Filling Triangular Mesh Model with All-Hex Mesh by Volume Subdivision Fitting

Hongwei Lin* Zhejiang University
Hongwei Liao† Zhejiang University
Chongyang Deng‡ Hangzhou Dianzi University

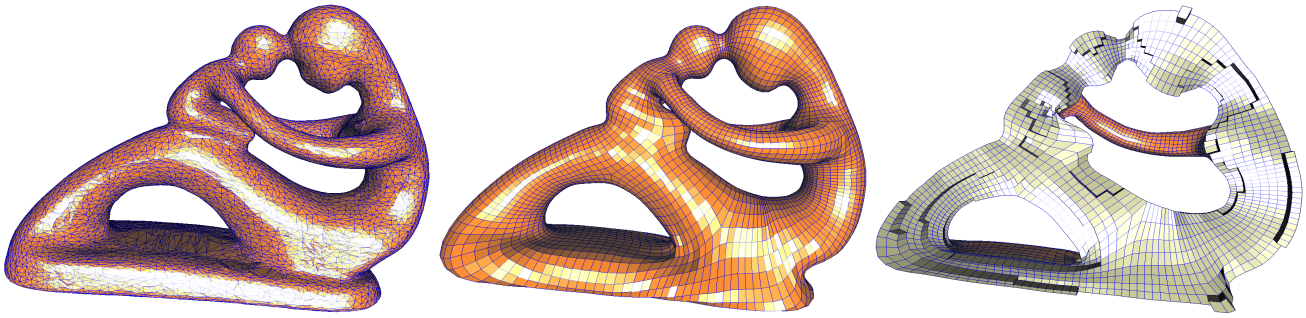


Figure 1: *Fertility model. Left: given triangular mesh model. Middle: all-hex volume mesh filling the left mesh model. Right: cutaway view of the all-hex volume mesh.*

Abstract

The hexahedral mesh (hex mesh) is preferred to the tetrahedral mesh (tet mesh) in finite element methods for numerical simulation. However, generating a hex mesh with desirable qualities often requires significant geometric decomposition and considerable user interactions. Therefore, this process may require days or even weeks in the case of complex shapes. In this paper, we develop a method based on subdivision fitting to fill a given triangular mesh model with an all-hex volume mesh. Our method first constructs an initial control solid, which consists of a face-to-face combination of several cubes, based on the skeleton of the given model. The orientation of each cube is determined by the local orientation of the skeleton and the local shape of the given model. Therefore, the shape of the control solid is close to that of the given model. Next, the surface mesh of the initial control solid is extracted and fitted to the given mesh model by an iterative subdivision fitting method. In each iteration, the movement of the surface mesh is diffused to the inner vertices by an optimization technique. After the iteration stops, an all-hex volume mesh that fills the given triangular mesh model can be generated by subdividing the control solid with the multi-linear cell-averaging (MLCA) volume subdivision rule. The smoothness of the MLCA subdivision rule guarantees that the quality of the generated all-hex volume mesh is good. Empirical data show that our algorithm is effective and efficient.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms;

Please cite this paper as:

Hongwei Lin, Hongwei Liao, Chongyang Deng. Filling Triangular Mesh Model with All-Hex Mesh by Volume Subdivision Fitting. Technical Report, TR_ZJUCAD_2012_002, State Key Lab. of CAD&CG, Zhejiang University

Keywords: solid modeling, hexahedral mesh generation, volume subdivision fitting

*e-mail: hwlin@cad.zju.edu.cn
†e-mail:liaohongwei@zjucadcg.cn
‡e-mail:dcy@hdu.edu.cn

Links: DL PDF

1 Introduction

In finite element methods for numerical simulation, the hexahedral mesh (hex mesh) is preferred to the tetrahedral mesh (tet mesh) owing to the reduced error and smaller number of elements [Shepherd and Johnson 2008]. 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 [Shepherd 2007].

In the hex mesh generation methods, the mapping and sub-mapping methods can usually generate high quality hex mesh. Recently, an efficient *sub-mapping* method based on PolyCube was presented [Han et al. 2010; Gregson et al. 2011]. Given a volume mesh model (usually a tet mesh), this method first constructs a PolyCube [Tarini et al. 2004]. A PolyCube 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 general, the closer the shape of the PolyCube to that of the input model and the smaller the distortion of the mapping between the PolyCube and the model, the better is the quality of the hex mesh transferred to the input model. However, the orientations of the cubes comprising the PolyCube are the same. Then, in cases where

there are some branches in the input model far away from the axes of the Polycube, the shape of the PolyCube will be far different from that of the input model. This will lead to large mapping distortions and a hex mesh of poor quality. Moreover, achieving low-distortion mapping is also difficult. In fact, calculating the PolyCube structure and identifying a low-distortion mapping between the PolyCube and the input model for general shapes remains an open problem [Xia et al. 2010; Gregson et al. 2011].

In this paper, we develop a method based on subdivision fitting to fill a given triangular mesh model with an all-hex volume mesh. This problem is more difficult than that converting a tet mesh to hex mesh, but very useful because the triangular mesh models are widely employed in CAD and computer graphics community. Given a triangular mesh model, we first construct an *initial control solid* based on its skeleton. This control solid is also a face-to-face combination of several cubes. However, unlike in the construction of a PolyCube, the orientation of each cube in the control solid is determined by the local orientation of the skeleton and the local shape of the input model. Therefore, the orientations of the cubes can differ and the shape of the initial control solid is closer to the input model than that of the PolyCube.

