Chapter 2

Hierarchical Grid System

The purpose of a grid in Computational Fluid Dynamics (CFD) is straightforward, it provides a skeleton upon which some discretized version of the equations of fluid motion may be solved numerically. The ideal grid would be one that is both simple to implement and computationally cheap to generate, places no unreasonable constraints on the method of flow solution, is able to match the local mesh spacing to the local physical length scale of the flow, and is able to describe complicated geometries. Unfortunately, it is not possible to realize this ideal grid. All practical grid systems are the result of a series of compromises. Consequently, there are almost as many different types of grid as there are methods to solve the flow equations. In this chapter we give a description for the computational grid system used by the AMR algorithm. This grid system has been developed with the aim of computing very high resolution solutions to unsteady shock hydrodynamic problems. For clarity, here we do not consider the interactions that have shaped the development of this grid system. However, these interactions will become clear in the light of subsequent chapters.

The AMR algorithm uses a hierarchical structure of embedded meshes to discretize the flow domain. We start this chapter with an overview of the hierarchical grid system. This is followed by a formal definition together with a specification of the information that must be supplied in order to fully describe a given set of meshes. To implement the embedded mesh system efficiently requires certain programming techniques, borrowed from computer science, which are unfamiliar to many CFD practitioners; these are explained at some length. Now, before we can solve the equations of fluid motion on our hierarchical grid we must know the mesh connectivity, that is which meshes need to communicate directly with one another. Algorithms are presented for extracting this information from the data describing the grid structure.

2.1 A Descriptive Overview

The AMR algorithm uses a hierarchical structure of embedded meshes to discretize the flow domain. The bottom of the hierarchy, level 0, consists of a set of coarse mesh patches which define the extent of the flow domain. Each of these mesh patches forms a logically rectangular unit ruled by I,J co-ordinate lines. These patches are restricted such that it is possible to reference all their cells by a single (i, j) co-ordinate system, see figure 2.1. This restriction ensures that there is continuity of grid lines between neighbouring patches and that if two patches overlap one another then the regions of overlap are identical. These mesh patches form the effective grid at level 0. Usually the terms mesh and grid are synonymous, but, throughout this work we reserve the term mesh for a single logically rectangular patch and the term grid for a collection of such patches.

The flow domain may be refined locally by embedding finer mesh patches into the coarse grid at level 0 to form the next grid level within the hierarchy, namely level 1. These embedded patches are formed by linearly sub-dividing rectangular groups of coarse cells as shown in figure 2.2. The choice for the number of sub-divisions along the edges of a coarse cell is arbitrary, but must be the same for every coarse cell that is refined. This restriction enables every mesh cell contained at level 1 to be referenced by its own (i, j) co-ordinate system. In their turn these embedded patches at grid level 1 may contain even finer embedded patches which form the grid at level 2. This process of refinement may be repeated indefinitely. The grids at different levels within the hierarchy co-exist, for underlying an embedded fine grid there is a complete coarse grid and a complete coarse field solution, see figure 2.3.

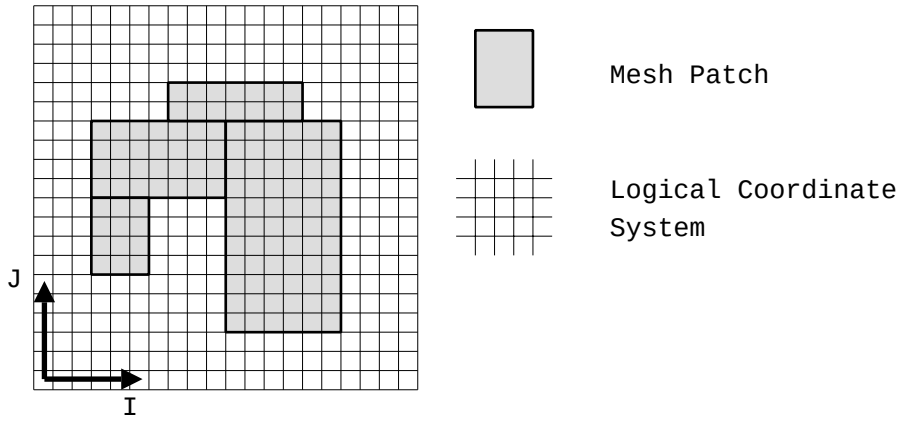

Figure 2.1: All meshes are fixed relative to a logical co-ordinate system.

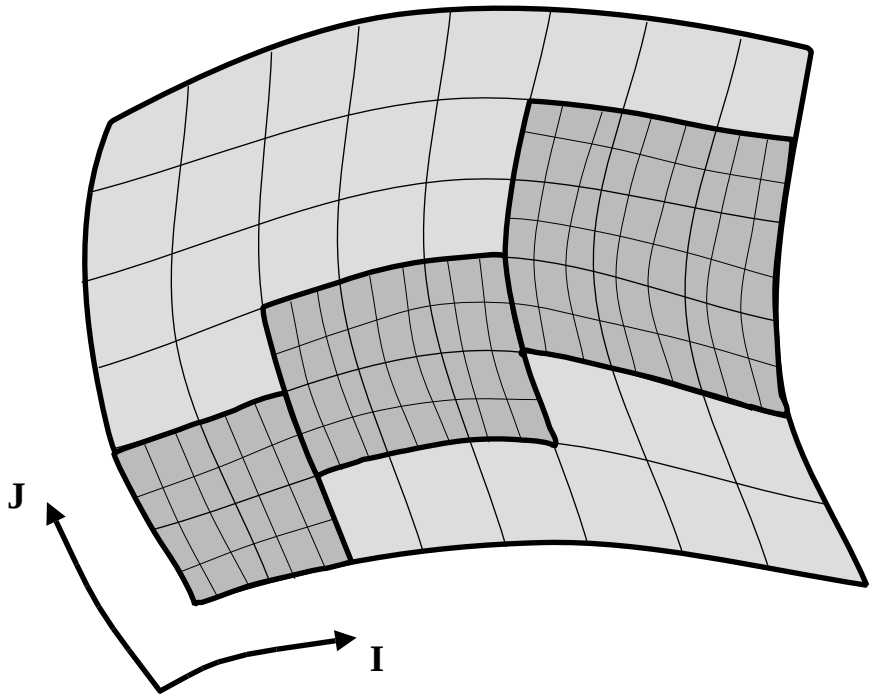

Figure 2.2: Fine meshes are generated by subdividing coarse meshes.

