INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids 2004; **45**:341–364 (DOI: 10.1002/fld.671)

Automatic unstructured surface mesh generation for complex configurations

Udo Tremel[1], Frank Deister[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

In this paper a new object-oriented (OO) approach is presented for automatic parallel advancing front based surface mesh generation and adaptive remeshing for complex configurations. Based on the ST++-system the advantages of the OO design and implementation compared to the traditional structural approach are described. Algorithmic enhancements to the advancing front method are explained, enabling a robust NURBS based triangulation process directly on B-rep CAD data. The message passing (MPI) parallelization strategy together with the achievable performance improvements are demonstrated. With the outlined parallel geometry analysis/rasterization a powerful method is described to derive automatically a well suited mesh size specification without any user-interaction from scratch. The application of this method to a complex 'real world' example finishes this paper. Copyright © 2004 John Wiley & Sons, Ltd.

1. INTRODUCTION

Surface mesh generation based on B-rep CAD data is an important task in the aerospace industry working with numerical (CFD) simulations. Especially for complex configurations consisting of several thousands of CAD patches the generation of a surface grid is a time-consuming task when the CAD geometry has to be cleaned manually again and again due to failures in the meshing procedure.

Therefore a robust and fast surface triangulation method capable of handling non-perfect CAD data would be extremely helpful.

2. OBJECT-ORIENTED APPROACH

In this paper a new object-oriented (OO) approach to surface meshing is presented. It has been realized in the ST++-system, currently developed at EADS M and based on the existing

[*]Correspondence to: O. Hassan, Civil Computational Engineering Centre, School of Engineering, University of Wales Swansea, Swansea SA2 8PP, U.K.
[†]E-mail: o.hassan@swansea.ac.uk

Received 13 January 2003
Revised 3 October 2003

Copyright © 2004 John Wiley & Sons, Ltd.

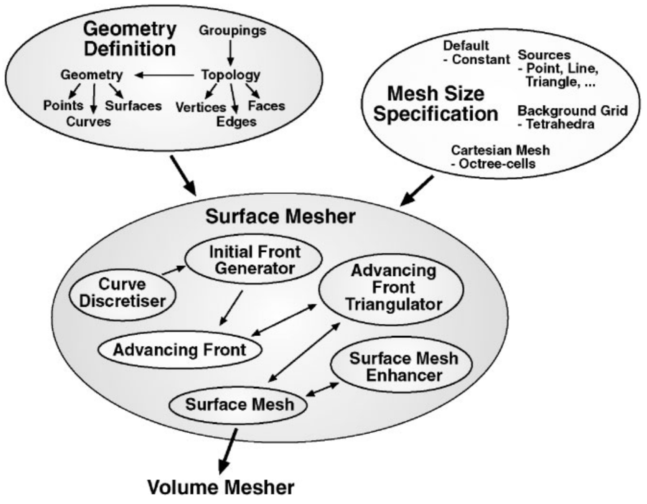

Figure 1. General design of the system.

FLITE–ST surface mesh generator from the Civil Engineering Department of the University of Wales, Swansea (UWS) [1–3]. The OO-methodology has been choosen because the OO concepts of polymorphism, inheritance and encapsulation [4, 5] support the building of large and complex software systems.

Compared to the procedural programming in, for example, Fortran77 or C normally also better readable, maintainable and extensible code results through the powerful features supported in OO languages such as strongly typed interfaces, templates, design patterns, etc. [6, 7]. Arguments of using C++ as implementation language are the rich OO features the language offers[‡] together with the downward compatibility to C which enables the integration and reuse of already available and validated C and Fortran77/90 routines.

The general design of the ST++-system is shown in Figure 1. It consists of three main components: the geometry definition, the mesh size specification and the surface mesher itself. To generate a surface mesh the following main steps are executed:

1. Initialize the geometry definition from CAD data.[§]
2. If only parts of the geometry are to be meshed, extract the selected subgeometry.
3. Initialize the mesh size specification.
4. Perform the advancing front triangulation.
5. Enhance the surface mesh.
6. Export/prepare the surface mesh for the volume mesh generation.

The geometry definition object is illustrated in detail in Figure 2. It encapsulates all the geometrical and topological entities imported from the CAD database. Supported import for-

[‡]Strong type checks, single and multiple inheritance, templates, abstract classes and interfaces, streams, exceptions, the standard template library (STL), ...

[§]Given in the form of a boundary representation (B-Rep) structure, i.e. with geometrical and topological types of data.

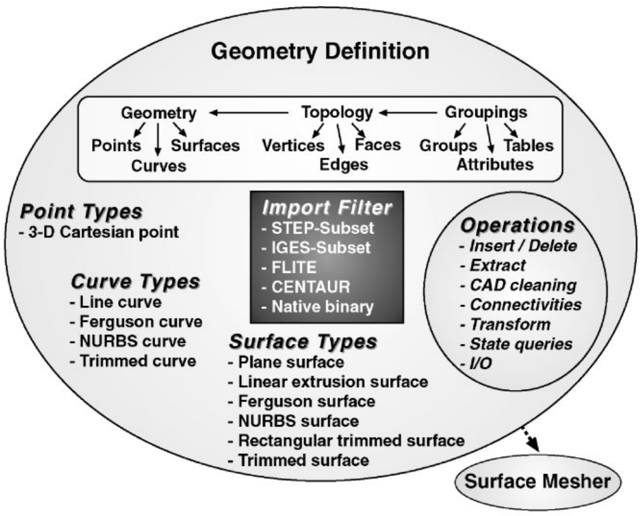

Figure 2. Layout of the geometry definition.

mats are the industrial relevant data exchange formats NASA-IGES [8], STEP [9] and the traditional FLITE format [1]. Various parametric curve and surface types represented by spline composite curves and tensor-product surfaces such as Ferguson [10], Bezier [11] or NURBS [12] are available. Important is that all specific point, curve and surface objects are derived from one abstract interface respectively so that all different types share a common interface that algorithms can work with. In this way it is possible to implement an advancing front triangulation algorithm independent of the underlying specific mathematical curve or surface definition. Compared to 'good old' Fortran77-code this is an enormous advantage because the developer need not take care of the type of curve (surface) given each time an operation on a curve (surface) is executed. Only the common interface for this specific curve (surface) type has to be implemented. Further, the integration of additional curve and surface types is supported transparently to codes only working with the abstract interface. For example, for curves, a part of the interface looks like:

- `Evaluate(doubleT u, doubleT *outXYZ, doubleT *outTangent=0),`
- `CalculateProjection(const doubleT *xyz, doubleT &outU),`
- `CalculateArcLength(doubleT u1, doubleT u2, doubleT &outL).`

The second major component of the ST++-system is the mesh size specification object shown in Figure 3. This enables the control of the spatial distribution of the nodes and the size and the shape of the elements to be generated. By means of background grids and sources the local mesh size together with certain stretching directions is defined for each spatial location. Starting from a constant mesh size all over the domain the mesh size can be modified locally by an optional tetrahedral background grid containing local sizes at each node together with stretching directions. Via superimposed (optional) sources the mesh size can be locally refined. Several types of sources are available (point, line, triangle, ...) to support a flexible selection and application depending on the geometrical demands. To allow

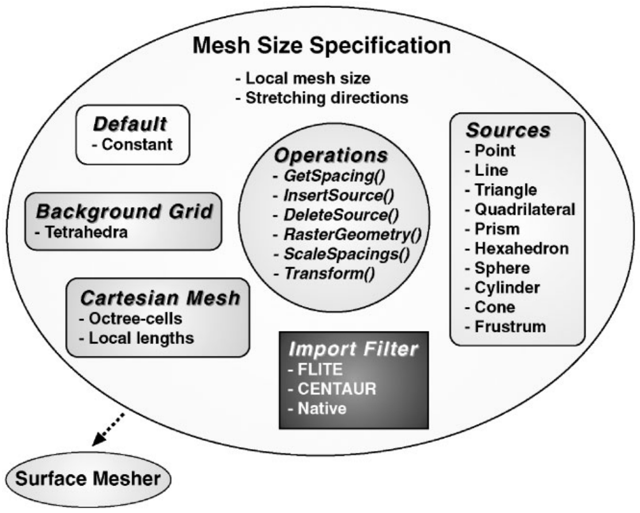

Figure 3. Ingredients of the mesh size specification.

an easy extension towards other source types an abstract source interface has been used from which all kinds of sources have been derived. To add other source types only the following lean interface has to be implemented:

- `GetMinSpacing(doubleT &outMinSpacing).`
- `GetSpacing(const doubleT *xyz, doubleT &outSpacing).`

Additionally a Cartesian background mesh can be used to define the mesh size completely or to refine the mesh size further at certain locations. The automatic determination of a well suited Cartesian background mesh is presented in Section 6. This approach differs from the octree based approaches presented in [13, 14], because the sizes of the Cartesian cells are largely independent from the local element sizes calculated. Only the scalar quantities defined in each Cartesian cell are used to derive the mesh size at a certain point in space. Experience has shown that the Cartesian cell size is generally two to eight times larger than the calculated lengths, hence, the Cartesian meshes created are much smaller than those grids generated in References [13, 14].

