Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 82 (2014) 325 – 337

Procedia Engineering

www.elsevier.com/locate/procedia

23rd International Meshing Roundtable (IMR23)

Surface Mesh Generation based on Imprinting of *S*-*T* Edge Patches

Shengyong Cai[a], Timothy J. Tautges[b]

[a] *University of Wisconsin-Madison, Madison, WI 53706, U.S.A*
[b] *CD-adapco, 1500 Engineering Drive, Madison, WI 53706, U.S.A*

Abstract

One of the most robust and widely used algorithms for all-hexahedral meshes is the sweeping algorithm. However, for multi-sweeping, the most difficult problems are the surface matching and interval assignment for edges on the *source* and *target* surfaces. In this paper, a new method to generate surface meshes by imprinting edge patches between the *source* and *target* surfaces is proposed. The edge patch imprinting is based on a cage-based morphing of edge patches on the different sweeping layers where deformed and undeformed cages are extracted by propagating edge patches on the *linking* surfaces. The imprinting results in that the *source* or *target* surfaces will be partitioned with the imprinted edge patches. After partitioning, every new *source* surface should be matched to a new specific *target* surface where surface mesh projection from one-to-one sweeping based on harmonic mapping[19] can be applied. In addition, 3D edge patches are projected onto 2D computational domains where every sweeping level is planar in order to increase the robustness of imprinting. Finally, the algorithm time complexity is discussed and examples are provided to verify the robustness of our proposed algorithm.

Peer-review under responsibility of organizing committee of the 23rd International Meshing Roundtable (IMR23)
Keywords: Imprinting, MultiSweeping, Hexahedral, Mesh Generation, *n*-to-*n*, Edge Patches;

1. Introduction

Computational simulations depend on the numerical approximation methods such as finite element, finite difference, and finite volume methods. A critical part of those numerical methods is to discretize domains or models. In many applications such as Computational Fluid Dynamics (*CFD*)[1] and Computational Structural Mechanics (*CSM*)[2], a hexahedral mesh is preferred over a tetrahedral mesh. However, fully automatic hexahedral mesh generation for any 3D objects is still an open problem[18]. The main difficulty to overcome in hexahedral meshing is that hexahedral meshes have a global topological structure that any meshing algorithm must take into account[18]. This characteristic is very restrictive and can explain why fully automatic hexahedral meshing is difficult. Current existing methods include *Division & Combination*[3], *Grid-based*[4], *Medial Surface generation*[5,6], *Plastering*[7], *Whisker Weaving*[8] and *Sweeping*[9]. While all-hexahedral meshes on general 3D geometries remains an elusive

* Corresponding author. Tel.: +1-608-265-8086 ; fax: +0-000-000-0000.
E-mail address: shengyongcai@gmail.com

Peer-review under responsibility of organizing committee of the 23rd International Meshing Roundtable (IMR23)
doi:10.1016/j.proeng.2014.10.394

goal, algorithms to mesh two-and-one-half dimensional geometries, generally referred to as sweeping or projection methods, continue to be important[9–12].

The general sweeping procedure for all-hex meshes consists of four steps: (1)generate surface meshes on the *source* surfaces; (2)map *source* surface meshes onto the *target* surfaces; (3)generate structured meshes on the *linking* surfaces; (4)generate hexahedral meshes including interior nodes and volume elements. The terms "two-and-one-half dimensional", "source", "target" and "many-to-one" are defined in Ref.[11,23]. Since other parts of the sweeping process have already addressed effectively such as surface mesh mapping[19], previous efforts on multisweeping and imprinting in particular are reviewed as follows.

Ruiz-Girones et al.[20] presented a new procedure to decompose geometries into Many-to-One sweepable sub-volumes based on the loop face projection and imprinting. The decomposition relied on a least-square approximation of affine mappings defined between loops of nodes that bound the sweeping levels. First, geometries were decomposed by advancing loops from the *target* surfaces to the *source* surfaces, which was achieved by computing a least-square approximation in the physical space and 3D representation of the computational space. The projection in the physical space was a first approximation to locations of inner nodes and was used for the imprinting procedure. Second, inner nodes mapped onto *source* surfaces were mapped back to the *target* surfaces by using least-square approximation in the physical domain. The final location of inner nodes was a weighted average of the above two projections. The projected and original loops were used to decompose the domain into single *source* and *target* sweepable volumes. This approach suffers from the fact that this decomposition constrains the final mesh geometrically, when really it is just an artifact of the meshing approach. What's more, inappropriate volume decomposition can be produced if there is a volume with multi-connected surfaces and complicated internal structure due to the affine transformation method used, does not work for multi-connected geometries and does not have a property of local deformation(i.e. local boundary deformations produce only local changes in the resulting mesh).