Furthermore, the surface mesh of the initial control solid is extracted and fitted to the input mesh model by an iterative subdivision fitting method. In each iteration, the movement of the surface mesh is diffused to the inner vertices by an optimization technique. Finally, the iteration stops when the fitting error between the subdivision surface mesh and the input mesh satisfies the termination condition. Thus, an all-hex volume mesh that fills the input triangular mesh model can be generated by subdividing the control solid with the multi-linear cell-averaging (MLCA) volume subdivision rule [Bajaj et al. 2002]. Owing to the smoothness of the MLCA subdivision volume, the quality of the generated all-hex mesh is good.

The structure of this paper is as follows. In Section 2, we briefly review related work. In Section 3, we present an overview of the entire algorithm. Moreover, we develop the initial control solid construction method in Section 4 and the volume subdivision fitting algorithm in Section 5. After presenting some results and discussions in Section 6, we conclude the paper in Section 7.

2 Related Work

Hex mesh generation: There is a great deal of literature on the generation of volume meshes including tet [Labelle and Shewchuk 2007; Tournois et al. 2009] and hex meshes. In this paper, we focus on hex mesh generation. According to Owen's classification [Owen 1998], hex mesh generation methods can be categorized into three classes, i.e., direct, indirect, and structured methods.

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 [Blacker and Meyers 1993; Staten et al. 2005].

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 [Melander et al. 1997; Owen and Saigal 2000].

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* [Cook and Oakes 1982], 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 [White et al. 1995]. 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 [Han et al. 2010]. In [Li et al. 2010], the method of fundamental solutions is employed to design a harmonic volumetric mapping. In [Xia et al. 2010], 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 [Gregson et al. 2011]. However, computing the PolyCube as well as a low-distortion mapping between it and the given model for general shapes remains an open problem [Xia et al. 2010].

For more work on hex mesh generation, we refer the reader to excellent surveys [Shepherd and Johnson 2008; Owen 1998].

Subdivision Surface Fitting: The limit surface of approximating subdivision schemes, such as Catmull-Clark scheme [Catmull and Clark 1978] 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 [Halstead et al. 1993], or local methods [Lai and Cheng 2006].

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 [Cheng et al. 2009], Doo-Sabin [Fan and Lai 2008], and Catmull-Clark schemes [Chen et al. 2008]. On the other hand, [Maekawa et al. 2007] develop a geometric interpolation algorithm for the Loop subdivision scheme. Moreover, [Nishiyama et al. 2008] present 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 an iterative subdivision surface 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. These schemes can be categorized into two classes: hexahedral and tetrahedral. To our best knowledge, the first volume subdivision scheme was developed in [MacCracken and Joy 1996] 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 [Bajaj et al. 2002]. The MLCA subdivision rule can be applied in any dimension, including 2D quadrilateral surface mesh, and 3D hex volume mesh.

On the other hand, tet mesh subdivision schemes have been devised by generalizing the subdivision rules for trivariate box splines, either with [Chang et al. 2002] or without a preferred direction [Schaefer et al. 2004].

In this paper, we develop a progressive volume subdivision fitting method for filling the given triangular mesh model.

3 Overview

Our algorithm takes a triangular mesh P as input and outputs an all-hex mesh that fills the inside of P. The algorithm can be divided into two steps: *initial control solid construction* and *progressive volume subdivision fitting*.

Initial Control solid construction: Given a triangular mesh P, we first extract its skeleton, and decompose it into several levels of branches, each of which is actually a piece of polyline. We then construct a sweeping solid along each branch. Finally, the sweeping solids are merged into a unified solid by a grafting operation. This unified solid of all-hex mesh is taken as the initial control solid, whose shape is much closer to the given mesh model P than that of the PolyCube.

Progressive volume subdivision fitting is an iterative method that consists of two parts: progressive MLCA surface subdivision fitting (abbr. progressive MSS fitting) and movement diffusion from the surface mesh vertices to the inner vertices. Given an initial control solid, we extract its quadrilateral surface mesh and develop a progressive MSS fitting algorithm to fit the surface mesh to the given mesh P. The progressive MSS fitting algorithm is performed iteratively, and the movements of the surface mesh vertices are transferred to the inner vertices of control solid after each iteration. When the iteration stops, the all-hex volume mesh that fills the given mesh model P can be generated by subdividing the final control solid with the MLCA rule.

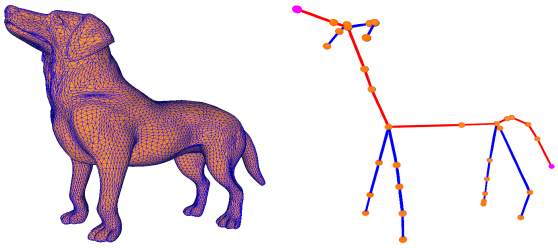
4 Initial Control Solid Construction

The shape of the initial control solid is one of the major factors that influences the quality of the generated hex mesh. In this section, we develop a method for constructing the initial control solid. The shape of the constructed control solid is not only very close to the given mesh P, but also easily discretized into an all-hex mesh.

