POST-SIMULATION PROCESSING FOR OVERLAPPING GRIDS
Minh-Duc Huynh and Wilfrid Lefer
University of Pau
France
{minh-duc.huynh,lefer}@univ-pau.fr

ABSTRACT
Nowadays, many engineering processes make intensive use of Computational Fluid Dynamics (CFD) simulations. Aeronautics has been a pioneer sector because numerical simulations not only limit security issues, as compared to “real” simulations, but it allows to drastically decrease both cost and time-to-market. CFD equations, typically Navier-Stokes ones, can only be solved using discrete approximation schemes. Hence the main task that needs to be achieved in order to set up the simulation consists in defining appropriate partitioning of the simulation domain using one or more 3D grids. This set of grids must perfectly fill the 3D area in which the flow is moving, which means that the boundary of the set of grids must be coincident with the boundary of the CAD model. For complex CAD geometries, finding the right set of of grids may be very consuming and even sometimes not possible. The chimera technique consists in allowing the grids to overlap in space, both with other grids and with the CAD geometry. The simulation itself is tuned for this particular configuration but produces results as the set of grids with their respective values. This raises new problems when the results have to be post-processed or for their visualization because the tools that are used for this purpose are not adapted to overlapping grids.

In this paper we present a post-processing technique for chimera simulation results, which aims at generating a multi-block grid featuring no overlap. Our method starts by hollowing out one of the overlapping grids and then uses constrained Delaunay tetrahedralization for filling the gap between grids, so that no hole remains in the resulting multi-block solution.

KEY WORDS
CFD simulations, chimera technique, computational geometry, volumetric meshes, constrained Delaunay tetrahedralization.

1. Introduction

The aeronautics sector makes intensive use of numerical CFD simulations for engine design. Examples include aircraft and gas turbine design for which numerical simulations can be run as many times as required, allowing simulation of the engine behavior for a multitude of operating conditions. An important issue is how to interpret reliably and effectively the results of these simulations. Scientific visualization involves the computation of a visual representation of the data characterizing the simulation: CAD models, temperature, pressure, velocity, fluid flow, etc. This representation should allow engineers to appreciate the dynamic behavior of the airplane in diverse flight situations or when the engine faces extreme conditions, such as very hot or very humid, by offering the possibility to interact with the visualization, such as change point of view, change parameters of the visual representation, trajectory to compute a virtual navigation, etc. This aims at optimizing engine shape, particularly so as to avoid heating phenomena that would lead on excessive degradation of material properties. For gaz turbines, high temperatures significantly modify behavior and resistance, running the risk of rupture, with extremely dramatic consequences in the case of a helicopter.

Fluid flow modeling is governed by the Navier-Stokes equations, for which an approximated solution is computed. These methods rest on a discretization of the 3D space according to a more or less regular grid. The resolution of the grid evidently has an impact on the quality of the numerical solution: the grid that is finer and better is the approximation, but over the cost of computation is important because it is proportional to the number of sampled points. To find a good compromise between computational cost and solution quality, engineers in fluid dynamics generally proceed as follows. They calculate a first solution based on a coarse grid and then identify the zones of high turbulence (the significant turbulences are created in certain key locations), in which the error is the most important because the fluid changes the direction at a higher frequency in the sampling frequency. The turbulences are therefore necessary to simulate with a much finer grid to fully apprehend the phenomena that occur there. They define a new grid of a higher resolution, called the fine grid, which only covers the area of turbulence. Limit conditions are extracted from the coarse grid and injected into the fine grid simulation. A new simulation is run in the region of interest. And so on, they are able to properly capture the behavior of the fluid. This process is perfectly supported by the chimera approach, which provides a number of techniques to deal with overlapping grids adequately [1][2].

Results of chimera simulations typically consist of several overlapping grids covering the simulation domain (see Figure 1). Post-processing of such a CFD solution raises issues where several grids overlap. Computing values in overlapping regions is currently achieved by two different techniques:
Weighting, which principle consists in using coefficients to weight the values coming from each grid, thus defining the contribution of each cell to the computed value. The computation of these coefficients is based on a polygon clipping algorithm applied to the overlapping cells. This technique is used in the USURP software [3].