White et al.[22] proposed CCSweep for all-hex meshes which decomposed multi-sweepable volumes into many-to-one sweepable volumes. The decomposition was based on projecting the *target* surfaces through the volume onto corresponding *source* surfaces. The new resulting volumes had only a single *target* surface. The affine transformation matrix and the corresponding nodal residual errors were used for projecting *target* surfaces onto the *source* surfaces. The problem is that this method fails for geometries with high distorted internal structures in that the affine transformation takes all the boundary nodes as a whole and does not work for geometries with local deformation on the boundaries.

Miyoshi et al.[24] presented a multi-axis cooper tool to generate all-hex meshes by using multi-axis imprinting sweeps. The geometries were automatically recognized and divided into hierarchical sub-volumes, which were then meshed by the existing single-axis sweep tools. The resulting volumes contained the individual meshes which were non-conformal or discontinuous at their interfaces. Those sections were then removed and replaced with a conformal mesh using the Cooper tool[25]. However, it fails for highly concave and non-simply connected geometries since it is a linear method. Also, it suffers from drawbacks of volume decomposition.

The existing multi-sweeping methods have several restrictions and limitations:

(1) *Indeterminate mesh edge sizing*[22] This occurs when edges on the *target* surfaces are projected onto edges of the *source* surfaces. Intersections requires node matching, that is, the edge sizing on the *target* surfaces must match that of the *source* surfaces. Prior to multi-sweeping, this kind of information is not known.
(2) *Over-dependence input mesh discretization*[22] This results from boolean operation of edge patches relying solely on node matching. The way in which the *source* and *target* faces will be imprinted depends on the boundary discretization. The false inherent assumption made by the existing algorithms is that the size used for meshing the volume is the best size for imprinting *source* and *target* surfaces. This is typically not the case.
(3) *Unstable loop imprinting when interior holes exist* There are two cases: the first one is: the failure when attempting to determine whether a 3D loop is completely inside another 3D loop by the *Winding Number* algorithm[27]. Fortunately, Ruiz-Girones et al.[20] have solved this kind of problem by proposing to project 3D loops onto the planar computational domain. The other case is that problems arise when interior hole size varies. The existing algorithms[20,22] handle this problem by using affine transformation. However, linear affine transformation takes all the boundary nodes as a whole and lacks the property of local deformation.

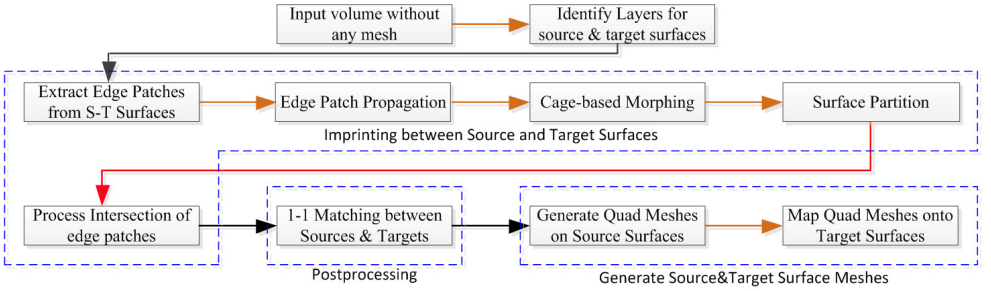

Fig. 1. Flowchart of surface mesh generation based on edge patch imprinting

(4) *Poor volume mesh quality from volume decomposition* This results when a volume is decomposed and it is impossible to move interior nodes from one subvolume to another subvolume during mesh smoothing. The existing algorithms[20,22] decompose a volume by connecting loops between two surfaces. This causes problems when a volume has a twisted and complicated internal structure such as an example in Fig.7.

Therefore, we propose an imprinting algorithm based on the cage-based morphing which deforms its interior objects by using its bounding cages. For sweeping, any quad element on the *source* surfaces must match a quad on the *target* surfaces. The imprinting operation between *source* and *target* surfaces can match edges between them and generate sweeping schemes: which *source* surface or parts of the *source* surfaces will be swept onto a specific *target* surface. Besides, the interval matching problem for edges on *source* and *target* surfaces can be avoided through imprinting. First, an imprinting algorithm by the cage-based morphing method is used to imprint edge patches between *source* and *target* surfaces. After edge patch imprinting, *source* and *target* surfaces are partitioned. Then every new *source* surface is matched to a specific *target* surface. Hence, during multi-sweeping, geometric surface matching and edge matching problems between the *source* and *target* surfaces are solved. Finally, One-to-One sweeping with ***S*-*T*** Harmonic Mappings[19] is applied to map all-quad meshes between the *source* and *target* surfaces.