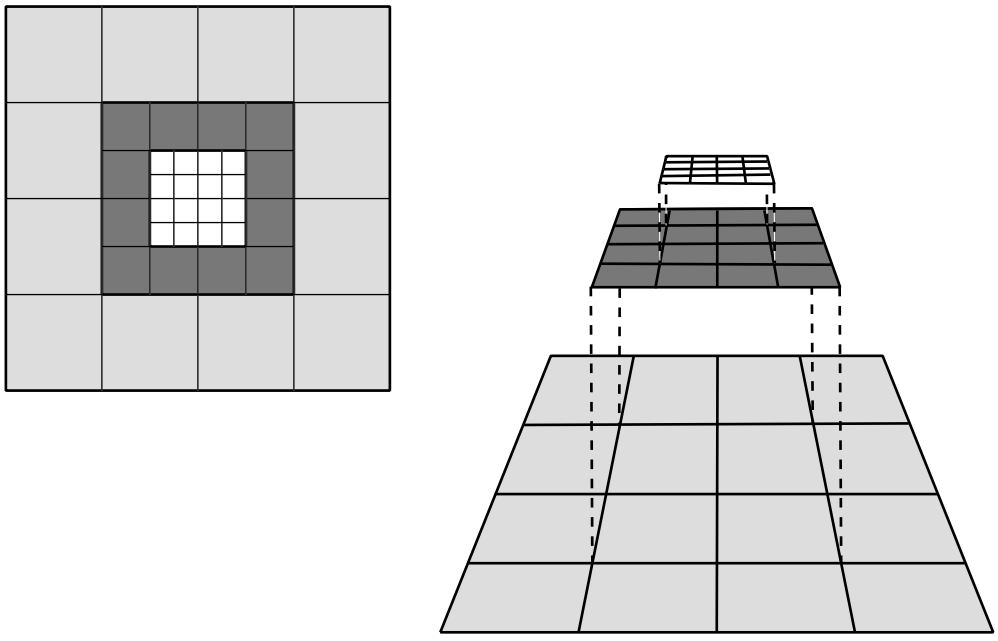

Figure 2.3: Below each embedded fine patch lies part of a coarse grid.

2.2 A Formal Definition

Following on from this descriptive overview, we now give a formal definition of the hierarchical grid system used by the AMR algorithm. The nomenclature developed in this section will be used extensively throughout the remainder of this thesis in an attempt to explain the intricacies of our scheme in an accurate yet concise manner.

An arbitrary grid structure, G, is formed from a set of $l_{max}+1$ grids, one grid for each level within the hierarchy. We label these grid levels, $0, 1, 2, \ldots$, and denote the grid at level l by G_l. Thus,

$$G = \{G_l : l \in \mathcal{N}, 0 \leq l \leq l_{max}\}. \tag{2.1}$$

Each grid is formed from the union of one or more mesh patches. If there are nG_l mesh patches at level l and we denote the k^{th} patch by $G_{l,k}$ then,

$$G_l = \bigcup_{k=1}^{nG_l} G_{l,k}. \tag{2.2}$$

A mesh $G_{l,k}$ can be identified by a unique grid index, $Gp_l + k$, where $Gp_0 = 0$ and,

$$Gp_{l+1} = Gp_l + nG_l. \tag{2.3}$$

A co-ordinate system $\mathbf{C}_l^2$ is used to index the mesh cells contained by the grid G_l,

$$\mathbf{C}_l^2 = \{(i_l, j_l) : i_l \in \mathcal{N}, j_l \in \mathcal{N}\}. \tag{2.4}$$

Using this co-ordinate system, a mesh patch $G_{l,k}$ is defined in terms of its rectangular extent, $\square_{l,k}$, see figure 2.4. We write an extent in the form $< IW, JS, IE, JN >$,

$$\square_{l,k} = \quad < IW_{l,k}, JS_{l,k}, IE_{l,k}, JN_{l,k} > . \tag{2.5}$$

The cells contained by a mesh $G_{l,k}$ may be referenced relative to the mesh using a co-ordinate system $\mathbf{M}_{l,k}^2$,

$$\mathbf{M}_{l,k}^2 = \{(i, j) : i \in \mathcal{N}, j \in \mathcal{N}, 1 \leq i \leq IM_{l,k}, 1 \leq j \leq JM_{l,k}\}. \tag{2.6}$$

Where,

$$\begin{aligned} IM_{l,k} &= IE_{l,k} - IW_{l,k} + 1 \\ JM_{l,k} &= JN_{l,k} - JS_{l,k} + 1. \end{aligned} \tag{2.7}$$

We denote the cell (i, j) contained by the mesh $G_{l,k}$ by $G_{l,k:i,j}$, while $G_{l,k:N;i}$ denotes the i^{th} interface along the northern boundary of this mesh. A similar notation is used to identify interfaces along the other three edges of the mesh.

