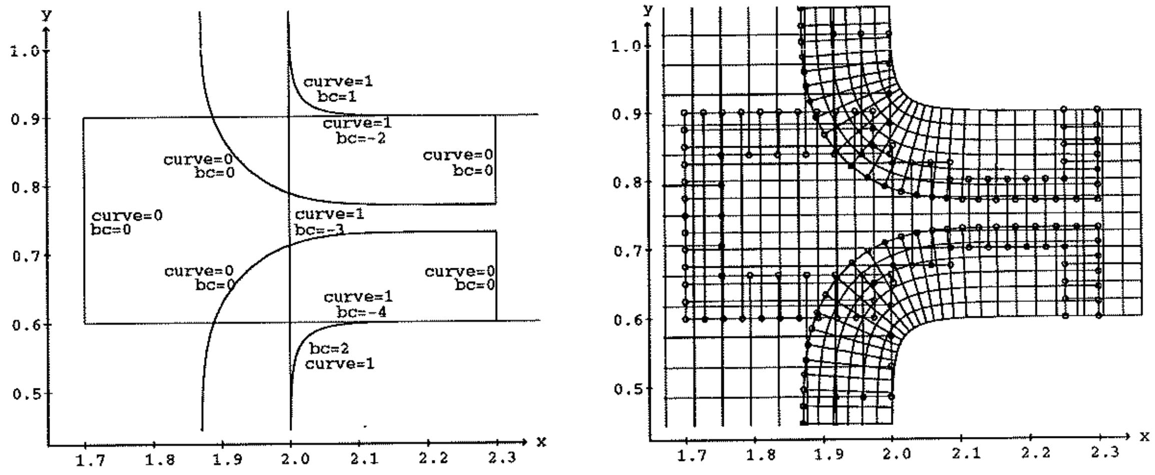

Figure 10: A closeup of the right part of the grid in figure 9. The left figure shows the boundary conditions and the `curve_label` markings that produced the overlapping grid shown on the right. Interpolation points are marked with circles.

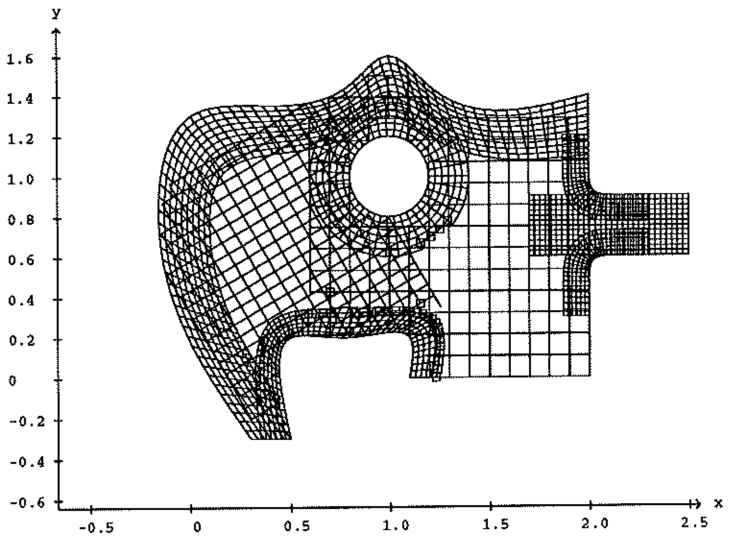

Figure 11: A coarser version of the grid in figure 9, with too few grid points in the two Cartesian grids that surround the circle. The bad discretization points are marked with boxes.

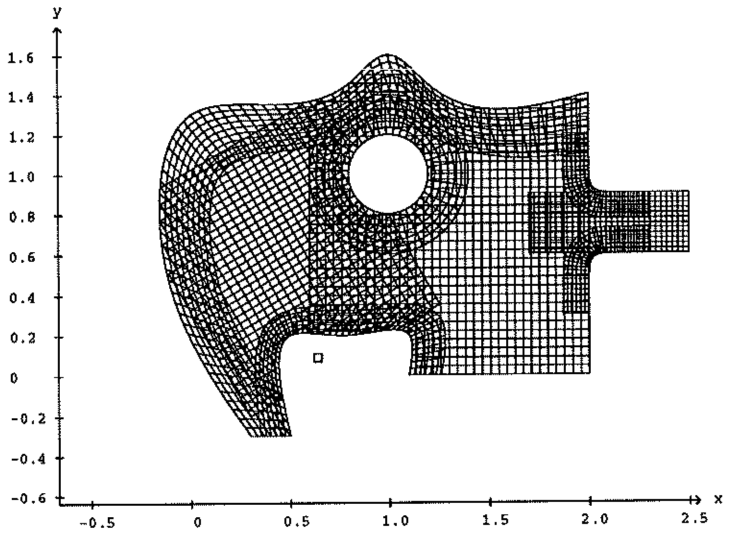

Figure 12: An example of an orphan point. This is the same grid as in figure 9, but where the lower left corner in the Cartesian grid to the right of the circle was not marked as exterior.

