Computer-Aided Design 67–68 (2015) 107–117

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

# Quality guaranteed all-hex mesh generation by a constrained volume iterative fitting algorithm☆

Hongwei Lin*, Sinan Jin, Hongwei Liao, Qun Jian

*Department of Mathematics, State Key Lab. of CAD&CG, Zhejiang University, Hangzhou, 310027, China*

## HIGHLIGHTS

- An iterative algorithm is developed to fill a triangular mesh with an all-hex mesh.
- The Jacobian values of the all-hex mesh are guaranteed to be positive.
- The convergence of the iterative algorithm is proved.

## ARTICLE INFO

*Article history:*
Received 23 May 2014
Accepted 23 May 2015

*Keywords:*
Solid modeling
Hexahedral mesh generation
Volume subdivision fitting
Guaranteed quality

## ABSTRACT

The hexahedral mesh (hex mesh) is usually preferred to the tetrahedral mesh (tet mesh) in finite element methods for numerical simulation. In finite element analysis, a valid hex mesh requires that the scaled Jacobian value at each mesh vertex is larger than 0. However, the hex mesh produced by lots of prevailing hex mesh generation methods cannot be guaranteed to be a valid hex mesh. In this paper, we develop a constrained volume iterative fitting (CVIF) algorithm to fill a given triangular mesh model with an all-hex volume mesh. Starting from an initial all-hex mesh model, which is generated by voxelizing the given triangular mesh model, CVIF algorithm fits the boundary mesh of the initial all-hex mesh to the given triangular mesh model by iteratively adjusting the boundary mesh vertices. In each iteration, the movements of the boundary mesh vertices are diffused to the inner all-hex mesh vertices. After the iteration stops, an all-hex volume mesh that fills the given triangular mesh model can be generated. In the CVIF algorithm, the movement of each all-hex mesh vertex is constrained to ensure that the scaled Jacobian value at each mesh vertex is larger than 0, etc. Therefore, the all-hex mesh generated by the CVIF algorithm is guaranteed to be a valid all-hex mesh.



## 1. Introduction

In finite element methods for numerical simulation, the hexahedral mesh (hex mesh) is usually preferred to the tetrahedral mesh (tet mesh) owing to the reduced error and smaller number of elements [1,2]. However, generating a hex mesh with desirable qualities often requires significant geometric decomposition. Therefore, hex mesh generation can be extremely difficult to perform and automate. As a result, it requires considerable user interactions and may require days or even weeks in the case of complex shapes [3].

Moreover, it is well known that a valid hex mesh in finite element analysis should satisfy the requirement that, the scaled Jacobian value at each mesh vertex is larger than 0 [4]. Unfortunately, there is little work which can generate an all-hex mesh with the quality guarantee stated above. On the other hand, though boundary representation models, especially triangular mesh models, are popular in current computer graphics and computer aided design applications, lots of existing all-hex mesh generation algorithms need a tet mesh model as an input [5,6]. So they cannot handle triangular mesh models directly, and it is inconvenient.

In this paper, we develop a *constrained volume iterative fitting* algorithm (abbr. CVIF) which can *fill* a given triangular mesh model using an all-hex volume mesh, with guaranteed quality that the scaled Jacobian value at each mesh vertex is larger than 0. Given a triangular mesh model, we first construct an initial all-hex mesh model by voxelizing the given model, and extract the boundary quadrilateral mesh of the initial all-hex mesh model. Then, the initial all-hex mesh model is fitted to the given triangular mesh model by the CVIF algorithm. In each iteration of the CVIF algorithm, there are two steps:

(i) The adjustment of the vertices of the boundary quadrilateral mesh, and
(ii) the diffusion of the movement of the boundary mesh vertices to the inner mesh vertices.

☆ This paper has been recommended for acceptance by Jaroslaw Rossignac.

* Corresponding author. Tel.: +86 571 87951860 8304; fax: +86 571 88206681.
*E-mail address:* hwlin@zju.edu.cn (H. Lin).

http://dx.doi.org/10.1016/j.cad.2015.05.004
0010-4485/

In the above two steps, the movement of the mesh vertices is so constrained that the scaled Jacobian value at each mesh vertex after movement is larger than 0. In this way, the mesh quality of the all-hex mesh generated by our algorithm is guaranteed.

Specifically, the iterative adjustments of the boundary quadrilateral mesh vertices make up of the *constrained surface iterative fitting* algorithm (abbr. CSIF), and we show its convergence in this paper.

The structure of this paper is as follows. In Section 2, we briefly review related work. In Section 3, we develop the constrained volume iterative fitting algorithm. After presenting some results and discussions in Section 4, we conclude the paper in Section 5.

## 2. Related work

In this section, we will briefly review previous work related to our method, including hex mesh generation, subdivision fitting, and volume subdivision.

**Hex mesh generation:** There is a great deal of literature on the generation of volume meshes including tet [7,8] and hex meshes. In this paper, we focus on hex mesh generation. According to Owen's classification [9], hex mesh generation methods can be categorized into three classes, i.e., direct, indirect, and structured methods. Usually, the quality of the generated hex volume mesh should be improved by postprocessing [10,11].

Starting with a quadrilateral boundary surface mesh, *direct methods* generate a hexahedron for each quadrilateral according to a heuristically advancing-front approach. However, when the algorithmic heuristics are exhausted, no additional hexahedra can be placed. Consequently, this will leave void regions in the generated hex mesh [12,13].

*Indirect methods* first generate a tet mesh, and then convert it to a hex mesh by tetrahedral decomposition or combination. The disadvantage of these methods is that the quality of resultant hex mesh can be very poor owing to the high valence nodes [14,15].

A structured hex mesh is a mesh whose inner vertex valence is only six. A popular *structured method* for hex mesh generation is known as *mapping* [16], by which a map from the given solid with six surfaces to a cuboid is constructed. A cuboid has a trivial hex mesh, and the hex mesh in the given solid can be generated by inverse mapping. Although the mapping method can generate a high-quality hex mesh, it can only deal with solids of relatively simple shape, i.e., those with six boundary surfaces.

To deal with complex solids, a *submapping* method has been developed [17]. This submapping algorithm decomposes the given solid into separate mappable subregions while ensuring that the constraints within each subregion are consistent with the adjacent subregions.

The recently proposed hex mesh generation methods based on the PolyCube are also submapping methods that focus on the construction of the mapping [5]. A PolyCube [18] is a solid formed by combining a number of cubes with the same orientation; hence, it has a trivial hex mesh. By devising a mapping between the PolyCube and the input model, the sub-mapping method transfers the hex mesh in the PolyCube to the input model. Therefore, the quality of the hex mesh is heavily related to the shape of the PolyCube and the mapping.

In [19], the method of fundamental solutions is employed to design a harmonic volumetric mapping. In [20], the given model is first decomposed into the direct product of a surface and curve and then parameterized; subsequently, the mapping between the model and the PolyCube is constructed. Moreover, a volumetric deformation method is utilized to construct the correspondence between the given model and its PolyCube [6]. However, computing the PolyCube as well as a low-distortion mapping between it and the given model for general shapes remains an open problem [20,21]. Recently, some work is devoted to calculate a desirable polycube. In Ref. [22], the polycube is constructed using a variational method, by deforming an input triangle mesh through minimizing the $l^1$ norm of the mesh normals. In Ref. [23], a constrained discrete optimization technique is developed to make the generated polycube balance parameterization distortion against singularity count.

It should be pointed out that, though the mapping and submapping methods can produce high quality hex mesh, they need a tet mesh as input. Therefore, the mapping and sub-mapping methods cannot be employed directly to transform a triangular mesh model into an all-hex mesh.

On the other hand, some hex mesh generation methods employ the frame field to guide the construction of mapping from the input tet mesh to the resulted hex mesh [24–26].

For more work on hex mesh generation, we refer the reader to excellent surveys [1,9].

**Subdivision surface fitting:** The limit surface of approximating subdivision schemes, such as Catmull–Clark scheme [27] will shrink, especially when the initial control mesh is sparse. Usually, this problem is solved by making the approximating subdivision surface fit the vertices of the initial mesh, by either global methods [28], or local methods [29].

Recently, some new methods, such as *progressive interpolation* (abbr. PI) and *geometric interpolation* (abbr. GI), have been proposed for subdivision surface fitting. They adjust the vertices of the control mesh iteratively, depending on either parametric distance in PI or geometric distance in GI, and the limit subdivision surface fits the initial mesh. The convergence of PI has been shown for the Loop [30], Doo–Sabin [31], and Catmull–Clark schemes [32]. On the other hand, Ref. [33] develops a geometric interpolation algorithm for the Loop subdivision scheme. Moreover, Ref. [34] presents a geometric approximation algorithm for the Loop subdivision surface by distributing the difference vector for each data point to the related control vertices. Given that the geometric interpolation (approximation) algorithm must compute the closest point on the limit surface for each data point in each iteration, it incurs great computational costs.

Different from existing subdivision fitting algorithms which take the limit surface of subdivision as the approximating surface, in this paper, we develop a constrained surface iterative fitting algorithm which takes the mesh surface after finite time subdivisions as the approximating surface.

**Volume subdivision:** Similar to the two-dimensional (2D) subdivision scheme, the volume subdivision scheme can generate a sequence of increasingly dense volume meshes by recursive subdivision starting from a coarse volume mesh. The volume subdivision is mainly applied to model deformation, and its smoothness analysis is difficult to handle.

