LEVEL SETS AS INPUT FOR HYBRID MESH GENERATION

O. Gloth, R. Vilsmeier

Institute for Combustion and Gas Dynamics, University of Duisburg, Germany oliver@vg.uni-duisburg.de

ABSTRACT

Level set methods are used to describe interfaces or discontinuous solutions on fixed grids. Hence they replace time consuming remeshing and adaptation procedures. This article, however, wants to show how information provided by level set functions can be used as an input for unstructured mesh generation. It is shown that scalar field functions can be used to describe complex, moving and even changing geometries. Furthermore level set functions usually give the distance to an interface and thus offer very valuable information for anisotropic mesh generation. This paper is restricted to two dimensional mesh generation, although the discussed approaches should in principle be extendable to three dimensions.

Keywords: level set, hybrid grids, moving grids, unstructured techniques

1. INTRODUCTION

In the past years, level set methods gained substantial attention to describe the location and motion of advancing interfaces or discontinuous solutions (e.g., shock waves in compressible gas dynamics). The basic idea is to describe the position of a front by a distinct iso-value of a scalar function. This scalar function can be stored on a given, fixed mesh.

Level set methods as a computational tool have been introduced by Osher and Sethian in 1988[2]. Further numerical variants have been developed by Milner et al. [1] and Sussman et al. [5] and applied to arbitrary problems in fluid dynamics. In [4] an application of level sets for a marching-front like, structured mesh generator can be found.

Non body-conformal meshes

A surface of an object could also be considered a front. It is thus possible to run computations with complex and curved boundaries on a Cartesian and non surface conformal grid. Boundary conditions would be realized in a way, that they are fulfilled on the iso-line representing the surface. This concept is not new and has various drawbacks. Anisotropic refinement normal to the surface is very difficult and even impossible for many cases. Singularities in the geometry (corners in two dimension) need local mesh refinement to be accurately described. This has to be done, even if the computational problem does not require refinement at those points. Hence it will lead to large numbers of nodes and thus increase runtimes for simulations. The obvious advantage of this method is its simple manner of describing complex boundaries. Furthermore it is able to deal with moving and changing shapes.

A body-conformal meshing approach

The meshing algorithm presented here tries to combine the advantages of boundaries described on a fixed grid and body conformal grids. The starting point is a uniform Cartesian grid. Shapes, roughly representing the surfaces of objects to be meshed, are cut out in the near vicinity. The later resulting mesh will, however, be body conformal. Unstructured techniques are used to fill the space between the surfaces and the Cartesian base grid. A point-moving algorithm, similar to classical unstructured mesh smoothers, is used to push the later boundary points onto the surfaces.
The mesh generator described in this article is integrated into a finite volume framework. For future applications it is planned to use it for a priori mesh generation, as well as for adaptation during a running simulation. The whole software project is written in C++ and is publicly available on the web (http://www.vug.uni-lich.de/MOUSE).

2. GEOMETRY DESCRIPTION

Boundaries will be represented by one or more scalar functions. These functions will in the following text also be referred to as $\phi$-functions $(\phi(x,y))$. $\phi$-functions can be used to describe geometric effects, which are one dimension below the basic dimension of the problem. This is done by referring to a distinct function value. All examples in this article have used a value of $\phi = 0$ to define surfaces. Doing so, iso-lines (or iso-surfaces in three dimensions) are obtained.

To describe anything with even less dimensions it is necessary to use more than one level set. Intersections between different shapes are singularities in the boundary and thus are one dimension below the original boundary description. For two dimensional problems only one kind of such singularities is possible. Geometrical corners in two dimensions are represented by two intersecting $\phi$-functions. If the method is extended to three dimensions, two different singularities are possible. Edges of three dimensional shapes can be described by the intersection of two $\phi$-functions, just like corners in two dimensions. Three-dimensional corners will be represented by the intersection of a geometric edge with another level set.

The actual shape of a geometrical object consists of faces and corners, represented by several bounding level-set functions. Boolean operators are required to declare what is inside and what outside. Figure 1 illustrates how two circular shapes could be combined. In this simple example, the corresponding $\phi$-function for a circular shape can be given in analytical form, therefore no discrete level-set representation on the base grid is required:

$$ g(x,y) = r - \sqrt{(x-x_m)^2 + (y-y_n)^2} $$

With $(x_m,y_n)$ as the middle point of the circle and $r$ as its radius. For definition values of $g > 0$ refer to regions inside a shape.