4.1 Skeleton extraction and decomposition

Given a mesh model P, the control solid construction algorithm begins with its skeleton. We employ the mesh contraction algorithm [Au et al. 2008] to generate the skeleton of mesh P (Fig. 2). The algorithm can output not only the skeleton of mesh P itself, but also information on which mesh vertices contract to a skeleton vertex. This information is useful in the following steps.

In fact, there are two types of vertices in the skeleton: *normal vertices* and *joint vertices*. A normal vertex has two adjacent edges, except the head and tail vertices, each of which has only one adjacent edge. A joint vertex has three or more adjacent edges (Fig. 2(b)).



(a) (b)

Figure 2: *Skeleton (b) of the given mesh model 'dog' (a). The polyline in red is the first level branch, and the polylines in blue are the second level branches.*

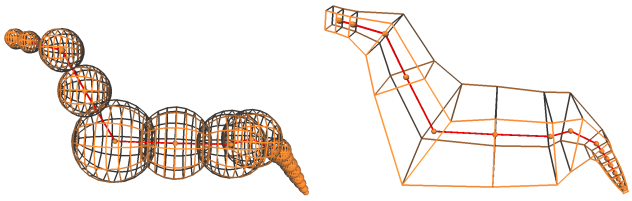
Next, regarding the so-constructed skeleton as a graph, it is decomposed into several levels of branches called *skeleton branches*, each of which is a piece of polyline (Fig. 2(b)). Of these branches, the first-level branch is the most important for the shape of the solid. To make the first-level branch desirable, we allow users to select a number of vertices of the skeleton and take the *shortest* path passing these vertices as the first-level branch. In our implementation, we use the Dijkstra algorithm to generate the shortest path.

Deleting the first-level branch from the skeleton, which is a graph, may leave a number of separate sub-graphs. In each sub-graph, we search the *longest* path as the second-level branch and delete it from the sub-graph. This procedure continues until no sub-graphs remain. In this way, the entire skeleton is decomposed into several branches (Fig. 2(b)).

Finally, based on the decomposed skeleton branches and the information about which mesh vertices contract to a skeleton vertex, the mesh model P can be segmented into several sub-meshes, each of which corresponds to a piece of skeleton branch.

4.2 Sweeping solid generation

As stated in Section 4.1, each branch level is a piece of polyline that corresponds to a sub-mesh. In this section, we construct a sweeping solid along each skeleton branch (Fig. 3).



(a) (b)

Figure 3: *Constructing a sweeping solid along a skeleton branch. (a) The sphere sequence. (b) The constructed sweeping solid.*

Given a piece of branch, we first insert spheres at its joint vertices, head and tail vertices, and *feature* vertices (i.e., those with angles less than 30°). The inserted spheres divide the branch into several segments. Along each segment, spheres are inserted by dichotomy until their union contains all of the branch. Sphere insertion is performed by the *averaging method* as follows.

Averaging method: To insert a sphere at a point on the branch, we

construct a plane perpendicular to the tangent vector of the branch at this point. This plane intersects with the mesh edges of the sub-mesh corresponding to the branch at a series of points $q_j, j = 1, 2, \cdots, m$. We take the averaging point $o = \frac{\sum_{j=1}^{m} q_j}{m}$ as the center of the inserted sphere, and the average value $\frac{\sum_{j=1}^{m} \|q_j - o\|}{m}$ as its radius.

If the sphere is inserted at a vertex $v_i, i = 1, 2, \cdots, n$ of a branch, its tangent vector is taken as,

$$\boldsymbol{t}_i = \begin{cases} v_2 - v_1, & i = 1; \\ \frac{v_{i+1}-v_i}{\|v_{i+1}-v_i\|} - \frac{v_{i-1}-v_i}{\|v_{i-1}-v_i\|}, & 1 < i < n; \\ v_n - v_{n-1}, & i = n. \end{cases} \quad (1)$$

On the other hand, if the sphere is inserted at an inner point on an edge of the branch, the edge is taken as the tangent line at this point.

Intersection and self-intersection elimination: Before constructing the sweeping solid, the inserted sphere sequence for each branch should be further processed to avoid self-intersection and unwanted intersection between branches.

In fact, such intersection and self-intersection can be avoided if the sphere sequences satisfy the following conditions:

- In each sphere sequence, one sphere intersects with at most two adjacent spheres;
- two sphere sequences at the same level can not intersect with each other;
- the sphere at the joint vertex intersects with each adjacent higher level sphere sequence at a single sphere.

If a sphere violates the conditions, the related spheres should be shrunk in proportion to the ratio of their radii until they satisfy the condition.

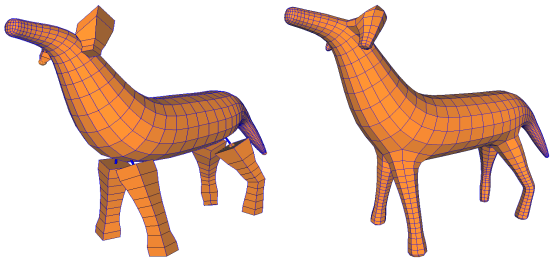
We are now in a position to construct the sweeping solid for each branch. Recall that in constructing the sphere sequence, we place a plane, which intersects with the corresponding sub-mesh at a series of points, at the center of each sphere. On each plane, we first establish a Cartesian coordinate system, (to be explained in the following paragraph), and then compute the axis-aligned bounding box of these intersection points. In this way, a sequence of rectangles is constructed, each on a plane at the center of a sphere. Finally, by connecting these rectangles in order, we can generate the sweeping solid, which is called a *control solid branch* (Fig. 3(b)).

