A Tetra-hex Hybrid Mesh Generation Method Based on Delaunay Triangulation

Pengfei Zhan¹,², Xianhai Meng¹,²,*, Zhongxiang Duan¹,² and Qin Yang¹,²
¹School of Computer Science and Engineering, Beihang University (BUAA), Beijing 100191, PR China
²State Key Laboratory of Software Development Environment, Beijing 100191, PR China
*Corresponding author

Abstract—This paper presents an automatic method to generate a hex-dominant hybrid mesh with the input being a Delaunay tetrahedral mesh. In this method, we replace the interior tetrahedra in the original mesh with regular hexahedra, and fill the cavity between the boundary tetrahedral and the interior structured elements with tetrahedral elements. The mesh generated by this method is a tetra-hex hybrid mesh which only consists of two types of grids. The result demonstrated that our method can perform on most tetrahedral mesh successfully. The total number of grids in the hybrid mesh is reduced markedly, and moreover, the resultant mesh can keep the conformity of the boundary and consist of grids with high quality.

Keywords—hybrid mesh; advancing front technique; delaunay triangulation

I. INTRODUCTION

Owing to the development of computing hardware, many numerical simulation methods, such as finite element method (FEM), have been widely used in industry. However, as an essential preprocessing step, the mesh generation directly impacts on the accuracy and convergence of the simulation [1]. Currently, the generated mesh can be roughly classified into the structured and unstructured mesh, while the simulation on the structured grids require less computational memory and cost [2] and the unstructured mesh can adapt to the complicated boundaries of the domain with arbitrary shape [3][4]. Since the hybrid grids combine the advantages of both unstructured and structured mesh, the methods for the hybrid grid generation have received much attention in recent decades.

The basic elements in hybrid grids include tetrahedra, prisms, hexahedrons, and pyramids. The researchers usually use two or more of basic elements above to create a hybrid mesh. For instance, the hybrid grids consisting of prismatic and tetrahedral elements were first proposed by Nakahashi [5], and developed extensively by Kallinderis et al. [6]. In 2009, Yamakawa and Shimada [7] presented a new method of prism-tetra hybrid mesh generation that replaced tetrahedral elements filling a sweepable volume with prismatic elements. This method effectively reduced the number of elements, and the resultant hybrid mesh yielded a more accurate solution than a tetrahedral mesh. In addition, the hex-dominant hybrid mesh [8] is also a popular type of hybrid mesh. Steven J. Owen [9] presented a hex-dominant mesh generation method which guarantees the boundary conformity using 3D constrained triangulation. In the work of Jesse Chan et al.[10], they create the hex-dominant hybrid mesh of hexahedra, wedge, pyramid, and tetrahedral with the same size by taking every regular hexahedron as the reference hexahedron. In most cases, the purpose of most hybrid mesh generation methods is to reduce the computational cost and to keep the computational accuracy as well.

In this paper, we propose an automatic method to convert a Delaunay tetrahedral mesh to a tetra-hex hybrid mesh. In our method, the interior tetrahedra in the original mesh are replaced with regular hexahedra, and the tetrahedra near the boundary remain. The hybrid mesh generated by our method can keep the conformity of the boundary and consist of grids with high quality. Moreover, the total number of grids can be markedly reduced.

II. METHOD FOR HYBRID GRID GENERATION

In this section, we introduce our algorithm to generate the tetra-hex hybrid mesh from the Delaunay tetrahedral mesh. The input of our algorithm is a Delaunay tetrahedral mesh with uniform size and high quality and we intend to fill the domain with structured elements inside and tetrahedra around the boundary. The main difficulty is to fill the cavity between boundary tetrahedral elements and inner structured elements. In our method, we design a modified advancing front Delaunay triangulation algorithm [11] to address this issue and we present the algorithm in the rest of this section.

A. Advancing front Initialization

Let $TeS_i$ denote the boundary tetrahedral set, where every tetrahedron in $TeS_i$ contains at least one vertex on the boundary. The initial front $TS_{ori}$ is the inner triangle faces of these tetrahedra which are stored in a linked list. Denote one single front triangle face as $T_n$ $(0 \leq n \leq n-1)$ and $n$ is the number of the front faces.

B. Hexahedron Insertion

In this step, the domain inside the initial front is filled with regular hexahedra $Cube_{ori}$. However, the insertion process has an important principle that the interior hexahedral grids need keep a distance with the initial front to avoid too many silvers generated. We use the extra bounding box to control the
Algorithm Choosing Process

1. Create an initial search region
2. Create an initial tetrahedron
3. While the list of the finishing surface is not empty
   (a) Choose first triangle from the list
   (b) If the list is empty, then return
   (c) Otherwise, choose another point
   (d) Add the chosen point to the list

C. Tetrahedron insertion

After the pretreatment above, we get an initial front and a finishing surface set. Then, the empty region between the front and the finishing surface set needs to be filled with new tetrahedra. Firstly, a list of candidate nodes is formed from all the nodes lying on the front and the finishing surface set. For the first triangle on the initial front, choose a best point from a Delaunay rule, which is known as the empty circumsphere criterion. We create the new tetrahedron in the empty region.