An abstract interface to level set information has been designed and used in the mesh generation software described here. With this interface it is now possible to use a variety of different input functions. The software interface requires the value of $g$ as function of $(x, y)$ as well as its gradient $\nabla g = (\partial g / \partial x, \partial g / \partial y)$, again as function of $(x, y)$. For development purposes, the mesh generator has mainly been used with analytic functions. For future developments, it is of course planned to use discretized $\phi$-functions. These could be created out of traditional boundary definitions, as well as images. Later in this article some details on the discrete reconstruction of $\phi$-functions are given, along with some small examples.

3. ISOTROPIC MESH-GENERATION

Before any anisotropic region can be refined, a basic isotropic grid has to be created. This grid has to exactly represent the shape defined by a boolean combination of different $\phi$-functions. A very important feature of the described meshing algorithm is that there is always a mesh with all logical connections intact. This is quite different from other algorithms which have to create the grid out of a boundary definition.

Both popular methods, marching front and elliptic mesh generation, share this. In order to create a triangular grid using an elliptic meshing algorithm, the mesh generator has to find an initial triangulation of a given set of boundary nodes. Nodes are then successively entered until the desired resolution has been reached. Also a reconfiguration of generated elements has to be performed to fulfill the Delaunay criterion.

Furthermore it is necessary to smooth the grid. A more detailed description of this elliptic mesh generation algorithm can be found in [3]. All this work has to be done to finally get a body conformal and isotropic triangular grid.

For isotropic regions a Cartesian mesh is very well suited. On Cartesian grids it is possible to find accurate and efficient discretizations. The generation of such a grid is trivial and requires a fraction of the effort which is needed for other grid-types. As already mentioned in the introduction, the meshing algorithm described here will begin with a Cartesian grid. This grid might have locally refined regions, though. It is, however, still Cartesian except for maybe small layers to provide connections between refined and non-refined parts. Using the information, provided by the level set functions, the Cartesian grid can be locally broken up into a triangular grid. Future surface nodes
can be detected and marked. An elliptic mesh generation method is then used to smooth these regions and to get the surface nodes on their corresponding surfaces. The initial grid covers a domain which completely encloses all shapes and boundaries. During the generation process it will stay this way. That means there will be a mesh inside the objects to be meshed as well. After the meshing is finished, it is possible to cut out inner parts or to leave them in the mesh. If the inner parts are cut, the result will be exactly the same as with other algorithms, a body conformal grid. As stated above a Cartesian grid is a very good starting point for this algorithm. It is, however, not restricted to that. Curvilinear or triangulated or even a hybrid grids might serve as initial mesh.

Starting from a Cartesian mesh the way how to obtain a body conformal multi element mesh shall be briefly described now. Figure 2 shows a zoom of an initial grid plus two intersecting surface lines. The desired shape is to the right of the two lines in figure 2. A boolean OR is used in this example. The points might be added or deleted within and outside the shape. It is, however, important to keep the once found surface intact. For example it is forbidden to reconfigure an inside with an outside triangle, since this would break up the surface. The so called surface nodes try to move onto their assigned surface-lines. An iterative procedure is used to achieve this. A new position for a surface node is computed as follows:

\[
\tilde{x}_{\text{new}} = \tilde{x}_{\text{old}} - \epsilon \left( g(\tilde{x}_{\text{old}}) \nabla g \right) \tag{2}
\]

Where \( \epsilon \) is a relaxation parameter. In case of \( |\nabla g| = \text{const} \) in the whole domain, \( \epsilon = 1 \) could be used. This means that a position on the zero iso-line of the \( g \)-function would be reached in one step. It is, however, possible that the surrounding mesh hinders a node moving onto its surface. In this case the node is moved as far as possible. In figure 4 a node can be seen which already moved onto its surface. This surface-line, however, is not the real surface of the shape. Here a simple rule has to be applied to assign the node to a different level set. Whenever a node has reached \( g = 0 \) of its level set, but it is still inside the total shape, the next best level set will be used for this node. The decision if a point is within the whole shape or not is
easy, since the values of all level set functions on the node are known. With this knowledge and the boolean operators between the different \( g \)-functions it can be decided if a position \((x, y)\) is inside the shape or not. Next best means the one with the lowest positive \( g \)-value. Please note that a node cannot be reassigned to a level set it once has been connected to before. By doing so loops are prevented from occurring in this iterative process. If a node would have to be reassigned, a similar movement method than it is used for corners will be applied.

