INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids 2007; **53**:1361–1379
Published online 17 February 2006 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/fld.1195

Parallel remeshing of unstructured volume grids for CFD applications

U. Tremel[1,*,†], K. A. Sørensen[1,‡], S. Hitzel[1,§], H. Rieger[1,¶], Oubay Hassan[2,‖] and Nigel P. Weatherill[2,**]

[1]*Flight Physics Department, MT632, EADS Military Aircraft, 81663 Munich, Germany*
[2]*School of Engineering, University of Wales Swansea, Swansea SA2 8PP, U.K.*

SUMMARY

This paper presents a new approach towards the parallel local remeshing of unstructured tetrahedral grids on distributed memory parallel computers based on the message passing paradigm (MPI). The overall remeshing approach consisting of the parallel determination of the regions to be remeshed and the parallel local surface and volume remeshing is described in detail. An application of the remeshing algorithms in a time-accurate simulation of bodies in relative motion demonstrates the capabilities of the approach. Copyright © 2006 John Wiley & Sons, Ltd.

KEY WORDS: parallel local remeshing; unstructured tetrahedral grids; CFD applications

1. INTRODUCTION

Many application areas demand for a modification of the computational grid during the simulation. This may be caused by bodies in relative motion which require a corresponding adjustment of the volume grid. Additionally, for solution adaptive procedures where regions are identified by error estimates, the mesh may have to be changed to improve the numerical accuracy.

Here unstructured grids have major advantage compared to multi-block structured grids since they allow for local modifications whereas the latter require an adjustment of the

*Correspondence to: Udo Tremel, Flight Physics Department, MT632, EADS Military Aircraft, 81663 Munich, Germany.
†E-mail: udo.tremel@eads.com
‡E-mail: kaare.sorensen@eads.com
§E-mail: stephan.hitzel@eads.com
¶E-mail: herbert.rieger@eads.com
‖E-mail: o.hassan@swansea.ac.uk
**E-mail: n.p.weatherill@swansea.ac.uk

Contract/grant sponsor: EADS Military Aircraft

Received 25 October 2005
Revised 9 January 2006
Accepted 9 January 2006

global structure. Several approaches have already been proposed for modifying an unstructured grid:

- Mesh movement methods [1, 2] do not alter the grid topology, only the node coordinates are adjusted. These methods can be used for both the simulation of moving bodies as well as for mesh adaptation purpose. Since the element connectivity remains unchanged, they also are applicable for multi-block structured grids. The limits of these methods are reached when inverted or badly shaped elements are created which usually happens for large movements.
- Hierarchical refinement is another commonly applied technique where the unstructured mesh is altered in regions where a higher node density is required for improving the solution accuracy [3–7]. This technique is limited to mesh adaptation and needs to be combined with a mesh movement method for simulations of bodies in relative motion. The limits of the mesh movement techniques thus remain.
- Another method applies local edge collapse operations for coarsening the mesh and an incremental Delaunay point insertion algorithm for refining it [8] for mesh adaptation purposes. Together with a mesh movement strategy this approach can also be applied for time-dependent problems including large motions. This approach always requires and operates on a valid mesh. If inverted elements are created after the mesh deformation, the sequence of mesh deformation, coarsening and refinement has to be repeated on a fraction of the initial displacements, until the full displacements are applied. Therefore, the efficiency of this approach may be reduced in case of large relative movements.
- The local remeshing of unstructured grids is a quite general but also a complex approach. A hole is extracted from a mesh, then remeshed and merged into the mesh again. This technique has been applied successfully in several dynamic simulations including bodies in relative motion [9–13]. In combination with a mesh deformation scheme the amount of remeshings can usually be reduced to a minimum which results in a very efficient approach.
- A global remeshing is the most time-consuming approach for a modification of a grid. Here the complete grid is re-generated based on the surfaces moved or influenced by the solution computed on the current grid. This technique has been applied for adaptive flow simulations [14–16] as well as for the simulation of moving bodies [12].

In this paper, the *local* remeshing approach has been selected since it represents a good compromise between general applicability, efficiency and mesh quality. Compared to the combination of edge-collapsing and point insertion, it allows larger movements without any substeps. In comparison to a global remeshing approach, it is much more efficient and faster to remesh a small subset of an existent grid only rather than to restart the mesh generation from scratch. The approach is believed to be particularly efficient for the time-accurate simulation of bodies in relative motion. Concerning mesh quality, the local remeshing can be adjusted according to the demands of the user. It can be limited to a minimum set of elements by allowing elements of lower quality which results in a faster execution time. If more stringent quality thresholds are applied, larger regions to be remeshed are selected. This normally results in a longer execution time but should improve the overall grid quality since a larger number of low-quality elements is removed.

The time-accurate prediction of fluid flow around complex configurations in relative motion is a very important application area where a local remeshing can be applied. This type

of simulation has become increasingly relevant in the aerospace industry over the last years [17]. Typical applications involve meshes consisting of millions of mesh points to represent the highly detailed geometric models with the necessary level of accuracy and to obtain an accurate flow solution. Due to the size of the problem exceeding the limited computational resources of a single machine, it has to be divided and distributed across multiple processors to be able to get a result in a reasonable amount of time. For this type of simulation, the flow solution is usually computed in parallel, whereas changes to the mesh connectivity induced by the relative movements are executed sequentially. On a shared memory parallel platform, this continuous switching between multiple processors and only one processor can easily be performed because of the globally accessible data. On the other hand, on a distributed memory platform without direct access to remote data, this approach forces the parallel computation to be stopped and requires the sequential local remeshing of the complete grid on an additional single machine with enough memory. After the locally modified grid has again been decomposed into subdomains, the parallel flow solution can be continued. Whether this approach is feasible or not, clearly depends on the frequency of the exchanges and the time required to perform the different tasks. Particularly for the commonly utilized distributed memory parallel clusters build from commodity off-the-shelf (COTS) components, this sequential bottleneck renders the procedure unfeasible if a frequent switching between flow computation and remeshing is required. Hence, in addition to a parallel CFD solver, a parallel volume remeshing capability is also mandatory for an efficient time-accurate simulation of bodies in relative motion on distributed memory machines including frequent changes of the unstructured grid.

This paper describes a new approach towards the parallel local remeshing of unstructured tetrahedral grids on distributed memory parallel computers based on the message passing paradigm (MPI). In Section 2, a general overview of the remeshing approach is presented. The first step of the local remeshing procedure is the determination of the regions to be remeshed which is described in Section 3. A detailed description of the parallel surface and volume remeshing algorithms is given in Sections 4 and 5. The paper concludes with examples demonstrating the capabilities of the local remeshing approach, Section 6.

2. PARALLEL REMESHING APPROACH

Given an unstructured tetrahedral mesh partitioned in n domains, the parallel local remeshing algorithm can be formulated as follows:

1. Determine all regions to be remeshed in parallel by an analysis of the local elements according to different quality criteria. Additionally, perform intersection checks to detect elements crossing each other due to local movements of surface and/or volume parts of the mesh introduced by objects in relative motion.
2. Identify all continuous regions[††] to be remeshed by a parallel colouring algorithm. Repartition the mesh such that every continuous region is completely contained in one domain and that the domains are approximately balanced in terms of meshing work load and number of elements and nodes.