The geometry of the grid G_0 is problem dependent. For all other grids, the grid at level l inherits its geometry from the grid at level $l-1$. A coarse cell at level $l-1$ is simply

linearly sub-divided to form rI_l by rJ_l finer cells at level l. The spatial refinement factors rI_l and rJ_l determine the number of sub-divisions made along the I and J co-ordinate lines respectively. Note that these factors are integers, typically they take values between 2 and 10. So, a simple relationship exists between the co-ordinate systems $\mathbf{C}_l^2$ and $\mathbf{C}_{l-1}^2$; a cell (i_l, j_l) at level l would be formed from the sub-division of a coarse cell at level $l-1$ whose co-ordinates (i_{l-1}, j_{l-1}) are given by,

$$\begin{aligned} i_{l-1} &= \frac{(i_l - 1)}{rI_l} + 1 \\ j_{l-1} &= \frac{(j_l - 1)}{rJ_l} + 1. \end{aligned} \tag{2.8}$$

All meshes must be *properly nested* that is, a mesh at level l is wholly contained within one or more coarse meshes at level $l-1$. Thus,

$$G_l \subseteq G_{l-1}. \tag{2.9}$$

The meshes shown in figure 2.5 satisfy this definition of *proper nesting*, while those in figure 2.6 do not. Note that this definition of *proper nesting* is less restrictive than that of Berger[7]. In order to check that the meshes at level $l+1$ are *properly nested* we require the extents for the meshes at level l using the co-ordinate system $\mathbf{C}_{l+1}^2$, for the mesh $G_{l,k}$ we denote such an extent by $\square_{l,k}^c$.

The AMR algorithm is a *cell-centred* scheme, the solution vector associated with each grid cell being an average value of the flow solution over the area of the cell. We denote the solution vector for the mesh cell $G_{l,k:i,j}$ by $\mathbf{W}_{l,k:i,j}$. While the field solutions for the grid structure G, the grid G_l and the mesh patch $G_{l,k}$ are denoted by $\mathbf{W}$, $\mathbf{W}_l$ and $\mathbf{W}_{l,k}$ respectively.

2.2.1 Grid Specification

In order to specify an arbitrary grid structure, G, the following information must be supplied: for each mesh, its grid level l and extent using $\mathbf{C}_l^2$ co-ordinates; for each grid level, the spatial refinement factors rI and rJ; the geometry for the grid G_0. All the other details for the grid structure may be gleaned from this basic data.

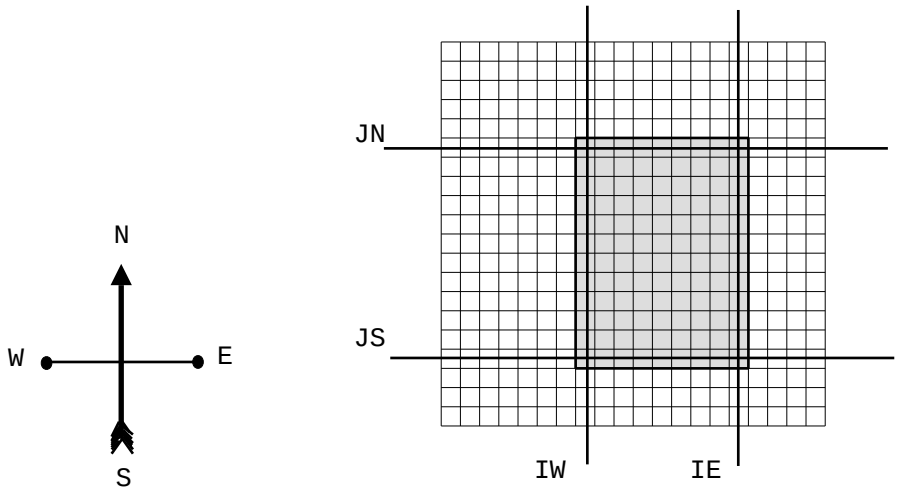

Figure 2.4: A mesh extent.

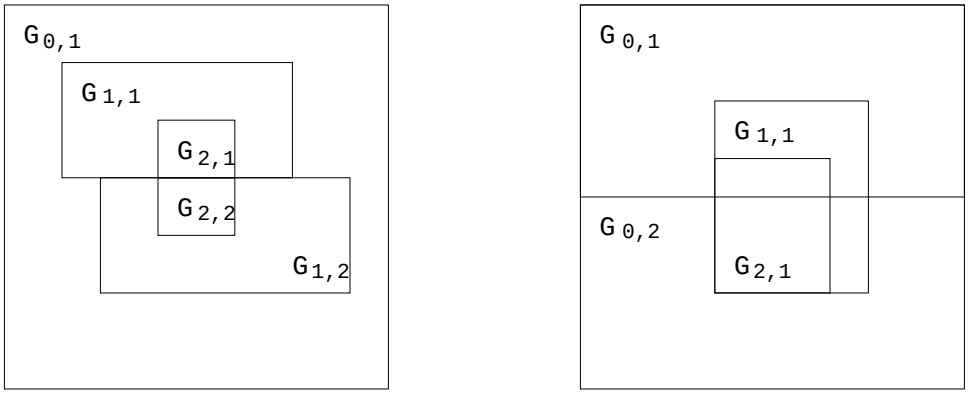

Figure 2.5: Properly nested meshes.

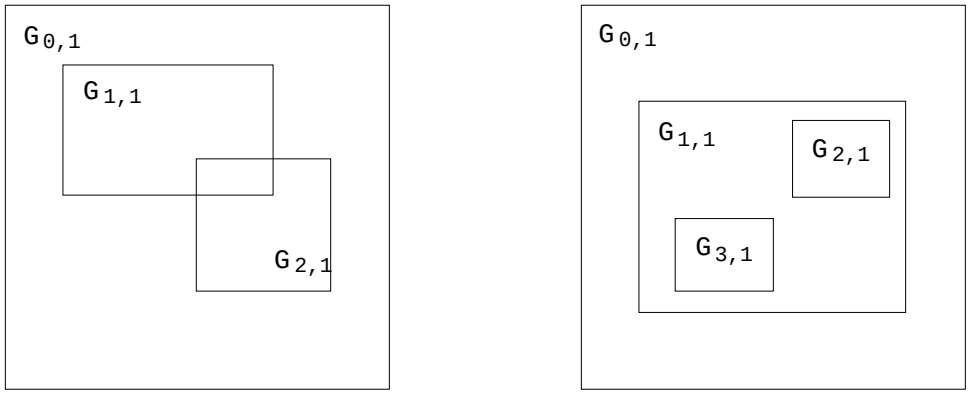

Figure 2.6: Badly nested meshes.