Another important topic is the recognition of corners. Corners are intersections of different level sets. Of course only those intersections are corners, which are on the real surface. In figure 5 two nodes are shown, which both reached the outer surface of the desired shape. These nodes are assigned to different \( g \)-functions. That means there should be a corner between the nodes. In this case a new node is created in the middle of the edge connecting both surface nodes (shown as a small square). The new node will then be assigned to both level sets. Movement of corners is hence performed using the following formulation:

\[
\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{old}} - c_1 \left[ g_1 \left( \frac{\nabla^2 \mathbf{x}_{\text{old}}}{(\nabla g_1)^2} \right) \nabla g_1 \right] - c_2 \left[ g_2 \left( \frac{\nabla^2 \mathbf{x}_{\text{old}}}{(\nabla g_2)^2} \right) \nabla g_2 \right]
\]

(3)

The relaxation parameters \( c_1 \) and \( c_2 \) should be less than 1 for corner nodes. Figure 6 shows the surrounding of the corner with all surface and corner nodes in their proper positions.

Now it has to be decided if the mesh within the shape should be cut out, or not. If it has to be cut out the former surface and corner nodes become boundary nodes. Generation of an isotropic unstructured grid is finished at this point. The method could be summarized as follows:

1. A Cartesian mesh is generated, which is large enough to cover the whole domain.
2. Optionally this base mesh can be refined in certain regions.
3. Quads which are close to the later boundaries are triangulated.
4. Later surface nodes are detected and marked.
5. Surface nodes are moved onto the real surfaces. During this process corners can be detected and moved to the correct positions.
6. If wanted so, inner parts of the grid can be deleted.

4. CREATION OF QUAD-LAYERS

Figure 7 shows a zoom around the left corner of the shape, together with iso-lines for both \( g \)-functions. It can be seen, that the iso-lines would be perfect grid lines for a boundary layer grid. Almost everything that is needed to create layers of quadrangular elements around the shape is already included in the basic mesh generation algorithm. To create one layer an initial set of quads is created around the shape. Figure 8 illustrates how a bilinear element is created on a surface edge. The same method can also be used to create a new quadrangular element on top of another.
one. Now the new nodes are moved in the same way as the surface nodes have been treated during the initial meshing. They do not move on the zero iso-line anymore. Instead a value \( g_{\text{line}} \) is used, which has to be stored for every node belonging to one of the quadrangular layers. The movement formula becomes:

\[
\bar{x}_{\text{new}} = \bar{x}_{i,d} - \epsilon \left[ \frac{g(\bar{x}_{i,d}) - g_{\text{line}}}{(\nabla g)^2} \nabla g \right]
\]

(4)

In figure 9 it can be seen how this algorithm creates an initial quadrangular layer for the example mesh. Figure 10 shows the insertion of two more layers into this grid. The final hybrid grid, including boundary layer, can be seen in figure 11. The mesh basically consists of three different zones. A Cartesian far field, a curvilinear boundary layer grid and a small triangular region to fill the space in between. Although these different zones can be observed, the mesh is not treated in zones. The whole mesh is unstructured containing different element types.

5. DISCRETIZED LEVEL SETS

In the above shown examples, distance functions were given in analytical form, that means a value \( g(x, y) \) can be made available in the whole field. For instance, the distance function to a circular line could easily be computed and lines of constant \( g \)-value were just sets of concentric circles. Although these analytical functions allow some test cases, the obvious objective of the development is to be able to treat arbitrary geometries of any complexity. To overcome the restriction, the level set function \( g \) can be computed in discretized version on a mesh.

Two possibilities exist. The mesh generation system relies in the first step on a given Cartesian mesh to be punched out. Discrete values of \( g \) can be computed on this grid. Remember however, that the level set function is in fact used twice. First to describe the position of the given boundaries and second as a "distance" function for the construction of quadrilateral layers. For the latter task, the level set function can be discretized directly on a developing mesh.

Before proceeding, it is important to know, in which form the surface description is provided. Usually, a polygonal line in 2D or sets of facets from CAD programs are available. However, the discrete points do neither coincide with the nodes of the background grid, nor with the later boundary nodes of a computational mesh.

Discretization on the background grid

The availability of a level set function is at least required to find the correct position of boundary nodes, since it serves as an attractor for these. However, concerning this task there is no need for a good precision
in regions apart from the surfaces. Very crude distance functions would thus be sufficient, as long as the iso-line for the surface of the object is given with reasonable accuracy. Since the functional values must be made available on the nodes, a reasonable resolution of the Cartesian mesh is required. This could be achieved, using an AMR-like refinement strategy.