Construction of Cartesian coordinate system: The orientation of the Cartesian coordinate system has an effect on the degree of distortion of the constructed sweeping solid. To reduce this distortion, in constructing the sweeping solid along the first-level skeleton branch, we compute the oriented bounding box (OBB) of its vertices and project the shortest edge (with direction) of the OBB to each plane as the first axis. Moreover, in constructing the sweeping solid along the second- or higher-level skeleton branch, the tangent vector (see Eq. (1)) of the lower-level branch at the joint vertex, which connects the current and the lower-level branch, is projected onto each plane as the first axis. The second axis is perpendicular to the first axis.

4.3 Hex mesh grafting

The so-constructed control solid branches are separated, each corresponding to a piece of skeleton branch. To integrate them into a unified control solid, we develop a level-by-level *mesh grafting* strategy. That is, the second-level solid branches are grafted to the first level, the third-level branches to the second level, $\cdots$, and so on.

Before grafting, the first-level solid branch is subdivided twice by using the MLCA volume subdivision rule [Bajaj et al. 2002] (Fig. 4(a)). In the following section, we take the second-level solid branch as an example to elucidate the mesh grafting method, which is identical for the next-level solid branches.



(a) (b)

Figure 4: *Mesh grafting. (a) Before grafting. (b) After grafting and Laplacian smoothing, the unified solid is taken as the control solid.*

Mesh grafting: The higher-level (the second-level) solid branch is grafted to the lower level (the first level) by first deleting the hexahedra of the higher-level branch, which intersect with the lower-level branch, and then inserting a number of hexahedra connecting the two branches (Fig.4(a)).

After deleting the hexahedra of the second-level branch, which intersect with the first level, the two levels of solid branches are still connected by the skeleton (Fig. 4(a)). Suppose the skeleton intersects the second-level solid at point p_s on a hexahedral face S, and intersects the first-level solid at point p_f on a hexahedral face F. In this way, the skeleton provides a natural correspondence of face F to face S between the solids of the two levels.

The mesh grafting algorithm searches a suitable $n \times n$ hexahedral region R_n around the face F, where the 1×1 region R_1 is simply the face F itself, and connects the corresponding vertices of R_n and S to form a hexahedron. The correspondence between the vertices should render the distortion of the generated hexahedron as small as possible.

Suppose the vertices of R_n and S are $r_i^n, s_i, i = 0, 1, 2, 3$, respectively. To determine the correspondence between them, we define two energy functions, E_{dis}^n for the distance, and E_{dir}^n for the direction, i.e.,

$$E_{dis}^n(j) = \sum_{i=0}^{3} \left\| s_i - r_{(j+i) \bmod 4}^n \right\|, \quad (2)$$

and,

$$E_{dir}^n(j) = \sum_{i=0}^{3} \frac{s_i - r_{(j+i) \bmod 4}^n}{\left\| s_i - r_{(j+i) \bmod 4}^n \right\|} \cdot \frac{p_s - p_f}{\|p_s - p_f\|}, \quad (3)$$

where $\cdot$ is the dot product. The correspondence between r_i^n and s_i is more desirable if the sum of the distances between them (i.e., E_{dis}^n) is smaller and if the direction of the vector $s_i r_i^n$ is closer to the direction of the local skeleton $p_s p_f$, (namely, the E_{dir}^n is greater). Therefore, combining the two energies yields the total energy function,

$$E_{total}^n(j) = \alpha E_{dis}^n(j) + \beta \frac{1}{E_{dir}^n(j)}, \quad (4)$$

where α and β are weights. In our implementation, they are taken as $\alpha = 0.6$, and $\beta = 0.4$.

For each n, we compute a correspondence between s_i and $r_i^n, i = 0, 1, 2, 3$ that minimizes E_{total}^n by enumeration, and denote it as $E_{total}^n(j_n)$. Then, with the expansion of the region R_n, we obtain an energy sequence $\{E_{total}^n(j_n), n = 1, 2, \cdots\}$. The energy will decrease if the region expansion can reduce the distortion. This means that when $E_{total}^{n+1}(j_{n+1}) > E_{total}^n(j_n)$, the region expansion will increase the distortion. In this case, we stop the expansion and connect the corresponding vertices between s_i and r_i^n, forming a hexahedron connecting the solids of the two levels (Fig. 4(b)).

Furthermore, each hexahedron in the second-level solid is subdivided into an $n \times n \times n$ volume mesh, and the correspondence between the vertices on the face S and R_n can be inferred from that between s_i and $r_i^n, i = 0, 1, 2, 3$. Finally, the two levels of solid branches are grafted by connecting the corresponding vertices on S and R_n. After Laplacian smoothing, it can be taken as the initial control solid (Fig. 4(b)).

5 Progressive Volume Subdivision Fitting

Currently, the solid branches are integrated into an initial control solid with a quadrilateral surface mesh V. Starting with the initial control solid, we develop a *progressive volume subdivision fitting* algorithm to make the subdivided volume fill the space enclosed by the given mesh P, while the subdivided surface mesh approximates the mesh P.