Important is that all components of the mesh size specification objects can be modified dynamically which is a preliminary requirement for a dynamic mesh modification process. Certain adaptation criteria will cause local changes of the mesh size which can then be used to adapt the surface grid accordingly.

The third component of the ST++-system is the advancing front surface mesh generator itself, illustrated in Figure 4. Inside this object the different steps of the meshing procedure are divided across several other objects. The mesh generator itself supervises and controls the execution and the exchange of data only. The different steps of the advancing front algorithm [2, 3, 15–17] are encapsulated in the curve discretizer, the initial front generator and the advancing front triangulator object. For all topological faces to be meshed, the curve discretizer discretizes all the topological edges connected to each face (but each edge only

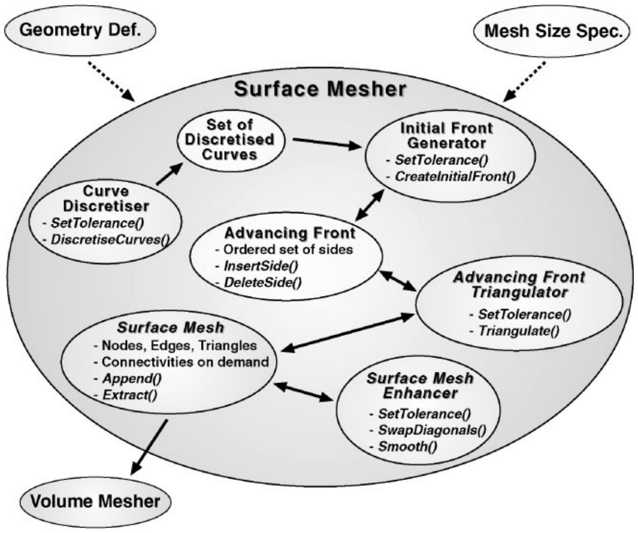

Figure 4. Components of the surface mesh generator.

once). This set of straight sides is given to the initial front generator object which builds up the initial advancing front. This starting front is then given to the triangulator together with a surface mesh object. When no sides are left in the front the triangulation is completed and the surface mesh is transferred to the mesh enhancer for optimization. After optional preparation steps (removal of duplicate nodes and edges, correcting the orientation of the facets, checks, ...) the merged surface mesh comprises the starting point for the volume triangulation.

3. ALGORITHMIC ENHANCEMENTS

Generally the advancing front triangulation algorithm can be formulated as follows (written in pseudo-code):

```
while(!advancingFront.Empty( )){
  side        = advancingFront.GetSide(sideSelectionStrategy);
  idealPoint  = CalculateIdealPoint(side);
  candidates  = DetermineCandidatePoints(advancingFront);
  newTriangle = FormTriangle(side, idealPoint, candidates);
  advancingFront.Update(side, newTriangle);
}
```

During the different steps of this algorithm several alternative ways are available to reach the desired goal.

Firstly one has to select a side from the front. Normally the sides are organized in a heap structure and the shortest side at the top of the heap is always taken. This is a reasonable and stable strategy to avoid crossing fronts. In the vicinity of large and small sides the smaller ones are taken first preventing large triangles, possibly crossing the front. An additional relevant

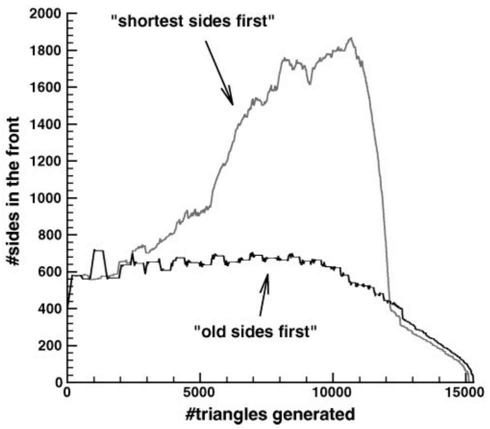

Figure 5. Effects of the different side selection strategies.

parameter is the age of the sides in the front. When there is a large number of sides with nearly the same length, this '*shortest sides first*'-strategy tends to produce outgrowths as can be seen in Figure 6. The proposed improved strategy takes additionally the age of the sides into account, so that, if sides have *nearly* the same length, the older side is privileged. In the real code this has been integrated by means of a discretization process of the length of the sides together with an age count. In the implementation the sides are now strictly sorted firstly by their discretized length and, if the length is equal, then by their age. The effects of this '*old sides first*'-strategy¶ are clearly visible in Figure 6. Especially for large areas, the improved strategy prioritises the old sides so that the front is closed more homogeneously. The real benefit gained from this '*old sides first*'-strategy is the reduction of the number of sides in the front during the entire advancing front algorithm because the boundary of the area to be meshed is kept as small as possible. In Figure 5, the history of the number of sides in the front is plotted during the triangulation process for the meshes shown in Figure 6. Because the advancing front algorithm scales nearly linearly with the number of sides in the front, this results in a noticeable total runtime reduction. For the example shown, the '*old sides first*'-strategy needs only 75% of the time needed for the '*shortest sides first*'-strategy.

The next critical step is the calculation of the 'ideal point', i.e. the computation of the spatial location of the point forming with the current side of the front the optimal triangle in size and shape according to the local mesh size given. The standard method is to calculate an initial guess and then to perform an iterative procedure to determine the correct position.‖

The initial guess is normally computed in the 2-D parametric space as the point on the perpendicular bisector of the side with length l_{side} at a distance equal to l_{side} away from the side midpoint. This point is then used as the starting point for a Newton method to solve the non-linear system of equations iteratively. Typically for a Newton method, the convergence is highly dependent on a good starting point. Assuming that the mapping from the 2-D parameter

¶More consistently it should be called '*shortest sides first (but when the lengths are nearly equal prefer the older side)*'-strategy, please allow us the shortcut and keep the intended meaning in mind.

‖For details see [2, 3].