Thus far, only a few volume subdivision schemes have been developed. To our best knowledge, the first volume subdivision scheme was developed in [35] for subdividing hex meshes. This scheme is an extension of the Catmull–Clark subdivision scheme. Furthermore, Bajaj et al. developed the MLCA subdivision rule and analyzed its smoothness [36]. The MLCA subdivision rule can be applied in any dimension, including 2D quadrilateral surface mesh, and 3D hex volume mesh. In Ref. [37], Pascucci introduced a subdivision scheme for 3D and higher dimensional meshes without the excessive vertex proliferation, which can be generalized to meshes of any dimension and with cells of virtually any type. Recently, inspired by the $\sqrt{3}$ subdivision scheme, an adaptive subdivision scheme for unstructured tetrahedral meshes was presented in [38], which generates only tetrahedra and supports adaptive refinement.

## 3. Constrained volume iterative fitting

The CVIF algorithm takes a triangular mesh $P$ as input (see Fig. 9(a)), and outputs an all-hex mesh that fills the inside of $P$. The outline of the algorithm is listed as follows.

**Algorithm 1:** Outline of the Constrained Volume Iterative Fitting Algorithm

**Input**: A triangular mesh model $P$ (Fig. 9(a))
**Output**: An all-hex mesh model
1 Construct the initial all-hex mesh model, and extract its boundary quadrilateral mesh as the initial boundary mesh $V$ ;
2 **while** *The termination condition is not satisfied* **do**
3 Adjust the vertices of the boundary mesh $V$ ;
4 Diffuse the movements of the boundary mesh vertices to the inner mesh vertices ;
5 **end**
6 Improve the quality of the generated all-hex mesh;

It should be pointed out that the iterative adjustments of the boundary quadrilateral mesh vertices make up the constrained surface iterative fitting algorithm (abbr. CSIF).

The details of the CVIF algorithm will be explained in this section.

### 3.1. Initial all-hex mesh construction and boundary mesh extraction

The first step of the CVIF algorithm 1 is to construct an initial all-hex mesh, and its boundary quadrilateral mesh $V$ will be fitted to the given triangular mesh $P$ (see Fig. 9(a)).

It should be noted that the finally generated all-hex mesh is heavily influenced by the initial all-hex mesh. Although there are several techniques to construct an initial all-hex mesh, such as the polycube mapping [20,21], it is not an easy task to construct a desirable initial all-hex mesh in general. Since the topic of this paper focuses on the CVIF algorithm, the initial all-hex mesh model is just constructed by voxelizing the triangular mesh $P$ in this paper. The voxelization size is taken as the half of the average edge length of the mesh $P$. Specifically, after ascertaining the *feature voxels*, which intersect with the triangular mesh $P$, the scan conversion algorithm categorizes the voxels into three classes: *outer voxels*, *inner voxels*, and *boundary voxels*, which are feature voxels adjacent to outer voxels. Note that, a boundary voxel must be a feature voxel, but a feature voxel is not necessarily a boundary voxel (refer to Fig. 2). For clarity, the processing to the voxelization is illustrated by 2-dimensional rectangular mesh in Figs. 2 and 3.

Moreover, to make the approximation of the boundary mesh $V$ to the triangular mesh $P$ as close as possible, each feature voxel should satisfy the following two conditions:

(i) Each feature voxel is a boundary voxel, and,
(ii) each feature voxel contains a continuous patch of the triangular mesh $P$.

Actually, if one feature voxel violates the two conditions above (see Fig. 2(a)), it will make that either the shape of the boundary mesh $V$ deviates from that of the triangular mesh $P$ too large, or more seriously, the boundary mesh $V$ is not topologically equivalent to the given mesh $P$. Therefore, if we encounter one feature voxel which violates the above two conditions, we will perform *adaptive subdivision*, i.e., subdividing each of the three voxel columns where the offending feature voxel lies into two columns voxels (see Fig. 2(b)). However, to avoid producing voxels with tiny side length, if the side length of voxels after subdivision is less than $\frac{1}{8}$ of the initial side length, we do not perform the adaptive subdivision at the offending feature voxel.

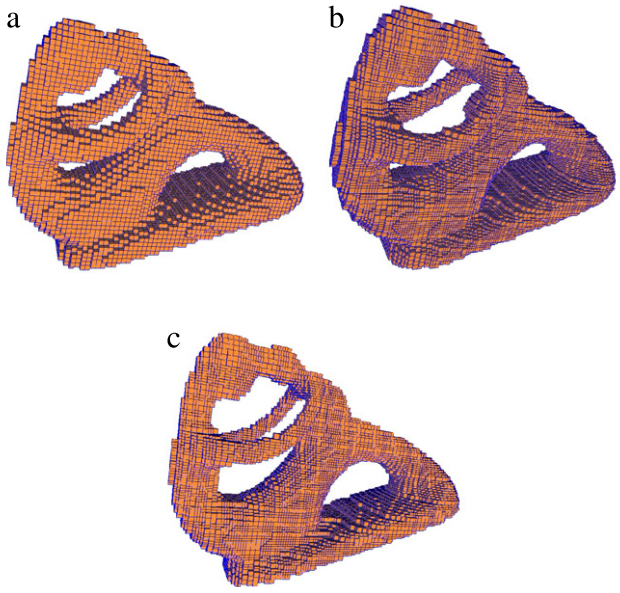


**Fig. 1.** Initial all-hex mesh construction. (a) Voxelization of the input triangular mesh. (b) Voxelization after adaptive subdivision. (c) The initial all-hex mesh is constructed after deleting some boundary voxels.

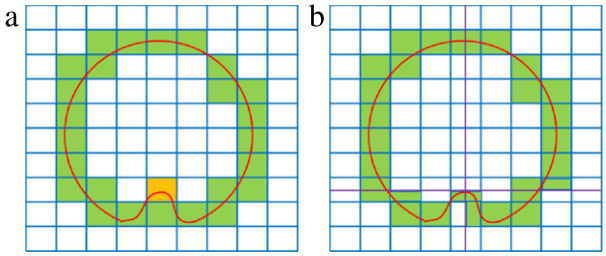


**Fig. 2.** The voxel in yellow is a feature voxel, but not a boundary voxel (a), so the adaptive subdivision should be performed, i.e., subdividing each of the two voxel columns where the offending feature voxel lies into two columns voxels (b). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Furthermore, to shorten the distance between the boundary mesh $V$ and the given triangular mesh $P$, some boundary voxels should be deleted (Fig. 3). Consider these boundary voxels whose three adjacent faces are all boundary faces. For each of these boundary voxels, denoting the vertex adjacent to the three boundary faces as $v$, supposing $d_1$ is the distance between $v$ and the triangular mesh $V$, and $d_2$ is the distance between the vertex opposite to $v$ and the triangular mesh $V$, if $d_2 < d_1$, the boundary voxel should be deleted, labeling as outer voxel (see Fig. 3). After deleting some boundary voxels in this way, the boundary mesh $V$ can be made closer to the triangular mesh $P$ (see Figs. 3 and 1(c)).

Fig. 1 demonstrates the procedure of the initial all-hex mesh construction. Fig. 9(a) is the input triangular mesh model $V$, and Fig. 1(a) is the voxelization of the input triangular mesh model. The initial all-hex mesh is generated after adaptive subdivision (Fig. 1(b)) and further deleting some boundary voxels (Fig. 1(c)).

Finally, the boundary quadrilateral mesh of the initial all-hex mesh (Fig. 1(c)) is extracted as the initial boundary mesh $V$, which will be fitted to the given triangular mesh $P$.

### 3.2. Constrained surface iterative fitting

As stated above, in the CVIF algorithm 1, the boundary quadrilateral mesh $V$ will be fitted to the given triangular mesh $P$. The iterative fitting procedure of the boundary mesh $V$ constitutes the

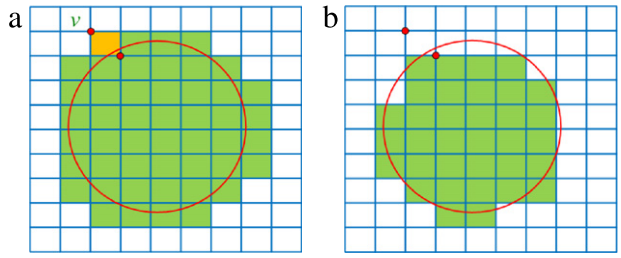


**Fig. 3.** Suppose $d_1$ is the distance between the vertex $v$ of the voxel (in yellow) and the given mesh model (represented by red curve), and $d_2$ is the distance between the vertex opposite to $v$ and the given mesh model (represented by red curve), respectively. If $d_1 > d_2$, the voxel in yellow should be deleted (a). This operation is performed iteratively till no voxel can be deleted (b). We can see that the boundary of the voxelization in (b) is closer to the red curve than that of the voxelization in (a). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

constrained surface iterative fitting algorithm (abbr. CSIF). Note that existing subdivision fitting algorithms take the limit surface of subdivision as the approximating surface. This does not meet the requirement of hex mesh generation, which relies on a mesh after finite time subdivision rather than the subdivision limit surface. Therefore, the CSIF algorithm developed in this section takes the mesh surface after finite time subdivisions as the approximating surface.