2.3 Mesh Connectivity

Any algorithm for integrating the flow solution, $\mathbf{W}$, contained by the hierarchical grid structure, G, must have some knowledge of the mesh connectivity. For example, if a fine mesh $G_{l,a}$ lies wholly within a single coarse mesh $G_{l-1,b}$ then in effect the coarse mesh must provide boundary conditions for the fine mesh. So, it would be necessary to know for each fine cell along the boundary of $G_{l,a}$ the coarse cell from $G_{l-1,b}$ that it abuts.

A cell interface that forms part of the boundary for a mesh $G_{l,a}$ may be categorized as one of three types: *external,* if it borders the edge of the computational domain; *fine-fine,* if it borders a cell from a mesh, $G_{l,b}$, which belongs to the same grid; *fine-coarse,* if it borders a cell from the underlying coarse grid G_{l-1}. These different types of boundary are shown in figure 2.7. Strictly, a single interface could be tagged as more than one type. However, *external* boundaries determine the problem to be solved. Consequently, an *external* tag should take precedence over both *fine-fine* and *fine-coarse* tags. While from accuracy considerations, see chapter 3, a *fine-fine* tag should take precedence over a *fine-coarse* tag. Note, only *fine-fine* and *fine-coarse* boundaries require knowledge of the mesh connectivity.

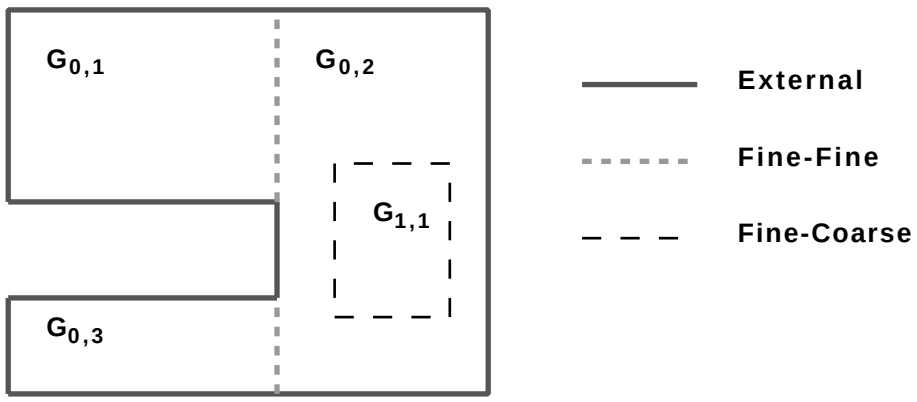

Figure 2.7: Different types of mesh boundary.

One of the strategies underpinning the AMR algorithm is that the basic mesh integrator should not need to know about the mesh connectivity; this burden is offloaded onto a separate process. The same numerical scheme that is used to compute the fluxes across cell interfaces which are internal to a mesh is also used to compute fluxes across cell interfaces which form mesh boundaries. Adopting this approach, to integrate a mesh IM cells by JM cells using an arbitrary second order *cell-centred* scheme requires data on a mesh $(IM + 4)$ by $(JM + 4)$ cells, see figure 2.8. We shall call those cells that lie outside the basic mesh of IM by JM cells dummy cells. Prior to the integration of a mesh the dummy cells for that mesh are primed with data. This data is chosen such that

the resultant numerical fluxes across the mesh boundary are consistent with the various boundary conditions that have to be met. Here we are concerned only with finding the connectivity information that allows this priming process to take place, details for the priming process are given in §3.2.

The problem of finding the mesh connectivity for the mesh $G_{l,k}$ has been reduced to finding those cells, either coarse ones from the grid G_{l-1} or fine ones from other meshes within the grid G_l, that overlap/underlay its dummy cells. Because meshes at the same grid level are allowed to overlap one another, it is conceivable that several cells simultaneously overlap/underlay a single dummy cell. But, the AMR algorithm guarantees that any overlapping mesh cells which belong to the same grid are identical. Moreover from accuracy considerations, an overlapping fine cell should take precedence over an underlying coarse cell. Consequently, a given dummy cell need at most have knowledge of either a single underlying coarse cell or a single overlapping fine cell.

Given a grid structure, G, the AMR algorithm gathers a set of information for every cell interface that forms part of the boundary of a mesh. If an interface lies along the edge of the computational domain then this information describes the type of external boundary procedure to be applied, otherwise it provides mesh connectivity information. In essence, two steps are required to gather the boundary information for a mesh $G_{l,a}$.

1. Compare $G_{l,a}$ with each mesh at level $l-1$, in turn. For each of those boundary interfaces that have not already been tagged as *external*, consider the dummy cell which abuts the interface. If this dummy cell overlaps a coarse cell $G_{l-1,b:i_c,j_c}$ then tag the interface as a *fine-coarse* boundary and save the connectivity information $Gp_{l-1}+b$ and (i_c, j_c). However, if this coarse cell is a dummy cell which borders the edge of the computational domain then use its boundary information to overwrite the *fine-coarse* information just saved.