The reminder of this paper is structured as follows: an overview of surface mesh generation during multi-sweeping is presented in Part 2. Afterwards, the cage-based morphing technique is presented to locate edge patches during propagation in Part 3. Part 4 describes the edge patch imprinting algorithm between the *source* and *target* surfaces. After imprinting, new *source* and *target* surfaces are matched and surface meshes are generated in Part 5. Finally, examples are provided and the algorithm time complexity is discussed.

2. Framework for surface mesh generation in multisweeping

In this paper, a new surface mesh generation algorithm based on the edge patch imprinting between the *source* and *target* surfaces during multisweeping is proposed. Figure 1 describes the flowchart of our proposed algorithm. It starts with volumes without any surface mesh while the parametric space $\{i, k\}$ and vertex types on the *linking* surfaces should be provided: this does not imply that all the boundary edges have to be meshed; there is no restriction on the number of points on the boundary edges. Prior to imprinting, sweeping layer number for each *source* and *target* surface should be identified in the sweeping direction. Then edge patch imprinting between the *source* and *target* surfaces is performed including edge patch extraction, edge patch propagation, cage-based morphing and intersection processing. Third, surfaces are split to make one-to-one matching between *source* and *target* surfaces. Finally, quad meshes on the *source* surfaces are mapped onto the *target* surface by morphing[19]. The pseudo code for surface mesh generation during multi-sweeping is described in **Algorithm 1**. Note, there are two imprintings in step 5: one is edge patch imprinting from *target* to *source* surfaces; the other one is imprinting from *source* to *target* surfaces.

3. Cage-based morphing

In multi-sweeping, the *linking* surfaces connect the *source* and *target* surfaces. During the edge patch propagation in the sweeping direction, the *linking* surfaces guide how to place edge patches from k layer to k+1 layer or vice versa:

Algorithm 1. Surface mesh generation for source and target surfaces.

Input: $vol = (\boldsymbol{V}_{geom}, \boldsymbol{E}_{geom}, \boldsymbol{F}_{geom})$ without any mesh, $F_{src} = \{\boldsymbol{F}^{i_1}_{geom},\ i_1 = 1,\cdots,m_1\}$, $F_{tgt} = \{\boldsymbol{F}^{i_2}_{geom},\ i_2 = 1,\cdots,m_2\}$, $F_{link} = \{\boldsymbol{F}^{i_3}_{geom},\ i_3 = 1,\cdots,m_3\}$ *with global consistent* $\{i,j,k\}$ *space*

Output: $\{\boldsymbol{M}_{src}, \boldsymbol{M}_{tgt}\}$

function surfMeshMultisweeping(vol, F_{src}, F_{tgt}, $F_{link}(i,j,k)$)

BUILD layers for source & target surfaces $layer(F_{src}/F_{tgt}) = buildLayers(F_{src}, F_{tgt}, F_{link}(i,j,k)) \in \{0,1,2,\cdots,K-1\}$.

EXTRACT edge patches and their corresponding edge facets on source & target surfaces

$$patch_i^k, i = 0,\cdots,n_{patch}, k = 0,\cdots,K-1$$

$F'_{src} = \emptyset$, $F'_{tgt} = \emptyset$, $Src\_tgt\_Map = \emptyset$

$(F'_{src}, F'_{tgt}) = imprintingEdgePatches(F_{src}, F_{tgt}, F_{link}(i,j,k), patch_i^k)$

$Src\_Tgt\_Map = surfMatching(F'_{src}, F'_{tgt})$

Generate surface meshes on F'_{src}, $\boldsymbol{M}_{src} = (\boldsymbol{V}^{src}_{node}, \boldsymbol{E}^{src}_{edge}, \boldsymbol{F}^{src}_{quad})$

CALL One-to-One sweeping for new target surface meshes F'_{tgt}

$$\boldsymbol{M}_{tgt} = (\boldsymbol{V}^{tgt}_{node}, \boldsymbol{E}^{tgt}_{edge}, \boldsymbol{F}^{tgt}_{quad}) = OneToOneSwept(\boldsymbol{M}_{src}, F'_{src}, F'_{tgt}, Src\_tgt\_Map)$$

for those edge patches directly connected by the *linking* surfaces, they can be placed precisely by using the parametric space $\{i,k\}$ on the *linking* surfaces; for other edge patches to be propagated which are not directly connected by the *linking* surfaces, they can be located by the cage-based morphing.

