SIAM J. Sci. Comput.
Vol. 21, No. 2, pp. 646–665

HOLE-CUTTING FOR THREE-DIMENSIONAL OVERLAPPING GRIDS [*]

N. ANDERS PETERSSON[†]

Abstract. A hole-cutting technique for assembling overlapping grid systems for solving partial differential equations is presented. In the present method, the closed surfaces on the boundary of the three-dimensional computational domain are represented by hybrid surface grids. The hybrid grids are composed of nonoverlapping structured surface grids joined by an unstructured layer of triangles. Each hybrid grid is constructed from a subset of the physical boundary points on the faces of the component grids.

The holes in the component grids are made by using a two-step mark-and-fill method. We first locate all grid cells that intersect the hybrid surface grids. We then apply the ray method to determine whether the grid points in these intersected grid cells are inside or outside the region bounded by the hybrid surface grids. As a result, the boundary of the hole region will be marked and it is a simple matter to then remove all points from the hole by starting from the outside points and traversing along all three grid directions until an inside point is found. To locate the intersecting grid cells, we employ the ray method augmented by an octree-based search technique together with Newton's method to invert the mappings corresponding to each component grid.

We demonstrate the hole-cutting method by running the code Chalmesh to generate overlapping grids for a sphere in a box, around the stern of a ship, and around a three-bladed ship propeller.

Key words. overlapping grids, overset grids, Chimera grids, hole-cutting

AMS subject classifications. 65M50, 65N50

PII. S1064827597329102

1. Introduction.

The most commonly used algorithm for assembling an overlapping grid from a set of structured component grids consists of two major parts. The first step is to detect all hole points outside the computational domain and the second step is to classify the remainder of the grid points as either discretization, interpolation, or hole points (see Figure 1.1). In this paper, we focus on the first part, which is also called the hole-cutting.

The basic overlap algorithm, developed by Benek, Buning, and Steger [1] and Kreiss [7], works well for simple cases, like the grid in Figure 1.1. Since each part of the boundary is completely described by one closed grid line, it is possible to use this grid line to determine whether a point in the background grid is inside or outside the computational domain. It becomes harder to detect the hole points when more than one component grid describes each part of the boundary of the computational domain, as in Figure 1.2. In this case, there is no closed grid surface from one component that can be used to cut out holes from the background grids. Furthermore, there can be a mismatch at the boundary where the components overlap each other. The mismatch makes the definition of the boundary slightly imprecise, which complicates the hole-cutting process since there is no unique definition of the extent of the computational domain close to the boundary in the overlap region. The mismatch problem becomes more pronounced when the grid is very fine in the direction normal to the boundary, and the grid cells have a large aspect ratio; see Petersson [11].

[*]Received by the editors October 21, 1997; accepted for publication (in revised form) May 26, 1998; published electronically October 14, 1999.
http://www.siam.org/journals/sisc/21-2/32910.html

[†]Hydromechanics Division, Naval Architecture and Ocean Engineering, Chalmers University of Technology, Gothenburg, Sweden (andersp@na.chalmers.se).

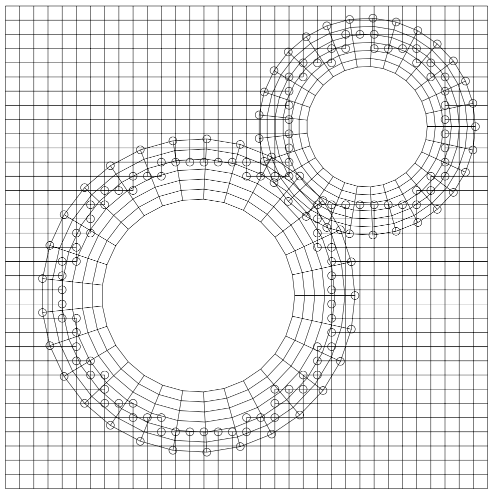

FIG. 1.1. *A simple overlapping grid where the hole points are blanked. The circles indicate interpolation points where the solution value is interpolated from the overlapping component grid. The remainder of the points are used to discretize the partial differential equation or the boundary conditions.*

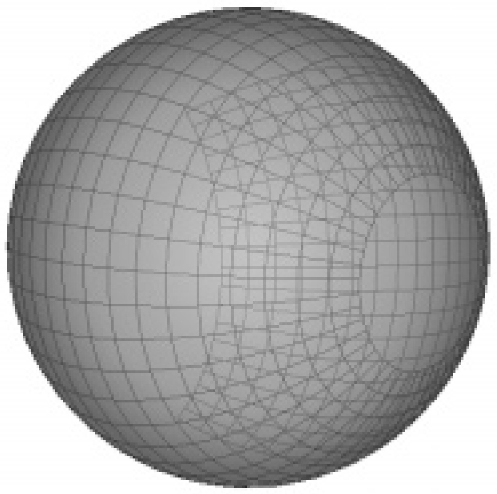

FIG. 1.2. *The component grids on the surface of a sphere. Each pole of the sphere is covered by a smooth component to avoid the polar singularity.*

One way to avoid the mismatch complications would be to use a globally valid boundary description, e.g., from a CAD system. To define the behavior of the components in between the grid points, that approach would require the overlapping grid algorithm to communicate closely with the component grid generators and the CAD system during the assembly of the overlapping grid. To code up such a monolithic system would require a lot of work, and it would also be hard to use component grids from external special-purpose grid generators in that system. For these reasons, we advocate a modular system that separates the overlapping grid assembly from the geometry description and the component grid generation.

A number of algorithms exist for constructing overlapping grids. Here we will review the hole-cutting aspects of the methods. For a more thorough description, we refer to Petersson [11]. The technique developed by Benek, Steger, and Dougherty [2] and Benek, Buning, and Steger [1] has been implemented and developed further in the code PEGSUS [13]. In order for the algorithm to cut the holes, the user must specify the hole-cutting surface as well as the hole grid, i.e., the grid in which the hole should be made. The hole-cutting surface can be either one or more grid surfaces or one or more rectangular boxes. Each grid point in the hole grid is checked to see if it is inside or outside the hole-cutting surface. This method requires the collection of hole-cutting grid surfaces to be convex, so holes with a concave boundary must be decomposed into convex parts by the user before the algorithm is applied.

Meakin [9] proposed a more efficient method, where hole-cutting surfaces can be defined by combinations of simple analytical shapes like cylinders and spheres. Although this imposes restrictions on the shape of the object to be gridded, it makes it straightforward and inexpensive to determine whether a grid point is inside or outside of the hole-cutting surface. This technique is implemented in the code DCF3D.

A combination of the overlapping grid method and the patched multiblock approach, where the component grids have common internal boundaries, is used in the technique developed by Maple and Belk [8] and implemented in the code Beggar. In this method, there can be one or several patched component grids within each super-block. The super-blocks overlap each other and communicate through interpolation. A data structure based on a combination of octrees and BSP-trees is used for determining whether a point is inside or outside of a super-block as well as providing an initial approximation for inverting the component grid mappings. The algorithm uses all grid surfaces with solid boundary condition to cut holes. The hole points outside of the computational domain are identified in a two-step mark-and-fill process, which requires the union of the hole-cutting grid surfaces to be closed but allows very thin holes.

Both PEGSUS and DCF3D require rather detailed user input concerning where holes should be cut out from the component grids. A more automatic method was developed by Chesshire and Henshaw [4] and originally implemented in the Fortran code CMPGRD [3]. The current version of the code is rewritten in C++ and is part of the Overture framework [5]. In that method, the algorithm determines iteratively where the holes should be made, based on where the physical boundary is located. Discrepancies between the boundary representations where the mappings overlap are handled by a user-specified mismatch tolerance together with a correction of the interpolation weights. This procedure is similar to the technique described by Petersson [11]. Although the iterative approach makes it possible to construct overlapping grids around very complicated objects, the algorithm encounters difficulties if the overlap between two component grids is too narrow. Then the number of hole points will increase during each iteration of the classification, resulting in an empty overlapping grid where all grid points are labeled as hole points.