Taking the quadrilateral surface mesh $V$ to be the initial control mesh of a surface subdivision procedure, a mesh surface denoted as $V_\gamma$ will be generated after $\gamma$ time surface subdivisions. Moreover, supposing that the vertices of $V$ and $P$ are $v$ and $p$, and called *control points* and *data points*, respectively, each iteration of the CSIF algorithm includes the following steps:

(i) For each data point $p$, compute the closest mesh vertex $v_{cl,\gamma}$ on mesh $V_\gamma$;
(ii) calculate the difference vector $\boldsymbol{\delta} = p - v_{cl,\gamma}$ for each data point $p$, and distribute it to the related control points on mesh $V$ that generate the vertex $v_{cl,\gamma}$;
(iii) for each control point $v$, averaging the difference vectors distributed to it produces the difference vector $\boldsymbol{\Delta}$ for the control point;
(iv) adding the difference vector $\boldsymbol{\Delta}$ to the control point $v$ generates the new control point $v^{new}$ of the new control mesh,

$$v^{new} = v + \boldsymbol{\Delta}. \tag{1}$$

The first step is to compute the closest vertex $v_{cl,\gamma}$ on the mesh surface $V_\gamma$ for each vertex $p$, which is performed using KD tree. To make the shape of subdivision fitting surface faithful to that of the given triangular mesh, the distance between two vertices should consider not only the location closeness, but also the normal similarity at the two vertices. Therefore, we map each vertex $v$ in three dimensional Euclidean space to six dimensional *feature sensitive space* $(v, w\boldsymbol{n})$ [39,40], where $\boldsymbol{n}$ is the normalized normal vector at the vertex $v$, $l$ is the diagonal length of the bounding box of the triangular mesh model, and $w$ is a weight, set as $w = 0.2$ in our implementation. For any given data point $p$, we map it to the six dimensional feature sensitive space, and then calculate the closest vertex in the six dimensional space.

It should be pointed out that in the first five iterations, we compute the closest vertex on the mesh surface $V_\gamma$ for each data point on mesh $P$. Afterwards, the parameters of the closest points on the mesh surface $V_\gamma$ are fixed for reducing the computational load. Moreover, to avoid self-intersection, the Laplace smoothing operation is performed after each of the first five iterations.

The second step is to construct the *difference vector for data point* $p$, i.e.,

$$\boldsymbol{\delta} = p - v_{cl,\gamma}. \tag{2}$$

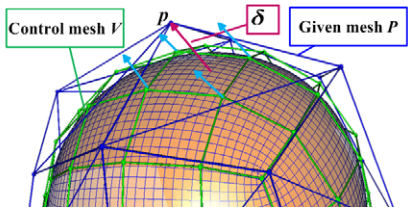


**Fig. 4.** The difference vector (in red) for data point is distributed to related control points (vectors in blue). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

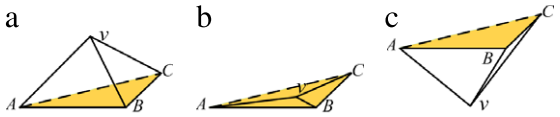


**Fig. 5.** Suppose the scaled Jacobian value at the vertex $v$ is positive in (a). When $v$ moves on the plane $ABC$ (b), its scaled Jacobian value is zero. When $v$ moves to the other side of the plane $ABC$ (c), its scaled Jacobian value changes to a negative value.

This difference vector will be distributed to the related control points of the control mesh $V$ (Fig. 4). Recall that the mesh vertex on mesh $V_\gamma$ is a linear combination of the related control points $v_1, v_2, \ldots, v_n$ of the control mesh $V$, i.e.,

$$v_{cl,\gamma} = c_1 v_1 + c_2 v_2 + \cdots + c_n v_n, \quad \text{with } \sum_{i=1}^{n} c_i = 1, \tag{3}$$

where the coefficients $c_i$, $i = 1, 2, \ldots, n$ can be easily obtained by the corresponding stationary linear surface subdivision rule. Thus, the weighted difference vector $c_i\boldsymbol{\delta}$ is distributed to the control point $v_i$, $i = 1, 2, \ldots, n$ (Fig. 4).

As the third step, all of the difference vectors $\{c_j\boldsymbol{\delta}_j\}$ distributed to a control point $v$ are collected and averaged in the following manner,

$$\boldsymbol{\Delta} = \frac{\sum_j c_j\boldsymbol{\delta}_j}{\sum_j c_j}, \tag{4}$$

to generate the *difference vector* $\boldsymbol{\Delta}$ *for control point* $v$.

Finally, adding $\boldsymbol{\Delta}$ to the control point $v$, generates the new control point $v^{new}$ (Eq. (1)) of the new control mesh.

The four steps above are performed iteratively until

$$\left|\frac{e^{(k+1)}}{e^{(k)}} - 1\right| < \varepsilon_0, \quad \text{with } e^{(k)} = \sqrt{\frac{\sum_{j=1}^{m} \left\|\boldsymbol{\delta}_j^{(k)}\right\|^2}{m}},$$

where $e^{(k)}$ is the root mean square (RMS) fitting error of the $k$th iteration (refer to Section 3.4), and $\varepsilon_0$ is a threshold. In our implementation, we take $\varepsilon_0 = 10^{-3}$.

*Construction of constrained region for quality guarantee:* As stated above, in each iteration of the CSIF algorithm, and the Laplace smoothing operation (5), the vertex of the boundary mesh $v$ is moved. The movement of the boundary mesh vertex should be constrained to ensure the quality of the all-hex mesh, i.e., the scaled Jacobian value at each mesh vertex is greater than 0.

As pointed out in Ref. [4] (see Fig. 5), the scaled Jacobian value at vertex $v$ is the scaled volume of the tetrahedron *v-ABC* (Fig. 5(a)). Suppose the scaled Jacobian value at $v$ is positive in Fig. 5(a). When $v$ moves on the plane *ABC* (see Fig. 5(b)), its scaled Jacobian value is zero. When $v$ moves to the other side of the plane *ABC* (Fig. 5(c)), its scaled Jacobian value changes to a negative value. Therefore, if the movement of the vertex $v$ keeps at the same side of the plane *ABC*, the sign of its Jacobian value will not change.

Take a regular inner vertex $v$ as an example. In an all-hex mesh, the regular inner mesh vertex $v$ is adjacent to six mesh vertices, and

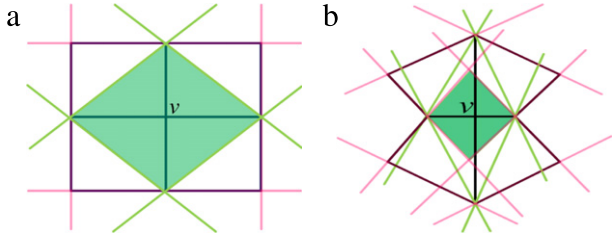


**Fig. 6.** The constrained region (the green region) of a mesh vertex $v$ in the initial all-hex mesh (a), and the all-hex mesh after some iterations (b), respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

eight hexahedra. So the movement of the vertex $v$ affects the scaled Jacobian values at seven mesh vertices, i.e., the vertex $v$ itself, and its six adjacent vertices. To make the sign of the Jacobian value at $v$ unchanged, its movement should lie in the same sides of 8 planes, each of which is determined by the three vertices adjacent to $v$ in the same hexahedron (corresponding to the green lines in Fig. 6). Similarly, to make the sign of the Jacobian value at a vertex $v'$ adjacent to $v$ unchanged, the movement of $v$ should keep in the same sides of 4 planes, each of which is determined by $v'$ and its two adjacent vertices on the boundary faces of the mesh model constituted by the eight hexahedra adjacent to $v$. Therefore, to make the signs of the Jacobian values at the six vertices adjacent to $v$ unchanged, the movement of $v$ should keep in the same sides of such $4 \times 6 = 24$ planes (corresponding to the red lines in Fig. 6). In conclusion, if the movement of the vertex $v$ keeps in the region enclosed by the so constructed $24 + 8 = 32$ planes, the signs of the Jacobian values at $v$ and its six adjacent vertices will remain unchanged. The so constructed region is named a *constrained region* for the vertex $v$. (see Fig. 6).

Suppose $v = (x_v, y_v, z_v)$, and the equations of the 32 planes aforementioned are, respectively,

$$P_i(x, y, z) = a_i x + b_i y + c_i z + d_i = 0, \quad i = 1, 2, \ldots, 32.$$

Moreover, suppose that the Jacobian value at each mesh vertex of an all-hex mesh before a new iteration is positive, to perform a new iteration, the constrained region can be modeled as the following system of inequalities,

$$\begin{cases} P_1(x_v, y_v, z_v)(a_1 x + b_1 y + c_1 z + d_1) > 0, \\ P_2(x_v, y_v, z_v)(a_2 x + b_2 y + c_2 z + d_2) > 0, \\ \cdots \\ P_{32}(x_v, y_v, z_v)(a_{32} x + b_{32} y + c_{32} z + d_{32}) > 0. \end{cases}$$

It should be pointed out that, the constrained region for an irregular mesh vertex can be constructed in the similar way.

In each iteration of the CSIF algorithm, the difference vector $\boldsymbol{\Delta}$ for control point is added to the boundary mesh vertex $v$. If the new vertex $v^{new} = v + \boldsymbol{\Delta}$ exceeds the constrained region of $v$, a weight $\omega$, $(0 \le \omega \le 1)$, as large as possible, is multiplied in front of $\boldsymbol{\Delta}$, to make that the new vertex $v^{new} = v + \omega\boldsymbol{\Delta}$ is guaranteed to stay in the constrained region of $v$. In our implementation, we uniformly discretize $[0, 1]$ into $\omega_i$, $i = 0, 1, \ldots, 10$, and take the largest $\omega_i$ which makes $v^{new} = v + \omega_i\boldsymbol{\Delta}$ be in the constrained region as the weight. In this way, the movement of the boundary mesh vertex is constrained, and the quality of the all-hex mesh is ensured.

Similarly, the movement of the mesh vertex in the Laplace smoothing operation (5) is also kept in its constrained region.