The volume subdivision fitting algorithm is an iterative method, where each iteration includes two procedures: the movement of the vertices of the surface mesh V, and the movement diffusion from the vertices of mesh V to the inner vertices of the control solid. In fact, the vertex iterations of mesh V constitute the *progressive MLCA surface subdivision fitting* (abbr. *progressive MSS fitting*) algorithm, which will be elucidated in Section 5.1.

When the iteration stops, an all-hex volume mesh that fills the space enclosed by the given mesh model P can be generated by subdividing the control solid with the MLCA volume subdivision rule. Owing to the smoothness of the MLCA rule, the generated all-hex volume mesh is of good quality.

5.1 Progressive MLCA surface subdivision fitting

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 progressive MSS fitting 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 MLCA surface subdivision procedure, a mesh surface denoted as V_γ will be generated after γ time MLCA 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 progressive MSS fitting includes the following steps:

1. For each data point p, compute the closest mesh vertex $v_{cl,\gamma}$ on mesh V_γ;
2. 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 $v_{cl,\gamma}$;
3. for each control point v, averaging the difference vectors distributed to it produces the difference vector $\boldsymbol{\Delta}$ for the control point;
4. adding the difference vector $\boldsymbol{\Delta}$ to the control point v generates the new control point v^{new} of the new control mesh,

$$v^{new} = v + \boldsymbol{\Delta}. \tag{5}$$

The first step is to compute the closest vertex $v_{cl,\gamma}$ on the mesh surface V_γ for each vertex p, which will cost a large amount of computation if the closest point is searched globally. To save computation, we select ten percent of vertices of mesh P with evenly spaced serial numbers as seeds. The vertices on the mesh V_γ which are closest to the seeds are first calculated *globally*. Then, the other vertices of mesh P are processed in order according to their proximity to the seeds, i.e., first the one-ring adjacencies, followed by the two-ring adjacencies, $\cdots$, and so on, by searching the vertices on V_γ with *local* minimum distances to them. For such a vertex p_c on P, the local search starts with a vertex on V_γ, which is closest to a vertex at the one ring adjacencies of p_c. The searching area is expanded gradually until a vertex on V_γ with the local minimum distance to p_c is found.

It should be pointed out that in the first five iterations, we compute the closest vertex on the mesh surface V_γ for each data point on mesh P. Afterwards, the parameters of the closest points on the mesh surface V_γ 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*, i.e.,

$$\boldsymbol{\delta} = p - v_{cl,\gamma}. \tag{6}$$

This difference vector will be distributed to the related control points of the control mesh V (Fig. 5). Recall that the mesh vertex on mesh V_γ is a linear combination of the related control points $v_1, v_2, \cdots, v_n$ of the control mesh V, i.e.,

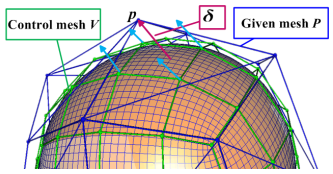


Figure 5: *The difference vector (in red) for data point is distributed to related control points (vectors in blue).*

$$v_{cl,\gamma} = c_1 v_1 + c_2 v_2 + \cdots + c_n v_n, \text{ with } \sum_{i=1}^{n} c_i = 1, \tag{7}$$

where the coefficients $c_i, i = 1, 2, \cdots, n$ can be easily obtained by the MLCA surface subdivision rule [Bajaj et al. 2002]. Thus, the weighted difference vector $c_i\boldsymbol{\delta}$ is distributed to the control point $v_i, i = 1, 2, \cdots, n$ (Fig. 5).

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,

$$\boldsymbol{\Delta} = \frac{\sum_j c_j \boldsymbol{\delta}_j}{\sum_j c_j}, \tag{8}$$

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. (5)) of the new control mesh.

The four steps above are performed iteratively until

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

where $e^{(k)}$ is the root mean square (RMS) fitting error of the k^{th} iteration (see Appendix), and ε_0 is a threshold. In our implementation, we take $\varepsilon_0 = 10^{-3}$.

The convergence of the progressive MSS fitting is shown in the Appendix.

5.2 Movement diffusion

In the above section, we presented the progressive MSS fitting algorithm for surface mesh fitting. After each iteration of the progressive MSS fitting, the movement of the surface mesh vertices should be diffused to the inner vertices of the control solid. In this section, a *level Laplacian operation* is proposed 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 surface mesh V. Specifically, the inner vertices in one-ring adjacency to the surface 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 surface mesh vertices, the first-level vertices v are moved to the new position v_{new} by the following Laplacian operation,

$$v_{new} = \frac{\sum_{j=1}^{d(v)} v_j}{d(v)}, \tag{9}$$

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 (9), $\cdots$, 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}.

The progressive MSS fitting and the movement diffusion therein constitute the progressive volume subdivision fitting algorithm. When this algorithm stops, we obtain a control solid. An all-hex volume mesh filling the given triangular mesh P can be generated by subdividing the control solid γ times with the MLCA volume subdivision rule.

6 Results and Discussions

The all-hex mesh filling algorithm has been implemented with Visual C++, and run on a PC with Intel Core2 Quad CPU Q9400 2.66GHz and 4G memory. We test our algorithm by a lot of examples. Some of them are illustrated in Figs. 1, 6, and 7. The all-hex mesh generated by our algorithm has very few irregular vertices (Table 1), and the scaled Jacobian values of most hexes lie in the interval [0.8,1.0]. In our implementation, we take the mesh surface after one time subdivision as the fitting surface. The fitting precision between the surface of the all-hex mesh and the given mesh surface is guaranteed by the convergence of the progressive MSS fitting, and adding the vertices of the control mesh can improve the fitting precision. The runtime of our algorithm usually needs tens of seconds (Tabel 1).

