VECTIS Mesher – A 3D Cartesian Approach Employing Marching Cubes

Lukáš Plaček

Ricardo Prague, s.r.o.
Thámova 11-13
186 00 Prague
Czech Republic
Lukas.Placek@ricardo.com

Abstract. This paper describes the main principles used in the development of the VECTIS mesher. The mesher produces unstructured 3D meshes suitable for Finite Volume Methods. It is based on the Cartesian approach. In contrary to the traditional approaches which use exact shape of boundary faces of cut cells, this mesher employs Marching Cubes method for generation of majority of boundary faces. Only in problematic parts of the geometry, when the danger of chamfering of sharp features occurs or when watertightness of the cell might not be ensured, the Exact Fit method is used to produce the patches. Because two different methods are used for generation of patches, additional effort needs to be made to tie the boundary polygons to prevent gaps. A new algorithm for determining the most suitable configuration of triangles of Marching Cubes patterns is proposed. In cartesian meshers, a problematic situation occurs whenever triangles of the surface lay exactly on a side of the intersecting box. In order to prevent these collisions, an approach called Dual Levels has been introduced. The implemented method of cell refinement is presented. The paper also explains the way how the problem of cells that are too concave was resolved. The algorithm of the whole meshing task is described in detail. The new mesher has significantly lower time and memory demands in comparison with its predecessor. The main approaches responsible for this improvement are discussed.

1 Introduction

For many applications, it is advantageous to use meshing based on the Cartesian approach. Although this type of meshing have a drawback in lower quality of cells near boundary, there are also significant advantages. Cartesian meshers are robust, so when the input surface geometry is prepared in a reasonable quality (it is closed and there are no self-intersections) the meshing process does not require further interaction with the user. Because generation of inner parts of meshes is very easy, Cartesian meshers can quickly produce millions of cells.

There are two basic ways how Cartesian meshes can be generated. The **first approach** uses so called cut cells, which means that the boundary

polygons have arbitrary shape and size defined by the intersection of the input geometry and the cutting box. When this approach is used, the solver that uses these meshes needs to be able to cope with the fact that neighbourhood of cells with significantly different size is possible. A finite volume scheme like this has been presented in [5]. Another possibility may be merging of cells (like in [3]). There is a lot of literature dealing with this kind of Cartesian meshing, for example [1], [2], [3] or [4]. The basic knowledge necessary to start working on the development of a Cartesian mesher is well summarised in [3]. In order to try to increase the quality of surface cells, the **second approach** is used by several development groups. In this case, the cartesian cells are generated only inside and then, the gap between the inner cells and the boundary is tried to be filled by cells with as high quality as possible. This approach is sometimes called I2B (interior to boundary). When using this way of mesh generation, the preservation of sharp features is the most challenging part of the task. This approach can be represented for example by these papers: [6], [7] or [8].

The mesher described in this paper uses the first approach (cut cells) and the generated 3D meshes are used by VECTIS solver, which is based on principles described in [5]. The mesher uses an unique approach of generation of patches. The exact shape of patches of cut cells (generated by a method in further text called Exact Fit) is used only in these boxes which are intersected by a sharp feature. All the other patches (the majority) are generated by Marching Cubes method (see [10]), which is very straightforward and quickly generates simple patches. Usage of Marching Cubes has a great advantage also in its ability to overcome problems with small flaws in the input geometry. The real world geometries often contain problems when healthy triangles are connected by a thin triangle with opposite orientation (this flaw is called "folded geometry"). When boundary faces are produced by Exact Fit, this kind of flaw leads to problematic patches and an additional cleaning algorithm needs to be employed. However, these flaws are invisible for Marching Cubes method; therefore, the surface is naturally cleaned up. Also, a new method for choice of the proper pattern of Marching Cubes is proposed (see section 4.7). Usage of Marching Cubes and the mixture of two different methods of patch generation creates a new problem which needs to be overcome: watertightness of the surface polygons need to be ensured. It is described in this paper how to do it (see section 4.8). There is another unique approach used in the mesher which ensures that collisions of the input surface with sides of cutting boxes are avoided. This technique is referred as Dual Levels in this paper. The approach is based on slight shifting of nodes of the input geometry to discrete levels and cutting planes are shifted to another discrete levels (shifted by half step), so as it can never happen that a triangle of input surface lies exactly on a cutting plane. The step of the discrete levels is far smaller than the manufacturing tolerance of the real component. However, the step is big enough to ensure robustness of the routines for generation of faces of cells. The technique is described in section 4.2. In order to cope with concave cells,

the mesher uses two techniques to overcome problematic situations. The first approach uses splitting of cells to convex features (called Cell Splitting in this text and described in section 4.11). For those situations when Cell Splitting fails (this can happen if the input surface contains flaws), a less exact but robust method is used (called IP patches in this paper and described in section 4.11). This second method is based on the maximal simplification of the inner parts of patch structure. The algorithm of the whole meshing task is described in detail in section 6. In comparison with its predecessor (the main principles on which the previous mesher was based were presented on 3rdInternational Meshing Roundtable [9]), the new mesher produces cells with much higher quality. This is caused mainly by better choice of Marching Cubes patterns, better strategy of decisions which method should be used (Exact Fit or Marching Cubes) when generating patches in a box and the new Dual Levels technique. Also, the new mesher has significantly lower time and memory demands. The main approaches responsible for the improvement in speed and memory consumption are discussed.