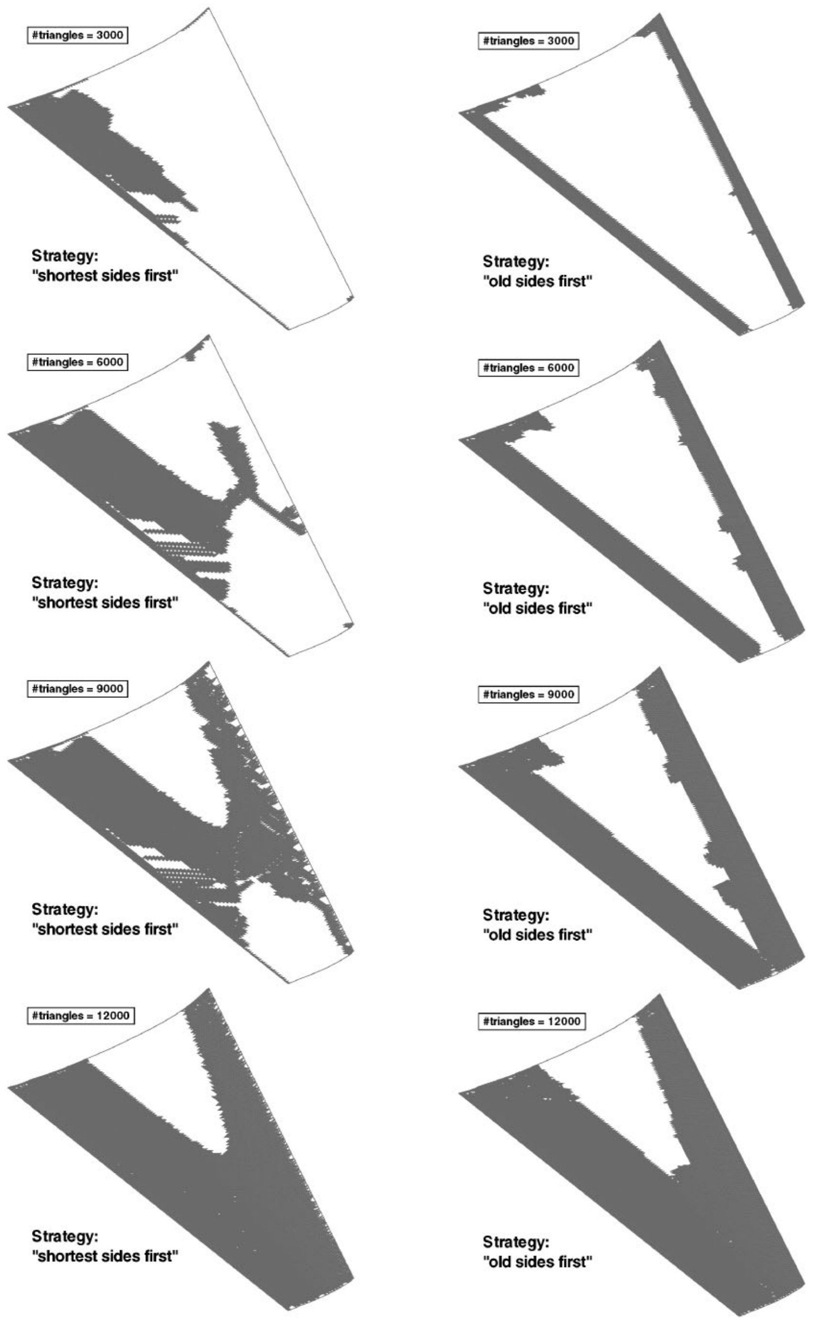

Figure 6. Comparison of both side selection strategies.

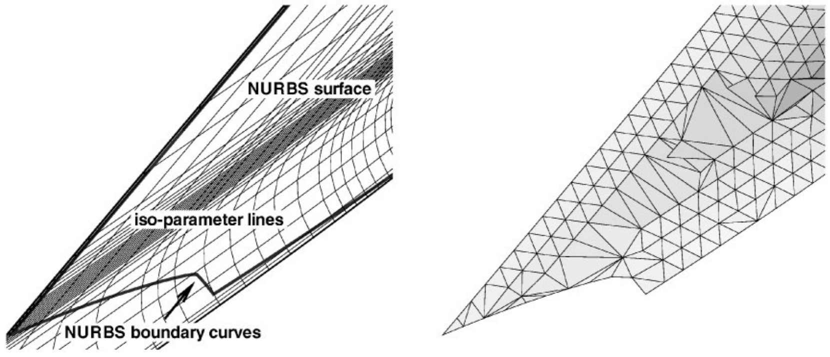

Figure 7. Parameter space and invalid mesh due to calculation errors of the 'ideal point'.

space into the 3-D physical space does not introduce significant anisotropy the starting point is in most cases quite close to the 'ideal point', and the iterative Newton method converges in a few steps. Therefore, this way of determining the 'ideal point' is highly efficient in terms of CPU-time and memory requirements. However, native unprepared CAD surfaces, especially those NURBS imported directly from the CAD system via IGES or STEP, do not necessarily fulfill the requirements to guarantee a nearly isotropic mapping between 2-D and 3-D space. Stretchings up to a ratio of 500 or more are not unusual. But this causes the standard procedure of calculating the 'ideal point' to fail completely. If this happens in the domain to be meshed, and no other points are available to form a new triangle, the result of the diverged Newton method will be used, generating an invalid mesh. This is illustrated in Figure 7.

Therefore an improved 'ideal point'-calculation method has been used in the ST++-system, consisting of two major stages. During the first stage, the midpoint of the side is iteratively calculated. In the second stage, this midpoint and modified parametric side normal vectors are used to calculate an initial guess good enough to reach convergence of the Newton method. In detail the improved algorithm can be formulated as follows:

1. Calculate the 2-D parametric coordinates $(u,v)_{\mathrm{mid}}$ of the 3-D midpoint $(x,y,z)_{\mathrm{mid}}$ of the side projected onto the surface. This can be achieved by trying to minimize the length difference of the two halves of the side in normalized physical space, e.g. again via an iterative Newton method.
2. When the midpoint has been determined up to a certain tolerance the initial guess $(u,v)^{(n)}_{\mathrm{new}}$ is computed as $(u,v)^{(n)}_{\mathrm{new}} = (u,v)_{\mathrm{mid}} + \alpha^{(n)}(u,v)_{\mathrm{normal}}$. α is used here as a relaxation factor reduced each time the Newton method has to be restarted again due to divergence. Depending on the change of the relaxation parameter the maximum number of restarts allowed has to be adjusted. Typically three to five restarts are applied in practice where the relaxation is reduced each time by a factor of 0.1 to 0.5. When convergence cannot be achieved, a monitoring and use of the best point reached during all steps is heuristically good enough to continue with the next side. One can expect that the enhancements later on improve these areas. Concerning $(u,v)_{\mathrm{normal}}$ care has to be taken to first transform the side vector into a normalized 2-D space where locally the anisotropy in the mapping into 3-D space is removed, before rotating the vector. The normal vector is then mapped

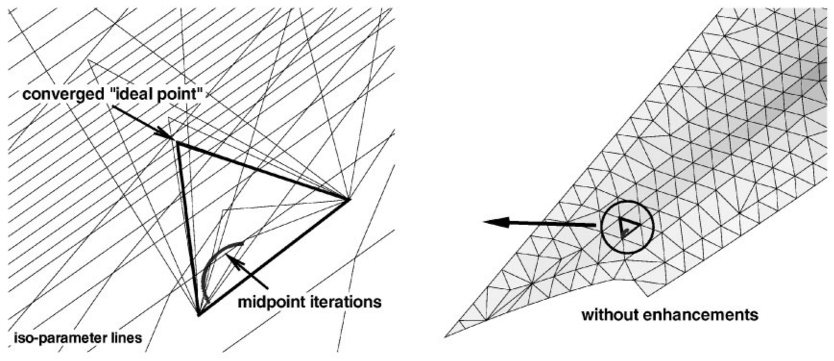

Figure 8. Convergence history of the iterative methods.

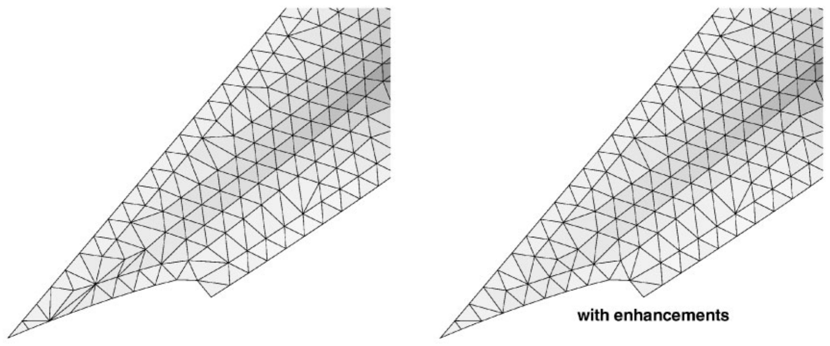

Figure 9. Valid mesh due to improved ‘ideal point’ calculations.

back into the original parametric space. Summerized the normal vector is calculated as

$$(u, v)_{\text{normal}} = TR(T^{-1}(\Delta_u, \Delta_v)) \text{ with } T = \begin{vmatrix} \mathrm{d}u/\mathrm{d}x & \mathrm{d}u/\mathrm{d}y \\ \mathrm{d}v/\mathrm{d}x & \mathrm{d}v/\mathrm{d}y \end{vmatrix} \text{ and } R = \begin{vmatrix} 0 & -1 \\ 1 & 0 \end{vmatrix}.$$

To illustrate the procedure a typical convergence history of the intermediate results during the iterations is given in Figure 8.