The algorithm of choosing a point satisfying the empty circumsphere criterion is shown as follows:

Algorithm Choosing Process($PS_0$, $Tri_0$, $P_0$

Input: the set of candidate points $PS_0$;
the first triangle $Tri_0$ on the front;
Output: the point $P_0$ satisfying the empty circumsphere criterion;
1. If $PS_0$ is empty, then return NULL.
2. Create an initial search region $D_1$ for $Tri_0$ (Fig 1), and then points inside $D_1$ forms a sub-candidate pointset $PS_{sub}$.
3. If $PS_{sub}$ is empty, then return NULL.
   Otherwise, $PS_{sub}$ contains at least one point.
   (a) If $PS_{sub}$ contains only one point $P_0$, then mark $P_0$ to be $P_0$, return $P_0$.
   (b) Otherwise, choose the point $P_0$ satisfying the empty circumsphere criterion from $PS_{sub}$.
4. return $P_0$.

In this algorithm above, we create a provisional circumsphere for $Tri_0$ for the purpose of removing the candidate points which is on the opposite direction or too far from $Tri_0$, so the distance $d$ in Fig 1 satisfies $d = R_c$. Moreover, if the output of this algorithm is NULL, we can insert a new point $P_1$ in a reasonable position, and use $P_1$ and $Tri_0$ to create the new tetrahedron.

After a new tetrahedron $Te_N$ has been created, $Te_N$ will be searched for any collisions with the $TS_{ori}$ or $FS_{hex}$. If $Te_N$ is intersected with any facet, this new tetrahedron will be discarded, and we will choose another point $P_0$ for $Tri_0$. If $Te_N$ passes the collision detection, the initial front $TS_{ori}$ will be updated by the new three triangles $T_{new}$ on $Te_N$. The result of front update is divided into three conditions as follows:

(a) If $TS_{ori}$ contains $T_{new}$, then remove $T_{new}$ from $TS_{ori}$.
(b) If $T_{new}$ overlaps with $FS_{hex}$, then do nothing.
(c) If $T_{new}$ overlaps with no facet of $TS_{ori}$ and $FS_{hex}$, then add $T_{new}$ into $TS_{ori}$.

Choose the first triangle $Tri_0$ from $TS_{ori}$ and repeat the processes above, until the $TS_{ori}$ is empty. Finally, we yield a set of new tetrahedra $Te_{Sh}$ between the front and the finishing surface. The whole algorithm of tetrahedron insertion is shown as follows:

Algorithm Tetrahedron Insertion($TS_{ori}$, $FS_{hex}$, $Te_{Sh}$)

Input: the list of the initial front $TS_{ori}$;
the set of the finishing surface $FS_{hex}$;
Output: the set of new tetrahedra $Te_{Sh}$;
1. Create $PS_0$ by $TS_{ori}$ and $FS_{hex}$.
2. While $TS_{ori}$ ≠ Ø
   (a) Choose first triangle $TS_{ori}$ ∈ $TS_{ori}$ and call
   Choosing Process($PS_i$, $Tri_i$, $P_i$).
   (b) If $P_0$ = NULL, then add a new point $P_i$ to $PS_i$, and mark $P_i$ to be $P_0$.
   (c) Create $Te_N$ by $P_0$ and $Tri_0$.
   (d) If $Te_N$ fails the intersection test, then discard $Te_N$, and choose another point $P_0$’; use $P_0$’ and $Tri_0$ to create new tetrahedron $Te_N$’.
   (e) Add $Te_N$ or $Te_N$’ to $Te_{Sh}$.
   (f) Update $TS_{ori}$.

End while.
3. return $Te_{Sh}$.

III. Algorithm Analysis

A. Termination

This method is an automatic hybrid mesh generation method which can terminate in a finite time. In our method, two issues may bother the process. The first problem is that the first-circumscribed sphere of $Tri_0$ contains no candidate point, and another one is that $Te_N$ fails the intersection test. We can solve the first problem by adding a suitable point $P_i$ into the cavity and choose $P_i$ as the fittest candidate point.