[††]All marked elements sharing a face are belonging to the same continuous region.

3. Extract in every domain the marked regions from the current grid and remesh them independently in parallel. After the meshing of the holes based on a constrained Delaunay triangulation [18] is completed, merge the local pieces together into a new distributed mesh.
4. If the imbalance in terms of number of elements and nodes per domain is above a user-defined threshold, repartition the mesh such that a better load balancing is obtained.
5. Continue with the flow computation.

If the regions to be remeshed reach sizes not handable by a single process, the procedure fails. Less critically if no reasonable balancing of the holes can be found, the scalability of the approach is reduced. In such cases alternative algorithms have to be developed for the parallel local volume remeshing of distributed unstructured tetrahedral meshes.

An option would be the application of a parallel volume mesh generation algorithm which fills the holes with elements after the surface parts have been remeshed in parallel. Possible candidates are the parallelized volume mesh generation algorithms based on octree, advancing front or Delaunay methods [19–27].

3. DETERMINATION OF REGIONS TO BE REMESHED

The first step in a local remeshing approach is the determination of the regions to be remeshed. Commonly applied strategies are the selection of invalid elements, the selection of highly distorted elements [12] or, more generally, the selection of elements not fulfilling the quality criterion applied. Another strategy is to select the elements which have a relative change in volume above a maximum limit [10]. After all elements are marked forming the holes to be remeshed, additional layers of elements adjacent to the marked element are added. By this enlargement of the holes the mesh generator has more freedom where to insert new interior nodes. Furthermore, this prevents a simple reconnect in very narrow and constrained regions.

The overall algorithm implemented for determining the regions to be remeshed can be formulated as follows:

1. Evaluate and cache the mesh size at each point.
2. Mark all edges, triangles and tetrahedra not respecting the local mesh size.
3. Mark all elements with a quality criterion such as the sliver value below the given threshold.
4. Mark all invalid elements.
5. If a motion analysis is activated by the user, check if any elements are intersecting each other after being moved.
6. Smooth and enlarge the marked regions by adding neighbour elements.

In detail every triangle and tetrahedron is analysed according to the triangle minimal inner angle $\alpha_{\min}$, the tetrahedron sliver value λ and the tetrahedron skewness value κ with

$$\lambda = \frac{\sqrt{2}}{12} \frac{\text{Average element edge length}}{\text{Volume}} \tag{1}$$

$$\kappa = \sqrt[3]{\frac{\text{Volume of ideal tetrahedron with edge length } l}{\text{Volume}}} \tag{2}$$

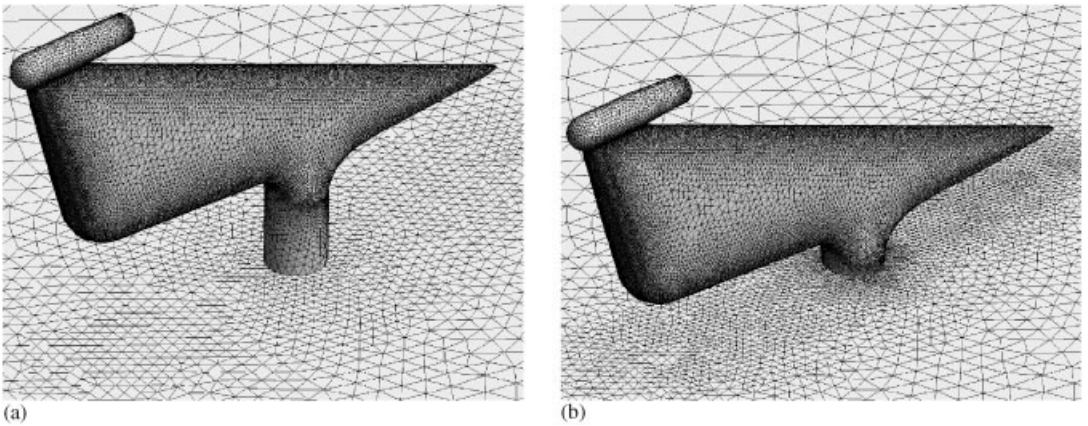

Figure 1. Influence of the five spacings on ground surface.

$$l = \frac{\text{Radius of circumsphere}}{\sqrt{\frac{5}{12}}} \tag{3}$$

Default threshold values found useful in practice are $\alpha_{\min} = 1°$, $\lambda = 5$ and $\kappa = 10$. In addition, elements having a non-positive volume are also marked to be remeshed.

In case the means to define the local mesh size are changed or the mesh has been deformed, also an analysis of the deviation from the prescribed spacing is applied. This is performed by detecting all elements larger or smaller than a user-defined relaxation factor σ[‡‡] times the desired local spacing computed from the average edge length of the element. A change of the spacing can be caused by, for example, the addition, modification or deletion of background grids or sources. By checking the local mesh spacing against the prescribed lengths, it can be guaranteed that the mesh always respects the initial specifications defined by the user. Particularly when surfaces come close to each other, the smaller spacing of one part should influence the other part (Figure 1).

In addition to the analysis of the element quality and the deviation from the prescribed mesh size, an analysis of the intersection of elements is performed. Figure 2 clearly demonstrates the necessity of such an investigation. Without an intersection check only the invalid and badly distorted elements at the moved boundary parts are marked to be remeshed which would result in a failure of the remeshing procedure. A brute-force approach testing each element against any other has a complexity of $\mathcal{O}(N^2)$ and is not practicable for larger problems. Thus, another algorithm is proposed for determining the elements intersecting each other efficiently:

1. Loop over all elements:
   (a) If no point of the element is moved: CONTINUE.
   (b) If the element is inverted or is at the boundary: Add the element to the set of elements to be tested.

[‡‡] In this work, a value of $\sigma = 1.9$ has been used.

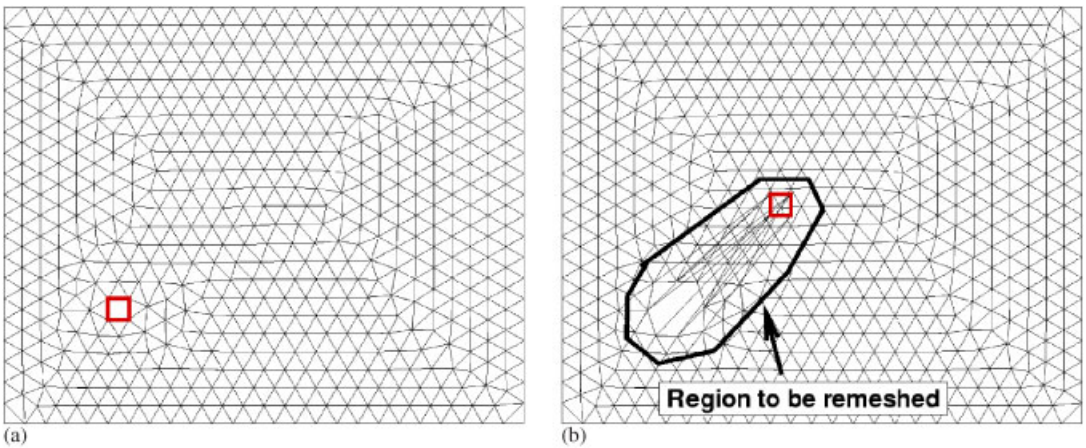

Figure 2. Region to be remeshed due to the movement of boundary parts.