The application of this improved approach to the same NURBS surface definition used for Figure 7 is demonstrated in Figure 9. It is clearly visible that now a valid mesh can be obtained by the advancing front method due to the improved ‘ideal point’-calculations.

In the very rare case of a divergence of all attempts the calculation of the ‘ideal point’ is performed in 3-D space by placing the point on the perpendicular bisector of the side normal to the side and normal to the surface normal at the midpoint of the side. This point is then projected back onto the surface and is used as the ‘ideal point’. Although this alternative procedure is more expensive it causes only negligible overhead due to the adaptive usage on demand. But concerning the robustness of the meshing procedure it is an important ingredient preventing the triangulation to fail.

Another aspect concerning robustness are the face-side intersection checks to be applied. When the mappings between 2-D parameter and 3-D physical space are highly anisotropic the standard 2-D intersection tests are sometimes not effective any more to detect crossing fronts. Due to the iterative procedures applied to determine a new point, a few sides crossing in 3-D space do not cross in 2-D space and *vice versa*. Hence, real 3-D intersection checks between the sides of the front and the new triangle to be generated have to be applied when parametric distortions are detected. To account for tolerance problems the intersection check is expanded to check also if sides of the front are too close to the new face causing the face to be rejected. Here a minimum distance of ten to hundred times the CAD tolerance has proven to be a useful measure. To minimize the overhead in the intersection calculations only sides in the vicinity of the new sides to be created have to be used. This search of close sides is performed efficiently by means of an alternating digital tree (ADT) containing the sides in the front.

Experience has shown that the 3-D intersection checks improved the robustness up to a production level where the triangulation procedure succeeds also for badly parameterized CAD data.

4. ADAPTIVE SURFACE REMESHING

Once a surface mesh has been created and used normally the grid has to be adapted locally, depending on various requirements such as refinements improving the spatial resolution, coarsenings reducing the number of triangles, etc. These modifications are important for stationary cases but even more important for transient calculations where a continuously changing state has to be managed.

For mesh adaptations by means of the ST++-system, the intended change of the spatial resolution has to be introduced via the mesh size specification object. This can be reached in various ways:

- Insertion/modification/deletion of sources.
- Modification of the tetrahedral background grid.
- Modification of the Cartesian background grid.

Once the new mesh size is defined, two options are available. Firstly, the mesh generation could be started from scratch again neglecting all the work invested during the last surface mesh generation. This option should be selected only if the mesh size has been modified in the majority of the domain. However, if changes have occurred only locally, keeping most of the previous mesh and remeshing only those parts not respecting the mesh size is the far more efficient strategy. Therefore, a local remeshing facility has been incorporated into the ST++-system, as illustrated in Figure 10. The local remeshing procedure starts with the determination of all elements violating the mesh size, i.e. all triangles with edges larger or smaller than the specified mesh size, times a certain tolerance factor, are flagged. After a few smoothing cycles of the markers, the selected parts of the mesh are extracted and removed from the mesh. From the extracted submesh all referenced surfaces are collected. For these surfaces, the orientation of the surface relative to the triangulation is determined to ensure the correct orientation of the advancing fronts later on. Then, all the boundary edges of the submesh are extracted and processed, so that for each topological face referenced all boundary

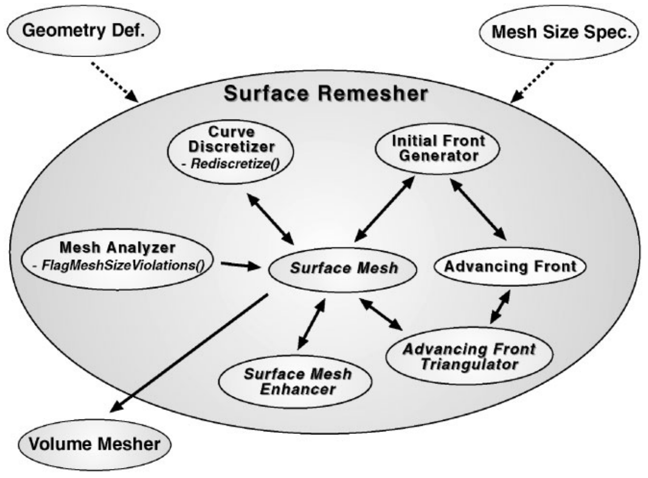

Figure 10. Objects involved during a local remeshing.

edges are known. After all edges lying on curves have been re-discretized according to the new mesh size, the new boundary mesh containing all the boundary edges is available. These edges are used for the creation of the initial fronts. When the advancing front triangulations together with the enhancements have been performed for each topological face the new mesh is merged with the old one. The resulting surface mesh can then be prepared again for other tasks such as visualisation, volume (re-)meshing, etc. In pseudo-code the algorithm can be formulated as follows:

```
geometry     = InitGeometry( );              // <-- old
meshSizeSpec = InitMeshSizeSpecification( ); // <-- modified
surfaceMesh  = InitSurfaceMesh( );           // <-- old
// --- analyse mesh ---
meshAnalyser.FlagMeshSizeViolations(surfaceMesh);
// --- extract selected elements ---
surfaceMesh.ExtractAndRemoveSelectedSubmesh(submesh);
// ===== remesh submesh =====
topoFacesSet = submesh.GetTopoFaceIDs( );
/* 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);
```

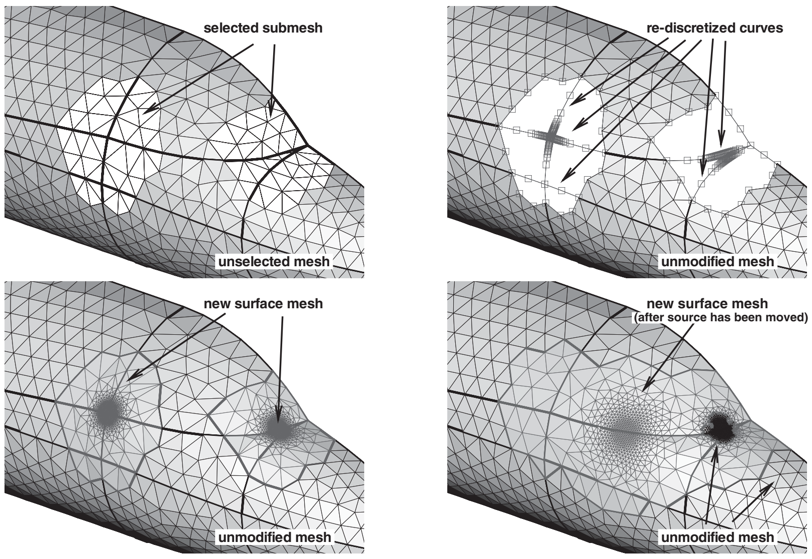

Figure 11. Remeshing example based on inserted/moving point sources.

```
    holeMesh = advFrontTriangulator.Triangulate(geometry,
                                                meshSizeSpec,
                                                advFront);
    meshEnhancer.Optimize(geometry, meshSizeSpec, holeMesh);
    surfaceMesh += holeMesh;
  }
  DoPostProcessing(surfaceMesh);
```

In Figure 11, the remeshing process is demonstrated on a real example. At the beginning, two point sources are inserted into the mesh size specification object. Then the part of the mesh now violating the mesh size is selected and remeshed. Subsequently one of the point sources is moved and the mesh size assigned to this source is modified which results in the last surface mesh shown.

In case of volume meshes it is planned to cut out holes consisting of elements not respecting the mesh size. The outer boundary of each hole lying on the CAD geometry will be remeshed as described above prior to the remeshing of the interior of the hole.

5. PARALLELIZATION

At present the CAD definition of complex configurations such as fighter aircrafts consists of thousands of patches. For such large geometries, the time needed to run the surface mesh generator is an important factor. Therefore, to reduce the runtime requirements, a parallelization of the initial surface mesh generation procedure itself and of the surface remeshing has been introduced in the ST++-system.