3.1. Introduction

Cage-based deformation allows an arbitrary closed mesh to act as a deformation cage around another mesh. Figure 3 is a 2D example of deformed edge patches between two layers. The deformed cage can be any shape of mesh but it must be closed.

There are four steps for a cage-based morphing: (1)automatically or manually create a cage to enclose an object to be deformed; (2)bind an object with its cage (cage vertices). In this step, the geometry of an enclosed object is associated with its bounding cage vertices. If any cage vertex is moved, the object will deform itself with its deformed cage as inputs. (3)deform a cage in order to deform an object; (4)interpolate the new object in response to the deformed cage. The most difficult one of the above 4 steps is step (2). The existing approaches for step (2) include *Mean Value Coordinates*[28], *Harmonic Coordinates*[13], *Green Coordinates*[29] and *Radial Basis Method*[30]. Current cage methods express a point η inside a cage P as an affine sum of its cage vertices $V = \{v_i\}_{i\in I_v} \subset R^3$. Let i be the cage vertex index, v_i be 3D location of a cage vertex i and Iv be a set of cage vertices, then we have

$$\eta = F(\eta; P) = \sum_{i\in I_V} \phi_i(\eta) v_i \tag{1}$$

where $\phi_i(\eta)$ is the weight for representing the deformation influence and is often referred as "coordinates". Equation (1) is an implicit equation and used to solve $\phi_i(\eta)$ in the undeformed cage. Then the deformation defined by a deformed cage P' can be computed as follows

$$\eta = F(\eta; P') = \sum_{i\in I_V} \phi_i(\eta) v'_i \tag{2}$$

3.2. Projection of 3D patches onto planar Domain

Generally, loops of points on edges on the real applications are non-planar. In order to simplify the edge patch propagation, 3D edge patches are projected onto the planar domain[20].

The pseudo-area vector, $\mathbf{a}$, of a loop of points $\{\mathbf{x}_i\}_{i=1,\cdots,n}$ is defined as

$$\mathbf{a} = \sum_{i=1}^{n} \mathbf{x}_i \times \mathbf{x}_{i+1} \tag{3}$$

where $\mathbf{x}_{n+1} = \mathbf{x}_1$. The pseudo-normal vector is defined as

$$\mathbf{n} = \frac{\mathbf{a}}{\|\mathbf{a}\|} \tag{4}$$

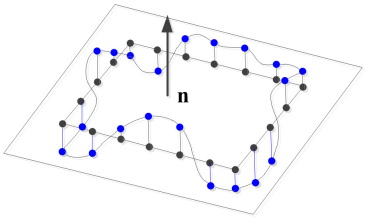

Fig. 2. Projection of 3D Edge Patches onto 2D Domain: 3D edge patches are represented with blue dots and 2D domain is denoted with black dots

The planar domain of a loop can be constructed as follows: first, the pseudo-normal of loop nodes **n** is computed; second,the computational position of $\mathbf{x}_i$ is defined as

$$\bar{\mathbf{x}}_i = \mathbf{x}_i - \langle \mathbf{x}_i, \mathbf{n} \rangle \mathbf{n} \tag{5}$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product and points $\mathbf{x}_i$, for $i = (1, \cdots, n)$, are projected onto the plane defined by the pseudo-normal.

3.3. Placement of edge patches during propagation

From Sect.3.2, 3D edge patches can be projected on a 2D domain. Therefore, 3D cage-based morphing is simplified into a 2D cage-based morphing problem. In Fig.3, the input of the cage-based morphing is the bounding edge nodes $\{\partial S_0^1, \cdots, \partial S_0^{m_1}\}$,$\{\partial S_1^1, \cdots, \partial S_1^{m_2}\}$ and interior edge nodes $\{\partial S_2^1, \cdots, \partial S_2^n\}$ on the layer k, as well as the deformed bounding edge nodes $\{\partial S'^1_0, \cdots, \partial S'^{m_1}_0\}$ and $\{\partial S'^1_1, \cdots, \partial S'^{m_2}_1\}$ on the layer $k+1$, where ∂S_0^i and ∂S_1^i propagate to $\partial S'^i_0$ and $\partial S'^i_1$ respectively through the *linking* surfaces. The goal is to find updated edge node positions $\{S'^1_2, \cdots, S'^n_2\}$ when ∂S_0^i is morphed to $\partial S'^i_0$ and ∂S_1^i is morphed to $\partial S'^i_1$. Our method proceeds by using the following steps to locate interior edge patches $\partial S'^i_2$.