2 Context of the Mesher in VECTIS-MAX System

The mesher described in this paper is part of the new version of VECTIS program (the new program is called VECTIS-MAX). VECTIS is a three-dimensional computational fluid dynamics program that has been developed specifically to address fluid flow simulations in the vehicle and engine industries. VECTIS allows the simulation of a number of applications: in-cylinder air motion and mixture preparation, spray dynamics, combustion modelling, intake system design and optimisation such as exhaust gas re-circulation or air/fuel ratio distribution, exhaust system development such as catalyst optimisation and thermal analysis, coolant jacket design and development, under-bonnet (under-hood) thermal simulations.

The whole VECTIS system consists of preprocessor, mesher, solver and postprocessor. In the preprocessor, user can load triangulated geometry in STL format and ensure (with help of implemented tools) that the triangulated surfaces are clean (i.e. there are no open edges nor mutual intersections of the triangles). Additionally, groups of triangles forming different boundary regions can be identified. In the preprocessor, the user also specifies how the mesher should create cells (regions with different density of cells can be specified). When this information is prepared, the mesher can be run. The meshing process is fully automatic and no further interaction with the user is required. Then, the graphical user interface of the solver can be used to define the input file for the solver. Here, boundary conditions are specified on the previously identified boundaries and parameters of the simulation can be set. Then, solver can be run and the results are visualised by the postprocessor.

3 Requirements for Mesh Quality

VECTIS-MAX solver uses an unstructured mesh. Cells can be formed by any number of polygons. They can have a nearly arbitrary shape; however, there are several criteria that each cell needs to fulfil.

Shape of polygons: It must be possible to split each polygon forming the cell into triangles starting from its centre (so called *star triangulation*). The polygon may be slightly concave; however, there must be direct visibility of each of its nodes from the face centre.

Watertight cells: Each cell needs to be properly enclosed by its faces. When polygons are split into triangles by the star triangulation (see the previous condition) and the *face surface vector* $\mathbf{A_j}$ is calculated for each triangle

$$\mathbf{A}_j = \sum_{i=1}^{N} \mathbf{A}_i = \frac{1}{2} \sum_{i=1}^{N} [(\mathbf{r}_{i-1} - \mathbf{r}_c) \times (\mathbf{r}_i - \mathbf{r}_c)] \tag{1}$$

(where N is the number of nodes of the polygon, which is equal to number of triangles used for the star triangulation; $\mathbf{r_i}$ is the position vector of i-th node; $\mathbf{r_c}$ is the position vector of the centre of j-th face), the geometric conservation law has to be ensured:

$$\oint_A d\mathbf{A} = \sum_{j=1}^{N_f} \mathbf{A}_j = 0 \tag{2}$$

(where N_f is the number of faces in the cell)

Angle condition for boundary faces: Angle between the vector *cell_centre* $\longrightarrow$ *face_centre* and the normal vector of the boundary face needs to be less than 90°(the scalar product of these vectors needs to be positive). In figure 1, there are examples of cells which do and do not meet the criterion. The cell on the left side satisfies the criterion, even though it is slightly concave. The cell on the right side does not fulfil the criterion, because the angle α is greater than 90°.

Angle condition for inner faces: A similar condition as for boundary faces needs to be fulfilled for inner faces. In this case, the angle of the two characteristic vectors needs to be less than 75°(the condition is more strict).

4 General Approach

The approach is based on cutting the whole domain according to a global mesh (defined by user) into boxes. When a box is intersected by the input geometry, it can be further refined. For each hexahedral box, input/output statuses of the eight vertices are remembered. Boundary faces (patches) are

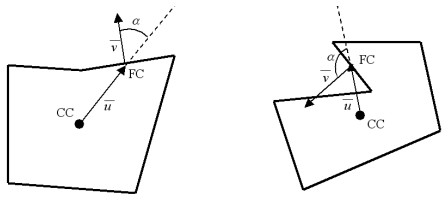

Fig. 1. A cell which does fulfil the angle criterion (left) and a cell which does not (right); CC represents the cell centre, FC is the centre of the tested face, **v** is the normal vector of the face; α is the tested angle

generated in all boxes intersected by the surface. Then, sides of the box are tested. If patches were already generated in both adjacent boxes, they are tied in order to close gaps; also, inner faces are generated on the common rectangular face of the two boxes. Then, all fully inner rectangular faces are generated on the rest of common sides of boxes. Finally, the output file is generated. In the next subsections, this general approach is described in detail.

4.1 Scaling

In order to eliminate the influence of dimensions of the input geometry (a fuel injector nozzle in millimetres, a boat in dozens of meters), the input geometry is proportionally scaled so as the longest dimension fits between 0.0 and 10.0.