2. Compare $G_{l,a}$ with each mesh $G_{l,b}$ at level l, $a \neq b$, in turn. For each of those boundary interfaces that have not already been tagged as *external*, consider the dummy cell which abuts the interface. If this dummy cell overlaps a cell $G_{l,b:i_f,j_f}$ then tag the interface as a *fine-fine* boundary and save the connectivity information Gp_l+b and (i_f, j_f). Note, if this cell is also a dummy cell then to prevent the cyclic interchange of data between the two dummy cells boundary information should only be gathered if $a > b$.

Each time a set of boundary information for a particular interface is stored it overwrites any previous information that may have been saved. Therefore multiple entries reduce to single entries and fine mesh information automatically takes precedence over coarse mesh information. If the above procedure fails to find a set of boundary information for any one interface of the mesh $G_{l,a}$ then this mesh cannot be *properly nested*. Algorithms for extracting the *fine-coarse* and the *fine-fine* mesh connectivity information from the grid

structure are shown in figures 2.9 and 2.10 respectively. Note, these algorithms extract the connectivity information for the inner border of dummy cells only, but if this information is known then that for the outer border follows trivially.

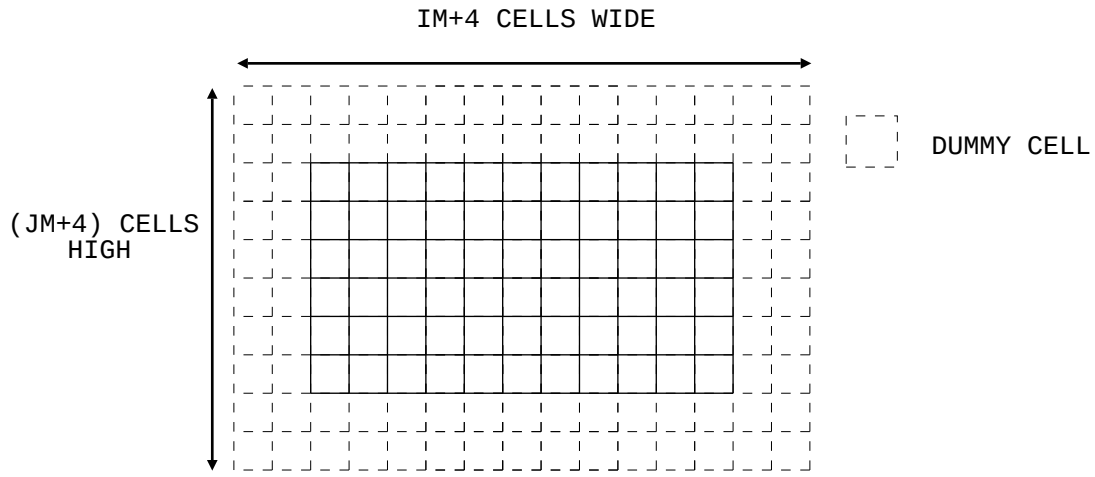

Figure 2.8: Dummy cells surround each mesh.

To Find *fine-coarse* Connectivity Information

1. For a dummy cell from the mesh $G_{l,a}$ to overlap either a mesh cell or a dummy cell from $G_{l-1,b}$ then the extents for the two meshes must overlap, that is

$$\square^c_{l-1,b} \cap \square_{l,a} \neq \emptyset.$$

The extent of this overlapping region $< IW, JS, IE, JN >$ is given by,

$$\begin{aligned} JN &= min(JN^c_{l-1,b}, JN_{l,a}) \\ JS &= max(JS^c_{l-1,b}, JS_{l,a}) \\ IE &= min(IE^c_{l-1,b}, IE_{l,a}) \\ IW &= max(IW^c_{l-1,b}, IW_{l,a}). \end{aligned} \tag{2.10}$$

2. If $JN = JN_{l,a}$ then the strip of dummy cells starting at $(IW, JN+1)$ and finishing at $(IE, JN+1)$ overlap coarse cells from $G_{l-1,b}$. A cell $(i, JN+1)$ along this strip overlays a coarse cell whose co-ordinates relative to the coarse mesh are,

$$\begin{aligned} i_c &= ((i-1)/rI_l + 1) - IW_{l-1,b} + 1 \\ j_c &= (JN/rJl + 1) - JS_{l-1,b} + 1. \end{aligned} \tag{2.11}$$

3. If $JS = JS_{l,a}$ then the strip of dummy cells starting at $(IW, JS-1)$ and finishing at $(IE, JS-1)$ overlap coarse cells from $G_{l-1,b}$. A cell $(i, JS-1)$ along this strip overlays a coarse cell whose co-ordinates relative to the coarse mesh are,

$$\begin{aligned} i_c &= ((i-1)/rI_l + 1) - IW_{l-1,b} + 1 \\ j_c &= ((JS-2)/rJl + 1) - JS_{l-1,b} + 1. \end{aligned} \tag{2.12}$$

4. The eastern and western edges may be dealt with similarly.

Figure 2.9: Algorithm to find *fine-coarse* connectivity information.

To Find *fine-fine* Connectivity Information

1. For a dummy cell from the mesh $G_{l,a}$ to overlap either a mesh cell or a dummy cell from $G_{l,b}$, $a \neq b$, then the extents for these two meshes, expanded to include the inner border of dummy cells, must overlap. If these modified extents are denoted by $\square'$ then,

$$\square'_{l,a} \cap \square'_{l,b} \neq \emptyset.$$

   The extent of this overlapping region $< IW, JS, IE, JN >$ is given by,

$$\begin{aligned} JN &= min(JN_{l,a}, JN_{l,b}) + 1 \\ JS &= max(JS_{l,a}, JS_{l,b}) - 1 \\ IE &= min(IE_{l,a}, IE_{l,b}) + 1 \\ IW &= max(IW_{l,a}, IW_{l,b}) - 1. \end{aligned} \tag{2.13}$$