The present method is an extension of the two-dimensional (2D) hole-cutting technique developed by Petersson [11] to the three-dimensional (3D) case. The boundary of a bounded 3D computational domain consists of a number of closed surfaces. We represent each closed boundary surface by a hybrid surface grid consisting of nonoverlapping structured surface grids joined by an unstructured layer of triangles; see Fig-

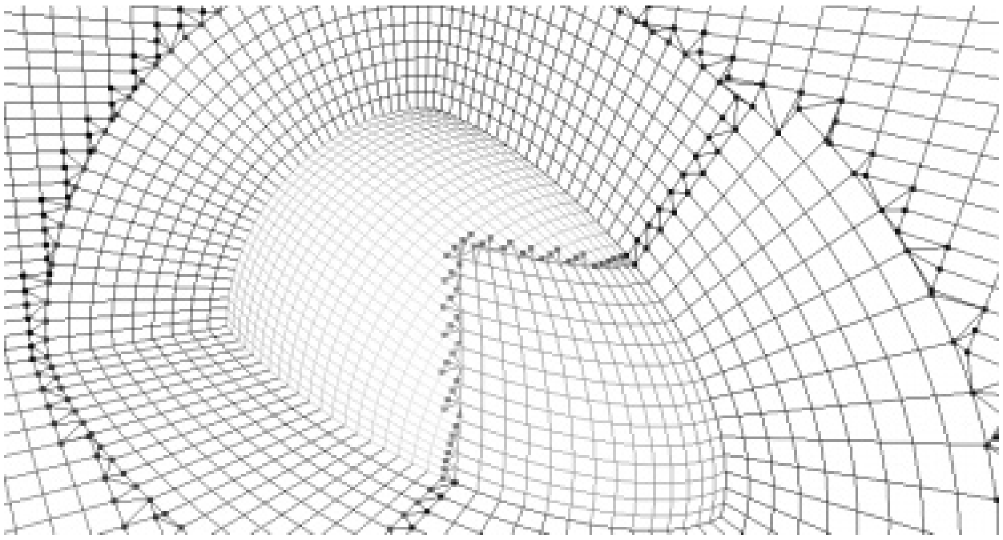

FIG. 1.3. *One hybrid grid covers each closed surface on the boundary of the computational domain.*

ure 1.3. A similar type of grid was used by Kao and Liou [6] for solving the equations of gas dynamics. Here, we use the hybrid grid to get a unique definition of the boundary of the computational domain. It is important that the boundary be unique since we employ the ray method (see Milgram [10]) to determine whether a point is inside or outside the computational domain during the hole-cutting algorithm.

Each hybrid grid is constructed from a subset of the physical boundary points on the faces of the component grids. In regions where the surface grids do not overlap, the hybrid grid is structured and consists of the quadrilateral grid cells. To make the hybrid grid unique where the surface grids overlap each other, some grid points in the overlap region are blanked, such that the remaining points in the surface grids do not overlap each other. The gap between the surface grids is filled by triangles that connect the internal boundaries. We present the full algorithm for constructing hybrid surface grids in section 2.

The holes in the component grids are made by using a two-step mark-and-fill method similar to the technique employed by Maple and Belk [8]. We first locate all grid cells that intersect the hybrid surface grids. Once the intersected grid cells have been located, we use the ray method to determine whether the grid points in these intersected grid cells are inside or outside the region bounded by the hybrid surface grids. The holes are then filled starting from the outside points by traversing along all three grid directions until an inside point is found. To locate the intersected grid cells, we must be able to invert the mapping corresponding to a component grid. For this purpose, we first employ the ray method to check if the point on the hybrid grid is inside or outside the boundary of the component. If it is inside, we proceed by locating the intersected grid cell by using an octree-based search technique together with Newton's method. The details of the hole-cutting algorithm are described in section 3.

The present hole-cutting method can handle boundaries that are described by several overlapping components and does not require the sides with a solid boundary condition to form closed surfaces. Similar to CMPGRD, the boundary mismatch

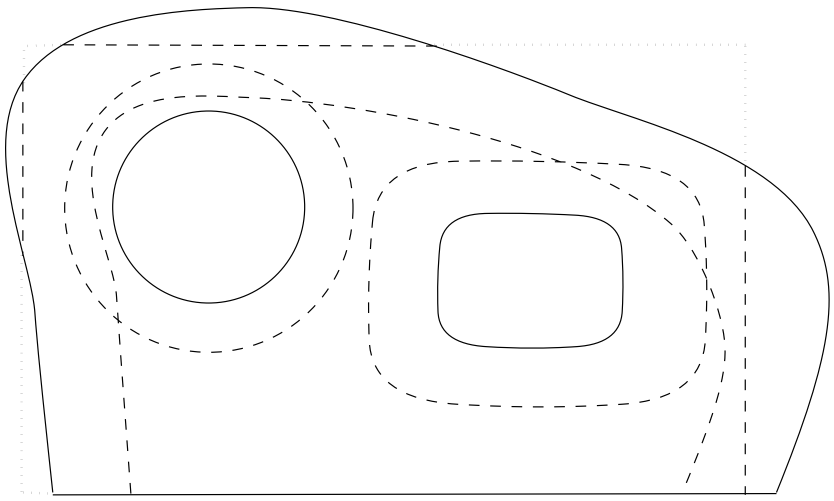

FIG. 2.1. *The extent of the computational domain is determined by the physical boundaries and by the position of the component grids relative to those boundaries. Here solid lines are physical boundaries, dashed lines represent interpolation boundaries, and dotted lines indicate external boundaries.*

problem is handled by using a mismatch tolerance, which in the present method is estimated automatically.

The method has been implemented in the overlapping grid code Chalmesh [12]. In section 4, we use this code to exemplify the method on three geometries: a sphere in a box, a ship hull, and a three-bladed ship propeller.

2. The hybrid surface grid.

The boundary points on the faces of the component grids can be physical or nonphysical. The physical boundary points are situated on the boundary of the computational domain. For example, in a fluid flow model, all grid points on no-slip, slip, inflow, outflow, and far-field boundaries are physical boundary points. The nonphysical points can be interpolating or external. An interpolating boundary point lies inside another component grid such that the solution value can be interpolated to that boundary, and an external boundary point is situated outside the computational domain (see Figure 2.1).

In the present method, each face of each component grid is labeled as either physical, mixed physical-external, or nonphysical. A surface label is assigned to all physical and mixed physical-external faces such that the surface label is the same for all faces of all components that are aligned with the same part of the boundary. The external portion of a mixed physical-external face in one component is bounded by an edge curve that is situated at the intersection between two physical grid faces in one or several other component grids; see Figure 2.2.

The hybrid grid on each part of the boundary, i.e., corresponding to the same surface label, is constructed in three steps. The first step is performed only if there are mixed physical-external faces on any component grid and consists of blanking the external surface points outside the edge curves. The second step is to construct a nonoverlapping surface grid, and the third step is to set up the surface triangulation that will join the nonoverlapping grids. To perform the first two steps, we must be able to invert a surface mapping to check whether a point on one surface grid is inside another surface grid. We start by describing that algorithm.

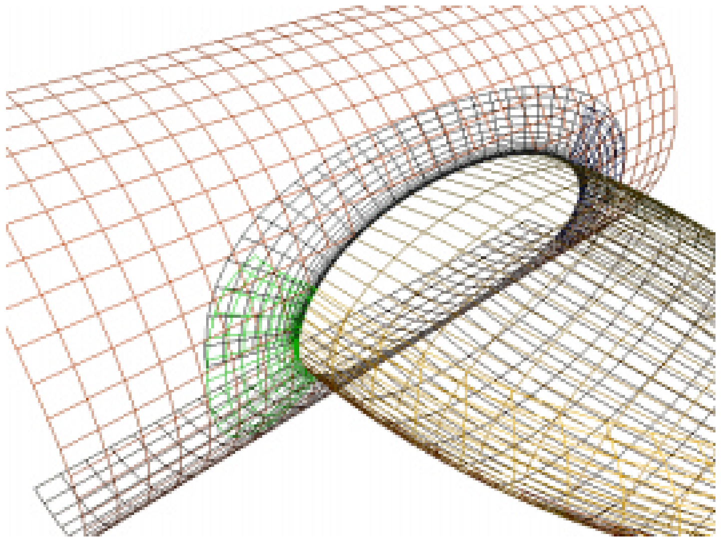

FIG. 2.2. *External surface points are not on the boundary of the computational domain and must therefore be removed before the hybrid surface grid is constructed. In this case, the grid points on the cylinder, inside the ellipse, are external. They are bounded by an edge curve, which is located at the intersection between the cylinder and the ellipse.*

2.1. Inverting a surface grid mapping. Let us consider inverting the mapping corresponding to a continuous curvilinear surface grid

$$\mathbf{x} = \mathbf{S}(r,s), \quad 0 \le r \le 1, \quad 0 \le s \le 1, \quad \mathbf{x} = (x,y,z).$$

It is not practical to restrict the inversion algorithm to points that are exactly on the surface $\mathbf{S}$, since there can be a mismatch between the surface grids that describe the same physical boundary. We handle the mismatch problem by introducing a surface tolerance $\epsilon_s > 0$ and consider all points that are within ϵ_s of the surface to be on the surface.