4.2 Dual Levels

The problematic situation when the input geometry intersects a box exactly on one of its sides (the triangle lies exactly on the cutting plane) needs to be avoided. In order to cope with this situation, the technique Dual Levels has been proposed. The approach is based on slight shifting of nodes of the input geometry to discrete levels and cutting planes are shifted to another discrete levels (shifted by half step), so as it can never happen that a triangle of input surface lies exactly on a cutting plane. In order to get the discrete levels of the two grids, the whole working space is confined to a cube that is divided to 4200000000 levels in each direction (x,y,z), so as the position of each node can be described by three unsigned integer values. Then, odd levels are used to find positions of vertices of surface triangles and even levels help to find new positions of the cutting planes. The technique of describing x,y and z positions of vertices by three unsigned integers is used also for storage of nodes and it is described in detail in section 7. Usage of this technique does not mean that the input geometry is changed. In fact there is no error introduced, because the step of the discrete levels is far smaller than the manufacturing tolerance of the real component. However, the step is big enough to ensure

robustness of the routines for generation of faces of cells. For example, even if the calculated domain has 10 meters in its longest dimension, the step defining the fine grid is 2.4×10^{-9} m. Then, the odd grid defining positions of vertices of input triangles has step 4.8×10^{-9} m, which is far below any used manufacturing tolerances. Usage of this technique makes the task of generation of polygons of cells much easier.

4.3 Shoeboxes

Shoeboxing is a system that helps to quickly limit number of elements that need to be taken into account when intersection tests are performed. The 3D space is divided to NI x NJ x NK boxes (so called shoeboxes). Then, for each input triangle all shoeboxes that are intersected by it are found. The index of the triangle is remembered by all affected shoeboxes. In the mesher, the shoeboxes are identical to the global boxes (that are defined by the user). Usually, users define higher density of meshlines in those regions where fine details in the geometry (modelled by many small triangles) occur, which naturally makes the searching system sufficiently balanced.

4.4 In/Out Status

In the stage of box generation, it is necessary to find the in/out statuses of the vertices of all potential boxes. In order to find the in/out status of a point, a ray-casting method (described for example in [3]) is used. Six rays are released in directions -x, +x, -y, +y, -z and +z. All intersections of the rays with the triangles of the surface are found. The in/out status is then evaluated according to the number of the intersections with the surface (odd number indicates inside status, even number means outside). Those rays containing surface intersections that are too close to each other are not taken into account to avoid further analysing whether the status should be reversed or not. Two different situations are possible when two very close intersections are found: the ray just hit edge of two triangles with similar unit normal vectors (the in/out status needs to be reversed) or it just touched a sharp feature (the in/out status needs to stay unchanged). If there is not any reliable ray, new set of rays needs to be released under different angles.

The described algorithm is used when the in/out status of a node needs to be found after the boxes are generated. However, for determining the in/out statuses of the initial box vertices, a slightly modified approach also based on the described principles is used: the released rays are used for whole row of vertices. This method is described for example in [11].

4.5 Box Generation

First, so called *global boxes* are constructed according to the given meshlines; in/out statuses of their vertices are found. Each global box is marked with

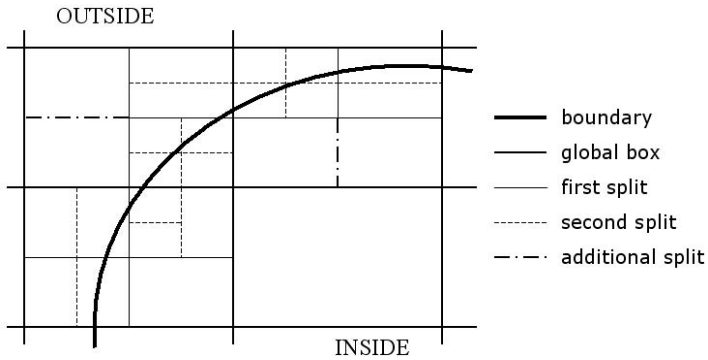

Fig. 2. 2D analogy of refinement of global boxes

one of the three statuses: *completely_inside*, *completely_outside* or *intersected*. Then, all completely inside and completely outside global boxes which neighbour to an intersected global box need to be tested for intersections on their twelve edges and intersections on their six sides. This test is necessary because it is possible that the surface penetrates the global box through an edge (there is even number of intersections on the edge) or through a side (there is a closed polygon forming the intersection on the side). If one of these cases is detected, the status of the global box needs to be changed to *intersected*.

Then, all the boxes marked as *intersected* are processed. For each the processed global box, the maximum possible refinement is found. According to the maximum refinement, a 3D array of the in/out statuses of the vertices of all the potential boxes is found [dimensions of the array depend on the maximum refinement depth D_{max}: $(2^{D_{max}}+1) \times (2^{D_{max}}+1) \times (2^{D_{max}}+1)$]. Also, a 3D array of statuses of the potential boxes is assembled (dimensions of the 3D array are $2^{D_{max}} \times 2^{D_{max}} \times 2^{D_{max}}$). The statuses are the same as for the global boxes: *completely_inside*, *completely_outside* or *intersected*. Then, $1 \times 1 \times 1$ boxes need to be combined in order to find the minimum number of boxes which can completely cover the space defined by $1 \times 1 \times 1$ boxes with inside or intersected status. This task is solved by searching for the minimum number of splits. The process of box generation is illustrated in figure 2. The algorithm of box generation does not allow neighbourhood of boxes with too different levels of refinement. When a big box is touched by more than one split line on one of its side (when 2D analogy is considered), it needs to be refined in the same direction. This situation can also be seen on the figure 2. The big box on the left part of upper-left global box needs to be split horizontally; otherwise, it would be touched by three horizontal splits from two different levels of refinement. Also, the box forming the lower part of upper-right global box needs to be split vertically because of the two split lines from two different levels of refinement.