The computation of the $g$-values at the nodes can be done by a recursive strategy, similar to a solution method for interface tracking on fixed meshes, as previously developed in our work group. Some details of this method are given in a later section below.

The discrete solution of the level set function on the background grid could also be used for the construction of anisotropic layers around the object as well. A slight drawback of this method is however, that a background grid must remain in a certain distance from the objects, rather than being only required to describe the surface.

**Discretization on the developing grid**

Accepting, that a sufficient discretization on the background grid is required for the surfaces, the level set function for the later construction of anisotropic layers relying on the $g$-function can also be discretized directly on the nodes of the developing mesh. This means, the function is recomputed as the mesh develops. The unstructured mesh generation employed here consists of the recursive application of a set of discrete and analog meshing tools. Recomputing the $g$-function would just be a further tool, similar to the relaxation of a mesh density function between given boundaries. Recall, however, the hyperbolic nature of the problem, which coincides with the normal marching process to create the layers.

A drawback of this variant is the necessity to recompute distance functions as the meshing process advances, and that smart storage concepts are required, if several level set functions are used. A variety of different generic data structures is already available from the underlying finite volume library. Advantage is however, that only the surface description requires a background grid.

**Approximate solution of the $g$-function**

The problem for the level set function in the vicinity of the surfaces of objects corresponds to the solution of the equation

$$|\nabla g| = f(x, y)$$

(5)

with a given constant boundary value $g_0$ defined at a line $L$ within the computational domain. For $f(x, y) = \text{const}$, the solution corresponds to the altitude of a sand hill, where the surface line $L$ can be thought to be an outer cut line at constant altitude.

The solution $g(x, y)$ apart the line $L$ is achieved with a point wise iteration process, [6]. This iteration process is split in two parts:

In **Step 1**, a gradient $\nabla g_i$ for a node $i$ is predicted by the following least squares ansatz:

$$\Phi_{\text{par}} = \sum_{j \in E_{\text{par}}(i)} \frac{(g_{i} - g_{j})^2}{w_{\text{par}}(m_{ij})}$$

(6)

where $E_{\text{par}}(i)$ is called a "parabolic" set of influencing neighbors and $m_{ij} = \mathbf{x}_i - \mathbf{x}_j$ is the coordinate difference vector from node $i$ to node $j$. $w_{\text{par}}(m_{ij})$ is a distance weight function. Differentiation and minimization yields the following system (exemplary for 2D):

$$\sum_{j \in E_{\text{par}}(i)} \frac{1}{w_{\text{par}}(m_{ij})} \left( \frac{\Delta x_{ij}^2}{\Delta x_{ij} \Delta y_{ij}} \frac{\Delta x_{ij} \Delta y_{ij}}{(\Delta x_{ij})^2} \right) (g_{i} - g_{j}) = 0$$

$$= \sum_{j \in E_{\text{par}}(i)} \frac{(g_{i} - g_{j})}{w_{\text{par}}(m_{ij})} \left( \frac{\Delta x_{ij}^2}{\Delta x_{ij} \Delta y_{ij}} \frac{\Delta x_{ij} \Delta y_{ij}}{(\Delta x_{ij})^2} \right) (g_{i} - g_{j})$$

(7)

allowing the computation of a one-sided gradient $\nabla g_i^2$ at iteration level $n$.

**Step 2**: Again with least squares, but now projected in the direction of the gradient $\nabla g_i^2$, the following ansatz is obtained:

$$\Phi_{\text{hyp}} = \sum_{j \in E_{\text{hyp}}(i)} \left( \frac{f(x, y) \nabla g_i^2 \cdot m_{ij} - (g_{i} - g_{j}^{n+1})^2}{w_{\text{hyp}}(m_{ij})} \right)$$

(8)

where $E_{\text{hyp}}(i)$ is called a "hyperbolic" set of influencing nodes and $w_{\text{hyp}}$ is again a weighting function.