The required size of the mismatch tolerance is related to the smoothness of the surface grid. Let there be N-by-M grid points in the surface grid with physical coordinates $\mathbf{x}_{i,j}$, $1 \le i \le N$, $1 \le j \le M$. We estimate the mismatch tolerance by

$$\epsilon_s = \max_{1<i<N, 1<j<M} |\mathbf{n}_{i,j} \cdot (\mathbf{x}_{i-1,j} + \mathbf{x}_{i+1,j} + \mathbf{x}_{i,j-1} + \mathbf{x}_{i,j+1} - 4\mathbf{x}_{i,j})/4|,$$

where the unit surface normal is

$$\mathbf{n}_{i,j} = \frac{(\mathbf{x}_{i+1,j} - \mathbf{x}_{i-1,j}) \times (\mathbf{x}_{i,j+1} - \mathbf{x}_{i,j-1})}{|(\mathbf{x}_{i+1,j} - \mathbf{x}_{i-1,j}) \times (\mathbf{x}_{i,j+1} - \mathbf{x}_{i,j-1})|}.$$

The estimated mismatch tolerance is sufficient in most cases, but it can be necessary to increase it further for very unsmooth grids.

To allow for boundary mismatch, we extend the surface to a thin volume around the surface, with thickness ϵ_s. Hence, the actual problem is to invert

$$\mathbf{x} = \mathbf{S}(r,s) + t\,\mathbf{n}(r,s), \quad |t| \le \epsilon_s, \tag{2.1}$$

where $\mathbf{n}$ is the unit normal of $\mathbf{S}$.

Before Newton's method can be applied to solve (2.1), we must determine if $\mathbf{x}$ is sufficiently close to $\mathbf{S}$ and, in that case, compute an initial guess of the solution. For this purpose, we apply a search technique based on quad trees.

The root of the quad tree contains the bounding box of the physical coordinates of all grid points in the surface grid. The bounding box is extended by ϵ_s on all sides to ensure that all points within an ϵ_s distance of $\mathbf{S}$ are also inside the bounding box. The root has four subnodes that contain the bounding boxes of the four subgrids obtained by dividing the surface grid along the grid lines $i = [N/2]$ and $j = [M/2]$. (Here $[a]$ denotes the integer part of a.) Each node of the tree is subdivided recursively into four subnodes until the corresponding subgrid contains only one grid cell. These nodes are the leaves of the quad tree.

To invert the surface grid mapping for a point $\mathbf{x}_p$, we first check whether it is inside the bounding box of the root of the quad tree. If it is, we repeat the test recursively in the subnodes until a leaf is found whose bounding box contains $\mathbf{x}_p$. If no such leaf is found, or if $\mathbf{x}_p$ is outside of all bounding boxes on one level of the tree, the point must be further than ϵ_s away from $\mathbf{S}$ and the search is terminated. If an enclosing leaf is found, we apply Newton's method to invert (2.1) starting from the parameter value in the middle of the corresponding grid cell. If the iteration converges to a point within ϵ_s of the surface, we terminate the search. Otherwise we continue to traverse the quad tree until all enclosing leaves have been checked. This is necessary since the bounding boxes of the subgrids on one level can overlap each other.

2.2. Blanking surface points outside edge curves. There are geometries where it is natural to extend the physical boundary to the outside of the computational domain to simplify the construction of the component grids. One example is a wing-fuselage junction on an aircraft, where the surface of the fuselage is also described inside the wing. A straightforward way of gridding this geometry is to put one component grid around the fuselage and one around the wing. Since the grid points on the surface of the fuselage, inside the wing, are not on the boundary of the computational domain, they must be blanked before the hybrid surface grid can be constructed.

In the above case, the external surface points inside the wing are bounded by the grid line in the wing grid that is on the edge between the wing surface and the fuselage. In the present method, we assume that an edge curve is always situated at the intersection between two physical grid surfaces. The simplest case occurs when only one component grid describes the physical surfaces, so that the edge curve is a grid line in that component. In the general case, the edge curve can consist of several overlapping parts from different component grids. In this case, we represent the edge curve by a polygon through all edge points from the different parts. To make it easy to estimate the tangent of the edge curve, we sort the points on the edge curve. To handle the case with thin holes, where the tangential distance between the edge points is larger than the thickness of the hole, we apply the algorithm for sorting boundary points in a 2D overlapping grid described in Petersson [11], extended to 3D curves.

The edge curve defines the boundary of the external surface points. To determine if a surface point is inside or outside the edge curve, we also save two test curves together with the edge curve. The test curves are grid lines on each of the two physical grid surfaces that join at the edge curve (see Figure 2.3). Since the test curves are on the boundary of the computational domain, they are always inside the edge curve.

To blank external points on a grid surface, we first search through the grid points

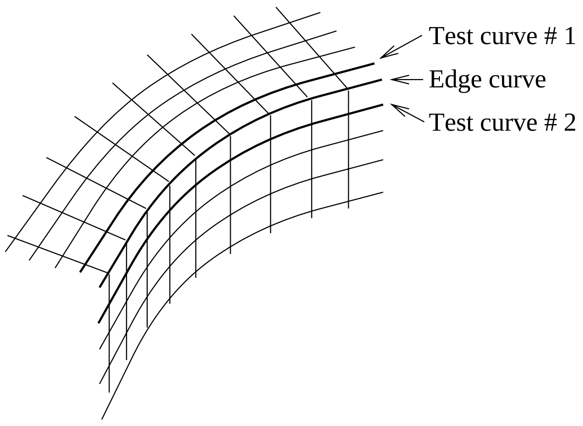

Fig. 2.3. *The test curves are located close to the edge curve, on the physical grid surfaces that join at the edge curve.*

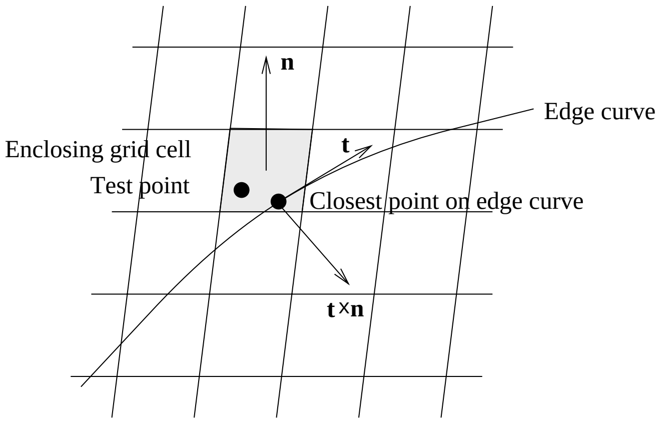

Fig. 2.4. *The surface normal, edge tangent, and edge normal in the surface plane. Observe that the edge curve and the test point do not belong to the surface grid shown in the figure.*

on the two test curves. If one point $\mathbf{x}_t$ is found to be on the grid surface, we compute the normal, $\mathbf{n}$, of the surface at the cell that contains the test point. We also locate the grid point $\mathbf{x}_e$ on the edge curve that is closest to the test point and compute the tangent, $\mathbf{t}$, of the edge curve at that point. The cross product between the surface normal and the edge curve tangent defines a normal of the edge curve that is tangential to the grid surface (see Figure 2.4). Consider

$$s(\mathbf{x}_t) = \operatorname{sign}((\mathbf{x}_t - \mathbf{x}_e) \cdot (\mathbf{t} \times \mathbf{n})).$$

Since $\mathbf{x}_t$ is on the boundary of the computational domain, that is, inside the edge curve, we can use $s(\mathbf{x}_t)$ to determine whether another surface grid point, $\mathbf{x}_p$, which is close to the edge curve, is inside or outside the edge curve. To do this, we compute the surface normal at the grid cell that contains $\mathbf{x}_p$, locate the closest point on the edge curve, and compute the tangent of the edge curve. The point $\mathbf{x}_p$ is inside the edge curve if $s(\mathbf{x}_p)$ has the same sign as $s(\mathbf{x}_t)$; otherwise it is outside.