The second one can also be solved by the step of choosing fittest candidate point. In practice, this step is simple as it puts points on the triangles, which are adjacent to $Tri_0$, into sub-candidate pointset $PS_{sub}$, and choose the fittest point $P_i$ from $PS_{sub}$. There must be at least one point that is in the forward direction of $Tri_0$, otherwise no triangular facet can intersect with the previous tetrahedron $Te_N$.
there is a point $P_i$ inside the new tetrahedron $Te_{Ni}$, the new tetrahedron $Te_{Ni}$ can surely pass the intersection test this time. Even if $P_i$ exists, we can choose $P_i$ as the fittest point at first time. As a result, it is certain to get a suitable point $P_0$ that can create a new tetrahedron which can pass the intersection test. Finally, all the triangular faces can connect to $FS_{hex}$, and the algorithm terminates.

### B. Algorithm complexity

There are two main parts in this method, the pretreatment and the process of tetrahedron insertion. In the pretreatment, we mainly traverse the set of all the input tetrahedra or all the hexahedra inserted, and use every hexahedra to do intersection test with the initial front, so the algorithm complexity is $O(Num_{tetra})$ and $O(Num_{cube} \times Num_{front})$.

However, the main process is an iterative process, and the algorithm complexity depends on the total number of input tetrahedra. At each iteration, we need create a candidate subset, using $O(Num_{pset})$, and then choose the fittest point, using $O(Num_{subpset})$ or $O(Num_{subpset}^2)$ in the worst case. While a new tetrahedron is created, we will use it to do intersection test with the front and the surface of inner hexahedra. The algorithm complexity of each intersection test is $O(Num_{front} + Num_{hex})$. With the increasing number of the total elements in the mesh, it will take more iterations to complete the hybrid mesh generation. For example, when the number of elements in the resultant mesh is 16905, it needs 9246 iterations. However, when the number of elements in the resultant mesh is 31584, it needs 14104 iterations to terminate.

### IV. EXPERIMENTAL RESULTS

We present several mesh examples of our hybrid grid generation algorithm in Figure 2 and the statistic on the reduced quantity of the grid elements by our method is shown in Table 1. In the examples, we choose the hexahedra edge length $l_{hex}$ to be 1.20 $l_{\min}$.

![FIGURE II. EXAMPLES OF HYBRID MESH, ARCH (UPPER LEFT), SPHERE (UPPER RIGHT), DUMBELL (LOWER LEFT), GRAIL (LOW RIGHT).](image1)

<table>
<thead>
<tr>
<th>Number of grids</th>
<th>Tetrahedral mesh</th>
<th>Resultant hybrid mesh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arch</td>
<td>57662</td>
<td>29948 1636 31584 45%</td>
</tr>
<tr>
<td>Sphere</td>
<td>65947</td>
<td>25053 3581 28634 56%</td>
</tr>
<tr>
<td>Dumbbell</td>
<td>50176</td>
<td>25144 1583 26727 46%</td>
</tr>
<tr>
<td>Grail</td>
<td>30888</td>
<td>18557 432 18989 38%</td>
</tr>
</tbody>
</table>

a. $H$ is the number of the grids reduced, and $T$ is the number of tetrahedral mesh.

![FIGURE III. THE GRID OF ARCH WITH DIFFERENT HEXAHEDRA SIZE. FROM LEFT TO RIGHT: LHEX EQUALS TO 1.00/1.15/1.30 LMIN.](image2)

![FIGURE IV. THE RATIO OF GRIDS REDUCED ON DIFFERENT SIZE OF HEXAHEDRA EDGE. (LHEX EQUALS TO X * LMIN, AND X IS THE VALUE OF X-AXIS.).](image3)

It is shown that our method can truly generate the qualified grids and decrease the number of the grid elements, especially the example of Sphere, where the grids can be reduced by more than a half. Although the hexahedra are much less than tetrahedra in each example, they play an important role in reducing total elements number as they not only can replace the tetrahedra, but also can reduce the number of points inside the mesh.

The proportions of the grid elements reduced when the hexahedra size is different are shown in Fig. 3 and Fig. 4. As shown from the line chart, as the size of hexahedra extends, the more grids are reduced. The reason for this phenomenon is that the extension of the hexahedral size decreases the number of the interior hexahedra and the candidate points. Besides, as we can see, our method always reduced more elements for the example Sphere compared to the Arch. This indicates that when the models contain larger cavities inside, the algorithm performs better on the reducing of the mesh quantity.

### V. CONCLUSION

In this paper we have presented an automatic hybrid generation mesh method with the input being a Delaunay tetrahedral mesh. This method replaces the interior tetrahedra with regular hexahedra and in the transition region, and it
connects two triangular faces to a quadrilateral face. Almost all tetrahedral mesh can be converted to a tetra-hex hybrid mesh by this method. The elements number of resultant mesh is reduced significantly and this method can perform much better on the model with a large cavity.

ACKNOWLEDGMENT

National Natural Science Foundation of China (Grant 61003110) and Fund of the State Key Laboratory of Software Development Environment (Grant SKLSDE-2010ZX-10) jointly supported this work.

REFERENCES