The convergence of the CSIF algorithm is shown in Section 3.4.

### 3.3. Movement diffusion

In the above section, we presented the CSIF algorithm for surface mesh fitting. After each iteration of the CSIF algorithm, the movement of the boundary mesh vertices should be diffused to the inner vertices of the all-hex mesh. In this section, to make the movement diffusion efficient, we develop a *level Laplacian operation* to diffuse the movement of the surface mesh vertices.

Firstly, the inner vertices are classified into levels according to their adjacency to the vertices of boundary mesh $V$. Specifically, the inner vertices in one-ring adjacency to the boundary mesh vertices are the first-level vertices; the left inner vertices in one-ring adjacency to the first-level vertices are the second-level vertices; $\cdots$; and so on.

Next, fixing the positions of the boundary mesh vertices, the first-level vertices $v$ are moved to the new position $v_{new}$ by the following Laplacian operation,

$$v_{new} = \frac{\sum_{j=1}^{d(v)} v_j}{d(v)}, \tag{5}$$

where, $d(v)$ is the degree of the vertex $v$, and $v_j$ are the vertices in the one-ring neighborhood of $v$. In succession, the first-level vertices are fixed and the second-level vertices are moved by (5), ..., and so on, until all levels of the inner vertices are adjusted. This procedure is performed iteratively until the ratio between the largest movement distance and the diagonal length of the bounding box of mesh $P$ is below a prescribed threshold $\mathcal{T}$. In our implementation, $\mathcal{T}$ is taken as $10^{-7}$.

To ensure the all-hex mesh quality, the vertex movement in the movement diffusion should also satisfy the vertex movement constraint stated in Section 3.2. The CSIF fitting and the constrained movement diffusion therein constitute the CVIF algorithm. When this algorithm stops, we obtain a control all-hex mesh. An all-hex volume mesh filling the given triangular mesh $P$ can be generated by subdividing the control all-hex mesh $\gamma$ times with the corresponding volume subdivision rule.

### 3.4. Convergence of the CSIF algorithm

In this section, we will show the convergence of the CSIF algorithm. Starting with the initial control mesh $V^{(0)} = V$, the control mesh after the $k$th iteration is denoted as $V^{(k)}$, which has $n$ mesh vertices $v_i^{(k)}$, $i = 1, 2, \ldots, n$, called *control points*. Subdividing the control mesh $V^{(k)}\gamma$ times generates the mesh surface $V_\gamma^{(k)}$. Moreover, suppose the given mesh $P$ has $m$ vertices $p_j$, $j = 1, 2, \ldots, m$, called *data points*, where $m > n$. To show the convergence of the CSIF algorithm, we rewrite Eqs. (2)–(4) with indices in the following.

In the $k$th iteration, the difference vector for the data point $p_j$ is,

$$\boldsymbol{\delta}_j^{(k)} = p_j - v_{j,\gamma}^{(k)}, \tag{6}$$

where $v_{j,\gamma}^{(k)}$ is the point on the mesh surface $V_\gamma^{(k)}$ with fixed parameter after the fifth iteration. It is a linear combination of related control points, i.e.,

$$v_{j,\gamma}^{(k)} = c_{j,1} v_1^{(k)} + c_{j,2} v_2^{(k)} + \cdots + c_{j,n} v_n^{(k)}, \quad \text{with} \sum_{i=1}^{n} c_{j,i} = 1. \tag{7}$$

By first distributing the weighted difference vector $c_{j,i}\boldsymbol{\delta}_j^{(k)}$ to the control point $v_i^{(k)}$, and then gathering the weighted difference vectors for each control point in the following manner,

$$\boldsymbol{\Delta}_i^{(k)} = \frac{\sum_{j \in I_i} c_{j,i}\boldsymbol{\delta}_j^{(k)}}{\sum_{j \in I_i} c_{j,i}} = \sum_{j \in I_i} \frac{c_{j,i}}{\sum_{j \in I_i} c_{j,i}} \boldsymbol{\delta}_j^{(k)}, \tag{8}$$

where $I_i$ is the index set of the data points which distribute their difference vectors to the control point $v_i^k$, we get the difference vector $\mathbf{\Delta}_i^{(k)}$ for the control point $v_i^{(k)}$. Finally, adding $\mathbf{\Delta}_i^{(k)}$ to the control point $v_i^{(k)}$ generates the new control point, i.e.,

$$v_i^{(k+1)} = v_i^{(k)} + \omega_i^{(k)} \mathbf{\Delta}_i^{(k)}, \quad i = 1, 2, \ldots, n,$$

where, $\omega_i^{(k)}$, $(0 < \omega_i^{(k)} \leq 1)$ is a weight to ensure that the new vertex $v_i^{(k+1)}$ satisfies the vertex movement constraint.

Arranging the difference vectors for control points in a sequence,

$$\Delta^{(k)} = [\mathbf{\Delta}_1^{(k)}, \mathbf{\Delta}_2^{(k)}, \ldots, \mathbf{\Delta}_n^{(k)}]^T,$$

the iterative format can be represented in matrix form,

$$\Delta^{(k+1)} = (I - \Lambda_1 A^T A \Lambda_2^{(k)}) \Delta^{(k)}, \tag{9}$$

where $I$ is an identity matrix, $\Lambda_1$ and $\Lambda_2^{(k)}$ are diagonal matrices,

$$\Lambda_1 = \text{diag}\left\{ \frac{1}{\sum_{j \in I_1} c_{j,1}}, \frac{1}{\sum_{j \in I_2} c_{j,2}}, \ldots, \frac{1}{\sum_{j \in I_n} c_{j,n}} \right\},$$

$$\Lambda_2^{(k)} = \text{diag}\{\omega_1^{(k)}, \omega_2^{(k)}, \ldots, \omega_n^{(k)}\},$$

and,

$$A = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n} \\ & \cdots & \cdots & \\ c_{m,1} & c_{m,2} & \cdots & c_{m,n} \end{bmatrix}.$$

Note that Eq. (9) is a non-stationary iterative format.

First, we will show that if the weights $\omega_i^{(k)}, i = 0, 1, \ldots, n$ do not change in the iterations, i.e., $\omega_i^{(k)} = \omega_i \in (0, 1], i = 0, 1, \ldots, n$, the iteration is convergent. In this case, $\Lambda_2^{(k)} = \Lambda_2 = \text{diag}\{\omega_1, \omega_2, \ldots, \omega_n\}$, and Eq. (9) becomes a stationary iterative format. Actually, on one hand, if $A^T A$ is nonsingular, it is positive definite, so all of its eigenvalues are positive. Because $\Lambda_2$ is also positive definite, all of the eigenvalues of $A^T A \Lambda_2$ are positive. Therefore, the eigenvalues of $\Lambda_1 A^T A \Lambda_2$ are also positive, i.e., $\lambda(\Lambda_1 A^T A \Lambda_2) > 0$. On the other hand, since $\left\| \Lambda_1 A^T A \Lambda_2 \right\| \leq 1$, all of its eigenvalues are less than or equal to 1, i.e., $\lambda(\Lambda_1 A^T A \Lambda_2) \leq 1$. Therefore, the spectral radius of the matrix $I - \Lambda_1 A^T A \Lambda_2$ satisfies $\rho(I - \Lambda_1 A^T A \Lambda_2) < 1$, and the corresponding iterative format is convergent.

Moreover, if the weights $\omega_i^{(k)}, i = 0, 1, \ldots, n$ change in each iteration, Eq. (9) becomes a non-stationary iterative format. In this case, we need a convergence condition,

$$\left\| I - \Lambda_1 A^T A \Lambda_2^{(k)} \right\|_\infty \leq q < 1. \tag{10}$$

Denote $\left\| \mathbf{\Delta}_i^{(k)} \right\|_E$ as the Euclidean norm of $\mathbf{\Delta}_i^{(k)}$, let,

$$\Phi^{(k)} = [\phi_{ij}^{(k)}]_{n \times n} = I - \Lambda_1 A^T A \Lambda_2^{(k)},$$

and define $\left\| \Delta^{(k)} \right\|_M = \max\{ \left\| \mathbf{\Delta}_i^{(k)} \right\|_E, i = 1, 2, \ldots, n\}$. Because,

$$\begin{aligned} \left\| \Phi^{(k)} \Delta^{(k)} \right\|_M &= \max_i \left\{ \left\| \sum_j \alpha_{ij} \mathbf{\Delta}_j^{(k)} \right\|_E \right\} \\ &\triangleq \left\| \sum_j \alpha_{mj} \mathbf{\Delta}_j^{(k)} \right\|_E \leq \sum_j |\alpha_{mj}| \left\| \mathbf{\Delta}_j^{(k)} \right\|_E \\ &\leq \left( \sum_j |\alpha_{mj}| \right) \max_j \left\| \mathbf{\Delta}_j^{(k)} \right\|_E \\ &\leq \left\| \Phi^{(k)} \right\|_\infty \left\| \Delta^{(k)} \right\|_M, \end{aligned}$$