4.6 Types of Refinement

As explained in the previous section (4.5), refinement depth (D) defines the maximum possible divisions of the global box. The maximal number of $1\times1\times1$ boxes that can be generated can be calculated as

$$N_{1\times1\times1} = 2^D \times 2^D \times 2^D \tag{3}$$

For the user, there are three ways how to affect the refinement of global boxes:

1) **Global refinement depth** defines the default refinement depth valid for all boxes with no other specification
2) **IJK refinement block** allows to set a refinement level to a rectangular block of global boxes. Each global box is defined by its I,J,K integer coordinates in the system of meshlines; therefore, the block can be specified by six integer values (IS, IE, JS, JE, KS, KE – where S stands for start and E stands for end) and values specifying the refinement depth: allowed *refinement level* and *forced refinement*. If the forced refinement is specified, the global box will be split regardless of whether the input surface intersects it or not.
3) **Boundary refinement** specifies the refinement depth which is to be used in the global boxes that are intersected by a particular boundary. With this type of refinement, three variables can be set for a boundary:
   - Refinement depth at the boundary
   - Refinement blending distance (This parameter specifies an integer value which is used to control how the refinement at the boundary blends into the refinement level of the surrounding cells. Blending is achieved by giving the cells at the boundary a forced refinement level which is less than or equal to the specified refinement depth, and propagating away from the boundary in layers of successively lower forced refinement. The blend distance defines how many layers of cells should be at each forced refinement level.)
   - Blend to boundary depth -1 (This is yes/no information specifying whether the blending should start from "refinement depth–1" instead from "refinement depth")

   Boundary refinement is applied after IJK refinement blocks. Therefore, the allowed refinement depth can be changed in global boxes that has previously been affected by an IJK refinement block. However, the forced refinement level and refinement depth are always only increased by boundary refinement specifications - never decreased.

4.7 Generation of Patches

Inside each box intersected by the surface, it is necessary to generate boundary faces (so called patches). First, patches are generated as triangles and later, triangles with similar unit normal vectors are combined to convex

polygons. For generation of the triangular patches the concept of combination of methods Marching Cubes and Exact Fit is used.

Marching Cubes

Marching Cubes is a well known method used in computer graphics (described in [10]). The method defines 14 basic patterns to create triangles on the intersections of edges of a cube (*box* in the terminology of VECTIS-MAX). The patterns differ according to the in/out statuses of vertices of the cube. The majority of the patterns have more than one possibility of how to create triangles (e.g. if there are four intersected edges on the cube in pattern no. 2, two configurations of triangles are possible: "[4,2,9]+[9,2,10]" or "[4,10,9] + [4,2,10]".

During the development of the new mesher, several ways to find the optimal configuration of triangles have been tried. Based on these experiments, the method used in the older 3D Cartesian mesher was rejected. The method was based on the comparison of the in/out statuses of selected nodes in the box when intersected by the original surface and in the box intersected by the tested configuration of triangles. There are also some other possibilities described in [13].

However, during the development of the mesher, a new simple method was found which seems to be sufficient and much quicker than the other tested methods. This method is based on evaluation of a criterion calculated from the scalar product of unit normal vectors of the proposed triangular patch ($\mathbf{n_p}$) and the triangle of the surface intersecting the edge of the box ($\mathbf{n_s}$). The criterion C can be calculated as

$$C = \frac{(C_{max} - 1)b^{1-s} + b^2 - C_{max}}{(b^2 - 1)} \tag{4}$$

where b is the base of the used logarithm, s is the scalar product $s = \mathbf{n_p} \cdot \mathbf{n_s}$ and C_{max} is the chosen maximum value of the criterion (the value for the worst case when the unit normal vectors are exactly opposite). This criterion is calculated for each node of each triangle and the average is taken as the value evaluating the configuration of triangles. The configuration with the minimum value of the criterion is chosen. The shape of the function of the criterion is visualised in figure 3. The criterion is designed to strictly refuse those configurations where unit normal vectors point to opposite half-spaces. Values $b = 7$ and $C_{max} = 4$ were found as reasonable for this criterion, so they are used in the mesher.

Exact Fit

If it is not possible to use the Marching Cubes method, the Exact Fit algorithm is applied. This approach of patch generation is based on the splitting

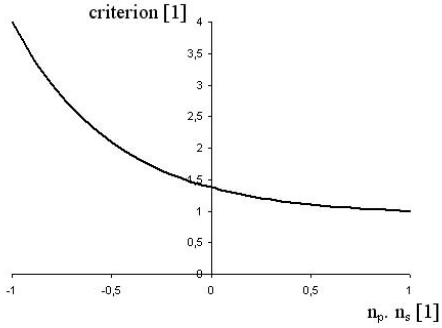

Fig. 3. Criterion for evaluation of different possibilities in Marching Cubes patterns

of surface triangles according to the sides of the box; only the triangles lying inside the box are kept as patches.