Reconstruction of a multi-block grid featuring coincident contact surfaces between adjacent blocks, i.e. with no overlaps. This technique is used for 2D surface grids in the FOMOCO software suite and especially the MIXUR module [4].

The main drawback of the weighting approach is that any post-processing technique needs to be adapted to the specific geometry of the computational domain, whereas reconstructing a multi-block domain allows using any post-processing tool, assuming the data format is supported of course. Particularly, there is a large offer of visualization tools to help scientists analyze simulation results, including Open Source ones, thus providing a large number of methods to drill into the always-growing simulation data. In this work, we target ParaView, an Open Source, client-server, parallel, solution for scientific visualization, extensible through a plug-in concept [5][6].

The purpose of this work was to develop a ParaView plug-in for the reconstruction of a manifold domain for post-processing of chimera simulations of gas turbine aerothermics.

Reconstruction of a manifold 3D grid from several overlapping, structured or unstructured, grids, has not been addressed widely in the CFD simulation community. For instance, the Chimera Grid Tools (CGT) library does not provide such functionality [7]. The zipper algorithm provided by the Post tool of ONERA's Elsa CFD simulator provides it but is restricted to surface grids [8]. As stated above, the FOMOCO software suite also implements this technique for 2D surfaces only [4].

The 2D case consists in triangulating a gap between 2 piecewise linear contours, which can be achieved by a step-by-step algorithm connecting vertices of the different contours so as to build non-overlapping triangles [3][8]. The problem is rather more complicated in 3D and such iterative method is far from being straightforward because there is not a unique direction for moving from one triangle to the next. Moreover we deal with tetrahedra rather than with triangles and ensuring that there is no gap in the middle of a tetrahedral decomposition of a domain is a complex issue.

Delaunay triangulations have been widely used for the decomposition of 2D and 3D domains in a collection of gap free simplices. Moreover, such decompositions feature nice properties - minimum edge angle maximization in 2D - for subsequent computations over the reconstructed domain, such as streamline generation, because they tend to minimize approximation errors. Since we have chosen to use this class of methods for computing a manifold multi-blocks 3D mesh, this rest of this background section is concerned with Delaunay triangulations.

2. Background

Two- and three-dimensional Delaunay triangulations are well-known geometric arrangements in computational geometry, which play a central role in many applications, such as mesh generation, rendering and visualization, interpolation and numerical methods for physical simulations such as heat transfer and fluid flow. A mesh is usually constructed using Delaunay triangulations from a finite set of vertices. However, in fact, there are a lot of meshes are generated not only from the set of vertices, but they also must satisfy a collection of segments, and faces called constraints (or sometimes called constraining boundaries). Then Delaunay triangulations satisfy these constraints that are called constrained Delaunay triangulations.

2.1. Constraining Boundary

In two dimensions, the concept of planar straight line graph (PSLG) in the computational geometry is often used to refer to the constraining boundaries of two-dimensional domains. A PSLG is a finite set of 0D-faces (vertices) and 1D-faces (segments) satisfying two conditions.

- Both endpoints of each segment of the PSLG must be in the PSLG.
- The intersection between two segments of the PSLG or between a segment and a vertex of the PSLG occurs only at the endpoints of the segment.

For the three-dimensional domains, the constraining boundaries are often described as a piecewise linear complex (PLC) [9]. A PLC consists of a finite collection of 0D-faces (vertices), 1D-faces (segments), and 2D-faces (facets). A PLC has properties similar to those of a PSLG:

- The boundary components of each face in the PLC are also the lower-dimensional faces in the PLC.
• The PLCs must be closed under intersections: all nonempty intersections between faces in the PLC must be the lower-dimensional faces in the PLC.

Figure 2: Constraining boundaries. (a) A PSLG. (b) A PLC: a cube with a hole in its center. (c) The Schönhardt polyhedron.

Figure 2 shows a planar straight line graph, a piecewise linear complex and a Schönhardt polyhedron. With the properties as above, we have seen that the polyhedrons are just a particular type of the PLCs because every polyhedron is a PLC.

Once the constrained Delaunay triangulations that were constructed from the constraining boundaries, these boundaries were part of the constrained Delaunay triangulations.