The surface grid points outside the edge curve are blanked by using a mark-and-fill technique, related to the method used by Maple and Belk [8]. All surface grid cells

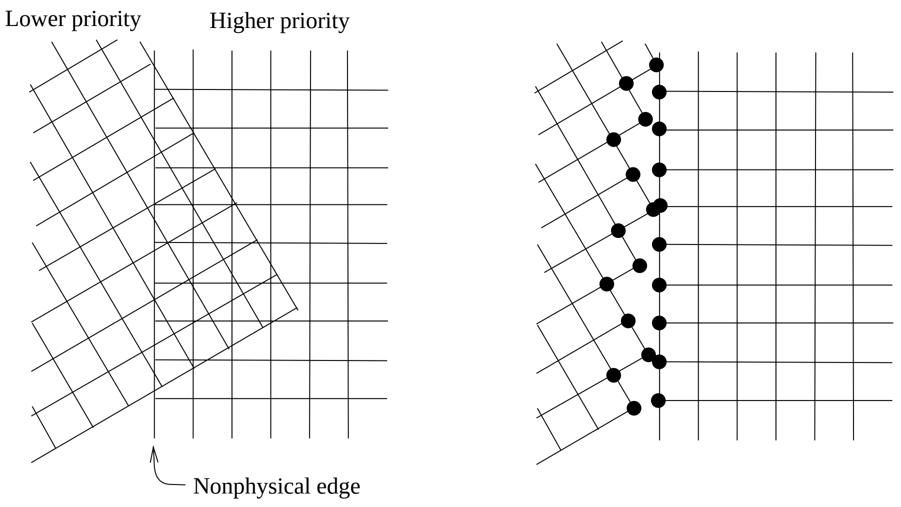

Fig. 2.5. *The blanking of the overlapping surface points depends on the priority of the surface grids. Here, boundary points are marked with circles.*

that are intersected by the edge curve are first identified. We start by locating the cells that contain the vertices of the edge curve. If those cells are not contiguous, we subdivide the segments between the vertices and repeat the identification recursively. The grid points in the intersected cells are then marked as either I- or O-points, depending on whether they are inside or outside the edge curve. The external surface points are identified by starting at the O-points and proceeding along the grid lines in both directions. The location of the I-points next to the O-points defines in which direction the external points are situated. We blank all points until we reach the next I-point along the grid line, or until the grid line ends.

2.3. Constructing the nonoverlapping surface grid. Let the structured surface grids covering each closed boundary surface be ordered in a linear hierarchy where the grid with the highest number has the highest priority. All edges of the surface grids correspond to edges in the component grids. We label all edges located between physical grid faces as physical. The other edges are labeled as nonphysical.

Each grid point in each surface grid is marked as internal or overlapping, depending on whether it is outside all other surface grids or inside at least one other surface grid. For the overlapping points, we save the priority of the highest overlapping surface grid so that the appropriate points in the overlap region can be blanked. A grid point is on an internal boundary that should be connected to another internal boundary in either of the following situations (see Figure 2.5): first, if it is an internal point and at least one of its nearest neighbors is an overlapping point inside a higher priority surface grid; second, if the point is located on a nonphysical edge and is inside of a lower priority surface grid. After all boundary points have been identified, we blank all overlapping points that are inside a higher priority surface grid.

Before the internal boundary points are connected by triangles, we determine to which surface grid each boundary point should be connected. For this purpose, we locate the closest boundary point with respect to a weighted distance function that

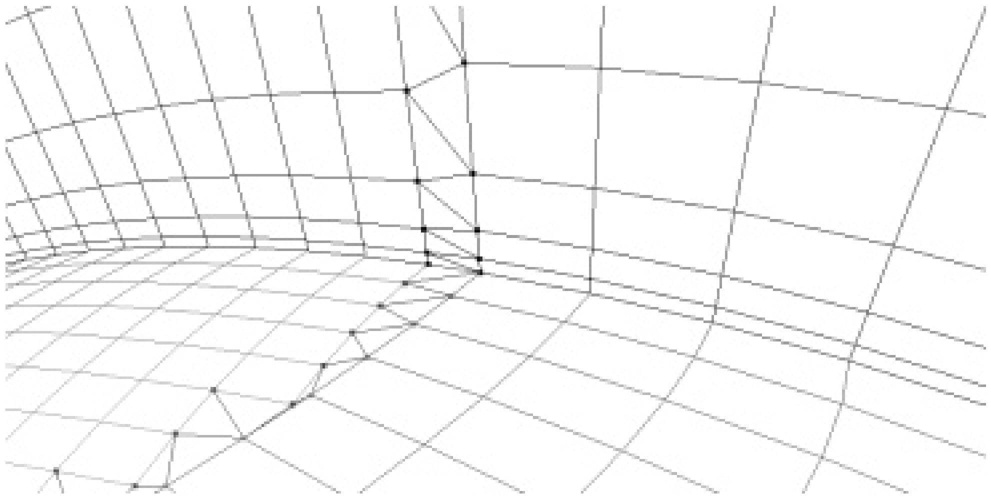

FIG. 2.6. *The closest boundary point near an edge curve can be on the other side of the edge. Here, boundary points are marked by filled squares.*

puts a penalty on distances in the normal direction:

$$d(\mathbf{x}_1, \mathbf{x}_2) = \Big(\sqrt{|\mathbf{x}_1 - \mathbf{x}_2|^2 + 100(\mathbf{n}_1 \cdot (\mathbf{x}_1 - \mathbf{x}_2))^2}, \\ \sqrt{|\mathbf{x}_1 - \mathbf{x}_2|^2 + 100(\mathbf{n}_2 \cdot (\mathbf{x}_1 - \mathbf{x}_2))^2} \Big),$$

where $\mathbf{n}_1$ and $\mathbf{n}_2$ are the unit normals of the surface grid at $\mathbf{x}_1$ and $\mathbf{x}_2$, respectively. The reason for penalizing normal distances is to avoid connecting boundary points on opposite sides of an edge (see Figure 2.6). We remark that the penalty factor 100 is arbitrary and could be insufficient in some cases. However, our experience indicates that it works well in most situations.

We group the boundary points according to which grid they belong and to which grid they will be connected. We assume that to each group A of points from grid G_A there corresponds a sister group B of points from the grid G_B. All points in group A will be connected to grid G_B and all points in group B will be connected to grid G_A. The points in each group are sorted so that nearest neighbors in the surface grid become contiguous in the group. Furthermore, to simplify the connection of the groups, the first point in the sister group B is chosen as the endpoint that is closest to the first point in group A.

2.4. Joining the nonoverlapping surface grids by triangles. The triangulation is done in two steps. First we connect each pair of groups of boundary points and then we triangulate the triple points, where three different surface grids meet (see Figure 2.7).

Beginning from the first points in groups A and B, we use an incremental triangulation algorithm where the first triangle connects the first points of groups A and B and the next point in either group A or B. Since the internal boundaries can be irregular, it is important to check that the triangle does not intersect the boundary. If neither of the two triangles intersects the boundary, we choose the triangle with the shortest diagonal. Once the first triangle is constructed, we move the starting point

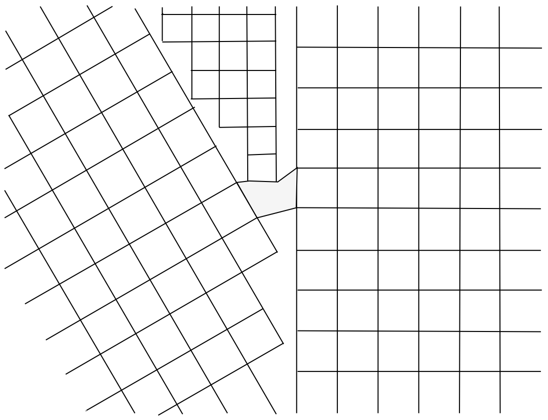

FIG. 2.7. *Each triple point is bounded by six boundary points.*

to the next point and the procedure is repeated until the boundary points of groups A and B are exhausted.