Choice of the patching method

Marching Cubes method can be used only when it is sure that the simplification would not cause chamfering of sharp features, problems with watertightness or damage of the border between different boundaries. When one of the following conditions is true, the Exact Fit must be used:

- one of the edges of the box is intersected more than once (in this case, tying of patches may not be feasible)
- the intersection polyline forms a closed polygon on one of the sides of the box (the geometry would be invisible for Marching Cubes)
- there is a sharp feature detected in the box (Marching Cubes would chamfer the feature)
- there is more than one boundary index detected among the triangles intersecting the box (Marching Cubes would damage the border between two boundaries)

4.8 Tying of Patches

Whenever patches are generated in a box, all rectangular faces of the box are tested to determine whether boxes on its both sides have already been processed. When a rectangular side is found, whose both adjacent boxes have been processed, the patches in the two boxes need to be tested for watertightness. If there is a gap between patches, they need to be tied so as the non-conformance is avoided. Two main cases when gaps appear are illustrated in figure 4. The picture on the left shows the situation when there is one box neighbouring with two boxes. Different simplifications from using the Marching Cubes method from both sides gives patches that are not properly tied. The picture on the right illustrates the situation when the Exact Fit is

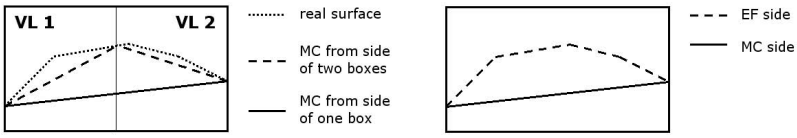

Fig. 4. Gap between patches caused by different refinement on two sides of common rectangular side of boxes (left side) and gap between patches caused by usage of different methods of patch generation (right side)

used from one side of the common rectangular side of boxes and Marching Cubes is used from the other side.

When tying of patches is used in the mesher, the corresponding nodes need to be found first. This action divides the problem to several simpler parts when several segments need to be tied to one segment. The single segment is divided so as the lengths of corresponding segments respect ratio of total lengths of the polylines:

$$\frac{L_j}{L'_j} = \left(\sum_{i=1}^{N} L_i\right) / \left(\sum_{i=1}^{N} L'_i\right) \tag{5}$$

where L_j is length of j-th segment of the polyline before tying of patches and L'_j is its length after movement of the nodes. Then, the nodes from the more complex side are moved to the new positions.

4.9 Generation of Inner Faces

Each generated inner face is tested to determine whether it can be divided into triangles by the star triangulation (see section 3). Those faces that are so concave that correct triangles cannot be formed, need to be split to more convex parts. The algorithm of searching the optimal cutting edge from the most concave angle (described in [12]) is used. Boundary faces do not require this treatment, because their convexity is ensured by the process of combination of triangular patches to polygons.

4.10 Removal of Small Cells

Volume of each generated cell is tested and compared with the size of its box. If the ratio V_{cell}/V_{box} is less than a defined constant R, the cell needs to be deleted. When a cell is removed, its inner faces need to become boundary faces of its neighbours. The constant R can be chosen by the user for each boundary region. Defaultly, $R = 1.0 \times 10^{-3}$ is set; for boundaries representing input/output or cyclic boundary, the value 1.0×10^{-100} is used.

4.11 Problem of Concave Cells

In order to overcome problems with cells that are so concave that the *angle condition for boundary faces* mentioned in section 3 is not fulfilled, two techniques are applied: *Cell splitting* and *IP patches*. Cell splitting inheres in searching for a cutting polygon which would divide the concave cell into two parts with better properties.

When this method fails (usually due to a flaw in the input surface), IP patches can be generated instead. This approach inheres in replacing the patch structure by polygons forming the intersection of the box with the surface (IP means intersection polygons). When IP patch method is applied, the shape of the geometry is not covered as well as if Cell splitting were used. Therefore, IP patches should be used only as a last chance mechanism when Cell splitting algorithm fails, just to avert failure of the whole meshing task.

Cell Splitting

First, the optimal cutting plane is found. In order to do this, the following procedure is performed. The edge representing the worst concave feature needs to be identified. Then, the adjacent edges are tested; if some of them are also not convex, the *concave feature polyline* will grow. Then, the cutting plane angle α_c is calculated as half of the average angles α_i adjacent to N edges of the concave feature polyline:

$$\alpha_c = \frac{1}{2}\left[\left(\sum_{i=1}^{N} \alpha_i l_i\right) / \left(\sum_{i=1}^{N} l_i\right)\right] \tag{6}$$

The average is weighted by lengths of the edges (l_i).

Then, the intersections of edges of the cell with the cutting plane are found and the cutting polygon (the closed polyline of the intersection) can be finished. Whenever an edge is intersected very close to one of its end vertices (the deviation of the two adjacent edges of the cutting polygon from the cutting plane is less than 5°), the end vertex is used instead.