(1) **Propagate bounding edge nodes:** Through the parametric space $\{i, k\}$ of *linking* surfaces, ∂S_0^i and ∂S_1^i can propagate to $\partial S'^i_0$ and $\partial S'^i_1$, respectively, where ∂S_0^i and ∂S_1^i are used as the undeformed cages and $\partial S'^i_0$ and $\partial S'^i_1$ are used as the deformed cages.

(2) **Binding:** The binding process of interior nodes ∂S_2^p with their cage vertices ∂S_0^i and ∂S_1^i can be achieved by using the Laplace Equation as Ref.[13,19]. It is used to solve φ_i with ∂S_0^i, ∂S_1^i and ∂S_2^p as inputs which are already known on the layer k.

$$\partial S_2^p = \sum_{i=1}^{m} \varphi_i(S_2^p) \partial S_j^i, p = \{1, \cdots, n\}, j = \{0, 1\} \tag{6}$$

where φ_i is called "harmonic coordinates" and has the following properties: (1)$\varphi_i(C_j) = \delta(i, j)$(Dirac delta function); (2)C^1 smooth inside a cage; (3)Non-negativity $\varphi_i >= 0$; (4)interior locality; (5)linear reproduction; (6)affine invariance; (7)strict generalization of barycentric coordinates. The binding process is used to solve

(3) **Interpolation:** With the deformed cage $\partial S'^i_0$ and $\partial S'^i_1$ as inputs, the interior nodes S'^p_2 can be interpolated as follows

$$\partial S'^p_2 = \sum_{i=1}^{m} \varphi_i(S_2^p) \partial S'^i_j, p = \{1, \cdots, n\}, j = \{0, 1\} \tag{7}$$

4. Edge imprinting between *S*-*T* surfaces

Prior to edge patch imprinting, layers in the sweeping direction for the *source* and *target* surfaces should be identified. There is an assumption that the global parametric space $\{i, k\}$ on the *linking* surfaces should be given before

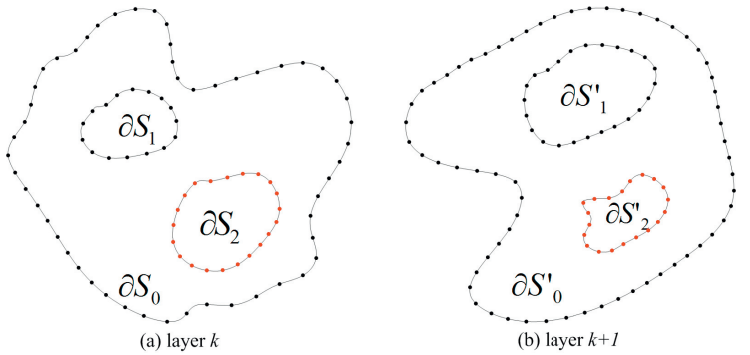

Fig. 3. Placement of interior edge patches during propagation from layer k to k+1

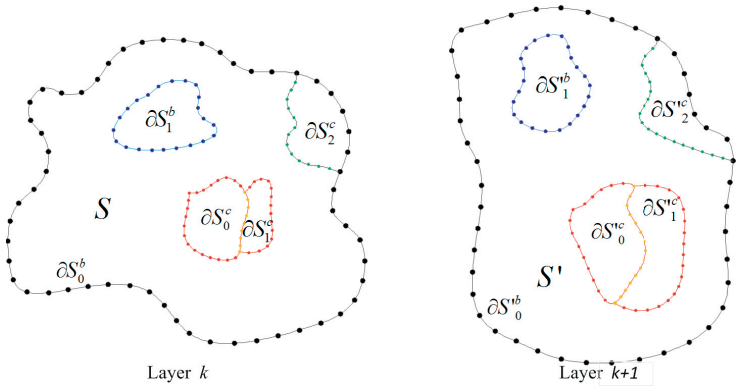

Fig. 4. Edge Patch Propagation between the layer k and layer k+1

imprinting. For sweeping, any quad element on the *source* surfaces must match a quad on the *target* surfaces. The imprinting operation between the *source* and *target* surfaces matches edges between them and generates the sweeping schemes: which *source* surfaces or which areas of the *source* surfaces will be swept onto a specific *target* surfaces.