2. While the set of elements to be tested is not empty:
   (a) Get the next element E to be tested.
   (b) Collect all neighbours N_f sharing a face with the current element.
   (c) Clear the set of intersection candidates (an intersection candidate is defined by a pair of elements).
   (d) Add all intersection candidates given by the pairs (neighbour N_f, element E) to the set of intersection candidates.
   (e) While the set of intersection candidates is not empty:
      i. Get the next intersection candidate (E_1, E_2).
      ii. If the pair of elements has already been tested: CONTINUE.
      iii. Test for intersections between the pair of elements.
      iv. If no intersection is found: CONTINUE.
      v. Insert the pair of elements in the set of candidates already processed to avoid a second check.
      vi. Mark both elements and their corresponding neighbours for remeshing.
      vii. For all neighbours $N_f^{E_1}$ sharing a face of the element E_1:
         A. If the pair of elements ($N_f^{E_1}$, E_1) has already been processed: CONTINUE.
         B. A new intersection candidate is found. Add the pair of elements to the set of intersection candidates.

This algorithm assumes a valid mesh (i.e. built from a conforming set of non-overlapping elements completely filling the space) based on the old coordinates. With this approach a minimum set of elements is determined which may have intersections with other elements based on the new coordinates, which are obtained after the mesh has been moved or deformed. Subsequently, all selected elements are tested for intersections with their neighbours sharing a face. By monitoring the already processed pairs of elements, multiple intersection tests of the same pair of elements can be avoided. This is implemented by means of a hash set

which supports an efficient look-up. If an intersection is found, the search is continued at the neighbour element until all intersection candidates are processed.

This sequential algorithm will not find intersections between elements in different domains of a distributed mesh since the search stops at the domain boundaries. Hence, additional steps have to be performed to enable the localization of elements intersecting each other also in a parallel environment.

For the parallel intersection algorithm proposed, the mesh is expected to be partitioned such that all domains have a one-cell overlap between each other. Then the parallel algorithm for the determination of intersecting elements within and across domains can be formulated as follows:

1. Determine the set of local elements to be tested for intersections and proceed as in step 1 of the sequential algorithm.
2. While the set of elements to be tested is not empty:
   (a) Proceed as in step 2(a)–2(e) of the sequential algorithm. Additionally, if an intersection candidate contains a ghost cell,[§§] add this pair of elements to the set of elements to be sent to the process owning this ghost cell.
3. While TRUE:
   (a) Determine the global number of intersection candidates to be exchanged from the local set of candidates to be sent.
   (b) If the global number of intersection candidates is zero: BREAK.
   (c) Send the own intersection candidates to the other domains and receive from them their intersection candidates.
   (d) Repeat step 2(a) with the received intersection candidates.

This algorithm computes iteratively the intersections between elements belonging to different domains by a parallel 'neighbour walking' approach. In each round, intersection candidates are exchanged and processed. If a domain boundary is reached, the intersection candidates are forwarded to the neighbour domain and the search has to be continued in the next round. After each round, a parallel synchronization point is reached enabling a consistent decision to be made whether to continue or to stop. This parallel algorithm will result in the same set of intersecting elements compared to the sequential version since the same intersection checks are performed.

After all elements to be remeshed have been selected, normally a post-processing of these regions is performed. In general, the selected elements will not form compact regions with a smooth outer boundary. They also may consist of narrow and constrained regions which may by simply reconnected in the remeshing stage. Therefore, to obtain compact regions with a smooth boundary and to allow the mesh generator more freedom where to place new internal points, the selected regions are smoothed and enlarged. This is performed by loops over the elements where each element sharing a node, an edge or a face of a marked element gets the same marker assigned. In this work, all neighbours sharing a node in combination with one–three smoothing loops have found to be a reasonable choice. An example of such a smoothing procedure is shown in Figure 3.

[§§] Ghost cells are replicated cells in the overlap area of parallel domains belonging to another domain.

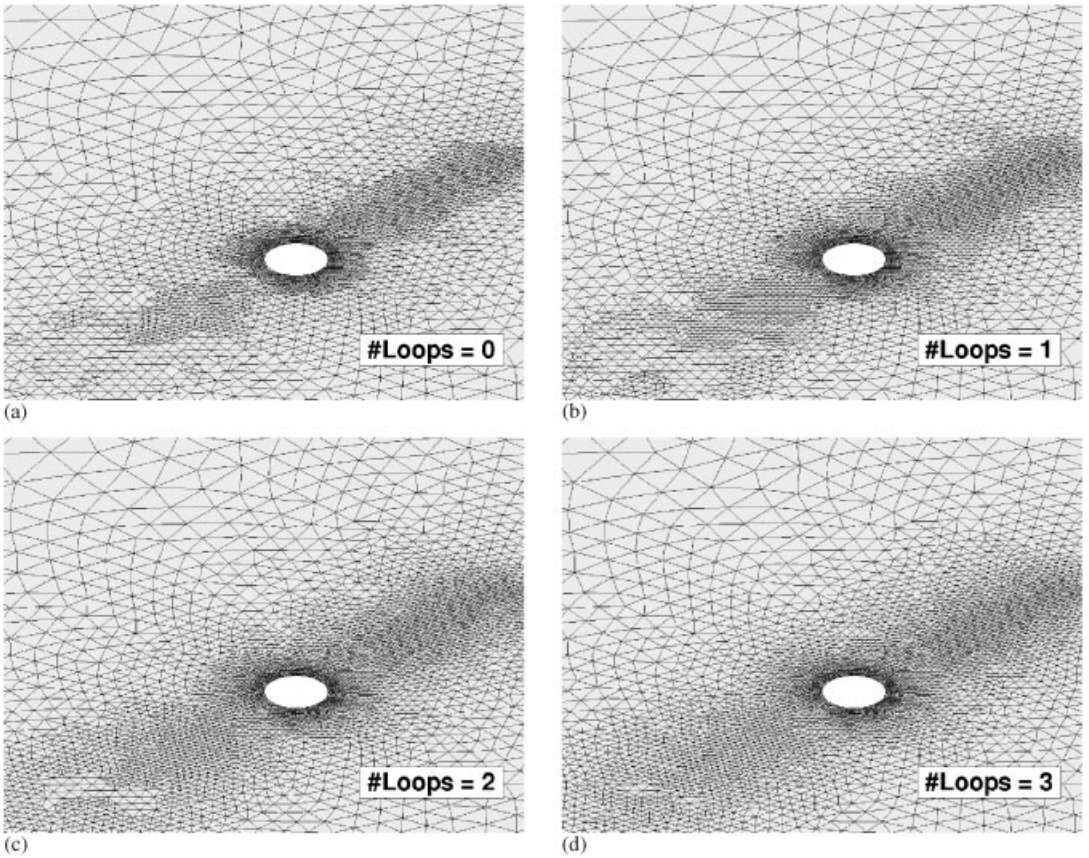

Figure 3. Influence of number of smoothing loops on remeshed surface.