2. If $JN = JN_{l,a} + 1$ then the strip of dummy cells starting at (IW, JN) and finishing at (IE, JN) overlap cells from $G_{l,b}$. If $JN = JN_{l,b} + 1$ then these cells from $G_{l,b}$ are also dummy cells, in which case the coarse mesh connectivity information should only be over written if $a > b$. A cell (i, JN) along this strip overlays a cell from $G_{l,b}$ whose relative co-ordinates are,

$$\begin{aligned} i_f &= i - IW_{l,b} + 1 \\ j_f &= JN - JS_{l,b} + 1. \end{aligned} \tag{2.14}$$

3. If $JS = JS_{l,a} - 1$ then the strip of dummy cells starting at (IW, JS) and finishing at (IE, JS) overlap cells from $G_{l,b}$. If $JS = JS_{l,j} - 1$ then these cells from $G_{l,b}$ are also dummy cells, in which case the coarse mesh connectivity information should only be over written if $a > b$. A cell (i, JS) along this strip overlays a cell from $G_{l,b}$ whose relative co-ordinates are,

$$\begin{aligned} i_f &= i - IW_{l,b} + 1 \\ j_f &= JS - JS_{l,b} + 1. \end{aligned} \tag{2.15}$$

4. The eastern and western edges may be dealt with similarly.

Figure 2.10: Algorithm to find *fine-fine* connectivity information.

2.4 Implementation Details

The implementation of the hierarchical grid system is done in two parts. The first part follows directly from the formal definition of the grid structure and is simply a bald statement of the program variables that are required to specify an arbitrary set of grids. the second part is more involved and is critical to the practicability of the AMR algorithm. It deals with the problem of how the data required for each mesh should be stored by a computer program. The solution although standard practice in computer science will be unfamiliar to many readers and so we give full details.

2.4.1 Grid description

The program variables that are required in order to specify an arbitrary set of grids that conform to our hierarchical grid structure follows directly from the formal definition of the grid structure. A full list of these variables is given in figure 2.11. From this list, it can be seen that the storage overhead associated with each mesh is quite small. This overhead could be reduced, albeit at some loss in programming convenience, for the variables IMX, JMX, JNc, JSc, IEc and IWc can all be calculated from JNf, JSf, IEf, IWf, rI and rJ. For example, IMX(GRD) = IEf(GRD)-IWf(GRD)+1. A pair of nested DO loops is all that is required to run through the grid structure and operate on every mesh, see figure 2.12.

2.4.2 Linked List Data Storage

In most CFD applications the computational domain is defined by a single structured mesh. In such cases it is both convenient and efficient to store geometrical and flow variables in multi-dimensional arrays. This would enable, for example, the k^{th} element of the solution vector for the ij^{th} cell to be accessed simply by the array element W(K,I,J). Now the AMR algorithm requires data to be stored for more than one structured mesh. So it might, at first sight, appear sensible to generalise the above example and store this vector element for the n^{th} mesh in the array element W(N,K,I,J). However, if the meshes are of disproportionate sizes, as is inevitable, then this would be extremely wasteful of memory. Such extravagant use of memory would severely limit the practicability of the AMR algorithm.

The solution to the above problem is to implement a data storage management system. The system needs to be flexible, invisible to the flow calculation, and should not introduce large overheads that might impair program efficiency. Such a system can be formed from a set of linked lists[34]. Each list consists of a continuous set of storage elements that may be viewed as a one-dimensional array.

All flow variables and geometrical data are stored in a series of lists called heaps; one

LMAX	l_{max}, maximum grid level.
NGA(L)	nGl, number of grids contained by G_l.
GP(L)	Gp_l, grid index pointer.
rI(L)	rI_l, Spatial refinement factor in I direction.
rJ(L)	rJ_l, Spatial refinement factor in J direction.
GRD = GP(L)+K	
LGRID(GRD)	Grid level for the mesh $G_{l,k}$.
IMX(GRD)	$IM_{l,k}$, Width of mesh $G_{l,k}$.
JMX(GRD)	$JM_{l,k}$, Height of mesh $G_{l,k}$.
JNf(GRD)	$\square_{l,k}$, extent of mesh $G_{l,k}$ using $\mathbf{C}_l^2$ co-ordinates.
JSf(GRD)	
IEf(GRD)	
IWf(GRD)	
JNc(GRD)	$\square_{l,k}^c$, extent of mesh $G_{l,k}$ using $\mathbf{C}_{l+1}^2$ co-ordinates.
JSc(GRD)	
IEc(GRD)	
IWc(GRD)	

Figure 2.11: List of variables stored to describe the grid structure.

```
DO L=0,LMAX
    DO K=1,NGA(L)
        GRD = GP(L)+K
        ...
        Operate on mesh G_{l,k}
        ...
    END DO
END DO
```

Figure 2.12: FORTRAN code fragment to visit every mesh of the grid structure.

heap for each variable. Each heap contains a contiguous set of blocks; one block for each mesh. While each block consists of a continuous set of storage elements; one element for each piece of data stored for the mesh. Another list provides an index into these data heaps pointing to the start of each mesh block. The AMR algorithm distinguishes between two different types of data heap. The first type is used for storing data that is associated with mesh boundaries, while the second type is used for data associated with mesh cells. The linked list data structure for *type1* data is shown in figure 2.13, that for *type2* data is identical in form. Note, for simplicity, four *type1* lists are used for each quantity stored along mesh boundaries, one list for northern edges, one list for southern edges *etc*.