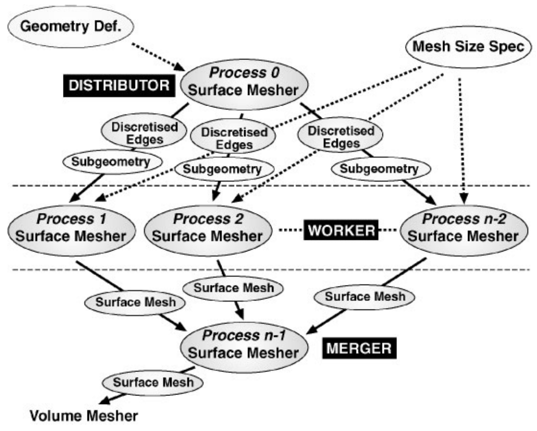

Figure 12. Architecture of the parallel surface mesher.

The parallelization of the surface meshing is incorporated via a pipelining approach where the loop over all the topological faces to be meshed is executed in parallel. In Figure 12 the architecture of the parallel system is outlined. Initially, the available processes are divided into three groups. One process has the role of the distributor, one of the merger and all the others are assigned to the worker group. The distributor process initializes the complete geometry definition and has then the task to extract and send subgeometries to the next worker waiting for data. When the worker process has meshed the received subgeometry, i.e. finished the advancing front triangulation together with the enhancements, the surface mesh is delivered to the merger. At the end of the pipeline this merger process receives all the single surface grids and takes care of their assembly into one large surface mesh. Based on the message passing programming model (MPI [18, 19]) the parallel algorithm consists of the following steps:

```
if(IAmTheDistributor){
  geometry = InitGeometry( );
  while(!allTopologicalFacesProcessed( )){
    setOfFaces = GetSetOfFacesToBeMeshed( ); // <-- chuncksize
    subgeometry = geometry.Extract(setOfFaces);
    worker = DetermineWaitingWorker( );
    SendGeometry(subgeometry, worker);
  }
  SendThankYouAndGoodByToAllWorker( ); // (;-)
}
else if(IAmAWorker){
  meshSizeSpec = InitMeshSizeSpecification( );
  while(!ReceivedThankYouAndGoodByFromDistributor( )){
    geometry = RequestGeometryFromDistributor( );
    surfaceMesh = GenerateSurfaceMesh(geometry, meshSizeSpec);
    SendSurfaceMesh(surfaceMesh, merger);
  }
  SendFinishedToMerger( );
}
else if(IAmTheMerger)
  while(!ReceivedFinishedFromAllWorkers( )){
    surfaceMesh = ReceiveSurfaceMeshFromAnyWorker( );
    completeSurfaceMesh += surfaceMesh;
  }
  DoPostProcessing(completeSurfaceMesh);
}
```

Nevertheless, a few pitfalls are inherent in the parallel execution. In the above outlined algorithm, the curve discretization is left to the worker for a better work share. This is valid if and only if it can be guaranteed that the result of this procedure is exactly the same on each worker process, because otherwise floating point roundoff errors might cause some curves to be discretized with one side more or one side less. It is a preliminary requirement for the merger that all curves are discretized in the same way, otherwise the assembly of the single surface grids fails due to non-corresponding boundaries. Therefore, the following modification is optionally available:

- The distributor pre-discretizes all the edges of the subgeometry (if not already available) and sends the subgeometry together with the discretized edges to the worker.
- The worker receives the subgeometry together with the discretized edges and immediately starts with the creation of the initial front based on the straight sides received.

Although this burdens the distributor with a higher work load, this is not critical due to the following reasons:

- The distributor keeps all discretized edges. If a subgeometry contains an already discretized edge, these sides are used again. This means that after a certain small startup delay (discretizing edges needs only a very small fraction of the time needed for the advancing front triangulation), the maximum performance is reached again because only the new edges needed have to be discretized. It should be mentioned that the overhead for the transmission of the sides is neglectible because they are only added to the message buffer. This means the number of messages to be send is the same, only their size increases.
- It is possible to overlap the different tasks. The distributor can continue pre-discretizing edges concurrently to the worker until at least one worker requests data. Only non-blocking communication calls have to be used.

The surface remeshing has been parallelized in a similar way. It starts with the parallel analysis of an existing surface mesh, which is partitioned across the available processes with one element overlap by means of the ***MeshLib***.[**] As partitioning strategy generally a graph based partitioning scheme is applied although a recursive coordinate bisection strategy also

[**]An EADS M internal OO library for the parallel handling of hybrid unstructured meshes, e.g. fully parallel on-the-fly partitioning, re-partitioning, creation of overlap cells across domains, quality analysis, feature extraction, submesh extraction, I/O, communication support, etc.

works well. Important is that compact partitions result and that the number of nodes is balanced across all domains to achieve a good load balancing. To ensure that both the sequential and the parallel marking algorithms result in the same set of markers, only the state of the external elements[††] has to be exchanged additionally after each marking loop in the parallel version. Having marked all parts of the surface grid to be remeshed, all partitions are collected on the distributor process.[‡‡] The distributor then extracts the parts to be remeshed and sends the unmodified parts immediately to the merger process. From the kept parts the corresponding boundary edges are extracted and all edges lying on curves have to be rediscretized. This is executed in parallel in the same pipelining approach described above. For each curve to be rediscretized the distributor determines the corresponding endpoints and send this information together with the curve itself to the next free worker process, which in turn sends the newly discretized edges to the merger process. After all edges have been rediscretized, the new boundary edges are returned to the distributor and used together with the unmodified edges to form the initial fronts of the holes to be remeshed. The now following loop over the faces to be remeshed is executed in parallel again in the pipelining scheme already described above, the only difference is, that the initial fronts are now always prepared on the distributor process based on the existing boundary edges. After all holes have been remeshed by the worker processes and sent to the merger process, the merger takes care about recombining all parts into a single (large) surface mesh. Having done this tasks together with the necessary post-processing operations like correcting the face orientation, etc., the merger is ready to deliver the new mesh for further tasks like e.g. volume remeshing.

Experience has shown that the parallel algorithm scales quite well with the number of processors. Depending on the type of hardware used, the communication network and the problem size, a linear speed-up is normally obtained also for small problems containing a sufficiently large number of patches. Especially for complex geometries containing a large number of patches the time needed to get a surface mesh is greatly reduced in parallel mode if multiple processors are available.

6. RASTERIZATION OF THE GEOMETRY

A fully automatic and parallel feature-based rasterization of native CAD data has been developed. The local curvature and characteristic length are investigated along CAD curves and inside trimmed CAD surfaces in order to define local sample lengths. A locally refined Cartesian background mesh (octree data-structure) is constructed to prolongate and therewith smooth the sample lengths. During the surface mesh generation, the Cartesian background mesh serves as the mesh size specification. Details are provided in Reference [20].

6.1. Rasterization of Native CAD Curves

The rasterization of CAD curves are controlled by three sampling parameters, which are specified by the user: minimal arc length $L_{\min}$, maximal arc length $L_{\max}$ and maximal curvature

[††]duplicated elements in the overlap area of a partition owned from another process.

[‡‡]It is assumed that the surface mesh fits completely into the memory of one process, which should normally be the case also for large surface meshes.

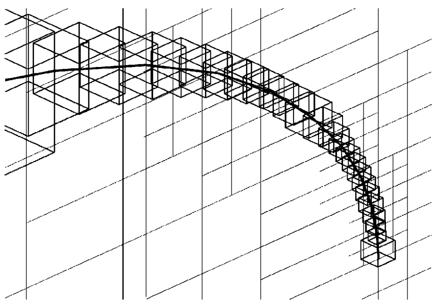

Figure 13. Raster boxes.

angle $\alpha_{\max}$. Now a CAD curve is subdivided into consecutive curve segments applying the following three sampling criteria: the arc length of a curve segment must not be smaller than the specified minimal arc length $L_{\min}$. Conversely the curve segment length must be smaller then the maximal arc length $L_{\max}$. Last but not least, the curvature angle must be smaller than the maximal curvature angle $\alpha_{\max}$. The last sampling criterion is only applied, if the arc length is larger than the minimal arc length $L_{\min}$. The curvature angle is taken as the angle between the tangential vectors at the two end points of a curve segment. The sample length cannot be larger than the length of the corresponding CAD curve.