IP patches

The unit normal vector and the face centre are assigned to each IP patch. These properties are calculated in the mesher and they are passed to the solver (in the contrary with other types of faces whose properties are found during the run of the solver). The normal vector of the IP patch is calculated from the condition described by equation (2). Sum of face surface vectors $\mathbf{A_j}$ of all other faces determines the face surface vector of the IP patch. A face centre needs to be found so as the angle criterion (see the section 3) is fulfilled. The process needs to be done iteratively, since each movement of the face centre affects the cell centre.

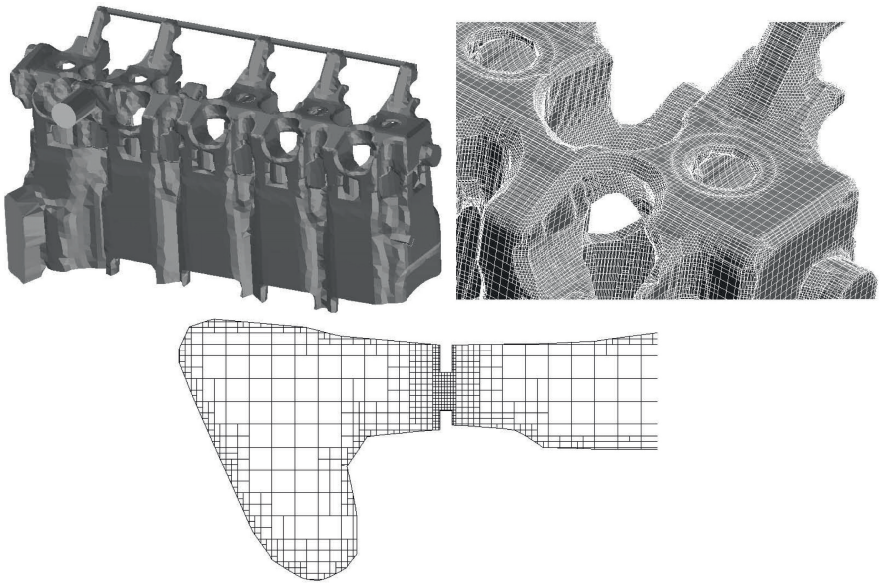

Fig. 5. Coolant jacket (input geometry, part of the mesh and a slice view of the gasket hole)

5 Examples of Generated Meshes

In order to show some examples of generated meshes, geometries of two typical problems often solved by VECTIS were chosen.

1) **Exhaust manifold with turbine:** The input geometries, and parts of generated meshes are illustrated in figure 6. The pictures represent exhaust manifold with a turbine housing. Images on the left side are linked to the solid part of the multi-domain simulation. On the right side, there is the fluid part of the simulation. When both geometries are meshed, the common interface needs to be made conformal in order to prepare them for the multi-domain simulation in the solver. Even distribution of meshlines with size of cells 3 mm and refinement depth 1 were chosen.
2) **Coolant jacket:** In figure 5 a typical example of geometry for modelling of flow of water in cooling channels of the engine is shown. The size of the global cells was chosen as 3 mm; together with refinement depth 2. The slice of the mesh in the lower part of the figure shows the critical part between the head and the block of the engine, called the gasket hole. IJK refinement block was used here to enforce refinement depth 3 in order to ensure sufficient number of cells to realistically simulate the flow in this narrow channel.

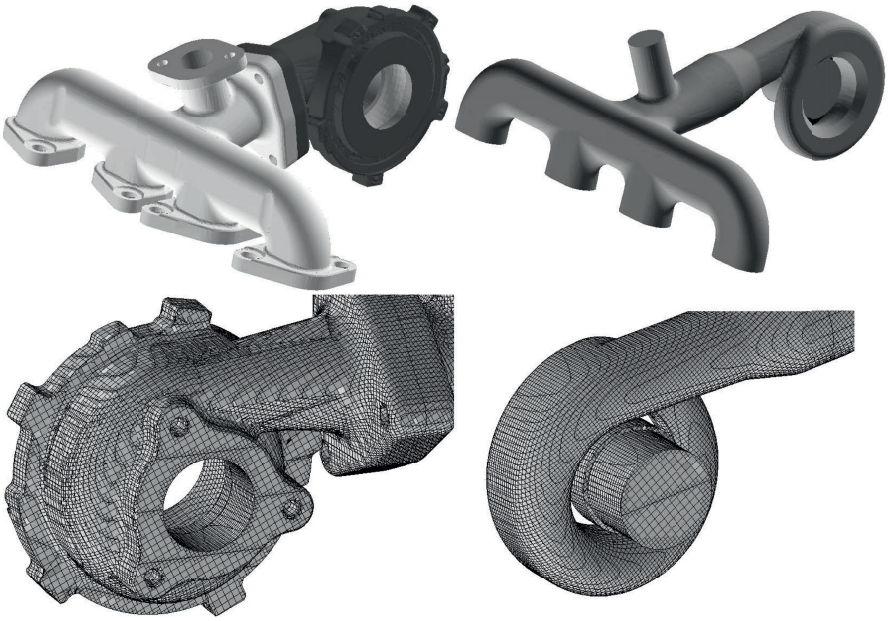

Fig. 6. Exhaust manifold with turbine (input geometries and parts of the meshes)