The two different types of data heap require a different mechanism for indexing into individual mesh data blocks. Figure 2.14 shows how the data is ordered for *type1* mesh data blocks. Note, for reasons which will be given later, along the northern and southern edges of a mesh $G_{l,k}$ it is sufficient to store just one piece of data for a group of rI_l cells. Similarly, along the eastern and western edges of a mesh it is sufficient to store just one piece of data for a group of rJ_l cells. So, the index into a *type1* heap for the storage elements associated with the interface $G_{l,k:E;j}$ is, assuming the variable GRD has the value GP(L)+K, simply given by

$$\mathrm{H1PTR(GRD)} + (\mathrm{J} - 1)/\mathrm{rJ(L)},$$

where the array element H1PTR(GRD) contains a pointer to the start of the *type1* data block for the mesh with grid index GRD. This calculation can be hidden away and returned via a function call

$$\mathrm{INDEXgk(GRD, J, rJ(L))}.$$

Figure 2.15 shows how the data is ordered for *type*2 mesh data blocks. Note, it is convenient to store a set of dummy or halo cells for each mesh. These dummy cells form a border, two cells deep, around each mesh. Hence a row of data stored for the grid GRD is actually IMX(GRD)+4 cells wide and not IMX(GRD) as one might expect. The function of these dummy cells will be described in the chapter dealing with boundary conditions. The index into a *type*2 heap for the mesh cell $G_{l,k:i,j}$ is given by,

$$\mathrm{H2PTR(GRD)} + (\mathrm{I} + 2) + (\mathrm{J} + 1) * (\mathrm{IMX(GRD)} + 4),$$

where the array element H2PTR(GRD) contains a pointer to the start of the *type2* data block for the mesh with grid index GRD. This calculation can also be hidden away and returned via a function call,

$$\mathrm{INDEXgij(GRD, I, J)}.$$

Our method of storing data may appear to be less efficient than using static arrays because it involves expressions to evaluate the offset indices. However, it should be remembered that similar expressions are generated by a compiler anyway[10]. Moreover, in many situations such an expression need only be evaluated once for each mesh. The code shown in figure 2.16 visits every cell of every mesh in the data structure. The indexing

overhead is reduced to one subroutine call per mesh in the grid structure. The subroutine UNPACKGRID sets the pointer IJ to,

$$\text{H2PTR(GRD)} + ([1] + 2) + ([1] + 1) * (\text{IMX(GRD)} + 4),$$

this is equivalent to INDEXgij(GRD,1,1). The variable Ibmp is the increment that must be added to the pointer to move one cell eastwards, so Ibmp is set to 1. While, the variable Jbmp is the increment that must be added to the pointer to move one cell northwards, so Jbmp is set to IMX(GRD)+4.

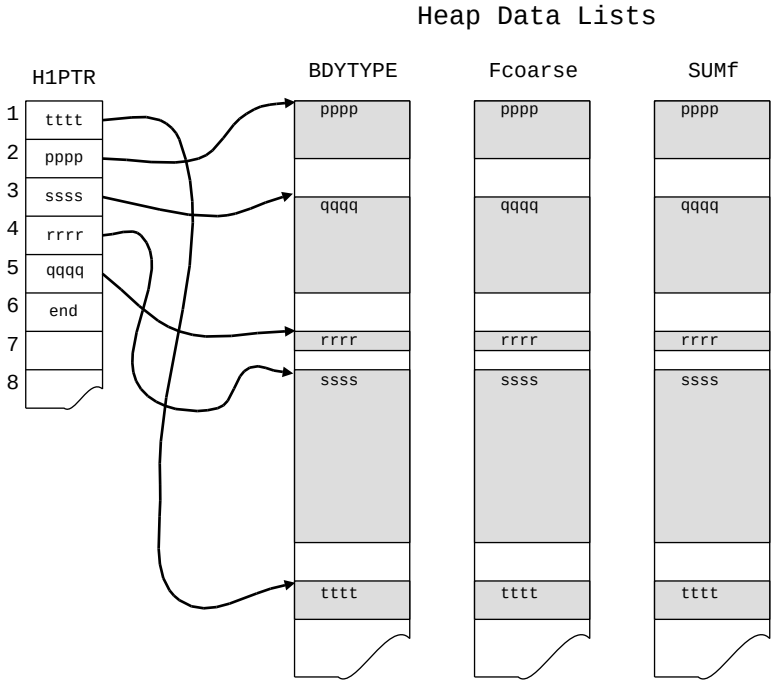

Figure 2.13: Linked list data structure for *type*1 data.

The computational grid structure used by the AMR algorithm is dynamic, during a calculation the grid adapts to the evolution of the flow solution. Consequently, the storage heaps are also dynamic. However, the manner in which the grid structure adapts obviates the need to incorporate *garbage collection*[34] routines into our heap management system. At every stage during the calculation the heap data block are stored nose to tail. Thus, simple recurrence formulae may be used to generate the heap data block pointers. For example, the set of pointers for a *type2* heap may be generated using the recurrence formula,

$$\begin{aligned} \text{H2PTR}(1) &= 0 \qquad (2.16) \\ \text{H2PTR}(\text{GRD}+1) &= \text{H2PTR(GRD)} + (\text{IMX(GRD)} + 4) * (\text{IMX(GRD)} + 4). \end{aligned}$$

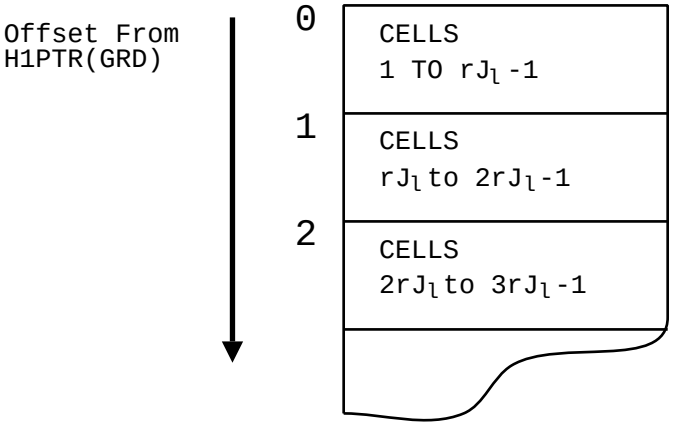

Figure 2.14: Order of data for *type*1 mesh data blocks.