We remark that there are surface grids that cannot be connected by triangles without intersecting the internal boundary. This situation arises when a very coarse grid is joined to a very fine grid with a nonsmooth internal boundary. One way to solve the problem could be to add extra surface points in between the boundaries. However, this technique will not be discussed further here.

After the gaps between all pairs of groups have been triangulated, there remain gaps at the triple points, where three surface grids meet. The starting and ending points of each pair of groups are saved during the triangulation algorithm. Each triple point is bounded by six boundary points from three pairs of starting and ending points. To cover a triple point by triangles, we again apply an incremental approach. We begin by locating three contiguous boundary points that form a triangle that does not intersect the boundary of the triple point. When that triangle has been subtracted from the triple point, the remaining domain has only five vertices and we repeat the procedure until all vertices have been connected by triangles. The triple point is completely covered when four triangles have been constructed.

3. Making holes in volume grids.

The holes in the component grids are made by using a two-step mark-and-fill method similar to the technique for removing external surface points that was described in section 2.2.

The hybrid grid corresponding to each surface label contains both triangular and quadrilateral cells. To simplify the mark step, we divide each quadrilateral cell into two triangles so that we need only to develop a technique for marking the cells that are intersected by a triangle. Each component grid is marked separately. The marking starts by locating the cells that contain the vertices of each triangle on the hybrid surface grids. If these cells are not contiguous, the triangle is subdivided into four subtriangles and the marking is repeated recursively (see Figure 3.1).

To apply the hole-cutting method to volume grids, we must be able to invert the mapping corresponding to a volume grid. We begin by describing that technique.

3.1. Inverting a component grid mapping.

We assume that the mapping for each component grid is one to one and that the corresponding grid in parameter space is Cartesian with constant step size. In the following, we will denote a coordinate in

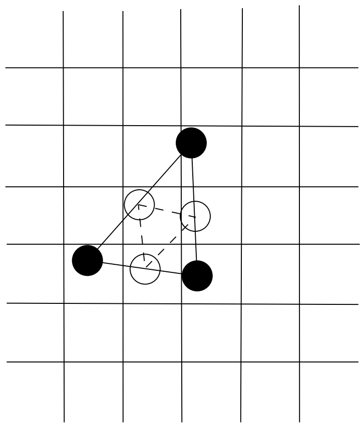

FIG. 3.1. *The recursive marking of the intersected cells proceeds until the marked cells corresponding to each triangle are contiguous. The filled circles label the vertices of the boundary triangles. Since the cells that contain the filled circles are separated, the triangle is subdivided to create three new vertices, indicated by unfilled circles. After one subdivision, the marked cells are contiguous and the recursion is stopped.*

parameter space by $\mathbf{r} = (r_1, r_2, r_3)$ and a coordinate in physical space by $\mathbf{x} = (x, y, z)$. Furthermore, let the mapping for component grid k be $\mathbf{x} = X^{(k)}(\mathbf{r})$ and let it map the unit cube in parameter space onto the domain $\mathbf{x} \in \Omega^{(k)}$ in physical space.

Let us consider inverting the mapping corresponding to a continuous curvilinear grid G in order to find the grid cell that contains a point with physical coordinate $\mathbf{x}_P$. Before we attempt to invert the mapping, we must determine if $\mathbf{x}_P \in \Omega^{(G)}$, since the mapping might not be well defined outside that domain. If the point is inside $\Omega^{(G)}$, the second problem is to generate a sufficiently good initial approximation for Newton's method, which converges only if the initial guess is sufficiently close to the solution. Once the parameter value corresponding to $\mathbf{x}_P$ has been computed, it is trivial to determine the enclosing grid cell.

The inversion of a component grid mapping can be complicated by mismatch close to a physical boundary. Similar to the 2D case (see Petersson [11]) we handle the mismatch problem by introducing a mismatch tolerance $\epsilon > 0$. Let $\mathbf{x}_P$ be a grid point in component grid C. If it is less than ϵ away from a physical face with surface label S in grid C, we say that it is an S-boundary point. An S-boundary point is inside component G if it is less than ϵ outside a physical face of $\Omega^{(G)}$ that also has surface label S. However, if $\mathbf{x}_P$ is further than ϵ away from all physical faces in grid C, the point is considered to be inside component G only if it is strictly inside $\Omega^{(G)}$.

It is desirable to keep ϵ as small as possible because the inversion algorithm becomes slower for a larger ϵ, since more points get treated specially by the search method. We therefore want to use the smallest ϵ that still enables all physical boundary points from an overlapping component to be treated as if they were inside the component. Furthermore, to correctly classify the points that remain after the hole cutting, the mismatch tolerance must be sufficiently large to allow all interpolation points close to a physical boundary in their own grid to interpolate from the appropriate component grid. In the present method, we take the mismatch tolerance ϵ for a component grid to be the largest surface tolerance ϵ_s in any of the physical grid faces of the component.

3.1.1. Is a point inside a component grid? To determine if a point with physical coordinate $\mathbf{x}_P$ is inside the domain $\Omega^{(G)}$, we first check if it is inside the approximate boundary consisting of a structured surface triangulation that connects all boundary grid points of the grid G.

The triangulated boundary is slightly outside the true boundary when it is concave and slightly inside the true boundary when it is convex. For points that are inside the triangulation, but outside the true boundary, the subsequent Newton iteration will converge to a parameter coordinate outside of the unit cube. If the nearest boundary is physical, it can be determined whether the point is within the ϵ mismatch tolerance. If the boundary is nonphysical, the point is classified as being outside of the grid G.

The situation is more complicated for points that are outside the triangulation but inside the true boundary. When $\mathbf{x}_P$ is not an S-boundary point for any surface label S, the problem is not critical because $\mathbf{x}_P$ is only slightly inside the grid G, so it would not be a valid interpolation point anyway. However, if $\mathbf{x}_P$ is an S-boundary point, we check if it is within the ϵ tolerance by computing the normal distances between $\mathbf{x}_P$ and all physical grid faces with surface label S. The normal distance is computed by inverting the surface grid mappings corresponding to physical grid faces of G by using the technique described in section 2.1.

The six grid faces of component grid G correspond to six structured surface grids $S^{(k)}$, $k = 1, 2, \dots, 6$. The vertices of the surface triangulation coincide with the vertices of the surface grids and each quadrilateral surface grid cell is divided into two triangles. The physical coordinates of a triangle with vertices $\mathbf{p}_i$, $i = 1, 2, 3$, can be expressed in barycentric coordinates according to

$$\mathbf{x}_{\text{tri}}(\beta_1, \beta_2) = \mathbf{p}_3 + \beta_1(\mathbf{p}_1 - \mathbf{p}_3) + \beta_2(\mathbf{p}_2 - \mathbf{p}_3), \quad 0 \le \beta_1, \beta_2 \le 1.$$

We use the ray method (see [10]) to check whether $\mathbf{x}_P$ is inside the surface triangulation of G. Let the ray start at $\mathbf{x}_P$ and tend to infinity in the negative x-direction. The physical coordinates of the ray are given by

$$\mathbf{x}_{\text{ray}}(t) = \mathbf{x}_P - t\mathbf{e}_x, \quad t > 0, \quad \mathbf{e}_x = (1, 0, 0).$$

The ray intersects the triangle if there is a solution of

$$\mathbf{x}_{\text{ray}}(t) = \mathbf{x}_{\text{tri}}(\beta_1, \beta_2) \tag{3.1}$$

with $0 \le \beta_1, \beta_2 \le 1$, $t > 0$. The linear system (3.1) is uniquely solvable if the normal of the triangle is not orthogonal to $\mathbf{e}_x$. If it is orthogonal, we define the ray not to intersect the triangle.

The physical coordinates of all intersection points between the ray and the triangles of $S^{(k)}$, $1 \le k \le 6$, are saved so that multiple intersection points can be removed before the intersections are counted. Multiple intersections occur, for instance, when the ray intersects a vertex. The ray will then intersect all six triangles that are connected to the vertex.