Finally, all curve segments are approximated by straight lines. For each straight line a bounding box is determined: the bounding box is called raster box, because it controls the local resolution of the later Cartesian background mesh. Figure 13 shows the raster boxes for a CAD curve around the leading edge of the ONERA M6 wing (in strongly magnified resolution). The entire CAD curve is enclosed by the raster boxes. Besides, the final Cartesian grid is visible in the plane cutting the CAD curve.

6.2. Rasterization of Native CAD Surfaces

The rasterization of CAD surfaces requires the same three user specified sampling parameters $L_{\min}$, $L_{\max}$ and $\alpha_{\max}$, which are used also for curve rasterization. Because only the part inside a trimmed CAD surface is considered, a scan-line algorithm from computer graphics is applied, [21, 22]. As first step, the trimming CAD curves are approximated by sequences of straight lines. For this, they are rasterized as described before. The resulting straight lines in physical (Cartesian) space are transformed to the $(u-v)$-parameter space of the CAD surface, in which the remaining computation takes place.

The second step consists of computing the stencil point distribution, where the local surface curvature will be investigated later. For both u- and v-direction equally distributed iso-curves (probes) of the CAD surface are rasterized applying again the previous curve rasterization algorithm. But this time the curvature angle is defined as the angle between the surface normal vectors at the end points of a curve segment. The end points of the evaluated curve segments are taken as the sought stencil points. For each direction, the final stencil point distribution is extracted from these probes. In Figure 14 the final stencil point distribution in u- and v-direction are represented by the circles.

Now the scan-line algorithm is applied separately for the u- and v-iso-curves, which are defined by the stencil points (third step). It identifies that part of the iso-curves, which are

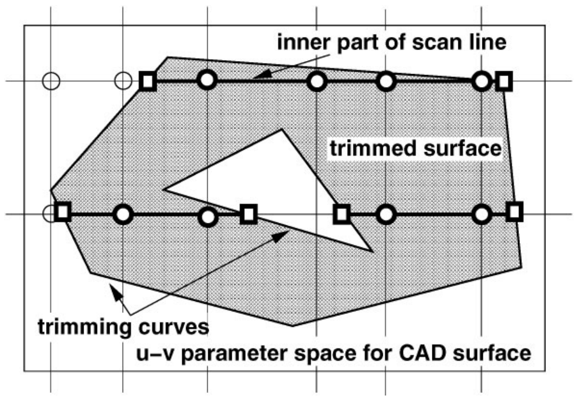

Figure 14. Rasterization of trimmed surface.

inside the trimmed surface or rather inside the polygon constituted by the sequences of straight lines. The demarcations are drawn as squares in Figure 14. However, these inner curve parts are rasterized again with the presented approach. Afterwards, the computed sample lengths are related to the stencil points located inside the trimmed surface (circles drawn with thick lines in Figure 14).

Every stencil point inside the trimmed surface gets a raster box according to the stored sample length. Additional raster boxes are created, if the distance between two stencil points is larger than their sample lengths. In this way, the trimmed surface is enclosed by raster boxes completely, which are used for the generation of the Cartesian background mesh.

6.3. Cartesian background mesh

The locally refined Cartesian background mesh specifies the mesh size required by the surface mesh generator. It bases on the hierarchical octree-data structure. which describes the connectivity between the Cartesian cells [23–25]. The raster boxes along the CAD curves and CAD surfaces, determine the local resolution of the Cartesian background mesh: all Cartesian cells are identified, which intersect a raster box. These Cartesian cells must not be larger than the current raster box. Besides, the sample length of the corresponding curve segment is stored in every intersected Cartesian cell. At the end, the Cartesian background mesh is smoothed due to an one-level difference rule: it is not allowed, that two neighbouring Cartesian cells differ by more than one refinement level.

The stored sample lengths are prolongated through the Cartesian mesh. The rate of change between adjacent Cartesian cells is limited by an user-defined slope. In this way, a smoothed sample length distribution is achieved in the complete flow domain. Moreover, the user is able to control the rate of coarsening of the triangulation by these slope parameter. Finally, the gradient of the sample length is calculated using a least square method.

During generation of the surface triangle mesh, the Cartesian background mesh specifies the local mesh size. Now the mesh size is required for a point. First, the Cartesian cell is identified applying the hierarchical octree data-structure, which encloses this point. Then the sought mesh size is interpolated linearly using the sample length and its gradient, which are

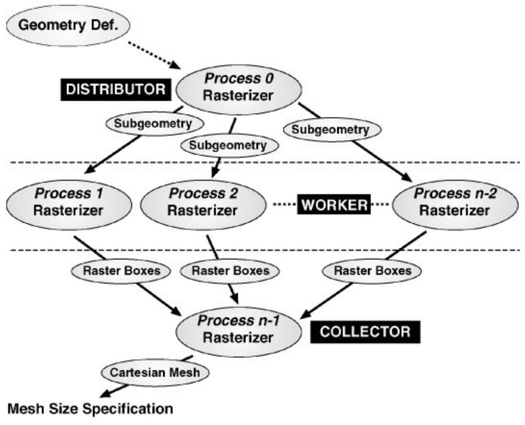

Figure 15. Architecture of the parallel rasterization program.

stored in the Cartesian cell. It is to emphasis, that the smoothed mesh size specification is also available in space and therefore usable by a tetrahedral mesh generator.

6.4. Parallelization

The parallelization of the rasterization process is incorporated via a pipelining approach: the loop over all topological curves and faces to be rasterized is executed in parallel. The parallel algorithm bases on the message passing programming model (MPI). In Figure 15 the architecture of the parallel system is outlined. Initially, the available processes are divided into three groups. One process has the role of the distributor, one of the collector and all others are assigned to the worker group. The distributor process initializes the complete geometry definition and has then the task to extract and send sub-geometries to the next worker waiting for data. When the worker process has rasterized the received sub-geometry, i.e. computed all the raster boxes enclosing the CAD curve or CAD surface, these raster boxes are delivered to the distributor. At the end of the pipeline this distributor receives all the raster boxes and generates the Cartesian background mesh and prolongates the sample lengths. Therefore, currently the operations for the Cartesian background mesh are implemented sequentially. The parallelization of these operations will be done soon.

7. 'REAL WORLD' EXAMPLE

To demonstrate the capabilities of the new ST++-system a highly detailed artificial fighter aircraft configuration has been selected equipped with all kinds of (large) appendages and (very small) sensors. This configurations consists of about 2750 topological faces and 6800 topological edges, resulting in an approximately 140 MB large STEP description (B-Rep model). Details of the configuration together with the created surface mesh are shown in Figures 16, 19.

The described surface mesh generation was completely performed in batch-mode on a Linux cluster equipped with dual 2 GHz XEON CPUs and 2 GB memory per node (RDRAM)

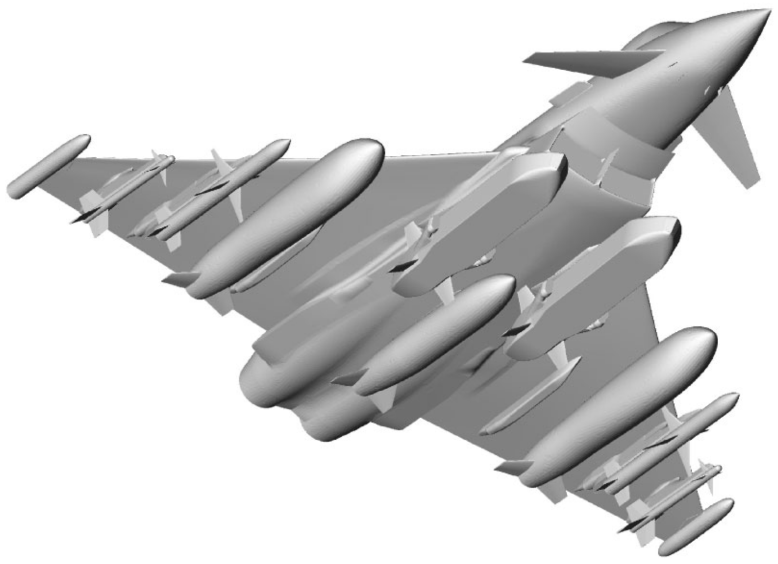

Figure 16. Bottom view of complete configuration.

connected via fast ethernet. In the following all timings refer to wall-clock timings if not otherwise stated.