4.1. Edge patch propagation

There are two types of propagating edge patches: one is direct propagation through the *linking* surfaces; the other one is to place edge patches by using cage-based morphing since those edge patches are not directly connected by the *linking* surfaces. First, the next sweeping level is determined based on the propagation direction. Then propagate the edge patches which are connected by the *linking* surfaces on two adjacent sweeping levels. Finally, apply the cage-based morphing and interpolate interior edge patches. The pseudo code for edge patch propagation is shown in **Algorithm 2**.

It is important to point out that the mapping function φ is solved by interpolating relationships between edge patches which can be directly propagated by themselves and those which can not be propagated by themselves on the current sweeping level. After the mapping function φ is solved, interior edge patches on the next sweeping level can be interpolated by using the bounding edge patches on the next sweeping level and interior edge patches on the previous sweeping level as inputs. For example, in Fig.4, ∂S_0^b and ∂S_1^b are two bounding edge patches which can propagate by themselves while ∂S_0^c,∂S_1^c and ∂S_2^c are three interior edge patches which can not propagate by themselves. By using $F_{link}(i, j, k)$, ∂S_0^b can propagate to $\partial S'^b_0$ and so are ∂S_1^b and $\partial S'^b_1$. The interior edge patches $\partial S'^c_0$, $\partial S'^c_1$ and $\partial S'^c_2$ can be interpolated by using Eqn.(2) with $\partial S'^b_0$, $\partial S'^b_1$, ∂S_0^c, ∂S_1^c and ∂S_2^c as inputs.

Algorithm 2. Propagate edge patches from k layer to $k-1$ *or* $k+1$ layer.

Input: $\partial S_p^b, p = 0, 1, \cdots, m$, $\partial S_p^c, p = 0, 1, \cdots, n$, $F_{link} = \{\boldsymbol{F}_{geom}^{i_3},\ i_3 = 1, \cdots, m_3\}$ *with global consistent* $\{i, j, k\}$ *space*
$ImprintDir = \{TgtToSrc, SrcToTgt\}$

Output: $\partial S'^b_p, p = 1, \cdots, m.$ $\partial S'^c_p, p = 0, 1, \cdots, n$

function $propagatePatches(k_{now}, \partial S_p^b, \partial S_p^c, F_{link}(i, j, k), patch_i^k, ImprintDir)$
  $k_{next} = k_{now} - 1$
  IF $ImprintDir = TgtToSrc$
    $k_{next} = k_{now} + 1$
  ENDIF
  FOR EACH point q on the $\partial S_p^b,\ p = 1, \cdots, m$
    $(i_{q'}, j_{q'}) = (i_q, j_q)$
    $k_{q'} = k_{next}$
    Interpolate $\partial S'^b_p$ coordinates by using parametric space $F_{link}(i_{q'}, j_{q'}, k_{q'})$
  ENDFOR
  Use point P of ∂S_p^b on the layer k_{now} as an undeformed cage, point v on ∂S_p^c as interior points. Solve mapping function φ:
    $v = F(P, v) = \sum_{j \in \partial S_p^b} \varphi_j(v) P$
  FOR EACH point v on $\partial S_p^c, p = 1, \cdots, m$
  Use point P' on $\partial S'^b_p$ as the deformed cage, locate point v' on $\partial S'^c_p$ as
    $v' = F(P', v) = \sum_{j \in \partial S_p^b} \varphi_j(v) P'$
  ENDFOR
  RETURN $\{\partial S'^b_p, \partial S'^c_p\}$

4.2. Imprinting of edge patches

During imprinting, edge patches on the *target* surfaces are imprinted onto the appropriate *source* surfaces or vice versa and topological operations (partition) are needed to process surfaces after the topology of one surface is modified as a result of imprinting. **Algorithm 3** describes the edge patch imprinting between the *source* and *target* surfaces.

There are two ways of imprinting edge patches: one is imprinting from the *target* surfaces to the *source* surfaces, which results in that multiple new *source* surfaces match one *target* surface; the other one is imprinting from the *source* surfaces to the *target* surfaces, which brings about the one-to-one matching between the new *source* and *target* surfaces. Prior to imprinting, bounding edge patches(directly connected to the next sweeping level by the *linking* surfaces) and interior edge patches should be detected in order to prepare for edge patch propagation. Interior edge patches are those which are not connected directly by the *linking* surfaces and can not propagate by themselves. Then edge patches can propagate as Sec.4.1. If there are imprinted edge patches on the *source* or *target* surfaces, surfaces need to be partitioned. Finally, intersections of edge patches are processed in that intersections mandate that a geometric vertex be placed there, so the point gets resolved by a mesh vertex.