It is not necessary to solve (3.1) for every triangle in the six surface grids to count all intersections. The number of operations is substantially reduced by noting that the ray can intersect the triangles in a surface grid only if it intersects the bounding box of the surface grid. To use this property, we construct a quad tree for each surface grid $S^{(k)}$ in the way that was described in section 2.1. During the search for intersection points, we need consider only the subgrids whose bounding boxes are intersected by the ray, and (3.1) is solved only for the leaves of the quad tree that are encountered during the traversal of the tree.

3.1.2. Generating an initial guess for Newton's method. Let the point $\mathbf{x}_P$ be inside of the surface triangulation of a component grid G. To generate a good initial guess for Newton's method we locate the grid cell that contains $\mathbf{x}_P$.

Let the grid points of grid G have physical coordinates $\mathbf{x}_{i,j,k}$, $1 \le i \le N_1$, $1 \le j \le N_2$, $1 \le k \le N_3$. We define the component grid mapping inside each grid cell (i_0, j_0, k_0) by trilinear interpolation:

$$\mathbf{x}_{\text{cell}}(\alpha_1,\alpha_2,\alpha_3) = \sum_{i=0}^{1}\sum_{j=0}^{1}\sum_{k=0}^{1} \gamma_i(\alpha_1)\gamma_j(\alpha_2)\gamma_k(\alpha_3)\mathbf{x}_{i_0+i,j_0+j,k_0+k}, \quad 0 \le \alpha_l \le 1, \tag{3.2}$$
$$l = 1,2,3.$$

Here, the linear base functions are $\gamma_0(\alpha) = 1 - \alpha$ and $\gamma_1(\alpha) = \alpha$.

The point $\mathbf{x}_P$ can be inside a grid cell only if the point is inside the bounding box of the physical coordinates of the cell. However, the bounding box can be much larger than the grid cell, for instance, when the cell is very thin in one direction and the cell is orientated diagonally to the (x, y, z)-directions. It can therefore happen that Newton's method for solving

$$\mathbf{x}_{\text{cell}}(\alpha_1,\alpha_2,\alpha_3) = \mathbf{x}_P \tag{3.3}$$

diverges, even though $\mathbf{x}_P$ is inside the bounding box of the cell. However, if $\mathbf{x}_P$ is inside the cell and the Jacobian of the trilinear mapping (3.2) is nonsingular, our experience indicates that Newton's method always converges, starting from the initial guess $(\alpha_1, \alpha_2, \alpha_3) = (0.5, 0.5, 0.5)$.

Corresponding to the search technique for surface grids, we use an octree to locate the enclosing grid cell in the component grid. The root of the octree contains the bounding box of $\mathbf{x}_{i,j,k}$, $1 \le i \le N_1$, $1 \le j \le N_2$, $1 \le k \le N_3$. The root has eight subnodes that contain the bounding boxes of the eight subgrids obtained by dividing the component grid along the grid lines $i = [N_1/2]$, $j = [N_2/2]$, and $k = [N_3/2]$. Each node of the octree is divided recursively into eight subnodes until the nodes contain only one grid cell. These are the leaves of the octree.

To find the grid cell that contains the point $\mathbf{x}_P$, we use a similar approach as when a quad tree is traversed (see section 2.1). When a leaf with enclosing bounding box is encountered, we apply Newton's method to solve (3.3). If it converges to a point inside the cell, the search is terminated. Otherwise, if the iteration diverges or converges to a point outside the cell, we continue to traverse the remainder of the octree. We remark that since $\mathbf{x}_P$ is inside the surface triangulation on the boundary of the component grid, there must be a grid cell that contains $\mathbf{x}_P$.

3.2. Is a point inside the computational domain? Before all points outside the computational domain can be removed, we must mark the grid points in the grid cells that contain the boundary of the computational domain. Points that are within the mismatch tolerance of a physical boundary in their own component are considered to be inside the computational domain. To mark points that are separated by more than the mismatch tolerance from all physical boundaries in their own grid, we apply the ray method and count the intersections between the ray and the hybrid surface grids on all parts of the boundary of the computational domain.

Each hybrid grid consists of two parts: structured surface grids, where some grid points are blanked, and unstructured triangles that join the boundaries of the

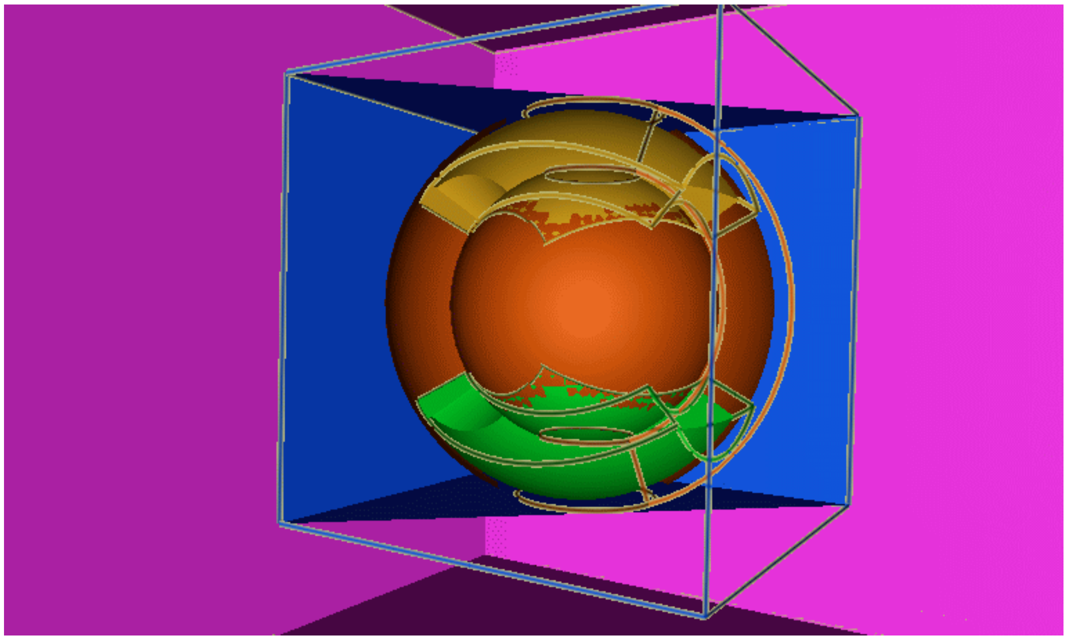

FIG. 4.1. *The five components in the overlapping grid for a sphere in a box.*

structured surface grids. It is straightforward to modify the technique in section 3.1.1 to count an intersection between the ray and a structured surface grid only if it occurs in a cell where no vertices have been blanked.

The unstructured triangles connect pairs of groups of boundary points and the triangles associated with each pair are ordered sequentially by construction. We therefore apply a binary search technique for counting intersections between the ray and the triangles from each pair. Finally, we must account for the triangles that cover the triple points on the surface. Since there are only four triangles on each triple point and there are usually only a few triple points in every surface triangulation, we apply the direct approach and check each of the triple point triangles for intersection.

4. Examples. We demonstrate the hole-cutting method by running the code Chalmesh [12], version 1.0-beta, compiled with optimization on a DEC-α station 200 with 64 Mbytes of RAM. Apart from the hole-cutting algorithm, the code contains a 3D version of the classification algorithm described in Petersson [11], some component grid generators, and an interactive graphics tool for visualizing the component grids and the overlapping grid in a variety of ways.

To indicate the CPU-time requirements of the hole-cutting algorithm, we consider the grid around a sphere inside a box, shown in Figure 4.1. To avoid the polar singularities, three components are used to describe the domain around the sphere. A coarse Cartesian grid extends to the boundary of the box and a finer Cartesian grid refines the discretization close to the sphere. Three overlapping grids were constructed. The number of grid points in each direction was increased by approximately 50% between the coarse and the medium grid and between the medium and the fine grid. The timings, presented in Table 4.1, indicate that the number of operations increases slower than linearly with the number of grid points.

The next example is the stern of a ship. The surface of the ship, together with the

TABLE 4.1
CPU time in seconds required by Chalmesh to set up the data structures and perform the hole cutting for the overlapping grid in Figure 4.1.