4. PARALLEL SURFACE REMESHING

The surface remeshing algorithm [28] starts with the extraction of the selected regions from the given surface mesh. First all mesh edges lying on CAD curves are rediscretized by the following algorithm:

1. Find all mesh edges in the surface region to be remeshed which are linked to CAD edges and are not at the boundary of this region (Figure 4):
2. Group all collected mesh edges and sort them according to their CAD edge number.
3. Colour all mesh edges which are connected to each other and share the same CAD edge number.
4. Loop over all colours:
   (a) Determine the start and the end point of the connected sequence of mesh edges sharing the same colour.
   (b) Project the start and the end point onto the underlying parametric CAD curve to determine the parameter interval in which the curve has to be rediscretized.

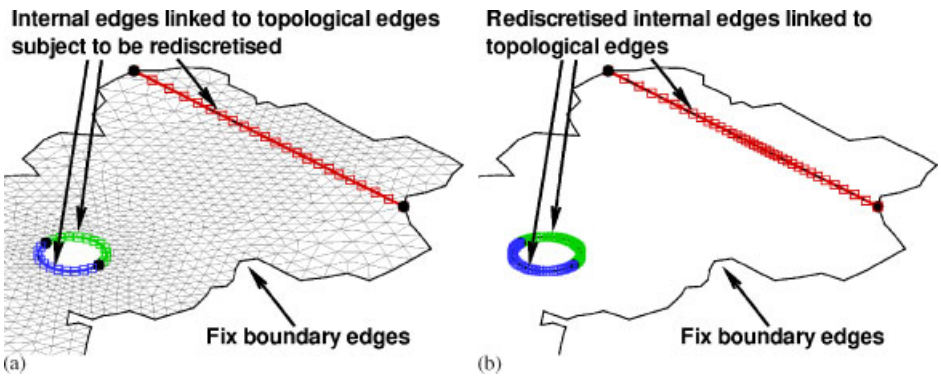

Figure 4. Internal mesh edges linked to topological edges which need to be replaced by mesh edges generated by the rediscretization of the underlying parametric CAD curve.

(c) Discretize the underlying parametric CAD curve from the start to the end point according to the specified mesh size by means of algorithm already used for the initial discretization of the curves [29].

The result of this curve rediscretization is shown in Figure 4. Then all rediscretized mesh edges linked to CAD edges and all mesh edges at the boundary of the extracted submesh are collected and used to form the initial fronts for the advancing front surface meshing. The correct orientation of these initial fronts is derived from the orientation of the underlying CAD surfaces relative to the surface mesh, which has been determined during the initial surface mesh generation. Subsequently, an advancing front triangulation of each CAD face is performed and the new mesh is merged with the old one.

In pseudo-code the algorithm can be formulated as follows:

```
geometry     = InitGeometry();
meshSizeSpec = InitMeshSizeSpecification();
surfaceMesh  = InitSurfaceMesh();
// --- analyse mesh ---
meshAnalyser.MarkRegionsToBeRemeshed(surfaceMesh);
// --- extract selected regions ---
subMesh = surfaceMesh.ExtractAndRemoveSelectedSubmesh();
// ===== remesh submesh =====
topoFacesSet = submesh.GetAllTopoFaceIDs();
/* determine orientation of surfaces relative to
   orientation of triangles */
DetermineSurfaceOrientations(geometry, topoFacesSet,
                             subMesh);
boundMesh = submesh.ExtractBoundaryEdges();
curveDiscretizer.Rediscretize(geometry, meshSizeSpec,
                              boundMesh);
// --- remesh holes ---
for(FI=topoFacesSet.begin(); FI!=topoFacesSet.end(); ++FI){
```



```
  advFront = initialFrontGenerator.InitFront(geometry, *FI,
                                             boundMesh);
  holeMesh = advFrontTriangulator.Triangulate(geometry,
                                              meshSizeSpec,
                                              advFront);
  meshEnhancer.Optimize(geometry, meshSizeSpec, holeMesh);
  surfaceMesh += holeMesh;
}
DoPostProcessing(surfaceMesh);
```

For the parallel surface remeshing two strategies have been implemented.

- The first strategy operates in parallel on the local domains and preserves the domain boundaries, hence, the sequential algorithms described above are applicable without any modifications. This is the approach used during the volume remeshing described in Section 5.
- In addition to the parallelization over the spatial domains, a parallelization over the different tasks is also possible. This can be achieved by parallelizing the loop over the topological edges to be rediscretized as well as the loop over the topological faces to be remeshed [30].

5. PARALLEL VOLUME REMESHING

The volume remeshing algorithm starts with the extraction of the selected regions from the given volume mesh. Then the missing submesh boundary faces are constructed. If also the remeshing of the surface boundary is enabled by the user, all parts of the extracted submesh boundary lying on the surface boundary of the mesh are remeshed by means of the procedures presented in Section 4. Then the volume holes are filled with elements by a Delaunay volume meshing algorithm [18] and the new submesh is merged with the unmodified parts of the mesh.

In pseudo-code the local volume remeshing algorithm can be formulated as follows:

```
geometry     = InitGeometry();
meshSizeSpec = InitMeshSizeSpecification();
volumeMesh   = InitVolumeMesh();
// --- analyse mesh ---
meshAnalyser.MarkRegionsToBeRemeshed(volumeMesh);
while(TRUE){
  // --- extract selected regions ---
  subMesh     = volumeMesh.ExtractSelectedSubmesh();
  surfaceMesh = subMesh.ExtractBoundary();
  if(remeshSurface){
    // --- remesh surface ---
    topoFacesSet = surfaceMesh.GetTopoFaceIDs();
    topoEdgesSet = surfaceMesh.GetTopoEdgeIDs();
    boundaryMesh = surfaceMesh.ExtractAndRemove(topoFacesSet,
```



```
                                                    topoEdgesSet);
    RemeshSurfaceMesh(geometry, meshSizeSpec, boundaryMesh);
    surfaceMesh += boundaryMesh;
  }
  // --- remesh volume ---
  holeMesh = DelaunayTriangulator.Triangulate(meshSizeSpec,
                                              surfaceMesh);
  meshOptimizer.Optimize(holeMesh);
  if(errorOccurred){
    if(enlargementCount < enlargementCountMax){
      meshAnalyser.EnlargeRegionsToBeRemeshed(volumeMesh);
      enlargementCount++;
      CONTINUE; // try again after region has been enlarged
    }
    else EXIT_WITH_ERROR;
  }
  else{
    volumeMesh.RemoveSelectedSubmesh();
    volumeMesh += holeMesh;
    BREAK; // remeshing succeeded
  }
}
```

In case of errors during the boundary recovery due to not recoverable configurations of hole boundary triangles, an automatic enlargement of the selected regions has been implemented. It has been found that this is an effective strategy in practice which enables the remeshing to succeed in the following attempts.