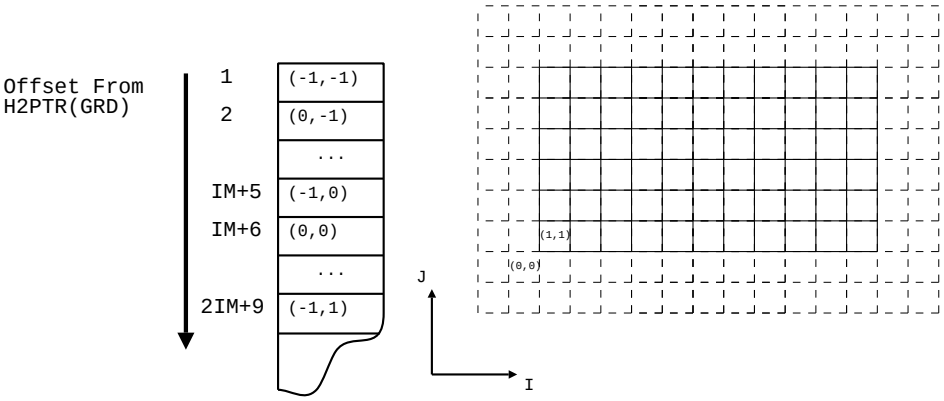

Figure 2.15: Order of data for *type*2 mesh data blocks.

```
DO L=0,LMAX
    DO K=1,NGA(L)
        GRD = GP(L)+K
        CALL UNPACKGRID(GRD,1,1,IJ,Ibmp,Jbmp)
        DO I=1,IMX(GRD)
            IJo = IJ
            DO J=1,JMX(GRD)

                IJ contains a pointer to the data stored
                for the cell G_{l,k:i,j}.

                IJ = IJ +Jbmp
            END DO
            IJ = IJo+Ibmp
        END DO
    END DO
END DO
```

Figure 2.16: FORTRAN code fragment to visit every mesh cell of the grid structure.

2.4.3 Mesh Boundary Information

Because of the manner in which a mesh $G_{l,a}$ is embedded in a grid G_{l-1}, it follows that all the cell interfaces contained by the set,

$$N_k = \{G_{l,a:N;i} : i \in \mathcal{N}, k \in \mathcal{N}, (k-1)rI_l + 1 \leq i \leq k(rI_l)\}, \tag{2.17}$$

will be tagged as having the same type of boundary. Furthermore, given the connectivity information for just one interface from the set N_k, it is trivial to deduce the connectivity information for the other $rI_l - 1$ interfaces. For example, if the interface $G_{l,a:N;1}$ abuts a cell $G_{l,b:i_f,j_f}$ then for $0 < k < rI_l$ an interface $G_{l,a:N;1+k}$ must abut a cell $G_{l,b:i_f+k,j_f}$. Consequently, we require just $IM_{l,a}/rI_l$ sets of data in order to provide the complete set of boundary information for the northern edge of the mesh. A similar state of affairs exists for the other three sides of the mesh.

For our implementation of the AMR algorithm we encode each set of mesh boundary information as a single 32 bit word. Thus, we can store the boundary information for the entire grid structure using just four *type1* data heaps. For example, the array element

$$\text{BDYTYPE}(\text{NORTH}, \text{INDEXgk}(\text{Gp}(\text{L}) + \text{A}, 1, \text{rI}(\text{L})))$$

would contain the encoded information for the interface $G_{l,a:N,1}$. To store a complete set of boundary information in this fashion is not an extravagant use of memory. To date, even

for our most ambitious calculations a heap of just 1000 words has proved ample. Moreover, we would strongly recommend this approach to any person who wished to implement the AMR algorithm. For during the early development of our code, whenever it crashed we could perform an autopsy on these mesh boundary storage heaps and thereby eliminate coding errors. Today, whilst we cannot claim that our code is completely free of bugs we are able to demonstrate that it is extremely robust, see §4.4.2.

2.5 Closing Comment

At this juncture our hierarchical grid system may appear flawed. For example, it cannot boast the geometric packing capabilities of an unstructured mesh. While compared with a structured grid system which employs cellular rather than embedded mesh refinement, it requires more mesh cells in order to refine features which are oblique to the mesh. But a scheme such as the AMR algorithm has to perform many tasks, some of which have conflicting demands. The hierarchical grid system enables the different parts of our algorithm to be combined in a harmonious fashion. So, it should not be judged in isolation. Indeed, in the light of the material given in chapters 3 and 4 it will become clear that our hierarchical grid system is actually an extremely effective means of producing high resolution solutions to shock hydrodynamic problems.

Finally, it is worth commenting on the relative ease with which our hierarchical grid system may be implemented. Now, in order to implement any adaptive mesh scheme careful consideration has to be given as to how data should be stored so that program efficiency is not impaired. For example, the overheads for accessing the data associated with a mesh cell or for finding which cells adjoin a specified cell must be kept small. For some schemes this necessitates sophisticated data structures such as quad- and oct-trees, complicated search algorithms, and *garbage collection* routines. The routines that maintain these sophisticated data structures can account for a sizeable part of the finished program. In contrast, an implementation of the AMR algorithm, at least for serial machines, can make do with a simple but nonetheless effective data storage mechanism. This simplicity stems from the structured nature of the grid system. Firstly, search operations for establishing the mesh connectivity work with mesh blocks rather than with individual mesh cells, and since there are significantly fewer mesh blocks than mesh cells such operations are inherently cheap. Secondly, the overheads associated with accessing cell data are reduced to a one off overhead for each mesh block, hence the effective overhead for a single cell is negligible. Before ending this chapter, it should be noted that our data storage mechanism would require a radical rethink if the AMR algorithm were to be ported onto a parallel computer which used distributed rather than shared memory.