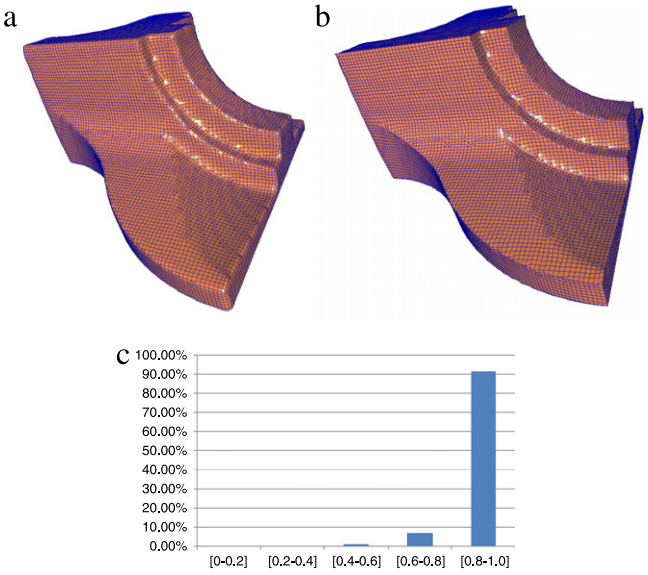


**Fig. 7.** Feature preservation. (a) Before feature preservation. (b) After feature preservation. (c) Diagram of the distribution of scaled Jacobian values of the all-hex mesh illustrated in (b).

we have,

$$\begin{aligned} \left\| \Delta^{(k+1)} \right\|_M &= \left\| \Phi^{(k)} \Delta^{(k)} \right\|_M \leq \left\| \Phi^{(k)} \right\|_\infty \left\| \Delta^{(k)} \right\|_M \\ &\leq \left\| \Phi^{(k)} \right\|_\infty \left\| \Phi^{(k-1)} \right\|_\infty \left\| \Delta^{(k-1)} \right\|_M \\ &\leq \cdots \\ &\leq \left\| \Phi^{(k)} \right\|_\infty \cdots \left\| \Phi^{(1)} \right\|_\infty \left\| \Delta^{(1)} \right\|_M \\ &\leq q^k \left\| \Delta^{(1)} \right\|_M. \end{aligned}$$

Therefore, if the convergence condition (10) holds, $\lim_{k \to \infty} \left\| \Delta^{(k)} \right\|_M = 0$, i.e., the iterative format (9) is convergent.

**Remark.** Note that the iterative format (9) and its convergence analysis do not involve any concrete volume subdivision rule, so they are valid for any stationary linear hex volume subdivision rule, including the subdivision rule developed in Ref. [35], and MLCA presented in Ref. [36].

### 3.5. Postprocessing

Till now, we have gotten an all-hex mesh each of whose vertices has a positive scaled Jacobian value. Moreover, postprocessing should be performed to make the all-hex mesh preserve the features, and improve its mesh quality.

**Feature preservation:** To make the all-hex mesh preserve the features in the given triangular mesh $P$, we select some edges in the boundary mesh of the generated all-hex mesh, whose vertices will be fitted to the features in the triangular mesh $P$. Then, extra CSIF iterations (without subdivision) are performed to the selected vertices, to make them fit the features in the given mesh $P$. In this way, most features in the mesh $P$ can be preserved. However, because the generated all-hex mesh contains lots of singular vertices, and lacks structure in the resulting edge flow, it is difficult to preserve all of the features.

Fig. 7 demonstrates the result of the feature preservation strategy stated above. In this example, the sharp features in the all-hex mesh model *fan disk* generated by the CVIF algorithm are smoothed (Fig. 7(a)). After performing the aforementioned feature preservation strategy, the features are recovered (Fig. 7(b)). Fig. 7(c) is the diagram of the distribution of scaled Jacobian values of the all-hex mesh demonstrated in Fig. 7(b). We can see that all of the Jacobian

**Table 1**
Statistics for the all-hex mesh filling algorithm.

| Model | #vert.[a] | #hex[b] | #total[c] | Jac. avg.[d] | Jac. min.[e] | J. ∈ (0, 0.2)[f] | J. ∈ [0.2, 0.4)[g] | J. ∈ [0.8, 1][h] | Precision | Time (s)[i] |
|---|---|---|---|---|---|---|---|---|---|---|
| Fig. 7(b) Fandisk | 17 576 | 93 948 | 102 511 | 0.9510 | 0.0381 | 45 | 103 | 86 026 | 0.0014 | 903 |
| Fig. 9(b) Fertility | 27 819 | 106 997 | 120 617 | 0.9029 | 0.0446 | 2 | 62 | 87 673 | 0.0009 | 1203 |
| Fig. 10(a) Rocker Arm | 15 114 | 62 670 | 71 214 | 0.8970 | 0.0585 | 3 | 39 | 51 553 | 0.0012 | 541 |
| Fig. 10(b) Torus Knits | 9 431 | 12 069 | 15 662 | 0.8066 | 0.0495 | 1 | 4 | 6 998 | 0.0021 | 110 |
| Fig. 10(c) Santa | 16 220 | 73 444 | 84 138 | 0.8792 | 0.0439 | 1 | 87 | 54 202 | 0.0007 | 647 |
| Fig. 10(d) Isis | 19 819 | 73 823 | 81 353 | 0.9178 | 0.0461 | 4 | 128 | 63 598 | 0.0012 | 772 |
| Fig. 11(a) Ball Joint | 10 134 | 32 188 | 36 260 | 0.8969 | 0.0571 | 1 | 24 | 25 874 | 0.0018 | 283 |
| Fig. 11(b) Rabbit | 10 379 | 58 531 | 64 324 | 0.9155 | 0.0374 | 2 | 56 | 50 132 | 0.0013 | 565 |

[a] #vert. of mesh: number of the vertices in the given triangular mesh;.
[b] #hex: number of the hexes in the all-hex mesh;.
[c] #total: number of the total vertices in the all-hex mesh;.
[d] Jac. avg.: averaged scaled Jacobian;.
[e] Jac. min.: minimum scaled Jacobian;.
[f] The numbers of hexes whose scaled Jacobian values lie in the interval (0, 0.2).
[g] The numbers of hexes whose scaled Jacobian values lie in the interval [0.2, 0.4).
[h] The numbers of hexes whose scaled Jacobian values lie in the interval [0.8, 1].
[i] Time is in seconds.

values are greater than 0, and over 90% of them lie in the interval [0.8, 1.0].

**Mesh quality improvement:** In the all-hex mesh generated by the CVIF algorithm, the poor hexahedra with too small positive scaled Jacobian values mainly concentrate in the boundary level. Therefore, to improve the mesh quality, a *pillowing* operation [41] is carried out just on the boundary level hexahedra.

Finally, the quality of the all-hex mesh is further improved by the *Mesquite* software [10].

Fig. 8 illustrates the result of the aforementioned mesh quality improvement technique. The cutaway view of the all-hex mesh before pillowing is presented in Fig. 8(a), where the quality of the boundary level hex mesh is very poor. After pillowing and mesh quality improvement by *Mesquite* software [10], the mesh quality is greatly improved, as shown by the cutaway view in Fig. 8(b). Zooms of the cutaway views in Fig. 8(a) and (b) are demonstrated in Fig. 8(c) and (d), respectively. Fig. 9(b) is the all-hex mesh model *Fertility* after mesh quality improvement, and Fig. 9(c) is the diagram of the distribution of scaled Jacobian values of the all-hex mesh model *Fertility*. Over 80% of them lie in the interval [0.8, 1].

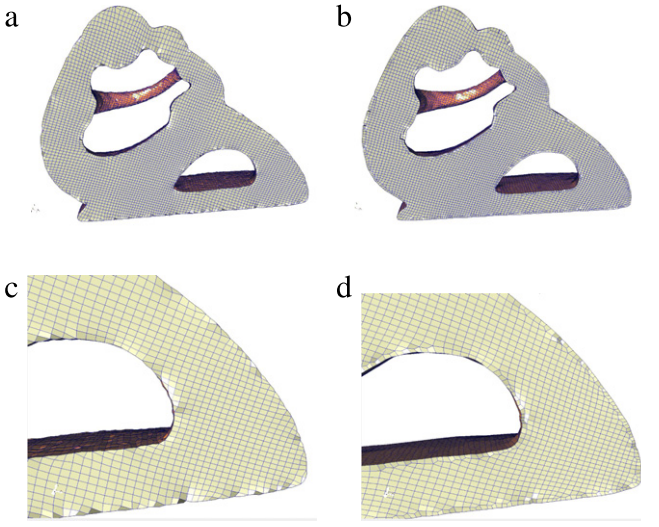


**Fig. 8.** All-hex Mesh quality improvement. (a) Cutaway view of the all-hex mesh model before pillowing. (b) Cutaway view of the all-hex mesh model after pillowing and mesh quality improvement by *Mesquite* software. (c) Zoom of the cutaway view of the model in (a). (d) Zoom of the cutaway view of the model in (b).

## 4. Results and discussions

The CVIF algorithm has been implemented with Visual C++, and run on a PC with Intel Core2 Quad CPU Q9400 2.66 GHz and 4G memory. We tested the CVIF algorithm by a lot of examples. Some of them were illustrated in Figs. 7–11. Moreover, we presented the diagram of the distribution of scaled Jacobian values along with each example. In these diagrams, the abscissa axis is the scaled Jacobian value intervals [2,4], and the vertical axis represents the ratio of the number of hexes whose scaled Jacobian values lie in the corresponding interval and the total number of the hexes of the all-hex mesh model.

Moreover, the statistics on these examples were listed in Table 1. In Table 1, the fitting precision is represented by the ratio between the last RMS error and the diagonal length of the bounding box of given mesh $P$, and the runtime is in seconds. It can be seen from Table 1 that, the scaled Jacobian values of these examples are all greater than 0, and the average scaled Jacobian values of these examples are all over 0.8. Of four of these example, the average scaled Jacobian values even exceed 0.9. On the other hand, the runtime cost by these examples varies between several minutes to twenty minutes.