Considering $\Phi_{\text{hyp}} / \partial g_i^{n+1} = 0$ for minimization and setting $f(x, y) = 1$ for simplicity yields:

$$g_{i}^{n+1} = \frac{\sum_{j \in E_{\text{hyp}}(i)} \frac{w_{\text{hyp}}(m_{ij})}{w_{\text{par}}(m_{ij})} \left( \frac{\nabla g_{i}^n \cdot m_{ij}}{\nabla g_{i}^n \cdot m_{ij}} + g_{j} \right)}{\sum_{j \in E_{\text{hyp}}(i)} w_{\text{hyp}}(m_{ij})^2}$$

(9)

The above formulation allows the iteration for a group of nodes, and a parabolic marching procedure stepping through neighboring layers has been proposed in [6]. The marching process starts at the set of nodes of the first neighborhood level to a given line, iterating up to convergence. It then proceeds from level to level in increasing order. For this marching method, the set $E_{\text{par}}(i)$ contains all nodes $j$ neighboring the node $i$ and being located in the same layer as $i$ or a layer below. The name-index par is thus justified. In contrast, the set $E_{\text{hyp}}(i)$ contains only those neighbors of
in a lower level, which has been previously computed. Both weighting functions were chosen as

\[ w_{\text{par}}(\mathbf{m}_{ij}) = w_{\text{perp}}(\mathbf{m}_{ij}) = |\mathbf{m}_{ij}|^{g}. \tag{10} \]

For the above discussed discretization on the developing mesh, the same strategy is promising. Starting at the boundary nodes located on the line \(L\) and setting \(g = 0\) on these nodes. The sets \(E_{\text{par}}(\mathbf{m})\) and \(E_{\text{perp}}\), as well as the weighting functions \(w_{\text{par}}(\mathbf{m}_{ij})\) and \(w_{\text{perp}}(\mathbf{m}_{ij})\) can be chosen as above.

The discretization on a given Cartesian background grid is more difficult. The reason is, that there’s not necessarily a reasonable matching between the given surface description, for instance the nodes of a polygonal line, and the nodes of the background grid. The development of a suitable method for this task is thus an important topic of current and further developments. A very general solution, employing only clouds of surface nodes seems possible. This would not require any surface connectivity, a big advantage for 3-D problems. If, however, the description of the surface is very accurate, for instance a very fine polygonal line, the solution is rather simple. In this case the crossing points of the polygonal line and the edges of the background grid can be assigned the \(g_n\)-value of the \(g\)-function and the above described marching method can start on these. Similarly the examples in the following sub chapter are obtained.

**Images and other high resolution inputs**

In the end of the previous sub chapter it was stated, that the construction of suitable level-set functions is still an open problem, if only very restricted amounts of data, for instance only few points on a boundary polygon, are provided. Lots of data are however available from images.

Figure 12 shows a small image which served as an input for the mesh generator. To get a discretized level set, equation 5 has to be solved on a background mesh. This could be the natural grid given by the pixel spacing of the image. If this is too large and only a moderate accuracy is desired, a coarser Cartesian grid can be used. \(f(x,y)\) should be a simple function. If \(f(x,y)\) is set to 1.0 in the whole domain the level set will give the distance to the desired surface. The initial position of the zero line can be obtained by comparing all pixel intensities with a threshold value. Figure 13 shows an isotropic grid which has been created using the above image. A single level-set function has been used and the values were discretized on the natural mesh given by the image.

![Figure 13. mesh generated out of image](image)

According to this experiment, a high resolution of a given surface description may always be a suitable input for the generation system. If a solution for a level-set function to describe the surface for low amounts of given data fails, the problem can thus be shifted back to the demand of a reasonable surface resolution of the input. An example is the computation of an airfoil, as shown below:

As input for this airfoil a reasonable resolution of the surface and the surface connectivity as polygonal line were available. In this example, the level-set function has not been solved farther apart the object. Instead, the values are only available in the very near vicinity of the two zero lines describing the upper and lower parts of the airfoil. Figure 14 shows an initial mesh plus two level sets, which are used to describe the geometry of an RAE2822 transonic airfoil. The next figure 15 contains the grid which has been created using the above shown \(g\)-functions. Please note that the grid was intended to be used for an invidious flow computation. The regions around the trailing and leading edge have been isotropically refined in order to resolve the stagnation points.
6. MOVING BOUNDARIES

One big advantage of using level sets as geometry description is the easy handling of moving boundaries. Even geometries, changing in time, would be possible to treat. In this case a transport equation for the scalar level-set function must be used:

$$\frac{\partial G}{\partial t} + \vec{c} \cdot \nabla G = 0$$  \hspace{1cm} (11)

This transport equation is in fact exactly what is used to track discontinuous solutions, while the term $\vec{c} \cdot \nabla G$ represents the local normal propagation speed. In analogy, applied to the boundaries here, the term represents the local normal surface speed.