In the first stage the rasterization of the boundary representation given in STEP format was performed in parallel on 16 processors in 6:24 (min). As input only the following parameters had been used:

- (min./max.) default raster length for topological edges and faces: (4/50) (mm).
- (min./max.) raster length for the group 'ADTs': (0.5/5) (mm).
- (min./max.) raster length for the group 'Nearfield': (0.5/ 5) (m).

The resulting Cartesian mesh contained approximately 1,4 million cells. This backround grid was solely used to define the mesh size for the surface mesh generator. No additional sources had been defined (Figures 17–21).

In the second stage the surface mesh generator was started with the geometry and the Cartesian mesh and took on 32 processors 6:40 (min) to generate the valid surface mesh shown containing approximately 2 million triangles. Compared to the 126 (min) needed for a sequential run this corresponds to a parallel speed-up of about 20. Speed-up values for other number of processors are shown Figure 22. Case A corresponds to runs where the distributor performed the edge discretization whereas in case B the worker discretized the edges themselves in parallel. Up to 16 processors the pipelining approach used for the parallelisation scales quite well but degrades for a larger number of CPUs. Tests have shown that a high performance network like Myrinet only has a small positive impact on the results. Therefore, a modified parallelization approach might be incorporated in the future consisting of multiple meshing pipelines with a balanced number of worker processes per distributor and merger together with a pre-partitioning of the geometry and a post-processing of the mesh parts.

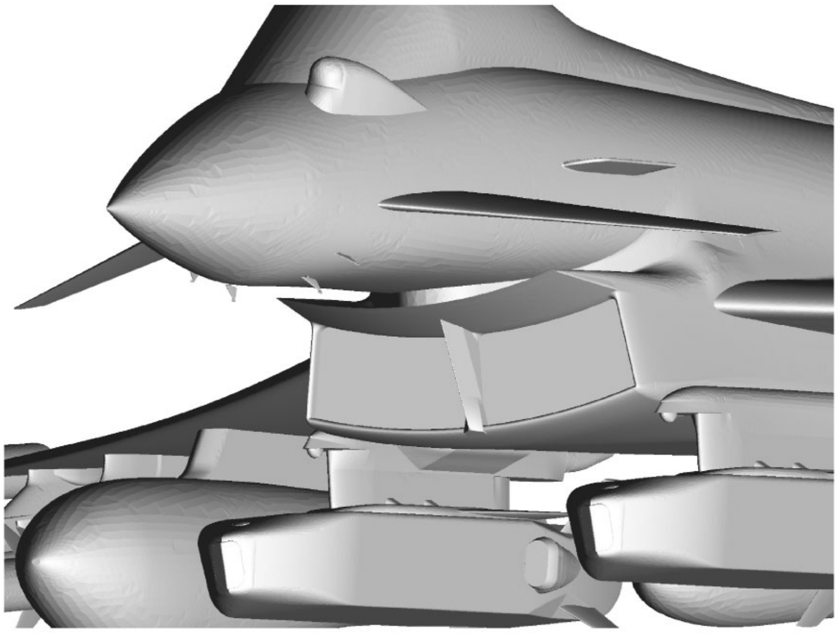

Figure 17. View from the front.

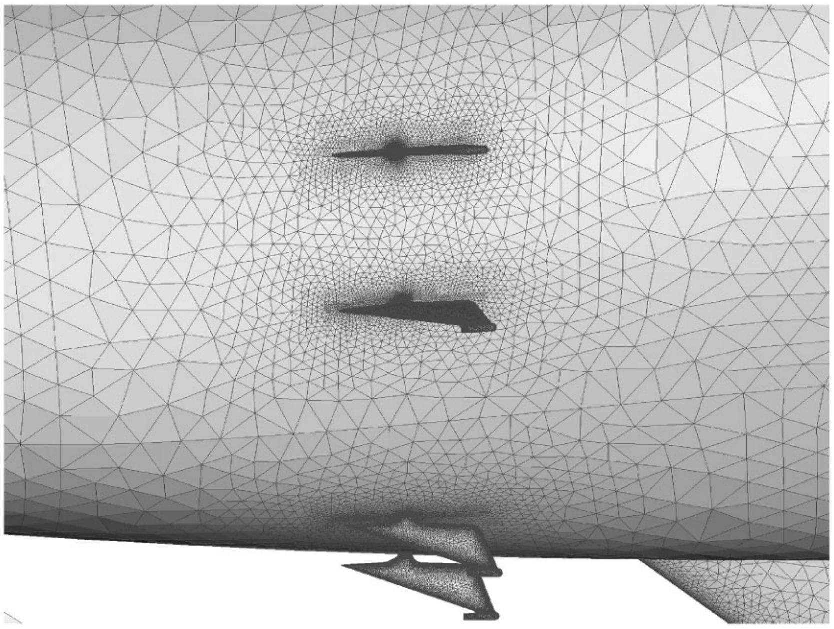

Figure 18. Zoom into ADT vane sensor area.

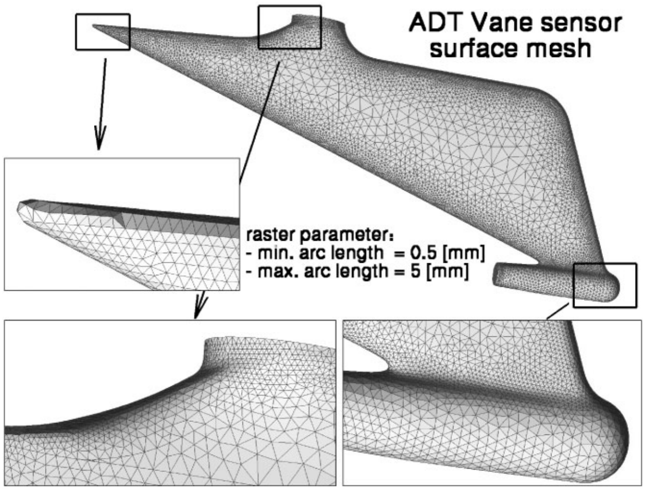

Figure 19. Surface mesh of single ADT vane sensor.

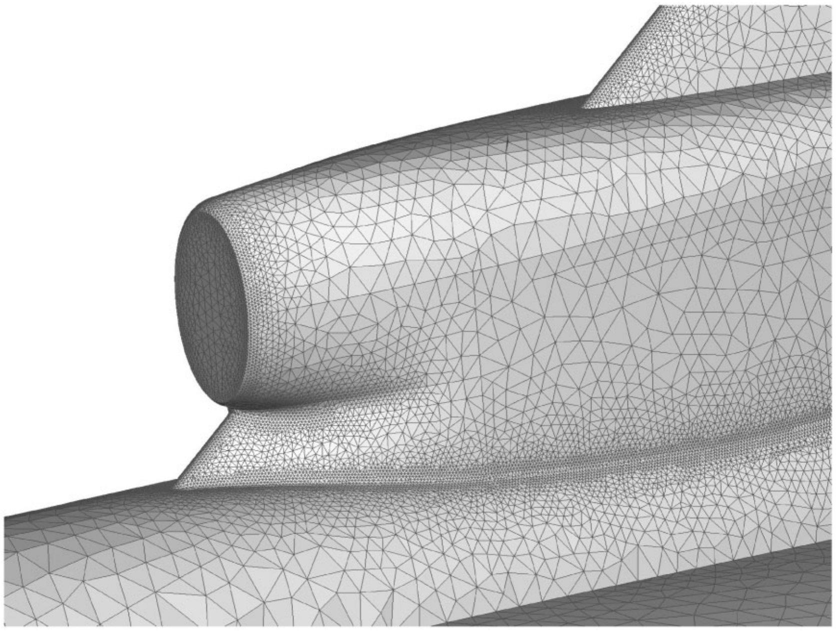

Figure 20. Surface mesh near fin.

Despite that, the important total time to generate the mesh can be reduced down to approximately 7 min for this configuration enabling fast turn-around times and efficient optimizations.