Figure 3: Constrained Delaunay triangulations. The constrained Delaunay triangulation (a), the conforming Delaunay triangulation (b), the conforming constrained Delaunay triangulation (c) and the quality Delaunay triangulation (d) of the PSLG represented in Figure 2(a).

2.2. Constrained Delaunay Triangulation

A constrained Delaunay triangulation is a variation of a Delaunay triangulation and is considered as a Delaunay-like triangulation because it has the properties similar to the Delaunay triangulation. In three dimensions, a constrained Delaunay triangulation is also known as a constrained Delaunay tetrahedralization. For the constrained Delaunay triangulation, as well as constrained Delaunay tetrahedralization, we introduce the following four types. Figure 3 shows four types of constrained Delaunay triangulation, and Figure 4 shows the constrained Delaunay tetrahedralization and the quality Delaunay tetrahedralization.

Figure 4: Constrained Delaunay tetrahedralizations. The constrained Delaunay tetrahedralization (a, b) and the quality Delaunay tetrahedralization (c, d) of the PLC represented in Figure 2(b).

The conforming Delaunay triangulation [10][11][12] is a triangulation that conforms to all the constraining boundary segments of the domain by inserting additional vertices (called Steiner points) into the boundary. In the conforming Delaunay triangulation, each boundary segment is the union of edges of the triangulation and all simplices satisfy the Delaunay criterion. Such the conforming Delaunay triangulation is exactly a Delaunay triangulation.

Unlike the conforming Delaunay triangulation, the constraining boundary segments in the constrained Delaunay triangulation [13][14][15] are kept and not separate into smaller edges. For the constrained Delaunay triangulation, every boundary segment is a single edge of the triangulation and not all the simplices satisfy the Delaunay criterion. The constrained Delaunay triangulation is not a true Delaunay triangulation but it keeps many of the properties of Delaunay triangulations. In three dimensions, there are some PLCs in which the constrained Delaunay tetrahedralization could not be generated and one of them is the Schönhardt polyhedron [16] (see Figure 2(c)). In this case, we must use the conforming Delaunay tetrahedralization to generate the mesh and Shewchuk has presented a condition to guarantee the existence of the constrained Delaunay tetrahedralizations [17].
Shewchuk has also introduced the conforming constrained Delaunay triangulation \[15\][17] because it is both the conforming Delaunay triangulation and the constrained Delaunay triangulation. The last type of the constrained Delaunay triangulation is the quality Delaunay triangulation \[18\][19]. A quality Delaunay triangulation is actually a special type of a conforming constrained Delaunay triangulation (also of a conforming Delaunay triangulation) because it is a refinement of a constrained Delaunay triangulation in which it must agree with the constraints on the angles (no small or large angles) or the radius-edge ratio of the simplices. Then the quality Delaunay triangulations are suitable for finite element methods.

3. Proposed Solution

In our application, the chimera technique that generate a multi-block grid for simulating fluid flow in a gas turbine and visualizing numerical simulation results are performed on the 3D grid models. The topologies of grid may be varied such as regular grid, rectilinear grid, curvilinear grid (structured grid), and unstructured grid (see Figure 6). In this paper, we would like to introduce a model including two grids. As main grid for the chimera technique, we call C the coarse grid which is typically curvilinear and as sub-grid, F the fine grid which is generally regular Cartesian. However, the model may be generalized into N grids that consist of one coarse grid C, which is regular, rectilinear, curvilinear or unstructured, and many find grids F.

3.1. Post-processing Pipeline

For merging two grids that overlap, we have implemented in two phases.
- First, we have looked for part of C that intersects F and removed it.
- Secondly, after removing this part in C, a gap will appear between the two grids and in order to fill this gap, we have constructed a new grid for joining C and F, which we call the gap grid G. This G is constructed as an unstructured grid with the good properties (see Figure 5).

The construction of G is achieved in two phases. First, we build the boundary $\partial G$ of G from the facets of cells of C and the facets of cells of F. $\partial G$ is a piecewise linear complex. Then, a mesh is generated by the constrained Delaunay tetrahedralization from constraining boundary $\partial G$.