In Figure 5 an example is given which illustrates the different steps of the local volume remeshing algorithm. At first, the mesh size specification is modified by an additional point source. All elements not respecting the mesh size any more are determined which results in the selected region shown. After the extraction of this submesh, the boundary faces of the hole are determined and the submesh surface part lying on the CAD surface is remeshed. Subsequently, a tetrahedral volume mesh is generated inside this hole formed by the unmodified and the new surface boundary triangles. Finally, the remeshed hole is merged with the unmodified parts of the mesh. The change of the local element size due to the additional point source is shown in a cutting plane. There the desired transition from coarse to fine elements can be seen.

In pseudo-code the parallel local volume remeshing algorithm can be formulated as follows:

```
geometry     = InitGeometry();              // <-- replicated
meshSizeSpec = InitMeshSizeSpecification(); // <-- replicated
volumeMesh   = InitVolumeMesh();            // <-- distributed
// --- analyse mesh ---
meshAnalyser.MarkRegionsToBeRemeshed(volumeMesh);
while(TRUE){
  // --- constrained mesh re-partitioning ---
  volumeMesh.RepartitionMesh(ensureLocalSelectedRegions);
```



```
  // --- extract selected regions ---
  subMeshPerProc  = volumeMesh.ExtractSelectedLocalSubmesh();
  surfMeshPerProc = subMeshPerProc.ExtractBoundary();
  if(remeshSurface){
    // --- remesh surface ---
    topoFacesSet     = surfMeshPerProc.GetTopoFaceIDs();
    topoEdgesSet     = surfMeshPerProc.GetTopoEdgeIDs();
    boundMeshPerProc = surfMeshPerProc.ExtractAndRemove(topoFacesSet,
                                                        topoEdgesSet);
    RemeshSurfaceMesh(geometry, meshSizeSpec, boundMeshPerProc);
    surfMeshPerProc += boundMeshPerProc;
  }
  // --- remesh volume ---
  holeMeshPerProc = DelaunayTriangulator.Triangulate(meshSizeSpec,
                                                     surfMeshPerProc);
  meshOptimizer.Optimize(holeMeshPerProc);
  if(errorOccurred){
    if(enlargementCount < enlargementCountMax){
      meshAnalyser.EnlargeRegionsToBeRemeshed(volumeMesh);
      enlargementCount++;
      CONTINUE; // try again after region has been enlarged
    }
    else EXIT_WITH_ERROR;
  }
  else{
    volumeMesh.RemoveSelectedSubmesh();
    volumeMesh += holeMeshPerProc;
    BREAK; // remeshing succeeded
  }
}
```

In Figure 6, the main steps of this parallel local remeshing algorithm are illustrated based on an example where two point sources are added to the mesh size specification used for the generation of the initial mesh. At first all parts to be remeshed are determined in parallel. The selected regions form two distinct areas which are distributed across the parallel domains. After the re-partitioning, purely local regions are obtained which are also balanced across the domains. Then the local remeshing can be performed in each domain in parallel resulting in the locally modified grid shown.

Compared to a parallel volume meshing working on a distributed hole it will clearly be case dependent whether this approach performs better or not. In case of a single large region to be remeshed, problems may arise for the presented approach because only one processor will perform the local remeshing. This may lead to a significant memory and time consumption. Then a distributed meshing may be the preferable approach as it avoids memory bottlenecks and reduces the total time required. The other extremum are many small holes, for which the presented approach may be the most efficient solution. In this case, the amount of time required to re-partition the grid to obtain purely local holes is small compared to the meshing time.

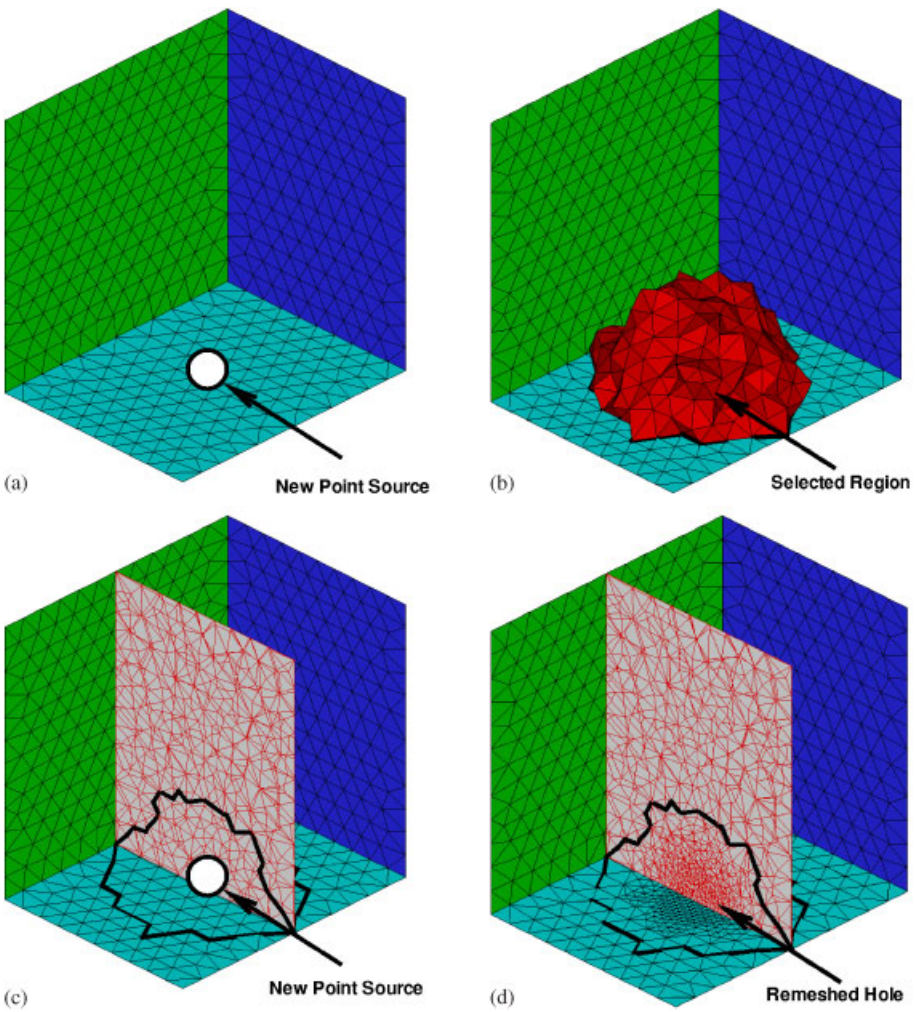

Figure 5. Example showing the different steps executed during the local volume remeshing.

Secondly, the meshing work is approximately balanced due to the distribution of the holes across all processes. And thirdly, no communication and synchronization is required during the local remeshing of each hole in parallel resulting in a fast parallel execution. Between these extreme cases, it will depend on the application in question and on the implementation itself which of the two approaches shows a faster run-time performance.