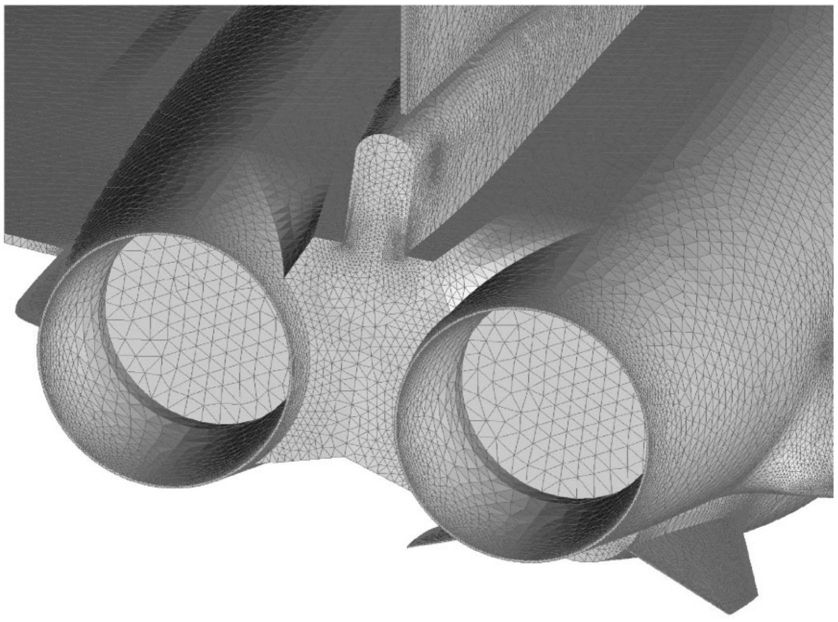
Figure 21. Surface mesh at engine exhausts.

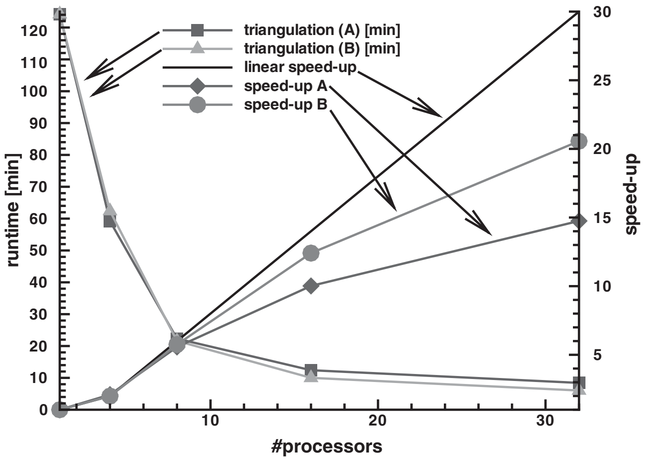

Figure 22. Speed-up and runtime measurements.

8. CONCLUDING REMARKS

In this paper a new object-oriented (OO) approach to surface meshing has been presented based on the ST++-system. The OO design and implementation have been explained together with the advantages of this approach, compared to the classical structured programming method. The different components of the system and how they interact together have been explained in detail. Algorithmic enhancements to the advancing front algorithm have been proposed. It has been shown that a modified side selection strategy improves the run-time performance to a noticeable amount. Additionally, an improved 'ideal point'-calculation has been demonstrated to be mandatory for the application of the advancing front triangulation directly on 'real world' CAD data. The performance improvements resulting from a parallelisation of the surface mesh generation procedure have been outlined, together with a discussion of the most important aspects inherent in the parallelisation applied. An elegant and automatic method to derive a well suited mesh size specification has been presented, relying on a geometry analysis/rasterization. Finally, the application of the system to a complex example demonstrates the capabilities of the approach.

ACKNOWLEDGEMENTS

We would like to thank EADS Military Aircraft, Ottobrunn, and all colleagues in the numerical simulation department for the support.

Special acknowledgements have to be given to Luciano Fornasier, Stephan M. Hitzel, Kaare A. Sørensen and Herbert Rieger for the great discussions and for the support concerning different topics of this work.

REFERENCES

1. Computational and Civil Engineering Department, *FLITE-3D User Manual*. University of Wales Swansea, Singleton Park, Swansea SA2 8PP, U.K.
2. Peraire J, Peiró J, Morgan K. Advancing Front Grid Generation. In *Handbook of Grid Generation*, (Chapter 17), Thompson JF, Soni BK, Weatherill NP. (eds). CRC Press: LLC, 1999.
3. Peiró J. Surface Grid Generation. In *Handbook of Grid Generation*, (Chapter 19). Thompson JF, Soni BK, Weatherill NP. (eds). CRC Press: LLC, 1999.
4. Gamma E, Helm R, Johnson R, Vlissides J. *Design Patterns*: *Elements of Reusable Object-Oriented Software*. Addison-Wesley: Reading MA, 1995.
5. Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley (3rd edn.). Reading MA, 1997.
6. Balzert H. *Lehrbuch der Software-Technik*. Spektrum Akademischer Verlag, 1998. Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung.
7. Balzert H. *Lehrbuch der Software-Technik*. Spektrum Akademischer Verlag: New York, 1998. Software-Entwicklung.
8. NIST (National Institute of Standards and Technology). *Initial Graphics Exchange Specification* (*IGES*), *Version* 5.3, 1990.
9. ISO (International Organisation for Standardization), Geneva. *Standard for the Exchange of Product Model Data* (*STEP*), *ISO 10303*.
10. Ferguson DR. Spline geometry: a numerical analysis view. In *Handbook of Grid Generation*, (Chapter 27), Thompson JF, Soni BK, Weatherill NP. (eds). CRC Press: LLC, 1999.
11. Gerald E. Farin. *Curves and Surfaces in Computer Aided Geometric Design* (4th edn.). Academic Press: New York, 1997.
12. Piegl L, Tiller W. *The NURBS Book* (2nd edn.). Springer: Berlin, 1997.
13. Harlan McMorris and Yannis Kallinderis. Octree-Advancing Front Method for Generation of Unstructured Surface and Volume Meshes. *AIAA Journal* 1997; **35**(6).
14. Aftosmis MJ, Delanaye M, Haimes R. Automatic generation of CFD-ready surface triangulations from CAD geometry. *AIAA Paper 99-0776*, 1999.
15. Löhner R. *Applied CFD Techniques*. Wiley: New York, 2001.
16. Cuilliere JC. An adaptive method for the automatic triangulation of 3D parametric surfaces. *Computer-Aided Design* 1998; **30**(2):139–149.
17. Tristano JR, Owen SJ, Canann SA. Advancing front surface mesh generation in parametric space using a Riemannian surface definition. In *IMR*. 1998; pages 429–445.
18. Snir M. *et al. MPI—The Complete Reference* (2nd edn.), vol. 1, (The MPI Core). The MIT Press: Cambridge, MA, 1998.
19. Gropp W. *et al. MPI—The Complete Reference*, vol. 2. The MIT Press: Cambridge, MA 1998. (The MPI Extensions).

20. Deister F, Tremel U, Hirschel EH, Rieger H. Automatic feature-based sampling of native CAD data for surface grid generation. In *Numerical Notes on Fluid Mechanics*, Hirschel EH. (ed.). Springer Verlag: Berlin, Heidelberg, New York, 2003, to appear.
21. Piegl LA, Richard AM. Tessellating trimmed nurbs surfaces. *Computer Aided Design* 1995; **27**(1):16–26.
22. Foley JD, van Dam A, Feiner SK, Hughes JF. *Computer Graphics: Principles and Practice* (2nd edn.). Addison-Wesley: Reading, MA, 1990.
23. Frank Deister. Selbstorganisierendes hybrid-kartesisches Netzverfahren zur Berechnung von Strömungen um komplexe Konfigurationen. *Ph.D. Thesis*, Universität Stuttgart, 2002.
24. Deister F, Hirschel EH. Self-Organizing Hybrid Cartesian Grid/Solution System with Multigrid. In *AIAA-2002-0112*, 2002.
25. Aftosmis MJ. Solution Adaptive cartesian grid methods for aerodynamic flows with complex geometries. In *Lecture Series CFD*, vol. 2. VKI, March 1997.