Fig. 1(left) is a mesh model with complex shape and topology. By selecting the outer ring as the first level branch, our algorithm successfully generates the all-hex mesh filling the mesh model. The number of the irregular vertices in the all-hex mesh is only 36, while the total vertex number is 16449. The ratio between the numbers of irregular and total vertices is just 0.2%. The shape of the model *Santa* in Fig. 6 is complicated, our algorithm fills the interior of the model using all-hex mesh with total 18885 vertices and just 32 irregular vertices. The model *Torus Knits* in Fig 7 has complex topology, which is difficult to handle with PolyCube based methods. Our algorithm fills the model with structured all-hex mesh.

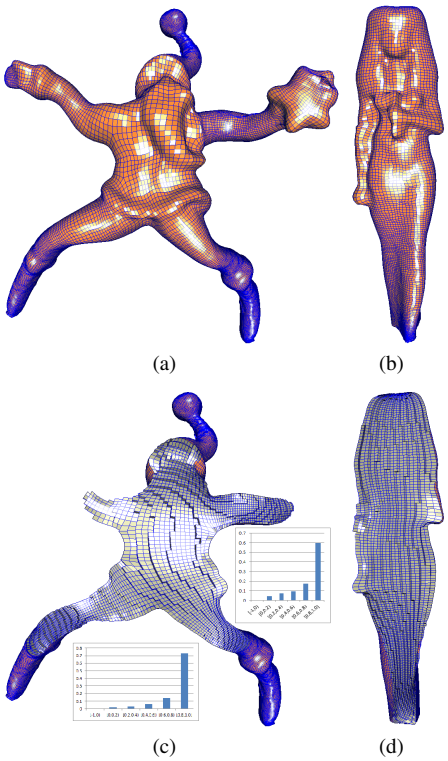


Figure 6: *Santa and Isis. (a,b) All-hex mesh filling a given triangular mesh. (c,d) Cutaway view of the all-hex mesh with the distribution diagram of the scaled Jacobian.*

Table 1 lists the statistics for our all-hex mesh filling algorithm. 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. The runtime is in seconds, not including the time for skeleton generation.

Limitations: At the vertex on highly concave area of the given mesh model, like the area near the ears of *Dog* in Fig. 7(b), the correspondence between it and the closest vertex at the subdivision surface may be wrong, this will lead to self-intersection on the resulted subdivision surface, and some hexahedra with negative scaled Jacobian values, such as the example in Fig. 7(b). In our implementation, after each of the initial five iterations of the progressive MSS fitting, the resulted control mesh is smoothed three times by the Laplacian operation. Although it can eliminate the self-intersection in most cases, it has not theoretical guarantee.

7 Conclusion

Given a triangular mesh model, an algorithm based on subdivision fitting is developed in this paper to fill the given mesh model with an all-hex volume mesh. Based on the skeleton of the triangular mesh model, this method first constructs an initial control solid comprising a face-to-face combination of several cubes. The orientation of each cube is determined by both the local orientation of the skeleton and the local shape of the given mesh model. Hence, the shape of the initial control solid is close to that of the mesh model. Moreover, the surface mesh of the initial control solid are fitted to the given mesh model by an iterative subdivision fitting procedure. In each iteration, the movements of the surface mesh vertices are diffused to the inner vertices of the control solid. When the iteration

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

Model	#vert. of mesh	#hex	#total vert. of hex	#irr. vert. of hex	Jac. avg.	Jac. min.	precision	time (s)
Fig. 1 Fertility	10994	12800	16449	36	0.752151	0.087822	0.007152	10.409
Fig. 6 Santa	10074	14472	18885	32	0.728218	0.086498	0.005058	17.910
Fig. 6 Isis	10957	52544	59941	27	0.743130	0.009021	0.003271	73.977
Fig. 7 Torus Knits	17136	33792	42768	0	0.865208	0.082744	0.002482	18.619
Fig 7 Dog	10058	11904	16097	62	0.721376	-0.099658	0.006301	9.274
Fig. 7 Rabbit	10389	32768	37281	8	0.764151	0.091468	0.005795	29.862
Fig. 7 Ball Joint	10134	41984	47685	31	0.738236	0.093269	0.004280	30.249

#vert. of mesh: number of the vertices in the given triangular mesh; #irr. vert. of hex: number of the irregular vertices in the all-hex mesh; #hex: number of the hexes in the all-hex mesh; Jac. avg. and Jac. min.: averaged scaled Jacobian, and minimum scaled Jacobian.