The test case shown here, however, does not have changing geometries. The $g$-functions are only moved through the grid. Two circular shapes exist in the domain. Both are also meshed in their inside. This is needed to enable these shapes to merge together. The upper circle slowly moves downward in the mesh. In the above equation 11, this corresponds to a constant propagation speed $\vec{c}$.

As the upper circle hits the lower one, both will merge and form a new body. The new shape will be similar to the one used in sections 3 and 4. Figure 16 shows six snapshots of the upper circle moving downward and merging with the lower one. Movement of bodies is realized in a very simple manner. First of all the origin of the $g$-function is shifted. After this is done the same inside-element detection as described in section 3 is performed. This might reduce the amount of mesh-smoothing needed. It becomes essential if two shapes have to merge. In figure 17 the process of merging together can be seen in more detail. State A shows the two circles just before they merge. In state B the situation after moving the level set and detecting inside-elements can be seen. Finally state C shows the shapes fully merged together.

This simple example shows how the meshing algorithm is able to deal with a changing mesh-topology. Also the implementation of parting shapes should be fairly straightforward.

7. VARIABLE QUAD-LAYER THICKNESS

This section intends to show a meshing example with a little more complex geometry. It is composed out of four level sets. Figure 18 shows the zero lines of the $g$-functions used. Quadrangular layers shall only be created for the parabolic shape. Its level set is
described by the following function:

\[ g(x, y) = -\frac{c(y^2 - x)}{cy^2 + 1} \]  

(12)

The constant \( c \) has been set to 3.0 in this example. This function does not fulfill \( |\nabla g| = \text{const} \) in the whole field, \( |\nabla g| \) is increasing with increasing \( x \). This will result in thicker bilinear elements for larger values of \( x \). An example where this can be useful is a boundary layer in a CFD simulation, as the thickness of a boundary layer will increase in the direction of the flow. The gradient of the parabolic level set is:

\[ \frac{\partial g}{\partial x} = \frac{c}{cy^2 + 1}, \quad \frac{\partial g}{\partial y} = -2 \frac{cy(1 + cx)}{(cy^2 + 1)^2} \]  

(13)

Figure 19 shows a plot of this function. This type of controlling the layer thickness by adjusting \( |\nabla g| \) could prove to be very useful for adaptive simulations. Consider for example the above mentioned boundary layer in fluid flow. Starting the simulation with an isotropic grid will lead to high velocity gradients on a wall. These gradients might then be used to adjust the \( g \)-function in the vicinity of the zero-line. For instance, the absolute value of the flow velocity could be used in a direct or indirect way. Created quad-layers would hence be well adapted to the needed resolution. If, during the simulation, these gradients change the corresponding bilinear elements would adapt themselves to the new needs. Figures 20 and 21 show the grid which has been generated with this simple combination of four level sets.

8. FUTURE PERSPECTIVES

As already outlined in the introduction, this meshing method is part of a larger ongoing software development. For the future it is planned to fully integrate the meshing algorithm into this finite volume framework. Thus it would be possible to run adaptive simulations or simulations with moving and changing geometries. This finite volume framework is entirely written in C++, making strong use of object oriented features. Object oriented design will in this case make the integration of meshing, visualization and simulation a
lot easier, compared to classical stand alone numerical codes. Computational efficiency of object oriented programs has a very bad reputation. Although this was true in the past, there exists now a variety of possibilities to overcome this. A good introduction to object oriented numerics can be found in [7]. Object oriented codes gain more and more efficiency and are close to that of FORTRAN codes now (see also [3]). This is also what could be observed in the case of the software mentioned here.

Another interesting field for future research could be the combination of level set based meshing with level set based discontinuity tracking. This would be very useful if anisotropic refinement is required in the vicinity of a discontinuous effect in a simulation. An example is the interaction between air and water flow on a free surface. A small overview of what has been done about discontinuity tracking in the frame of this software development can be found in [6]. An efficient anisotropic and hybrid adaptation algorithm might require more sophisticated conversion techniques to get from one to another element type. An outline how this could be done can be found in [8].

Very important for future applications is the extension of the algorithm to three dimensions. Principally the same approach as in two dimensions should be applicable. The most important difference is the existence of two different boundary singularities (edges and corners). It also has to be investigated if the reconstruction of surfaces is possible in the same manner as described in this article.

REFERENCES