The re-partitioning of the distributed unstructured mesh across n domains is performed to obtain purely local regions to be remeshed and to balance the regions and therewith the meshing work across all processes. This is performed by the following algorithm:

1. Ensure that the mesh is partitioned such that all domains have a one-cell overlap between each other.

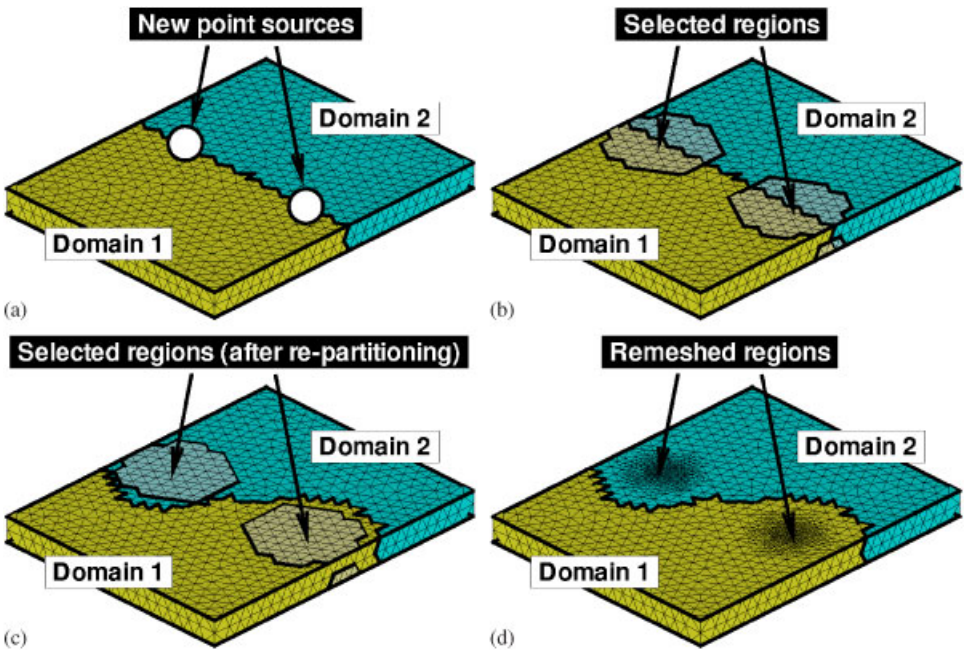

Figure 6. Example showing the main steps of the *parallel remeshing algorithm.*

2. Colour all regions in an advancing front manner to determine all elements belonging to the same region. This is performed in several rounds until no update needs to be made. In each round, the colours between the ghost elements and the corresponding original elements are exchanged to advance also across domain boundaries. This results in the set of independent regions R.
3. Switch the representation of the distributed mesh such that all domains have no element overlap any more but only a one-node overlap between each other.
4. Construct the distributed graph $G=(V,E)$ of the given mesh where the vertices $v \in V$ are identical to the global node numbers of the mesh nodes and the graph edges $e \in E$ are defined by the node-to-node connectivity. Assign a weight of one to each vertex.
5. Modify the graph G such that for each independent region $r \in R$ a vertex v_r is added to G. For each new vertex v_r determine the set of processes $P \subseteq 1,\ldots,n$ containing nodes of this region and add the vertex v_r to the process $p=\min_{k \in P}(k)$ with the smallest process number of the processes in P. Additionally, for each local vertex v contained in a region r add an edge $e=(v_r,v)$ from the corresponding region vertex to this local vertex to the set of edges E.
6. Agglomerate all nodes directly connected with a region node to this region node and accumulate the vertex weights.
7. Compute the destination process for each vertex of the agglomerated graph. To achieve a balance of both the number of nodes and the remeshing work later on, a multi-constraint approach is required. The first set of weights $w_v^{(1)}$ to be balanced are the vertex weights assigned and accumulated for each vertex as defined above. The second set of weights

is given by the function

$$w_v^{(2)} = \begin{cases} 1000 : v = v_r, r \in R \\ 1 : \text{otherwise} \end{cases} \quad (4)$$

By this definition of weights all region nodes are assigned a high weight to force them to be distributed across the processes by balancing the weights. In this work, the parallel PARMETIS library [31] is utilized to compute the new destination processes by means of the integrated multi-constraint graph partitioning scheme [32].

8. Propagate the destination process to each vertex agglomerated previously from the corresponding region vertices.
9. Determine the destination process for each element, face and edge, respectively, depending on their region affiliation and, if not contained in any region, depending on the destination processes of their nodes.
10. Re-distribute the nodes, elements, faces and edges accordingly.

By this constrained re-partitioning it can be assured that each region to be remeshed is contained completely in one process and that the meshing work is approximately balanced across the processes.

6. APPLICATIONS

A time-accurate CFD simulation of bodies in relative motion is presented where local remeshing had to be frequently applied to restore a valid grid throughout the computation. The simulation was performed by means of the EADS M *SimServer* simulation environment [33].

Table I. Reference conditions for the two flight cases.

	Case 1	Case 2
Mach number	0.962	1.055
Altitude (ft)	6332	10832
Angle of attack (deg)	0.46	−0.65
Dive angle (deg)	43	44

Table II. Breakdown of total time spent in the major simulation stages.

Task	Fraction of total time (%)
CFD solution	40
Mesh deformation	10.3
Volume hole enlargements	0.3
Volume mesh analysis	3.4
Volume remeshing	37
Re-partitioning	6.1
Motion application	$\sum$ 57.1
Others (I/O, ...)	2.9

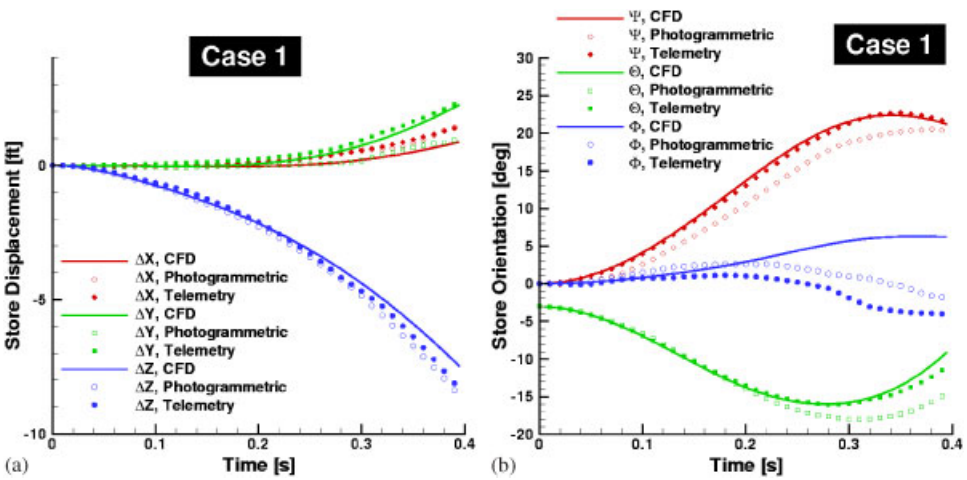

Figure 7. Comparison of the computed displacements and orientations against flight measurement data for *Case 1*.