In Fig. 10, we illustrated four examples that are generated by the CVIF algorithm without subdivision. Figs. 10(a–d) are the four all-hex mesh models, *Rocker Arm*, *Torus Knits*, *Santa*, and *Isis*, respectively. Figs. 10(e–h) are their cutaway views. Figs. 10(i–l) are the diagrams of the distribution of their scaled Jacobian values. In the all-hex mesh models of *Rocker Arm* and *Isis*, over 80% scaled Jacobian values lie in the interval [0.8, 1]. In the all-hex mesh models of *Santa* and *Torus Knits*, over 70% and nearly 60% scaled Jacobian values are in the interval [0.8, 1], respectively. It should be pointed out that, in Figs. 7 and 8, the all-hex mesh models *Fan disk* and *Fertility* are also generated by the CVIF algorithm without subdivision.

In Fig. 11, there are two examples that are generated by the CVIF algorithm with one MLCA [36] subdivision. Figs. 11(a, b) are the two all-hex mesh models *Ball Joint* and *Rabbit*. Figs. 11(c, d) are their cutaway views. Figs. 11(e, f) are diagrams of the distribution of their scaled Jacobian values. In the two all-hex mesh models, 80% scaled Jacobian values or so lie in the interval [0.8, 1]. It can also be seen from Table 1 that, the average scaled Jacobian values of the all-hex mesh models *Ball Joint* Fig. 11(a) and *Rabbit* Fig. 11(b) are 0.8969 and 0.9155, respectively.

We can see from Table 1 that, except the model *Fandisk* in Fig. 7(b), there are only several (not more than 5) hexes whose scaled Jacobian values are less than 0.2. In the model *Fandisk* in Fig. 7(b), even the shape preserving operation deteriorates the hex mesh quality, there are just 45 hexes whose scaled Jacobian values are less than 0.2.

Because there are usually just several hexes whose scaled Jacobian values are less than 0.2, we can improve the mesh quality

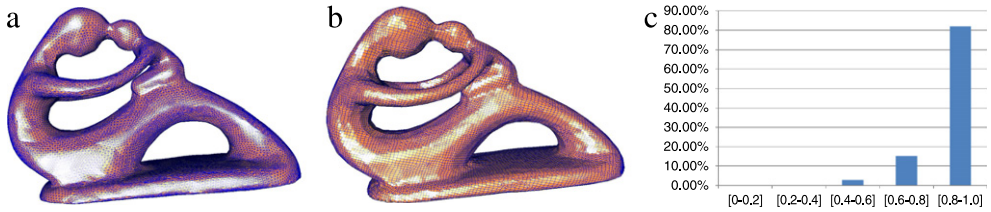


**Fig. 9.** *Fertility* model. (a) The input to our algorithm is a triangular mesh. (b) The all-hex mesh model *Fertility* generated by our algorithm, after mesh quality improvement. (Please also refer to Fig. 8(b) for the cutaway view.) (c) Diagram of the scaled Jacobian value distribution of the all-hex model *Fertility* in (b).

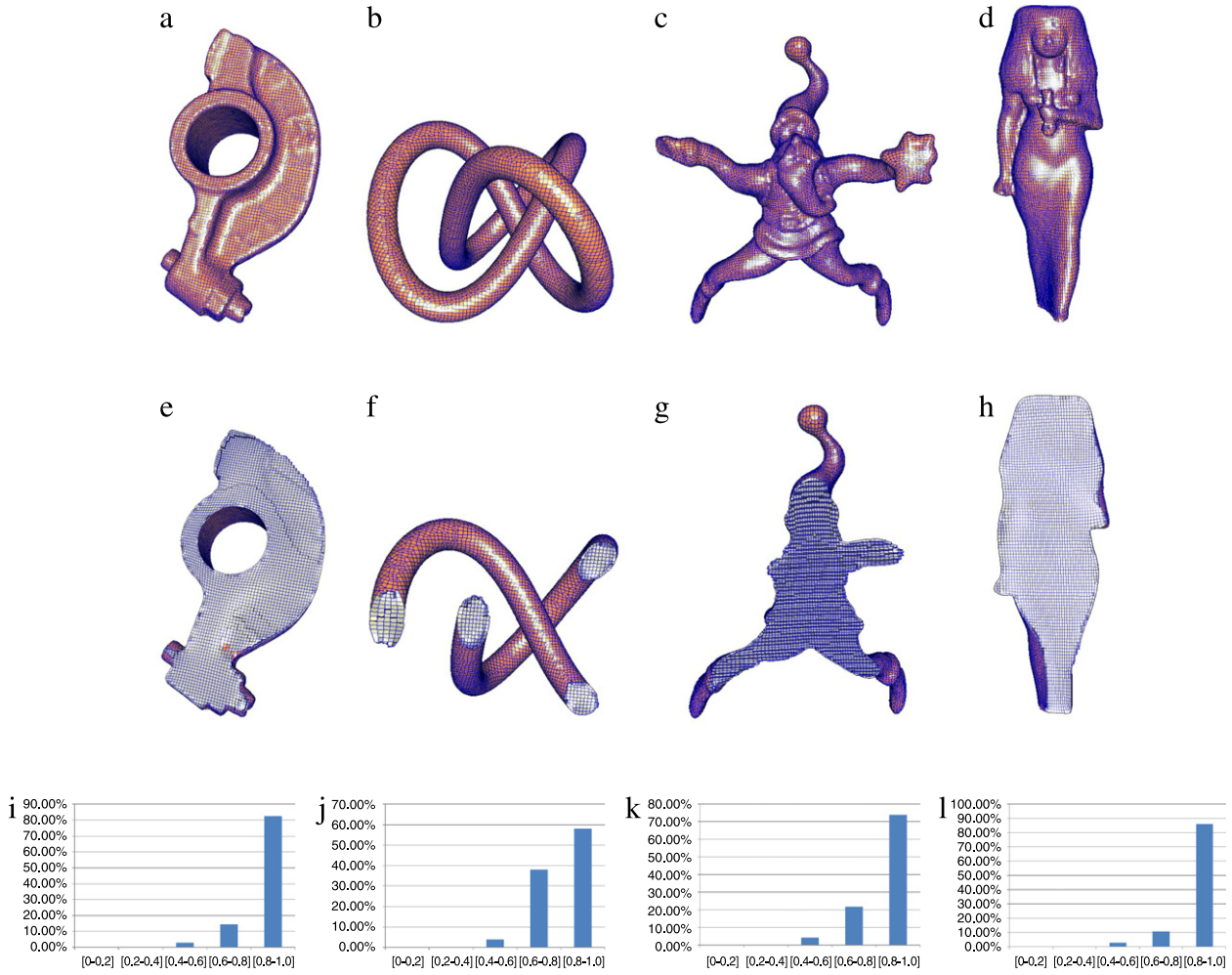


**Fig. 10.** All-hex meshes filling triangular meshes and their cutaway views, generated by the CVIF algorithm without subdivision. (a–d) All-hex meshes of *Rocker Arm*, *Torus Knits*, *Santa* and *Isis*. (e–h) Their cutaway views. (i–l) Distribution diagrams of scaled Jacobian.

at these hexes by some interactions. The statistics of the hex mesh model after some interactions were listed in Table 2. It can be seen that the minimum scaled Jacobian values have been improved greatly.

**Comparison with prevailing hex mesh generation methods:** The differences between our CVIF algorithm and other prevailing all-hex mesh generation methods, such as polycube based methods [5,6], and frame field guided methods [24–26], mainly lie in two aspects. Firstly, while the prevailing methods need to input a *tet* mesh, the input to the CVIF is just a triangular mesh, which are widely employed in computer graphics and CAD community. Secondly and more importantly, though these prevailing hex mesh generation methods can produce all-hex mesh with high quality, they lose strategy to ensure that the generated all-hex mesh is a valid mesh, i.e., they cannot ensure that the scaled Jacobian value of each hex is larger than 0. However, the CVIF algorithm developed in this paper can guarantee that all of the scaled Jacobian values of the generated all-hex mesh model are greater than 0. Last but not least, the quality of the hex mesh generated by our CVIF algorithm is comparable to that produced by the state-of-the-art hex mesh generation algorithms (refer to Tables 1 and 2 and Refs. [22,23]).

Moreover, we compared our CVIF algorithm with the state-of-the-art octree-based all-hex mesh generation method, i.e., Hexotic [42], which has been integrated to the software MeshGems as a component Hexa [43] (v1.2-1). MeshGems-Hexa should be run through SALOME (v7.4.0) [44]. In our implementation, the MeshGems-Hexa (through SALOME) was run on a MacBook Pro with 2.6 GHz Intel Core i5 CPU, 8 GB 1600 MHz DDR3 memory, Intel Iris graphic card, and 256G solid state disk, which is much more powerful than the platform where our CVIF algorithm was run. The statistics on MeshGems-Hexa were listed in Table 3. It should be pointed out that, because MeshGems-Hexa exploits

**Table 2**
Statistics for the all-hex mesh after some interactions.

| Model | Jac. avg.[a] | Jac. min.[b] | J. ∈ (0, 0.2)[c] | J. ∈ [0.2, 0.4)[d] | J. ∈ [0.8, 1][e] |
|---|---|---|---|---|---|
| Fig. 9(b) Fertility | 0.9131 | 0.2042 | 0 | 60 | 87 677 |
| Fig. 10(a) Rocker Arm | 0.8982 | 0.1991 | 1 | 41 | 51 551 |
| Fig. 10(b) Torus Knits | 0.8160 | 0.3146 | 0 | 5 | 6 998 |
| Fig. 10(c) Santa | 0.8793 | 0.2100 | 0 | 87 | 54 207 |
| Fig. 10(d) Isis | 0.9268 | 0.1384 | 4 | 126 | 63 600 |
| Fig. 11(a) Ball Joint | 0.9067 | 0.3156 | 0 | 25 | 25 877 |
| Fig. 11(b) Rabbit | 0.9247 | 0.1809 | 3 | 58 | 50 129 |

[a] Jac. avg.: averaged scaled Jacobian;.
[b] Jac. min.: minimum scaled Jacobian;.
[c] The numbers of hexes whose scaled Jacobian values lie in the interval (0, 0.2).
[d] The numbers of hexes whose scaled Jacobian values lie in the interval [0.2, 0.4).
[e] The numbers of hexes whose scaled Jacobian values lie in the interval [0.8, 1].