3.2. Gap Boundary Construction

A constraining boundary of a three-dimensional domain that is represented by a piecewise linear complex is more general than the boundary that is represented by the other three-dimensional geometric solid such as polyhedron. The piecewise linear complex may contain the holes, the slits and the isolated vertices inside it, and also inside its facets. The facets of the piecewise linear complex are arbitrary polygons which cross at the shared vertices or the shared segments and they may be non-convex. We have used the piecewise linear complex to construct $\partial G$ of G. Because this boundary consists of two parts which contact C and F. So we find the first part that contacts C, and then the second part that contacts F.
3.2.1. A Coarse Grid Extrusion and Boundary Construction

Before getting ∂G, we have to remove the intersection between C and F by finding and erasing all cells C_d of C which intersect F. For finding these C_d, we get all cells C_s of C and check the relative position between all vertices and facets of C_c and F. If there is at least one facet of C_c that is in F or at least one facet of C_c which intersects F then C_c is C_d.

We call ∂G_c is part of ∂G that contacts C. The determination of ∂G_c does not depend on the position between C and F and it is based on C_d. Before deleting one C_d, we take all facets f_s of C_d which do not contain any vertices of C_d inside F and do not intersect F. Every f is a facet of ∂G_c and the set of all f_s form ∂G_c. The algorithm for deleting C_c and building ∂G_c is:

**Input:**
Coarse grid C
Fine grid F

**Output:**
∂G_c = (∂G_c(v): vertex, ∂G_c(f): facet)

Begin
∂G_c ← ∅
For Each cell C_c of C Do
N_c = number of vertices of C_c
N_i = number of vertices of C_c inside F
If N_i > 0 Then
If N_i < (N_c - 2) Then
S_i ← set of facets of C_c
For Each vertex v of C_c Do
If v is inside F Then
S_i ← S_i + incident facets of v
Else
∂G_c ← ∂G_c + v
EndIf
EndFor Each
For Each facet f of S_i Do
If f intersects F Then
S_i ← S_i – f
EndIf
EndFor Each
∂G_c ← ∂G_c + S_i
EndIf
C_d ← C_c
Delete C_c
EndFor Each
End

3.2.2. Fine Grid Boundary Construction

Depending on the position between C and F, part ∂G_f of ∂G in which ∂G_f contacts F is boundary ∂F of F or just part of ∂F. To determine the ∂G_f we have the following cases:

If F is inside C then ∂G_f is ∂F. For extracting ∂F and the boundary ∂C of C we have used the boundary extraction method that was given by VTK lib. If C contains part of ∂F, then ∂G_f is part of ∂F. ∂G_f is a collection of facets of ∂F that are in C. So we checked the position between facets f_s of ∂F and C. Note that F is regular grid then f_s is rectangular or square. If f_s is inside C then it is a facet of ∂G_f. In the case that f_s cross the boundary ∂C of C, we determine the points of intersection of ∂C and segments of f_s using an approximate method is described in three steps as follows. Given AB segment of f_s that cuts ∂C in which A is vertex outside C and B is in C:

- Step 1: Take the midpoint D of AB
- Step 2: Check the distance d between D and ∂C. If d is smaller than ε then D is the point of intersection of AB and ∂C. If d is greater than ε, then we move to step 3
- Step 3: Check the position of D and C. If D is in C, we take the midpoint D_1 of AD otherwise we take the midpoint D_1 of the BD and return to step 2.

The algorithm for finding the point of intersection between AB and ∂C is given at the end of the section 3 (end of page 6).

The points of intersection of ∂C and f_s and all vertices of f_s that are inside C form a facet of ∂G_f. The algorithm for building ∂G_f is:

**Input:**
Coarse grid C
Fine grid F

**Output:**
∂G_f = (∂G_f(v): vertex, ∂G_f(f): facet)

Begin
∂G_f ← ∅
P = set of points
For Each face f_f of ∂F Do
N_f = number of vertices of f_f
N_i = number of vertices of f_f inside C
If N_i = N_f Then
∂G_f ← ∂G_f + all vertices v of f_f
∂G_f ← ∂G_f + f_f
ElseIf (0 < N_i < N_f) Then
P ← ∅
For Each segment AB of f_f Do
If AB intersects ∂C Then
P ← Intersection(∂C, AB, ε)
EndIf
EndIf
EndFor Each
P ← vertex of f_f inside C
Create a facet f_f from P
∂G_f ← ∂G_f + P
∂G_f ← ∂G_f + f_f
EndIf
EndFor Each
End
3.2.3. Boundary Closing