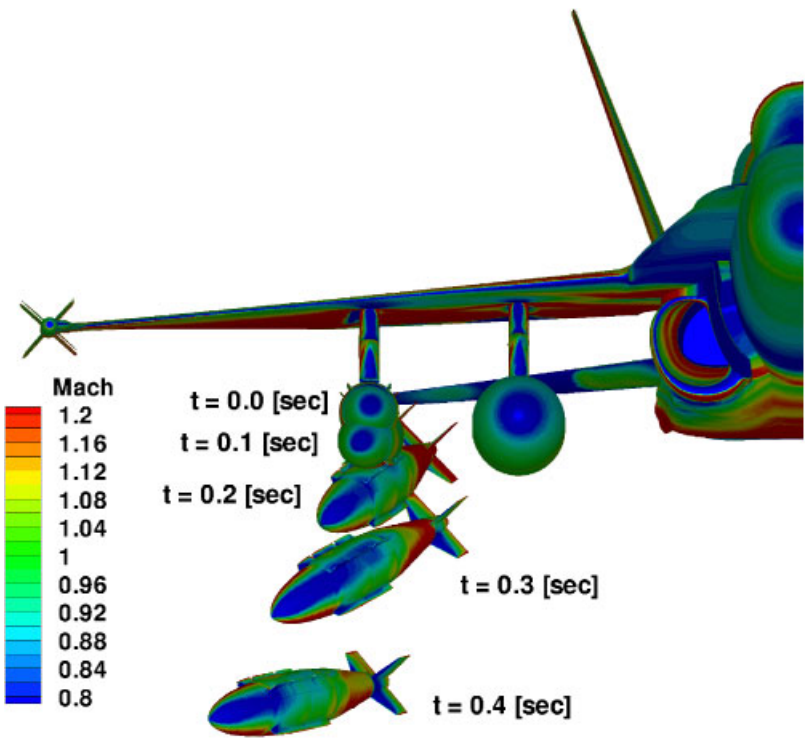

Figure 8. Contour plot of the computed Mach number distributions during the separation.

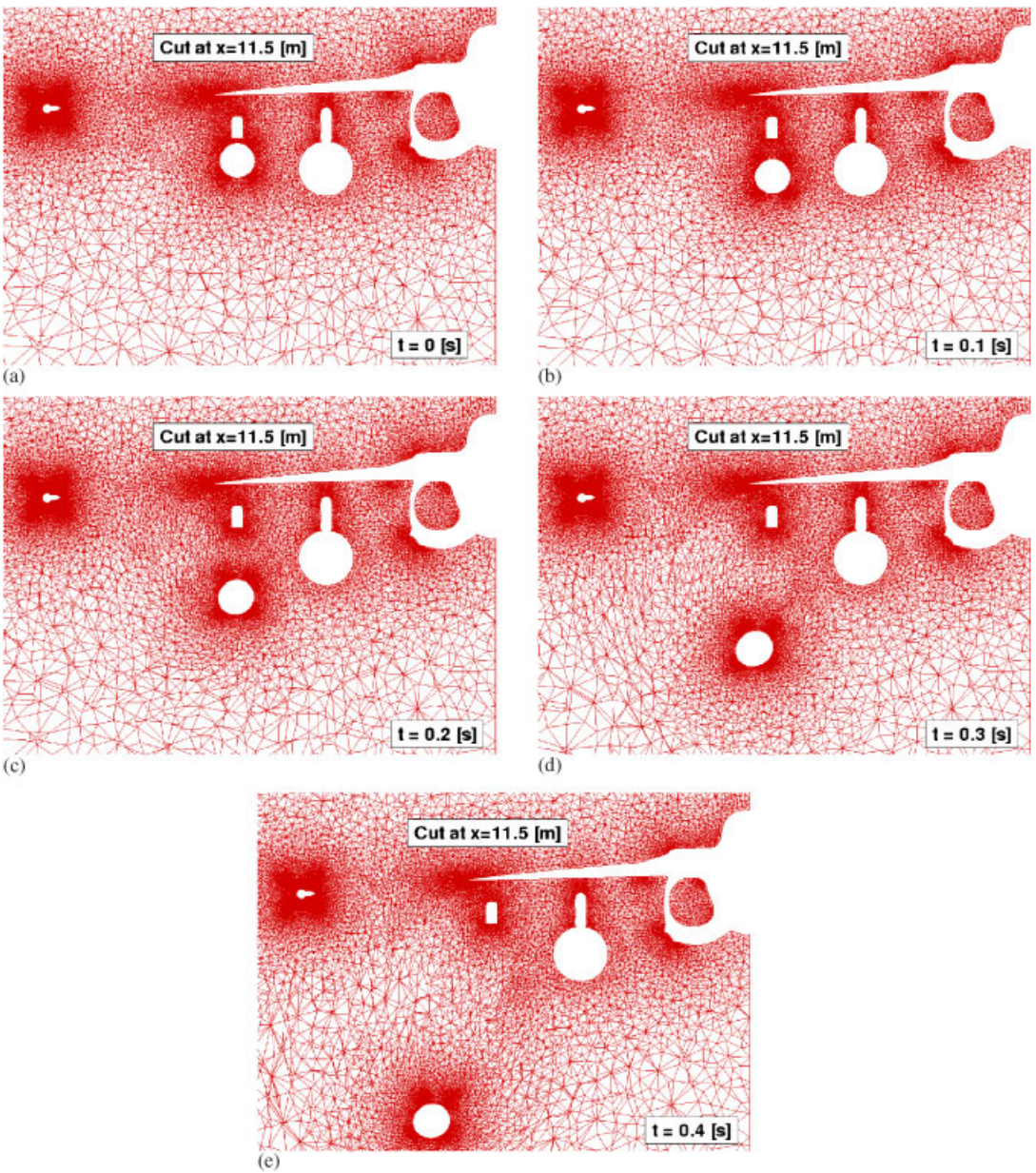

Figure 9. Cuts through the mesh at $x = 11.5$ m at different stages of the simulation.

It has been designed for the efficient parallel execution of multi-disciplinary simulations, whereby existing, validated, and highly optimised parallel CFD solvers are applied for the flow solution part. Into this environment the meshing algorithms described above has been integrated and closely coupled with the flow solver.

The example presented shows the release of a JDAM from a F/A-18C, which has been selected as a common test case in the Applied CFD Challenge II [17, 34]. Both release conditions are listed in Table I.

For *Case 1* the time-accurate simulation lasted 10.4 h on 24 Intel XEON processors running at 2.66 GHz, the parts required for the different tasks are summarized in Table II. About 150 local remeshings steps were performed within the 40 physical time steps simulated. After all remeshings the number of nodes and elements in the mesh increased by approximately 10% from ≈ 2.1 to ≈ 2.3 million nodes and from ≈ 12.1 to ≈ 13.4 million elements.

Similar timings and numbers had been obtained for *Case 2*.