**Table 3**
Comparison with the Hexotic [42] framework (i.e., MeshGems-Hexa (v1.2-1) [43]).

| Model | #hex[a] | #total[b] | Jac. avg.[c] | Jac. min.[d] | J. ∈ (0, 0.2)[e] | J. ∈ [0.2, 0.4)[f] | J. ∈ [0.8, 1][g] | Precision | Time (s)[h] |
|---|---|---|---|---|---|---|---|---|---|
| Fig. 7(b) Fandisk | 50 381 | 56 610 | 0.9158 | 0.0467 | 164 | 1023 | 42 876 | 0.0025 | 3.1 |
| Fig. 9(b) Fertility | 62 584 | 72 743 | 0.8153 | 0.0685 | 42 | 882 | 37 585 | 0.0028 | 4.7 |
| Fig. 10(a) Rocker Arm | 42 965 | 49 794 | 0.8106 | 0.0278 | 77 | 804 | 26 029 | 0.0043 | 3.3 |
| Fig. 10(b) Torus Knits | 28 000 | 34 855 | 0.8298 | 0.1624 | 3 | 47 | 17 563 | 0.0028 | 2.6 |
| Fig. 10(c) Santa | 83 077 | 95 915 | 0.7945 | 0.0496 | 23 | 1280 | 45 665 | 0.0023 | 4.9 |
| Fig. 10(d) Isis | 67 621 | 76 724 | 0.8079 | 0.0466 | 103 | 1160 | 40 334 | 0.0027 | 4.3 |
| Fig. 11(a) Ball Joint | 20 479 | 23 636 | 0.8196 | 0.0582 | 5 | 194 | 12 404 | 0.0053 | 2.1 |
| Fig. 11(b) Rabbit | 42 887 | 48 645 | 0.8561 | 0.1038 | 14 | 394 | 29 339 | 0.0032 | 3.5 |

[a] #hex: number of the hexes in the all-hex mesh;.
[b] #total: number of the total vertices in the all-hex mesh;.
[c] Jac. avg.: averaged scaled Jacobian;.
[d] Jac. min.: minimum scaled Jacobian;.
[e] The numbers of hexes whose scaled Jacobian values lie in the interval (0, 0.2).
[f] The numbers of hexes whose scaled Jacobian values lie in the interval [0.2, 0.4).
[g] The numbers of hexes whose scaled Jacobian values lie in the interval [0.8, 1].
[h] Time is in seconds.

parallel computing, and the platform it was run is much more powerful than that our algorithm was run, the computing time of MeshGems-Hexa is shorter than our algorithm. However, comparing Table 3 with Tables 1 and 2, we can see that, even in the case that the fitting precisions of the CVIF algorithm are higher than those of MeshGems-Hexa, the averaged and minimum scaled Jacobian values of the all-hexes generated by CVIF algorithm (Table 2) are still better than those produced by MeshGems-Hexa (Table 3). Furthermore, the numbers of hexes with scaled Jacobian in (0, 0.2) and [0.2, 0.4) of the all-hex meshes generated by our CVIF algorithm (Table 2) are much smaller than those produced by MeshGems-Hexa (Table 3).

### 4.1. Limitations and future work

One limitation of the CVIF algorithm lies in the feature preservation. Because the sharp edges on the triangular mesh model are not considered in the voxelization procedure, not all of the sharp edges on the triangular mesh model have correspondences on the boundary mesh of the all-hex mesh model. This makes the feature preservation results unsatisfactory in some cases, and restricts the application domain of the developed method. As the future work, the feature preservation capability will be improved by both considering the sharp edges in the voxelization procedure, and performing pillowing operation in the post-processing procedure.

On the other hand, the speed of the CVIF algorithm is a bit slow. However, this algorithm is easy to parallelize. In the future, we will implement the algorithm by GPU to improve the running speed.

Another limitation of the CVIF algorithm is that the number of the singular vertices on the boundary mesh of the generated all-hex mesh model is a bit large. The singular vertices on the boundary mesh are introduced in the initial all-hex control mesh by voxelization. As a future work, the voxelization rule will be improved to reduce the number of the singular vertices.

As stated above, the initial all-hex mesh has great influence on the finally generated all-hex mesh. In Fig. 12, we illustrate an all-hex mesh model filling the triangular mesh model Fig. 9(a), generated by the CVIF algorithm. But the initial all-hex mesh is constructed by the skeleton-based method developed in the technical report [45]. It can be seen that the two all-hex mesh models, illustrated in Figs. 9(b) and 12(a) are different. Therefore, one of our future work is to develop a method to construct the desirable initial all-hex mesh.

On the other hand, we note that, in each iteration of the CVIF algorithm, if the weights are chosen appropriately (refer to Section 3.2), the scaled Jacobian values of the generated all-hex mesh can be made greater than a suitably assigned value. In the all-hex mesh model illustrated in Fig. 13, the weights are so selected that the scaled Jacobian values are greater than 0.2. But, meanwhile, the fitting precision is made worse when the termination condition of the iteration is satisfied. In Fig. 13, while the minimum scaled Jacobian value is 0.2, the fitting precision is only 0.0017. As a future work, we will improve the CVIF algorithm so that it not only guarantees the scaled Jacobian values greater than a suitable value, e.g., 0.1, 0.2, but also ensures the fitting precision at the same time.

## 5. Conclusion

In this paper, we presented the CVIF algorithm to filling a given triangular mesh model $P$ with an all-hex mesh. First, we constructed an initial all-hex mesh by voxelizing the given triangular mesh $P$. By adaptive subdivision and boundary mesh extraction, the boundary quadrilateral mesh $V$ of the generated voxelization is faithful to the triangular mesh $P$, thus getting a

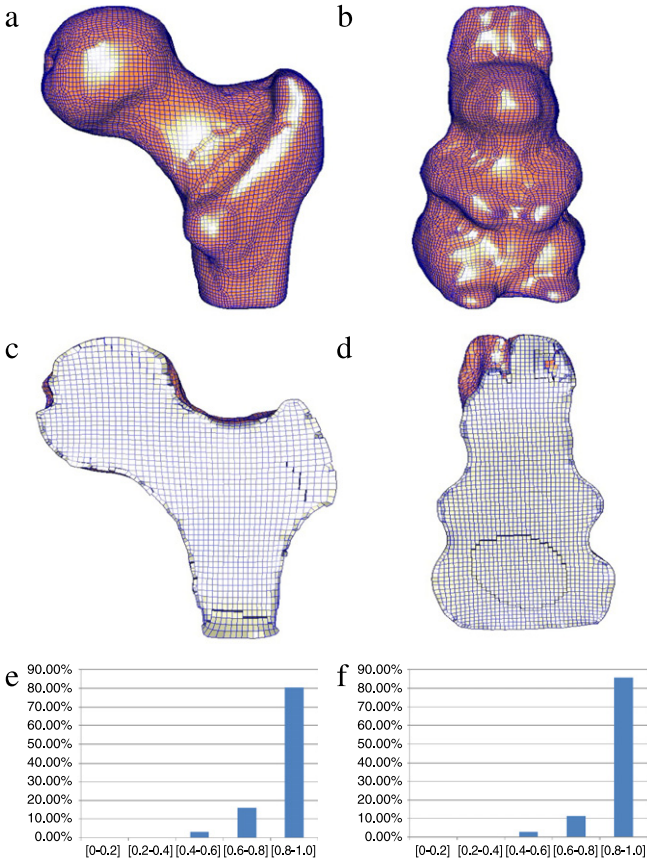


**Fig. 11.** All-hex meshes filling triangular meshes and their cutaway views, generated by the CVIF algorithm with one subdivision. (a, b) All-hex meshes of *Ball Joint* and *Rabbit*. (c, d) Cutaway views of the all-hex meshes. (e, f) Distribution diagrams of the scaled Jacobian.

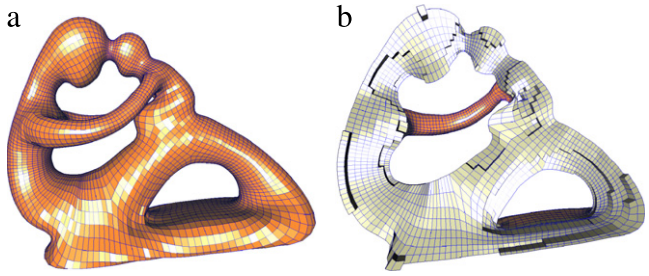


**Fig. 12.** The all-hex mesh generated by the CVIF algorithm, starting from the initial all-hex mesh constructed by a skeleton-based method. (a) The generated all-hex mesh. (b) The cutaway view of the all-hex mesh in (a).