Grid	# grid points	Initialization	Hole cutting	Total	Total/grid point
Coarse	27050	2.2	6.3	8.5	3.14×10^{-4}
Medium	73598	4.6	14.7	19.3	2.62×10^{-4}
Fine	216400	10.2	42.0	52.2	2.41×10^{-4}

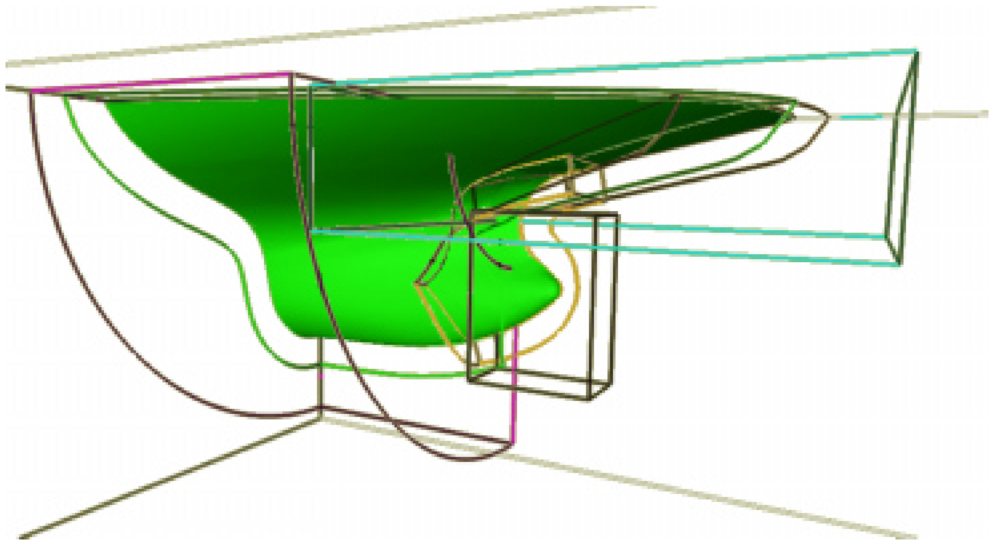

FIG. 4.2. *The surface of the ship stern and the outline of the* 7 *component grids. This grid contains* 137, 264 *grid points.*

outline of the component grids, is shown in Figure 4.2. In this case, three components are aligned with the surface of the ship to avoid the singularity at the aft bulb, where the propeller normally is attached. The body-fitted components are grown out from the surface by using a hyperbolic technique. Since there are concave regions on the surface, the thickness of these grids is of the order of the radius of curvature in the concave areas. The ship is enclosed in a rectangular box that extends to the far-field boundary. Since the flow field far from the ship can be expected to vary slowly, we make the Cartesian background grid in the rectangular box rather coarse. On the other hand, the geometrical properties of the ship surface restrict the thickness of the body-fitted components. We therefore use intermediate components to refine the region in the vicinity of the ship but outside the body-fitted grids. The thickness of the intermediate grids is not limited; thus they can be made sufficiently wide and fine to capture the details in the flow field close to the ship that cannot be resolved on the Cartesian background grid.

The boundary of the computational domain consists of one closed surface. Before the hybrid grid is constructed, we use three edge curves to remove external parts from mixed physical-external faces of the background grid and the intermediate components. The edge curves are located along the ship surface in the symmetry plane, the water line, and the inflow cross section. Note that the edge curve along the water line consists of parts from two components and the edge curve along the symmetry line is described by edge points from all three boundary-fitted components. The re-

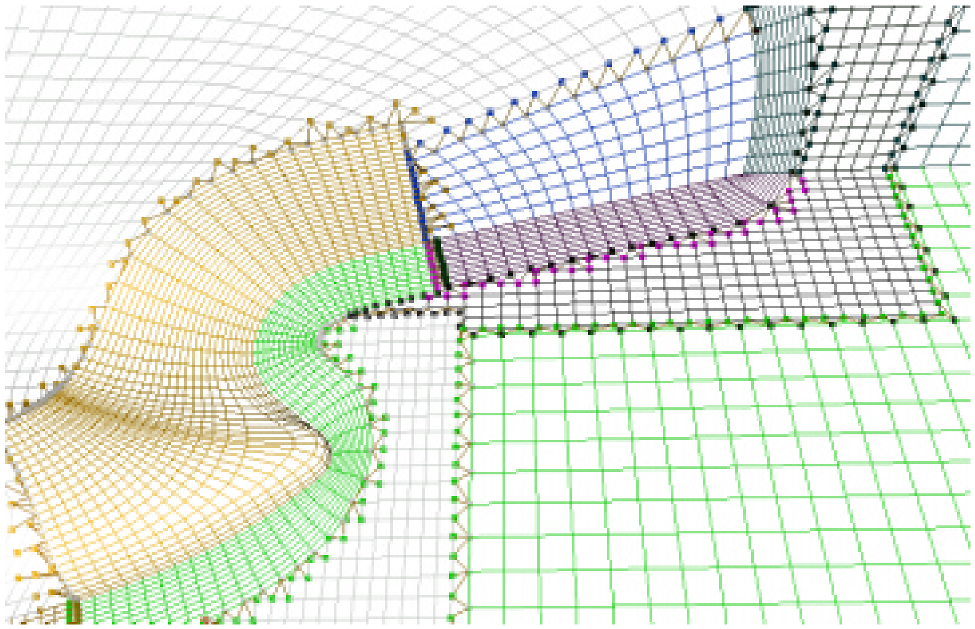

Fig. 4.3. *A closeup of the hybrid grid close to the aft bulb of the ship.*

sulting hybrid grid is shown in Figure 4.3. It took 22.3 seconds to set up the data structures and the hole cutting required 27.4 seconds. The result of the hole-cutting algorithm is presented in Figure 4.4.

The final example is a three-bladed DTRC–4119 ship propeller. We show the propeller surface together with the hub in Figure 4.5. The surface grids on the propeller surface are also outlined in the figure. On each propeller blade, we use one surface grid on each side of the blade and one surface grid around the edge of the blade. This decomposition is convenient because the curvature on each side of the blade is rather small, while it is very large at the leading and trailing edges, as well as around the tip of the blade. Furthermore, the singularity at the tip is avoided in this way. Since the curvature is high close to the edge, we concentrate grid points in this region. The body-fitted component grids are grown out from the blade using a hyperbolic technique, and the grid faces at the root of the blades are projected onto the hub surface. Since the curvature at the edge is very large, the grid cells in the edge grid expand significantly from the blade surface toward the outer grid surface. We therefore want to keep the edge grid rather thin. To resolve the tip vortex and the wake behind the blades, we again employ intermediate components. In this case each blade is contained in a twisted box grid whose thickness and grid sizes can be chosen to resolve the flow features in the vicinity of the blades. We embed the propeller in a rather coarse cylindrical background grid. To better resolve the flow close to the hub, we resolve this region with a finer cylindrical grid.

The boundary of the computational domain consists of one closed surface. Before the hybrid grid is constructed, three edge curves are used to blank the surface grid points on the hub, inside the propeller blades. In this case, each edge curve is periodic and consists of parts from four grid faces. We show the resulting hybrid grid in Figure 4.6. The overlapping grid contains 14 components with 487,008 grid points. The program required 72.0 seconds to set up the data structures and 751.0 seconds