If \( F \) is inside \( C \) then \( \partial G \) that is combined by \( \partial G_c \) and \( \partial G_f \) is closed, and otherwise \( \partial G \) is not closed. Set of additional facets \( \partial G_a \) of \( \partial G \) are determined based on facet \( f_c \) of \( \partial C \). To find additional facets \( \partial G_a \), we get all facets \( f_c \)s of \( \partial C \) which are inside \( F \) or intersect \( \partial F \). These \( f_c \)s are also facets of deleted cells \( C_s \) of \( C \) that belong to \( \partial C \). If \( f_c \) is in \( F \) then it is also a hole. In the case that \( f_c \) intersects \( \partial F \) then it is split into parts by \( \partial F \) in which one inside \( F \) is a hole of \( f_c \). This hole of \( f_c \) is determined by all vertices of \( f_c \) that are in \( F \) and all points of intersection \( P_s \) of \( f_c \) and \( \partial F \). Each \( P \) may be found by using algorithm \text{point of intersection}. \( \partial G \) which is the union of \( \partial G_c \), \( \partial G_f \), and \( \partial G_a \) is closed. \( \partial G \) with the hole that is bounded by \( F \) is a piecewise linear complex. The algorithm for building \( \partial G \) is:

\begin{verbatim}
Input:
Set of deleted cells \( C_s \) of \( C \)
Boundary \( \partial C \) of \( C \)
Fine grid \( F \)
Output:
\( \partial G_a = (\partial G_a(v): \text{vertex}, \partial G_a(f, h): \text{facet, hole}) \)
Begin
\( \partial G_a \leftarrow \emptyset \)
\( P \leftarrow \text{set of points} \)
ForEach cell \( C_d \) Do
   ForEach facet \( f_c \) of \( C_d \) Do
      If \( f_c \) is facet of \( \partial C \) Then
         \( \partial G_a \leftarrow \partial G_a + \text{all vertices } v \text{ of } f_c \)
      Else
         \( \partial G_a \leftarrow \partial G_a + f_c \) // \( f_c \) is a hole
      EndIf
   EndForEach
   P \leftarrow \text{set of points} \)
   ForEach segment \( AB \) of \( f_c \) Do
      If \( AB \) intersects \( \partial F \) Then
         \( P \leftarrow \text{Intersection}(\partial F, AB, \varepsilon) \)
      EndIf
   EndForEach
   P \leftarrow \text{vertex of } f_c \text{ inside } F
   Create a hole \( h \) of \( f_c \) from \( P \)
   \( \partial G_a \leftarrow \partial G_a + P \)
   \( \partial G_a \leftarrow \partial G_a + h \)
EndIf
EndForEach
End
\end{verbatim}

3.3. Gap Tetrahedralization

In this paper, we have used constrained Delaunay tetrahedralizations to construct gap grid when there are not any constraints that are imposed on the gap cells such as minimum radius-edge ratio, maximum volume constraint. Method for meshing a piecewise linear complex by using the constrained Delaunay tetrahedralization that we have used here is the method given by Si and Gärtner [20]. This method always guarantees the existence of the constrained Delaunay tetrahedralization of any piecewise linear complex. In the case that the constrained Delaunay tetrahedralization may not be generated it modifies the input piecewise linear complex by inserting few Steiner points into the segments (splitting segments) or reversing the vertex. New piecewise linear complex has the same geometry and topology of the input piecewise linear complex. Besides it may also recover the missing facets (segments and triangles) of the constrained Delaunay tetrahedralization. In addition the quality Delaunay tetrahedralization is also used to refine the gap cells when they must satisfy the given constraints such as their edge ratio threshold is less than or equal to a given number. To generate quality Delaunay tetrahedralization, we use the method introduced by Si [19][21]. This method is an extension of the basic Delaunay refinement method was proposed. It generates an isotropic mesh that corresponds to a sizing function. This sizing function is specified by the user or automatically derived from the geometric data and good mesh may be obtained for smoothly changing sizing information.