After imprinting, the surface matching and interval assignment problem for edges of *source* and *target* surfaces can be solved. Note: volumes are not partitioned at all; only *source* and *target* surfaces are partitioned with the imprinted edge patches on them. Even though the *source* and *target* surfaces may be partitioned after imprinting, quad mesh quality on those surfaces is guaranteed to be good since cage-based morphing places the propagated edge patches appropriately constrained by using the *linking* surfaces and it has an important property: local deformation.

4.3. Intersection processing of edge patches

After edge patches propagate from k layer to k+1 layer, an intersection problem needs to be solved: which parts of edge patches need to be propagated; which parts of edge patches need to stop propagating; which parts of edge patches need to be transformed from bounding patches to interior patches. Meanwhile, intersection mandates that a geometric vertex should be placed there, so that new edges are matched between *source* and *target* surfaces.

An overview of processing intersection of edge patches is given in **Algorithm 4**. In the algorithm, whether edge patches continue to propagate or not depends on whether they are imprinted on surfaces or not. An example for Boolean operation between edge patches is shown in Fig.5 where two edge patches P_a and P_b are given: *Union*, *Subtraction* and *Intersection*.

Algorithm 3. Imprint edge patches from k layer to $k-1$ *or* $k+1$ layer.
Input: $S_i^{k_1}, S_i^{k_2}, F_{link} = \{\boldsymbol{F}_{geom}^{i_3}, i_3 = 1,\cdots,m_3\}$ *with global consistent* $\{i,j,k\}$ *space*
$ImprintDir = \{TgtToSrc, SrcToTgt\}$
Output: F'_{src}, F'_{tgt}

1: function $imprintPatches(S_i^{k_1}, S_i^{k_2}, F_{link}(i,j,k), ImprintDir)$
2: $S'^0_i = S^0_i$
3: FOR $k = 1,2,\cdots,K-1$
4 DETECT bounding and interior patches $\{S_i^b, S_i^c\}$ from S'^{k-1}_i.
5: CALL $S'^k_i = propagatePatches(S_i^b, S_i^c, F_{link}(i,j,k), TgtToSrc)$
6: IF $F^k \epsilon F_{src}$ *and* $S'^k_i \cap F^k \neq \emptyset$ $(i = 1,\cdots,n'_{patch})$
7: PARTITION F^k with propagated patches S'^k_i.
8: PUSH new partitioned surfaces f_{src} into F'_{src}.
9: ELSE IF $F^k \epsilon F_{src}$
10: PUSH surfaces F^k into F'_{src}.
11: ENDIF
12: PROCESS intersections of edge patches $S'^k_i = processIntersection(S'^k_i, S_i^k, TgtToSrc)$
13: ENDFOR
14: $S''^{K-1}_i = S'^{K-1}_i$
15: FOR $k = K-2,\cdots,1$
16: DETECT bounding and interior patches $\{S_i^b, S_i^c\}$ from S''^{k+1}_i.
17: CALL $S''^k_i = propagatePatches(S_i^b, S_i^c, F_{link}(i,j,k), SrcToTgt)$
18: IF $F^k \epsilon F_{tgt}$ *and* $S''^k_i \cap F^k \neq \emptyset$ $(i = 1,\cdots,n'_{patch})$
19: PARTITION F^k with propagated patches S''^k_i.
20: PUSH new partitioned surfaces f_{tgt} into F'_{tgt}.
21: ELSE IF $F^k \epsilon F_{tgt}$
22: PUSH surfaces F^k into F'_{tgt}.
23: ENDIF
24: PROCESS intersections of edge patches $S''^k_i = processIntersection(S''^k_i, S_i^k, SrcToTgt)$
25: ENDFOR
26: RETURN $\{F'_{src}, F'_{tgt}\}$

4.4. Surface partition

The surface partition happens when there are imprinted edge patches on them. The surfaces may be partitioned with a list of point coordinates on the edge patches. This results in a curve represented with a list of facet line segments. In order to avoid too many line segments, first, polylines are constructed by using a set of point coordinates. Then polylines are projected onto surfaces (*sources* or *targets*) to define curves for splitting surfaces. If only one position is specified, a zero-length curve with a single vertex is created. In this paper, we use virtual partition functions in CGM[14] to partition surfaces with any imprinted edge patches.