In Figure 7, the computed trajectory is plotted against the flight test measurements for *Case 1*. It can be observed that the two separation predictions compare well with flight test data. In addition, they have been performed fully in parallel on a distributed memory platform. Due to the efficiently utilized parallel computing power, the simulations were performed also in an acceptable amount of time. This clearly highlights the powerful capabilities of the parallel local remeshing approach developed in this work (Figures 8 and 9).

ACKNOWLEDGEMENTS

We gratefully acknowledge the support from EADS Military Aircraft for performing this work. In addition, we would like to express our gratitude to A. Cenko for the support regarding the F/A-18C JDAM test case.

REFERENCES

1. McRae DS, Laflin KR. Dynamic grid adaptation and grid quality. In *Handbook of Grid Generation*, Thompson JF, Soni BK, Weatherill NP (eds), Chapter 34. CRC Press LLC: Boca Raton, FL, 1999.
2. Zegeling PA. Moving grid techniques. In *Handbook of Grid Generation*, Thompson JF, Soni BK, Weatherill NP (eds), Chapter 37. CRC Press LLC: Boca Raton, FL, 1999.
3. Löhner R. Adaptive h-refinement on 3-d unstructured grids for transient problems. *AIAA Conference*, *AIAA Paper 89-0365*, 1989.
4. Rivara M-C. A 3-d refinement algorithm suitable for adaptive and multigrid techniques. *Communications in Applied Numerical Methods* 1992; **8**:281–290.
5. Bänsch E. Local mesh refinement in 2 and 3 dimensions. *Impact of Computing in Science and Engineering* 1991; **3**:181–191.
6. Bey J. Tetrahedral grid refinement. *Computing* 1995; **55**(4):355–378.
7. Mavriplis DJ. Adaptive meshing techniques for viscous flow calculations on mixed element unstructured meshes. *Report 97-20*, ICASE, Hampton, VA, May 1997.
8. Baker TJ. Adaptive modifications for time evolving meshes. *Journal of Materials Science* 2003; **38**(20): 4175–4182.
9. Baum JD, Löhner R, Marquette TJ, Luo H. Numerical simulation of aircraft canopy trajectory. *28th AIAA Fluid Dynamics Conference*, *AIAA Paper 97-1885*, Snowmass Village, CO, 1997.
10. Hassan O, Probert EJ, Morgan K. Unstructured mesh procedures for the simulation of three-dimensional transient compressible inviscid flows with moving boundary components. *International Journal for Numerical Methods in Fluids* 1998; **27**:41–55.
11. Hassan O, Probert EJ, Morgan K, Weatherill NP. Unsteady flow simulation using unstructured meshes. *Computer Methods in Applied Mechanics and Engineering* 2000; **189**:1247–1275.
12. Löhner R. *Applied CFD Techniques*: *An Introduction Based on Finite Element Methods*. Wiley: New York, 2001.
13. Sørensen KA. A multigrid accelerated procedure for the solution of compressible fluid flows on unstructured hybrid meshes. *Ph.D. Thesis*, University of Wales Swansea, December 2001. C/Ph/251.
14. Peraire J, Vahdati M, Morgan K, Zienkiewicz OC. Adaptive remeshing for compressible flow computations. *Journal of Computational Physics* 1987; **72**:449–466.
15. Peraire J, Morgan K, Peiró J. Adaptive Remeshing in 3-D. *Journal of Computational Physics* 1992; **103**: 269–285.

16. Löhner R. Adaptive remeshing for transient problems. *Computer Methods in Applied Mechanics and Engineering* 1989; **75**:195–214.
17. Cenko A, Gowanlock D, Lutton M, Tutty M. F/A-18C/JDAM applied computational fluid dynamics challenge II results. *38th AIAA Aerospace Sciences Meeting*, *AIAA Paper 2000-0795*, Reno, NV, January 2000.
18. Weatherill NP, Hassan O. Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints. *International Journal for Numerical Methods in Engineering* 1994; **37**: 2005–2039.
19. Shostko A, Löhner R. Three-dimensional parallel unstructured grid generation. *International Journal for Numerical Methods in Engineering* 1995; **38**:905–925.
20. Galtier J, George P-L. Prepartitioning as a way to mesh subdomains in parallel. *5th International Meshing Roundtable*. Sandia National Laboratories, 1996; 107–122.
21. Wu P, Houstis EN. Parallel adaptive mesh generation and decomposition. *Engineering with Computers* 1996; **12**:155–167.
22. de Cougny HL, Shephard MS, Özturan C. Parallel three-dimensional mesh generation on distributed memory MIMD computers. *Engineering with Computers* 1996; **12**:94–106.
23. Chew LP, Chrisochoides N, Sukup F. Parallel constrained Delaunay meshing. *Trends in Unstructured Mesh Generation* 1997; **220**:89–96.
24. Okunsanya T, Peraire J. 3D parallel unstructured mesh generation. *Trends in Unstructured Mesh Generation* 1997; **220**:109–115.
25. Said R, Weatherill NP, Morgan K, Verhoeven NA. Distributed parallel Delaunay mesh generation. *Computer Methods in Applied Mechanics and Engineering* 1999; **177**:109–125.
26. Larwood BG, Weatherill NP, Hassan O, Morgan K. Domain decomposition approach for parallel unstructured mesh generation. *International Journal for Numerical Methods in Engineering* 2003; **58**:177–188.
27. Chrisochoides N, Nave D. Parallel Delaunay mesh generation kernel. *International Journal for Numerical Methods in Engineering* 2003; **58**:161–176.
28. Tremel U, Deister F, Hassan O, Weatherill NP. Automatic unstructured surface mesh generation for complex configurations. *International Journal for Numerical Methods in Fluids* 2004; **45**:341–364.
29. Peiró J. Surface grid generation. In *Handbook of Grid Generation*, Thompson JF, Soni BK, Weatherill NP (eds). Chapter 19, CRC Press LLC: Boca Raton, FL, 1999.
30. Tremel U, Deister F, Hassan O, Weatherill NP. Parallel generation of unstructured surface grids. *Engineering with Computers* 2004. Special issue devoted to selected papers from the 12th IMR, in press.
31. Karypis G, Schloegel K, Kumar V. *ParMetis—Parallel Graph Partitioning and Sparse Matrix Ordering Library*, Version 3.2. Department of Computer Science and Engineering, University of Minnesota, 2003.
32. Schloegel K, Karypis G, Kunar V. Parallel multilevel algorithms for multi-constraint graph partitioning. *Technical Report TR 99-031*, Army HPC Research Center, Department of Computer Science and Engineering, University of Minnesota, 1999.
33. Tremel U, Deister F, Sørensen KA, Rieger H, Weatherill NP. An object-oriented parallel multidisciplinary simulation system—the SimServer. In *Parallel Computing*: *Software Technology*, *Algorithms*, *Architectures and Applications*, *Advances in Parallel Computing*, Joubert GR, Nagel WE, Peters FJ, Walter WV (eds). Elsevier: Amsterdam, 2004; 331–338.
34. Sickles WL, Denny AG, Nichols RH. Time-accurate CFD predictions of the JDAM separation from an F-18C aircraft. *38th AIAA Aerospace Sciences Meeting*, *AIAA Paper 2000-0796*, Reno, NV, January 2000.