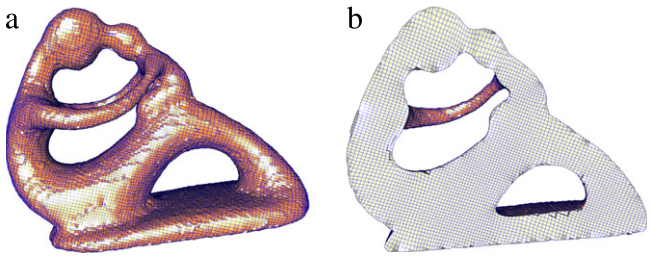


**Fig. 13.** If the weights in each iteration of the CVIF algorithm are chosen appropriately (refer to Section 3.2), the scaled Jacobian values of the generated all-hex mesh can be made greater than a suitably assigned value, e.g., 0.2. (a) The generated all-hex mesh. (b) The cutaway view of the all-hex mesh in (a).

desirable initial all-hex mesh for the CVIF algorithm. Next, the boundary mesh $V$ is fitted to the triangular mesh $P$ iteratively. After each iteration of the boundary mesh $V$, the movements of the mesh vertices of $V$ are diffused into the inner mesh vertices of the all-hex mesh. In the above iterations, the movements of the boundary and inner mesh vertices are constrained to ensure that the scaled Jacobian value at each mesh vertex is larger than 0. Therefore, the mesh quality of the generated all-hex mesh model is guaranteed.

## Acknowledgments

This paper is supported by National Natural Science Foundation of China (Nos. 61379072, 60970150).

## References

[1] Shepherd J, Johnson C. Hexahedral mesh generation constraints. Eng Comput 2008;24(3):195–213.
[2] Knupp P. Achieving finite element mesh quality via optimization of the jacobian matrix norm and associated quantities. Part II—A framework for volume mesh optimization and the condition number of the jacobian matrix. Internat J Numer Methods Engrg 2000;48(8):1165–85.
[3] Shepherd J. Topologic and geometric constraint-based hexahedral mesh generation (Ph.D. thesis), The University of Utah; 2007.
[4] Knupp P. A method for hexahedral mesh shape optimization. Internat J Numer Methods Engrg 2003;58(2):319–32.
[5] Han S, Xia J, He Y. Hexahedral shell mesh construction via volumetric polycube map. In: Proceedings of the 14th ACM symposium on solid and physical modeling. ACM; 2010. p. 127–36.
[6] Gregson J, Sheffer A, Zhang E. All-hex mesh generation via volumetric polycube deformation. In: Computer graphics forum, vol. 30. Wiley Online Library; 2011. p. 1407–16.
[7] Labelle F, Shewchuk J. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. ACM Trans Graph (TOG) 2007;26(3):57.
[8] Tournois J, Wormser C, Alliez P, Desbrun M. Interleaving delaunay refinement and optimization for practical isotropic tetrahedron mesh generation. In: ACM transactions on graphics (TOG), vol. 28. ACM; 2009. p. 75.
[9] Owen S. A survey of unstructured mesh generation technology. In: 7th international meshing roundtable, vol. 3, Citeseer. 1998.
[10] Brewer M, Diachin L, Knupp P, Leurent T, Melander D. The mesquite mesh quality improvement toolkit. In: Proceedings, 12th international meshing roundtable, Sandia National Laboratories report SAND.
[11] Shepherd J, Zhang Y, Tuttle C, Silva C. Quality improvement and Boolean-like cutting operations in hexahedral meshes. In: Proceedings of the 10th international conference on numerical grid generation in computational field simulations. 2007.
[12] Blacker T, Meyers R. Seams and wedges in plastering: a 3-d hexahedral mesh generation algorithm. Eng Comput 1993;9(2):83–93.
[13] Staten M, Owen S, Blacker T. Unconstrained paving & plastering: A new idea for all hexahedral mesh generation. In: Proceedings of the 14th international meshing roundtable. Springer; 2005. p. 399–416.
[14] Melander D, Benzley S, Tautges T. Generation of multi-million element meshes for solid model-based geometries: The dicer algorithm, Tech. rep. Albuquerque (NM, United States): Sandia National Labs; 1997.
[15] Owen S, Saigal S. H-morph: an indirect approach to advancing front hex meshing. Internat J Numer Methods Engrg 2000;49(1–2):289–312.
[16] Cook W, Oakes W. Mapping methods for generating three-dimensional meshes. Comput Mech Eng 1982;1(1):67–72.
[17] White D, Mingwu L, Benzley S, Sjaardema G. Automated hexahedral mesh generation by virtual decomposition. In: Proceedings of the 4th international meshing roundtable. Albuquerque (USA, Citeseer): Sandia National Laboratories; 1995. p. 165–76.
[18] Tarini M, Hormann K, Cignoni P, Montani C. Polycube-maps. In: ACM transactions on graphics (TOG), vol. 23. ACM; 2004. p. 853–60.
[19] Li X, Xu H, Wan S, Yin Z, Yu W. Feature-aligned harmonic volumetric mapping using MFS. Comput Graph 2010;34(3):242–51.
[20] Xia J, He Y, Yin X, Han S, Gu X. Direct-product volumetric parameterization of handlebodies via harmonic fields. In: Shape modeling international conference (SMI), 2010. IEEE; 2010. p. 3–12.
[21] Li B, Li X, Wang K, Qin H. Surface mesh to volumetric spline conversion with generalized poly-cubes. IEEE Trans Vis Comput Graphics 2012;99:1–14.
[22] Huang J, Jiang T, Shi Z, Tong Y, Bao H, Desbrun M. $l^1$-based construction of polycube maps from complex shapes. ACM Trans Graph (TOG) 2014;33(3):25.
[23] Livesu M, Vining N, Sheffer A, Gregson J, Scateni R. Polycut: monotone graph-cuts for polycube base-complex construction. ACM Trans Graph (TOG) 2013;32(6):171.
[24] Nieser M, Reitebuch U, Polthier K. Cubecover–parameterization of 3D volumes. In: Computer graphics forum, vol. 30. Wiley Online Library; 2011. p. 1397–406.

[25] Huang J, Tong Y, Wei H, Bao H. Boundary aligned smooth 3D cross-frame field. In: ACM transactions on graphics (TOG), vol. 30. ACM; 2011. p. 143.
[26] Li Y, Liu Y, Xu W, Wang W, Guo B. All-hex meshing using singularity-restricted field. ACM Trans Graph (TOG) 2012;31(6):177.
[27] Catmull E, Clark J. Recursively generated b-spline surfaces on arbitrary topological meshes. Comput-Aided Des 1978;10(6):350–5.
[28] Halstead M, Kass M, DeRose T. Efficient, fair interpolation using Catmull–Clark surfaces. In: Proc. SIGGRAPH'93. 1993. p. 47–61.
[29] Lai S, Cheng F. Similarity based interpolation using Catmull–Clark subdivision surfaces. Vis Comput 2006;22(9):865–73.
[30] Cheng F, Fan F, Lai S, Huang C, Wang J, Yong J. Loop subdivision surface based progressive interpolation. J Comput Sci Tech 2009;24(1):39–46.
[31] Fan F, Lai S. Subdivision based interpolation with shape control. Comput Aided Des Appl 2008;5(1–4):539–47.
[32] Chen Z, Luo X, Tan L, Ye B, Chen J. Progressive interpolation based on Catmull–Clark subdivision surfaces. Comput Graph Forum 2008;27(7):1823–7.
[33] Maekawa T, Matsumoto Y, Namiki K. Interpolation by geometric algorithm. Comput-Aided Des 2007;39:313–23.
[34] Nishiyama Y, Morioka M, Maekawa T. Loop subdivision surface fitting by geometric algorithms. In: T. Igarashi, N. Max, F. Sillion (Eds.), Poster proceedings of pacific graphics 2008. 2008. p. 67–74.
[35] MacCracken R, Joy K. Free-form deformations with lattices of arbitrary topology. In: Proceedings of the 23rd annual conference on computer graphics and interactive techniques. ACM; 1996. p. 181–8.
[36] Bajaj C, Schaefer S, Warren J, Xu G. A subdivision scheme for hexahedral meshes. Vis Comput 2002;18(5):343–56.
[37] Pascucci V. Slow growing subdivision (SGS) in any dimension: Towards removing the curse of dimensionality. In: Computer graphics forum, vol. 21. Wiley Online Library; 2002. p. 451–60.
[38] Burkhart D, Hamann B, Umlauf G. Adaptive and feature-preserving subdivision for high-quality tetrahedral meshes. In: Computer graphics forum, vol. 29. Wiley Online Library; 2010. p. 117–27.
[39] Pottmann H, Steiner T, Hofer M, Haider C, Hanbury A. The isophotic metric and its application to feature sensitive morphology on surfaces. In: Pajdla T, Matas J, editors. Computer vision—ECCV 2004. Lecture notes in computer science, vol. 3024. Berlin (Heidelberg): Springer; 2004. p. 18–23.
[40] Lai Y, Hu S, Pottmann H. Surface fitting based on a feature sensitive parametrization. Comput-Aided Des 2006;38(7):800–7.
[41] Tautgesa TJ, Knoopb SE. Topology modification of hexahedral meshes using atomic dual-based operations. Algorithms 2003;11:12.
[42] https://www.rocq.inria.fr/gamma/gamma/Membres/CIPD/Loic.Marechal/Research/Hexotic.html [Online; accessed 15.12.14].
[43] http://www.meshgems.com/volume-meshing-meshgems-hexa.html [Online; accessed 15.12.14].
[44] http://www.salome-platform.org/ [Online; accessed 15.12.14].
[45] Lin H, Liao H, Deng C. Filling triangular mesh model with all-hex mesh by volume subdivision fitting, Tech. Rep. TR-ZJUCAD-2012-002. Zhejiang University; 2012.