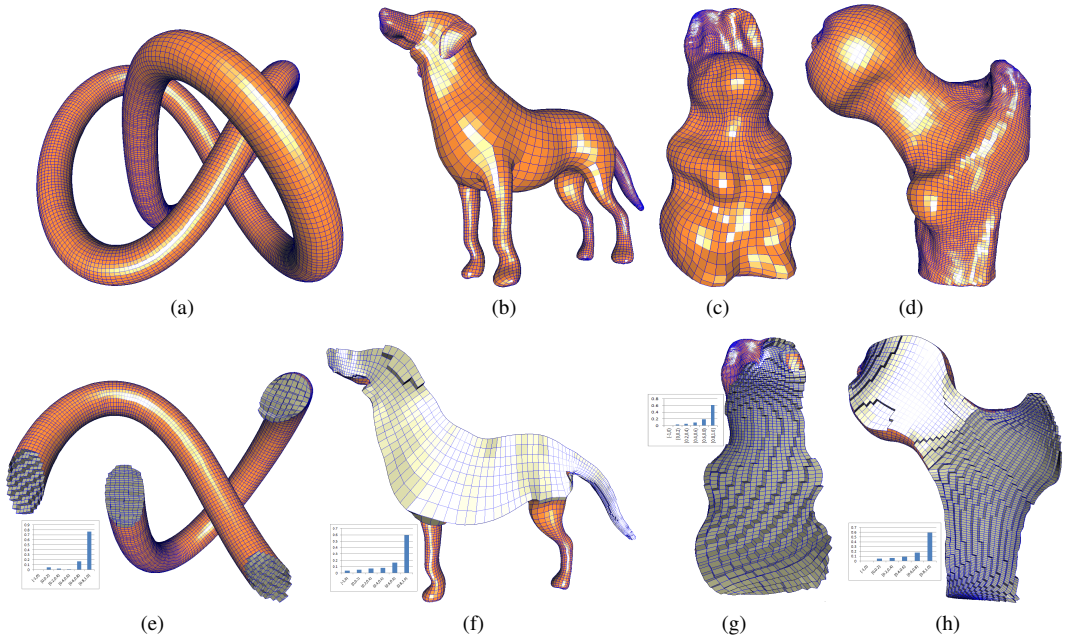


Figure 7: *All-hex meshes filling triangular meshes and their cutaway views. (a, b, c, d) All-hex meshes of Torus Knits, Dog, Rabbit, and Ball Joint. (e,f,g,h) Their cutaway views with the distribution diagrams of scaled Jacobian.*

stops, an all-hex volume mesh that fills the given mesh model can be generated by subdividing the control solid with the MLCA subdivision rule. The smoothness property of the MLCA rule guarantees that the all-hex volume mesh will be of good quality.

References

AU, O., TAI, C., CHU, H., COHEN-OR, D., AND LEE, T. 2008. Skeleton extraction by mesh contraction. In *ACM Transactions on Graphics (TOG)*, vol. 27, ACM, 44.

BAJAJ, C., SCHAEFER, S., WARREN, J., AND XU, G. 2002. A subdivision scheme for hexahedral meshes. *The visual computer 18*, 5, 343–356.

BLACKER, T., AND MEYERS, R. 1993. Seams and wedges in plastering: a 3-d hexahedral mesh generation algorithm. *Engineering with computers 9*, 2, 83–93.

CATMULL, E., AND CLARK, J. 1978. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design 10*, 6, 350–355.

CHANG, Y., MCDONNELL, K., AND QIN, H. 2002. A new solid subdivision scheme based on box splines. In *Proceedings of the seventh ACM symposium on Solid modeling and applications*, ACM, 226–233.

CHEN, Z., LUO, X., TAN, L., YE, B., AND CHEN, J. 2008. Progressive interpolation based on catmull-clark subdivision surfaces. *Computer Graphics Forum 27*, 7, 1823–1827.

CHENG, F., FAN, F., LAI, S., HUANG, C., WANG, J., AND YONG, J. 2009. Loop subdivision surface based progressive interpolation. *Journal of Computer Science and Technology 24*, 1, 39–46.

COOK, W., AND OAKES, W. 1982. Mapping methods for generating three-dimensional meshes. *Computers in Mechanical Engineering 1*, 1, 67–72.

FAN, F., AND LAI, S. 2008. Subdivision based interpolation with shape control. *Computer Aided Design & Applications 5*, 1-4, 539–547.

GREGSON, J., SHEFFER, A., AND ZHANG, E. 2011. All-hex mesh generation via volumetric polycube deformation. In *Computer Graphics Forum*, vol. 30, Wiley Online Library, 1407–1416.

HALSTEAD, M., KASS, M., AND DEROSE, T. 1993. Efficient, fair interpolation using catmull-clark surfaces. In *Proc. SIGGRAPH '93*, 47–61.

HAN, S., XIA, J., AND HE, Y. 2010. Hexahedral shell mesh construction via volumetric polycube map. In *Proceedings of the 14th ACM Symposium on Solid and Physical Modeling*, ACM, 127–136.

LABELLE, F., AND SHEWCHUK, J. 2007. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. *ACM Transactions on Graphics (TOG) 26*, 3, 57.

LAI, S., AND CHENG, F. 2006. Similarity based interpolation using catmull-clark subdivision surfaces. *The Visual Computer 22*, 9, 865–873.

LI, X., XU, H., WAN, S., YIN, Z., AND YU, W. 2010. Feature-aligned harmonic volumetric mapping using mfs. *Computers & Graphics 34*, 3, 242–251.

MACCRACKEN, R., AND JOY, K. 1996. Free-form deformations with lattices of arbitrary topology. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM, 181–188.

MAEKAWA, T., MATSUMOTO, Y., AND NAMIKI, K. 2007. Interpolation by geometric algorithm. *Computer-Aided Design 39*, 313–323.

MELANDER, D., BENZLEY, S., AND TAUTGES, T. 1997. Generation of multi-million element meshes for solid model-based geometries: The dicer algorithm. Tech. rep., Sandia National Labs., Albuquerque, NM (United States).