6 Description of the Meshing Algorithm

When running in the meshing mode, the scheme of the work of the mesher is this:

1) **READ INPUT FILE FOR MESHING TASK:** The input ASCII file (containing information about positions of meshlines and some other specifications for the meshing task) is read.
2) **READ SURFACE TRIANGLES:** The input file containing information about surface triangles (trifile) is read. Coordinates of the triangles are scaled (see the subsection 4.1), the geometrical extents of triangles are found and connectivity information is assembled.
3) **APPLY DUAL LEVELS:** The x,y and z positions of meshlines (defining the cutting planes) are shifted to discrete positions on a very fine grid. At the same time, nodes defining triangles of the input surface are moved to lay on a different grid (with its levels shifted by half-step). This averts collisions of cutting planes with with triangles perpendicular to the principal axes. (see the subsection 4.2)
4) **DETECTION OF OVERLAPPED TRIANGLES:** The algorithm for detection of overlapped triangles is run here. If there are some problematic triangles, a warning is printed together with the list of indexes of the triangles. If problematic triangles are detected, the mesher will continue its work. Usually, the mesher automatically overcomes small

problems in the input triangulated surfaces. However, if it happens that the final gridfile contains cells with low quality, the user should try to improve the flaws in the surface detected in this step and run the mesher again.

5) **PREPARE SHOEBOXES:** The system of shoeboxes is established here (see the subsection 4.3).

6) **CONSISTENTLY ORIENT TRIANGLES:** Since the orientation of the input triangulated surfaces is random in the input file, it is necessary to orient them so as normal vectors of the triangles always point into the flow domain.

7) **PREPARE IN/OUT STATUSES:** Here, in/out statuses of the vertices of the global boxes (that are defined by meshlines) are found by ray-casting method (see the subsection 4.4).

8) **GENERATION OF BOXES:** The boxes are generated in the global boxes (see the subsection 4.5) according to the prescribed refinement specification (see the subsection 4.6).

9) **PREPARE COMMON RECTANGULAR SIDES OF BOXES:** In this part of the algorithm, the boxes generated in the previous step are indexed first. Then, the common rectangular sides of the generated boxes are generated. According to which direction they are perpendicular, U, V and W common sides are distinguished (perpendicular to x, y and z, respectively). They serve for navigation through boxes and for generation of inner faces on them.

10) **GENERATE CELLS IN BOUNDARY BOXES:** For all boxes, it is determined whether the box is intersected by the surface or fully inner or fully outer. The optimal order of processing boundary boxes is found (smaller boxes need to be processed first to be sure that when a box is processed which has more than one neighbour in a direction, all the adjacent boxes are already done). Then, all boxes intersected by the surface are looped and their faces are generated in these steps:

— **POLYGON GENERATION PART** —

A1) Generate boundary faces (patches): Marching Cubes or Exact Fit method is used for the generation of boundary faces (see the subsection 4.7).

A2) Polygon Simplification: This technique simplifies the patch structure where possible. The algorithm preserves sharp features and borders between different boundaries.

A3) Generate inner faces if neighbours are processed: All rectangular sides common with the adjacent boxes are looped. On those locations where patches of the two adjacent boxes have already been generated from both sides, the patches are tied (described in the subsection 4.8) and polygons of inner faces are generated. If any generated inner face is concave, it is split to convex parts here.

— **CELL ASSEMBLING PART** —

B1) Find a complete cell: It is found whether there is a cell that has all its polygons already generated. If there is no such a cell, continue with the next box (go to A1).

B2) Distinguish separated volumes: The connectivity of faces is found and the polygons are painted in order to find separated volumes in the box.

B3) Cell splitting: On each separated volume, it is tested whether there are concave features. If there are, split the cell to convex parts.

B4) Delete small cell: The volume of the cell is found. If it is too small, the cell needs to be removed (described in subsection 4.10).

B5) Save faces: The generated boundary and inner faces are saved to auxiliary files so as they can be retrieved back later when assembling the final grid (this concept is described below in subsection 7.2). The algorithm continues with B1.

11) FINISH COMMON FACES: Rectangular faces of those cells that are fully inner are generated here.

12) PRINT STATISTICS OF GENERATED MESH: The statistics of the generated grid is printed. The report contains information about number of generated cells, numbers of boxes processed by Exact Fit and by Marching Cubes methods, number of cells that needed to be split to convex parts. If problems occur, number of cells with negative volume, gaps or angle problems is reported here (see the section 3). Problems like this are usually linked to topological problems of the input surface.

13) WRITING THE MESH FILE: The polygons stored in the auxiliary files (saved in the step B5) are subsequently read while the output arrays are assembled at the same time. Then, the output gridfile is written out. Finally, the auxiliary files are deleted.

7 Tools Helping to Decrease Time and Memory Demands

It is hard to compare the previous VECTIS mesher with the new approach since both are doing different tasks. The new VECTIS-MAX mesher needs to perform more actions in order to meet the higher requirements for the mesh quality. Despite this fact, measurement on several typical cases has shown that the new program consumes 74 % of memory and 64 % of time in comparison with the previous system. In the following text, the main features are described that are believed to be responsible for the significant improvement.