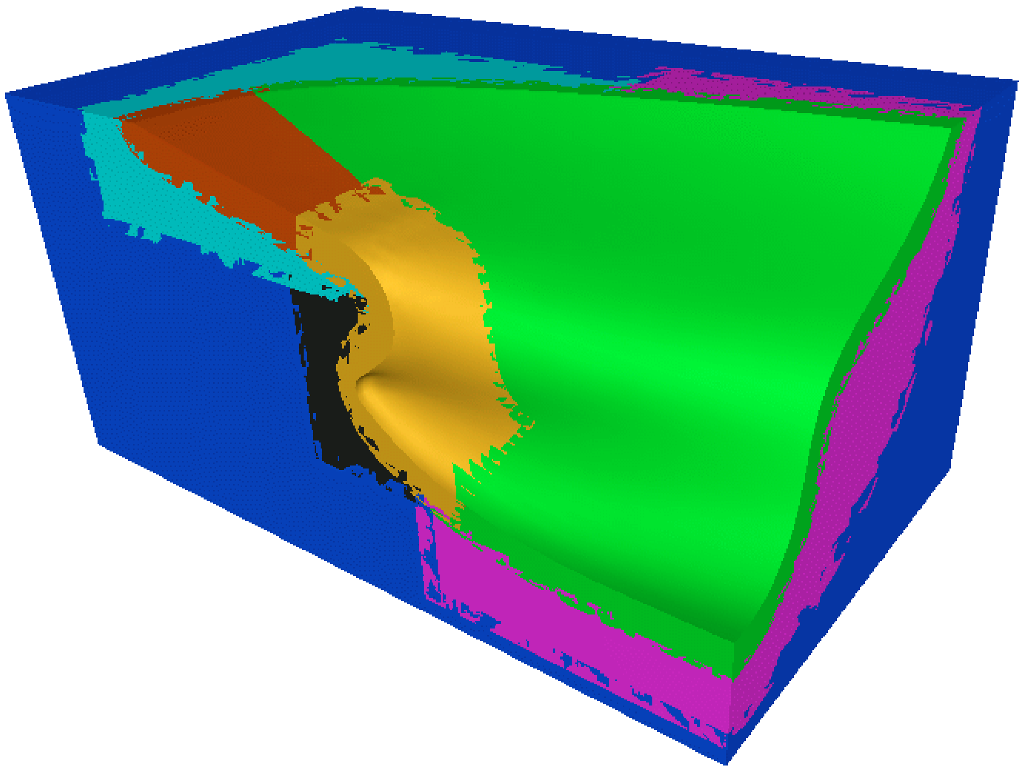

FIG. 4.4. *The overlapping grid around the ship stern after the hole cutting.*

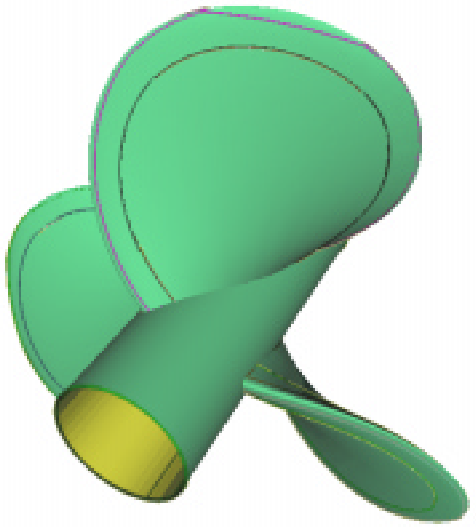

FIG. 4.5. *The surface of a three-bladed* DTRC–4119 *ship propeller on a hub. The thick lines show the boundaries of the surface grids.*

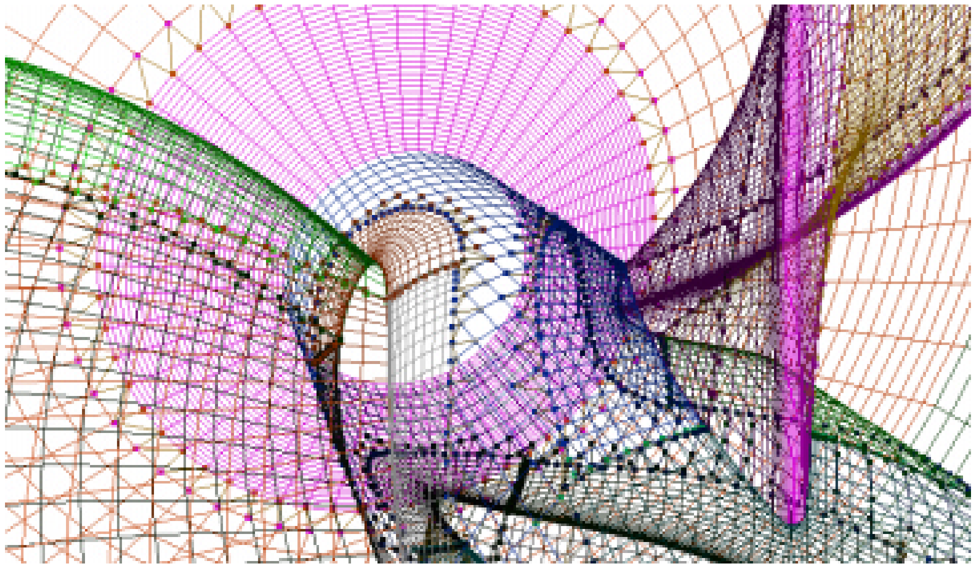

FIG. 4.6. *A closeup of the hybrid grid on the propeller.*

for the hole cutting. This grid was made on a DEC-α 3000 with 192 Mbytes of RAM since it did not fit in memory on the machine that was used in the previous tests. To compare the CPU timings, we note that the DEC-α 3000 is approximately 60% faster than the DEC-α station 200 (based on SPEC fp-95).

5. Conclusions. A hole-cutting method for 3D overlapping grids has been described. The algorithm has been implemented in the code Chalmesh [12], which is distributed under the GNU general public license. (See online license information at www.gnu.org.)

The 2D version of the algorithm [11] corrects the interpolation weights to compensate for boundary mismatch. This feature has not been discussed here, but the extension to the 3D case is rather straightforward. One part of the 3D method that needs more work is the algorithm for connecting surface patches by triangles. As was mentioned in section 2.4, the present technique sometimes encounters problems when it attempts to connect a fine and a coarse surface patch. Furthermore, the algorithm for generating nonoverlapping surface grids, described in section 2.3, fails in some degenerate cases when the grids are very coarse. These issues will be investigated in future work.

Acknowledgments. I thank Dr. Da-Qing Li for providing the surface grids for the ship hull and the ship propeller as well as for testing the Chalmesh code during its development. I also thank Dr. Geoff Chesshire as well as two anonymous referees for pointing out shortcomings in earlier versions of this article.

REFERENCES

[1] J. A. BENEK, P. G. BUNING, AND J. L. STEGER, *A 3-D Chimera Grid Embedding Technique*, AIAA paper 83–1944, American Institute of Aeronautics and Astronautics, Reston, VA, 1983, pp. 373–382.

[2] J. A. Benek, J. L. Steger, and F. C. Dougherty, *A Flexible Grid Embedding Technique with Application to the Euler Equations*, AIAA paper 85–1523, American Institute of Aeronautics and Astronautics, Reston, VA, 1985, pp. 322–331.

[3] 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, Los Alamos, NM, 1989.

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

[5] W. D. Henshaw, *Ogen: An Overlapping Grid Generator for Overture*, LA-UR 96-3466, Los Alamos National Laboratory, Los Alamos, NM, 1996.

[6] K.-H. Kao and M.-S. Liou, *Advance in overset grid schemes: From Chimera to DRAGON grids*, AIAA J., 33 (1995), pp. 1809–1815.

[7] B. Kreiss, *Construction of a curvilinear grid*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 270–279.

[8] R. C. Maple and D. M. Belk, *A new approach to domain decomposition: The Beggar code*, in Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, N. P. Weatherill, ed., Pine Ridge Press, 1994, pp. 305–314.

[9] R. L. Meakin, *A New Method for Establishing Intergrid Communication Among Systems of Overset Grids*, AIAA paper 91–1586–CP, American Institute of Aeronautics and Astronautics, Reston, VA, 1991, pp. 662–670.

[10] M. S. Milgram, *Does a point lie inside a polygon?*, J. Comput. Phys., 84 (1989), pp. 134–144.

[11] N. A. Petersson, *An Algorithm for Assembling Overlapping Grid Systems*, Tech. Report CHA/NAV/R-97/0049, Hydromechanics Division, Naval Architecture and Ocean Engineering, Chalmers University of Technology, Gothenburg, Sweden, 1997; SIAM J. Sci. Comput., 20 (1999), pp. 1995–2022.

[12] N. A. Petersson, *User's Guide to Chalmesh Version* 1.0, Tech. Report CHA/NAV/R-97/0051, Hydromechanics Division, Naval Architecture. and Ocean Engineering, Chalmers University of Technology, Gothenburg, Sweden, 1997; also available online from http://www.na.chalmers.se/~andersp/chalmesh/chalmesh.html.

[13] N. E. Suhs, *Tutorial: PEGSUS Version* 4.0, in 2nd Symposium on Overset Composite Grid and Solution Technology, AIAA, New York, 1994.