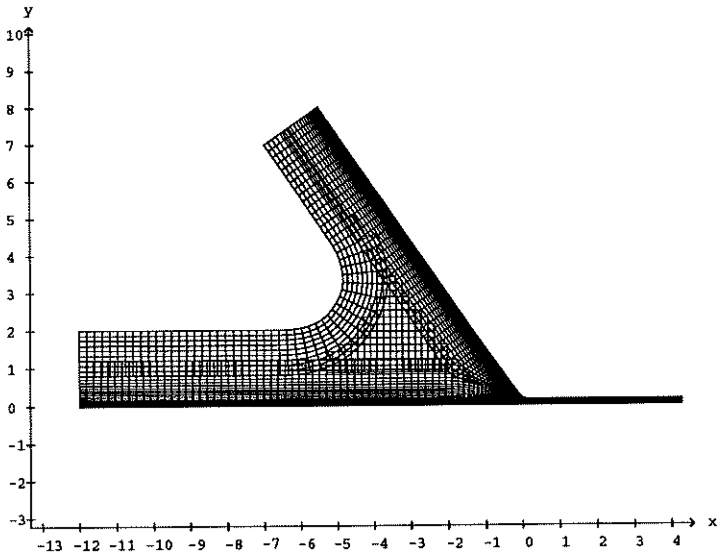

Figure 13: The overlapping grid used for the comparison between Xcog and CMPGRD. The component grids following the upper-left, upper-right, and bottom boundaries have 60×10, 100×24, and 120×24 grid points, respectively. The Cartesian grid in the middle has 30×24 grid points, so the total number of grid points is 6600.

Grid	# grid points	Xcog	CMPGRD
Coarse	1650	0.3	2.7
Medium	6600	1.2	8.8
Fine	26400	5.0	31.0

Table 1: CPU-time in seconds required by Xcog and CMPGRD to compute the overlapping grid in figure 13. The tests were performed on a Tatung 385 with 48 MB of RAM.

information. Because CMPGRD is written in FORTRAN but Xcog is written in C, the same compiler is not available for both codes. To still make a reasonable comparison, we compiled both codes with maximum optimization. The tests were performed on a Tatung 385 (comparable to a SUN-20/51) with 48 MB of RAM. CMPGRD (version June 19, 1995) was compiled with f77 (version SC2.0.1) and Xcog (version 1.0) was compiled with gcc (version 2.6). Both codes were using 64 bits floating point precision during the tests. The execution time for the three grid refinements can be found in table 1. The results indicate that both Xcog and CMPGRD need of the order $\mathcal{O}(\sum_k N_k M_k)$ operations to compute the overlapping grid. Also note that Xcog was more than six times faster than CMPGRD in the present example.

6 Summary

A highly automated general purpose algorithm for constructing overlapping grids has been described. The basic feature of the algorithm is its ablility to directly classify most grid points in the overlapping grid; it only needs to iterate on the classification of interpolation points where three or more component grids overlap each other. In addition to making the algorithm efficient, the perhaps most important advantage of the direct classification manifests itself when there are too few grid points in the component grids to form valid interpolation relations. Because the present method only iterates on the classification of interpolation points, the main parts of the grid will be valid even after the algorithm fails. By checking the overlapping grid after the main classification is completed, the algorithm marks any inconsistent grid points and reports them graphically to the user, who then rather easily can determine why the algorithm failed and modify the component grids to improve the situation.

In order to determine if a grid point can be an interpolation point, the overlapping grid algorithm must calculate if the grid point is inside or outside of the donor grid. For this purpose, the present method uses a fast and memory efficient implementation of the ray-method, which counts the number of intersections between the boundary of the donor grid and a ray that starts at the grid point and ends at infinity.

The overlapping grid algorithm has been implemented in an interactive code called Xcog. It has been exemplified that Xcog requires of the order of the total number of grid points to compute the overlapping grid and that it is significantly faster than the overlapping grid code CMPGRD.

7 Acknowledgments

I would like to thank the people who patiently tested Xcog during its development. They made valuable suggestions for improvements and located bugs and weaknesses in previous versions of the code. These people include: Per Hammarlund, Johan Malmliden, Lotta, Fredrik and Pelle Olsson, Olof Runborg, Björn Sjögreen, Jon Tegnèr and Jacob Yström. I would also like to thank David Brown, Geoff Chesshire, Bill Henshaw and Heinz Kreiss for initiating my interest in overlapping grids and sharing their thorough knowledge on the subject.

References

[1] J. F. Thompson, Z. U. A. Warsi, and C. W. Mastin. *Numerical Grid Generation.* North-Holland, New York, 1985.

[2] G. Chesshire and W. D. Henshaw. Conservation on composite overlapping grids. *SIAM J. Sci. Comput.*, 15:819–845, 1994.

[3] J. A. Benek, J. L. Steger, and F. C. Dougherty. A flexible grid embedding technique with application to the Euler equations. *AIAA*, 83-1944, 1983.

[4] J. A. Benek, P. G. Buning, and J. L. Steger. A 3-D Chimera grid embedding technique. *AIAA*, 85–1523, 1985.

[5] J. L. Steger and J. A. Benek. On the use of composite grid schemes in computational aerodynamics. *Computer Methods in Applied Mechanics and Engineering*, 64:301–320, 1987.

[6] G. Chesshire and W. D. Henshaw. Composite overlapping meshes for the solution of partial differential equations. *J. Comput. Phys.*, 90(1):1–64, 1990.

[7] R. L. Meakin. A new method for establishing intergrid communication among systems of overset grids. *AIAA*, 91–1586–CP, 1991.

[8] D. L. Brown, G. Chesshire, and W. D. Henshaw. Getting started with CMPGRD. Introductory user's guide and reference manual. LA–UR 90-3729, Los Alamos National Laboratory, 1989.

[9] M. S. Milgram. Does a point lie inside a polygon? *J. Comput. Phys.*, 84:134–144, 1989.

[10] S. J. Parks, P. G. Bunig, J. L. Steger, and W. M. Chan. Collar grids for intersecting geometric components within the Chimera overlapped grid scheme. *AIAA*, 91–1587–CP, 1991.