7.1 Features Improving Time Efficiency

Integer storage of coordinates of vertices

In order to ensure the test for existence of a node to be efficient, the approach recommended in [3] and [12] was used for storage of nodes. The whole

working space is confined to a cube that is divided to 4200000000 levels in each direction (x,y,z), so as the position of each node can be described by three unsigned integer values. The choice of the number of levels is linked to the capacity of unsigned int type on 32-bit computers, which is 4294967295. All existing nodes are kept sorted according to their coordinates. When comparing two nodes, x determines which node is less; if both nodes have the same x, the decision is done according y, etc. Whenever a new node needs to be created, it is very quick to find out whether the node already exists or not, because of the binary search in the sorted array.

This concept turned out to be very efficient. Of course, it saves certain amount of memory (*unsigned int* consumes half the number of bytes than the *double* type on 32-bit computers). However, more important is its time efficiency. It is believed that this concept is mainly responsible for the higher time efficiency of the new mesher.

This usage of integer values is limited only to storage of coordinates of vertices. The full integer arithmetic (described in [3]) was not implemented.

7.2 Features Improving Memory Efficiency

Avoidance of STL containers

During the development of the mesher, it was found that when containers from the Standard Template Library (set, map and list) had been replaced by simple classes using "malloc" allocation, the time and memory requirements were significantly lower. A comparison of STL and non-STL approach was done on storage of one million nodes to a map. Memory requirements fell down to 16 % when std::map was avoided. Time necessary for storage of the nodes was decreased to 23 %; time needed to retrieve the nodes decreased to 56 %. The development team believes that the savings are caused by allocations and reallocations with a reasonable step. If an STL container is to be extensively used in a program, its allocator should be changed to avoid too many allocations by small chunks.

Temporary storage of polygons

Whenever a cell is generated (all its polygons are prepared), those polygons that are no longer needed can be stored in an auxiliary file. The memory occupied by the polygons can be reused for polygons of another cell. In order to make this technique efficient, the optimal order in which boxes are processed needs to be found so as neighbours of already done cells are processed as soon as possible.

Arrays of low-bit information

During the run of the mesher, it is often needed to store long arrays of information that requires only low number of bits (e.g. in/out status of nodes,

done status of a box, ...). However, allocation of an array of *bool* type consumes eight bits for each entry. Therefore, a tool that can store 1D, 2D or 3D array of entries of arbitrary bit-length in a chunk of memory (unsigned char type is used) has been prepared. For storage or retrieval of each entry some additional time is consumed. This is caused by necessity to find the proper unsigned char(s) containing the information and perform appropriate bit operations in order to find the bits that should be used. However, the additional time disadvantage seems to be low price for the memory reduction benefit.

References

1. Aftosmis, M.J., Berger, M., Melton, J.: Robust and Efficient Cartesian Mesh Generation for Component-Based Geometry. In: 35th AIAA Aerospace Sciences Meeting, Reno NV (January 1997)
2. Berger, M., Aftosmis, M.J.: Aspects (and Aspect Ratios) of Cartesian Mesh Methods. In: 16th International Conf. on Num. Meth. in Fluid Dynamics (July 1998)
3. Aftosmis, M.J.: Solution Adaptive Cartesian Grid Methods for Aerodynamic Flows with Complex Geometries. In: 28th Computational Fluid Dynamics Lecture Series. Von Karman Institute for Fluid Dynamics, Belgium (1997)
4. Ingram, D.M., Causon, D.M., Mingham, C.G.: Developments in Cartesian cut cell methods. Mathematics and Computers in Simulation 61, 561–572 (2003)
5. Przulj, V., Birkby, P., Mason, P.: Finite Volume Method for Conjugate Heat Transfer in Complex Geometries Using Cartesian Cut-cell Grids. CHT-08: Advance. In: Computational Heat Transfer, Marrakech (May 2008)
6. Kini, S., Thoms, R., Zhu, F.: A Fast and Fully Automated Cartesian Meshing Solution for Dirty CAD Geometries. SAE International paper 2008-01-2998 (December 2008)
7. Kanade, K., Lietz, R., et al.: Rapid Meshing for CFD Simulations of Vehicle Aerodynamics: SAE International paper 2009-01-0335 (April 2009)
8. Kini, S., Thoms, R.: Multi-domain Meshes for Automobile Underhood Applications. SAE International paper 2009-01-1149 (April 2009)
9. Bardsley, M.: Automatic Mesh Generation for CFD. In: 3rd International Meshing Roundtable, Albuquerque, New Mexico (1994)
10. Lorensen, W.E., Cline, H.E.: Marching Cubes: a High Resolution 3D Surface Construction Algorithm. Computer Graphics 21, 163–169 (1987)
11. Tarini, M., Callieri, M., Montani, C., Rocchini, C.: Marching Intersections: An Efficient Approach to Shape-from-Silhouette. In: Vision Modeling and Visualization Conference, Erlangen, Germany (2002)
12. O'Rourke, J.: Computational Geometry in C. Cambridge University Press, Cambridge (1994)
13. Frey, P.J., George, P.L.: Mesh Generation. HERMES Science Publishing, Oxford (2000)