Both of these methods and their algorithms have been implemented in the program TetGen [22]. TetGen may be used to generate the pure constrained Delaunay tetrahedralizations, the conforming Delaunay tetrahedralizations and the Voronoi tessellation for any 3D domains, and the quality Delaunay tetrahedralizations for solving many problems of the finite element and finite volume methods.

\text{Function Intersection}(\partial C, AB, \varepsilon)
\begin{verbatim}
Input:
Boundary \( \partial C \) of \( C \)
Segment \( AB \), such as \( A \) is outside \( C \) and \( B \) is in \( C \)
Number \( \varepsilon \)
Output:
Point of intersection \( P \)
Begin
\( \text{stop} \leftarrow \text{false} \)
\( D \leftarrow \text{midpoint of } AB \)
\( d \leftarrow \text{distance between } D \text{ and } \partial C \)
If \( d < \varepsilon \) Then
   \( P \leftarrow D \)
Else
   If \( D \) is inside \( C \) Then
      \( A \leftarrow D \)
   Else
      \( B \leftarrow D \)
EndIf
EndIf
While (\text{stop} = \text{false})
End
\end{verbatim}
4. Implementation and Results

Our algorithms have been implemented as a plugin of ParaView [5]. We also use TetGen [22] for the tetrahedralization of the gap from the gap boundary, a piecewise linear complex.

How our plug-in operates is described in three steps (see Figure 7):

- Step 1: VTK object reads datasets input that are coarse grid and fine grid datasets, then creates gap boundary data and new coarse grid dataset by deleting all cells which intersect fine grid. Fine grid is unchanged.
- Step 2: TetGen reads gap boundary data includes arrays of vertices, constraining facets, and holes, then generates gap grid and returns gap grid data to VTK object.
- Step 3: VTK object creates gap grid dataset, then it combines coarse grid, fine grid and gap grid datasets into a multi-block dataset output.

In order Tetgen can read and write VTK data, we have defined two new methods for TetGen that are readVTKData and writeVTKData methods. readVTKData is a method that allows TetGen can read all VTK data and writeVTKData allows TetGen can write all its data as VTK data.

Supported grids topologies include regular, rectilinear, and unstructured grids, thus making our software able to post-process almost any Chimera simulation solution. There is no constraint on the relative positions of the different grids in the 3D space. Figure 8 shows two unstructured grids of different granularity that are overlapping. It turns out that the gap created by hollowing out the coarse grid is the same for both configurations – it could have been different although since the tetrahedra on Figure 8 have been obtained by decomposing each original cube independently, it is likely to lead to the same gap shape. Figure 8 demonstrates two important properties of our solution:

- The new block perfectly fills the gap between the coarse and the fine grid, that is there is no remaining gap in between,
- The new tetrahedralization is “Delaunay optimal”, thus making it a good candidate for further manipulations, such as streamline computation.

5. Conclusion and Future Work

We see that constrained Delaunay tetrahedralization is an important geometric structure in computational geometry, which has many applications in a wide variety of fields of computer graphics and computer-aided design, and especially for mesh generation. Recently it has also been used for tracing a ray through a scene [23]. We have solved our problem by using the constrained Delaunay tetrahedralization to fill the gap between create a gap mesh. The results that we obtained show that we have chosen constrained Delaunay tetrahedralization among geometry structures is reasonable because it is easy to use and can be applied to many different topologies such as regular grid, rectilinear grid and unstructured grid. It also demonstrates the stability of the gap grid because it does not depend on the topology of the input grids and just depends on its constraining boundary. Grids are merged...
using a constrained Delaunay tetrahedralization, which guarantee the simulation results of fluid flow that are processed on them retain the stability as the initial results. Besides, the operation of this implementation seems to be flexible, fast and powerful.

The expansion of this application, first we implement it on the other topologies such as curvilinear grid, unstructured grid whose cells are parallelepiped, secondly we extend to cases that the number of overlapping grids is greater two. Moreover these grids are used for fluid flow simulation and visualization of simulation results so that a problem is posed: in addition to merging all grids we have to add new fluid flow properties such as the pressure, the temperature, the velocity on gap grid. Finally that is evaluating the performance of the constrained Delaunay tetrahedralization and applications on a variety of the topologies and sizes of input grids.

References