NISHIYAMA, Y., MORIOKA, M., AND MAEKAWA, T. 2008. Loop subdivision surface fitting by geometric algorithms. In *Poster Proceedings of Pacific Graphics 2008*, T. Igarashi, N. Max, and F. Sillion, Eds., 67–74.

OWEN, S., AND SAIGAL, S. 2000. H-morph: an indirect approach to advancing front hex meshing. *International Journal for Numerical Methods in Engineering 49*, 1-2, 289–312.

OWEN, S. 1998. A survey of unstructured mesh generation technology. In *7th International Meshing Roundtable*, vol. 3, Citeseer.

SCHAEFER, S., HAKENBERG, J., AND WARREN, J. 2004. Smooth subdivision of tetrahedral meshes. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, ACM, 147–154.

SHEPHERD, J., AND JOHNSON, C. 2008. Hexahedral mesh generation constraints. *Engineering with Computers 24*, 3, 195–213.

SHEPHERD, J. 2007. *Topologic and geometric constraint-based hexahedral mesh generation*. PhD thesis, The University of Utah.

STATEN, M., OWEN, S., AND BLACKER, T. 2005. Unconstrained paving & plastering: A new idea for all hexahedral mesh generation. In *Proceedings of the 14th International Meshing Roundtable*, Springer, 399–416.

TARINI, M., HORMANN, K., CIGNONI, P., AND MONTANI, C. 2004. Polycube-maps. In *ACM Transactions on Graphics (TOG)*, vol. 23, ACM, 853–860.

TOURNOIS, J., WORMSER, C., ALLIEZ, P., AND DESBRUN, M. 2009. Interleaving delaunay refinement and optimization for practical isotropic tetrahedron mesh generation. In *ACM Transactions on Graphics (TOG)*, vol. 28, ACM, 75.

WHITE, D., MINGWU, L., BENZLEY, S., AND SJAARDEMA, G. 1995. Automated hexahedral mesh generation by virtual decomposition. In *Proceedings of the 4th International Meshing Roundtable, Sandia National Laboratories, Albuquerque, USA*, Citeseer, 165–176.

XIA, J., HE, Y., YIN, X., HAN, S., AND GU, X. 2010. Direct-product volumetric parameterization of handlebodies via harmonic fields. In *Shape Modeling International Conference (SMI), 2010*, IEEE, 3–12.

Appendix: Convergence of the progressive MSS fitting

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 $v_i^{(k)}, i = 1, 2, \cdots, n$, called *control points*. Subdividing the control mesh $V^{(k)}$ γ times generates the mesh surface $V_\gamma^{(k)}$. Moreover, suppose the given mesh P have m vertices $p_j, j = 1, 2, \cdots, m$, called *data points*, where $m > n$. To show the convergence of the progressive MSS fitting, we rewrite Eqs. (6)-(8) with indices in the following.

In the k^{th} iteration, the difference vector for the data point p_j is,

$$\boldsymbol{\delta}_j^{(k)} = p_j - v_{j,\gamma}^{(k)}, \tag{10}$$

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.,

$$v_{j,\gamma}^{(k)} = c_{j,1}v_1^{(k)} + c_{j,2}v_2^{(k)} + \cdots + c_{j,n}v_n^{(k)}, \text{ with } \sum_{i=1}^{n} c_{j,i} = 1. \tag{11}$$

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

$$\boldsymbol{\Delta}_i^{(k)} = \frac{\sum_{j\in I_i} c_{j,i}\boldsymbol{\delta}_j^{(k)}}{\sum_{j\in I_i} c_{j,i}} = \sum_{j\in I_i} \frac{c_{j,i}}{\sum_{j\in I_i} c_{j,i}}\boldsymbol{\delta}_j^{(k)}, \tag{12}$$

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 $\boldsymbol{\Delta}_i^{(k)}$ for the control point $v_i^{(k)}$. Finally, adding $\boldsymbol{\Delta}_i^k$ to the control point v_i^k generates the new control point, i.e.,

$$v_i^{(k+1)} = v_i^{(k)} + \boldsymbol{\Delta}_i^{(k)}, i = 1, 2, \cdots, n.$$

Arranging the difference vectors for control points in a sequence,

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

the iterative format can be represented in matrix form,

$$\Delta^{(k+1)} = (I - \Lambda A^T A)\Delta^{(k)}, \tag{13}$$

where I is an identity matrix, Λ is a diagonal matrix,

$$\Lambda = diag\{\frac{1}{\sum_{j\in I_1} c_{j,1}}, \frac{1}{\sum_{j\in I_2} c_{j,2}}, \cdots, \frac{1}{\sum_{j\in I_n} c_{j,n}}\},$$

and,

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

On one hand, if $A^T A$ is nonsingular, it is positive definite, so all of its eigenvalues are positive, as well as that of $\Lambda A^T A$, i.e., $\lambda(\Lambda A^T A) > 0$. On the other hand, since $\left\|\Lambda A^T A\right\| = 1$, all of its eigenvalues are less than or equal to 1, i.e., $\lambda(\Lambda A^T A) \leq 1$. Therefore, the eigenvalues of the matrix $I - \Lambda A^T A$ satisfy $0 \leq \lambda(I - \Lambda A^T A) < 1$. It means that the progressive MSS fitting (13) is convergent.