5. Surface mesh generation for *S-T* surfaces

After imprinting, there are new *source* and *target* surfaces. In essence, every new *source* surface should match a specific new *target* surface. The new surface matching should be done in order to find one-to-one surface matching relationship between the *source* and *target* surfaces. Once the surface matching is done, an existing all-quad mesh algorithm could be employed to generate quad meshes on the *source* surfaces. Finally, One-to-One sweeping based on Harmonic Mapping[19] could be used to generate quadrilateral meshes on the *target* surfaces.

5.1. Surface matching between S-T surfaces

After surface partition, the multi-sweeping problem for the *source* and *target* surface meshes is reduced to multiple One-to-One sweeping for surface meshes. The next is to match each new *source* surface with a specific *target* surface.

An overview of matching *source* and *target* surfaces is given in **Algorithm 5** where F'^{tgt}_i is the stack of new *target* surfaces and new *source* surfaces are in the list of F'^{src}_i. Whether one pair of *source* and *target* surfaces is matched or not is based on the fact that whether edge patches on one surface can propagate to those on the other surface. If so, two surfaces are matched. Otherwise, they are not. The edge matching between the *source* and *target* surfaces could be done as **Algorithm 5** so that interval assignment for edges on *source* and *target* surfaces can be solved.

Algorithm 4. Intersection processing for edge patches.
Input: $P_i^a, i = 1, \cdots, m, P_j^b, j = 1, \cdots, n,\ ImprintDir = \{TgtToSrc, SrcToTgt\}$
Output: $P'^a_i, i = 1, \cdots, m'$

```
function processIntersection(P_i^a, P_i^b, ImprintDir)
  P'^a_i = ∅
  FOR each patch P_i^a (i = 1, ···, m)
    P_tmp = P_i^a
    FOR each patch P_j^b (j = 1, ···, n)
      IF (P_tmp ∩ P_j^b ≠ ∅)
        IF (ImprintDir == TgtToSrc)
          IF (P_i^b ∈ Face_tgt)
            PUSH back {P_tmp − P_j^b},{P_i^b − P_tmp},{P_tmp ∩ P_j^b} and {P_tmp ∪ P_j^b} into P_tmp
          ELSE
            PUSH back {P_tmp − P_j^b} into P_tmp
          ENDIF
        ELSE
          IF (P_i^b ∈ Face_tgt)
            PUSH back {P_tmp − P_j^b} into P_tmp
          ELSE
            PUSH back {P_tmp − P_j^b},{P_i^b − P_tmp},{P_tmp ∩ P_j^b} and {P_tmp ∪ P_j^b} into P_tmp
          ENDIF
        ENDIF
      ELSE
        IF (ImprintDir == TgtToSrc)
          IF (P_i^b ∈ Face_tgt && P_i^b ∩ P^a = ∅)
            PUSH back P_i^b into P_tmp; break;
          ENDIF
        ELSE
          IF (P_i^b ∈ Face_src && P_i^b ∩ P^a = ∅)
            PUSH back P_i^b into P_tmp;  break;
          ENDIF
        ENDIF
      ENDIF
    ENDFOR
    PUSH back P_tmp into P'^a_i.
  ENDFOR
  RETURN P'^a_i
```

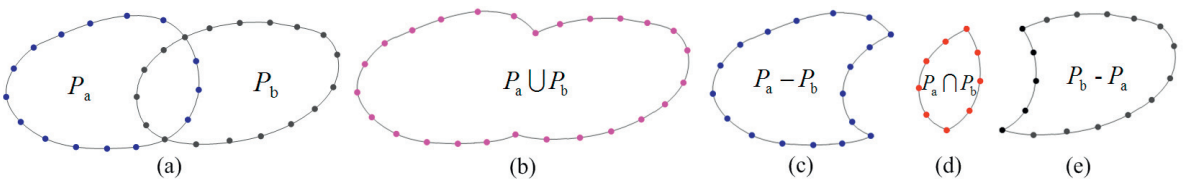

Fig. 5. Intersection processing for edge patches: (a)patch P_a and patch P_b; (b)Union of patch P_a and patch P_b; (c)Subtraction of patch P_a and P_b; (d)Intersection of patch P_a and P_b; (e)Subtraction of patch P_b and P_a

5.2. Surface mesh generation

At this point, every *source* surface (after partitioning) has a corresponding *target* surface (after partitioning). Therefore, morphing based on harmonic mapping[19] is applied to generate all-quad meshes on the *target* surfaces for multisweeping.

6. Examples and discussion

This section presents three examples of *source* and *target* surface meshes which have been generated by using the edge patch imprinting algorithm and morphing algorithm[19]. Users can manually match edge patches. Otherwise, the imprinting algorithm will use the cage-based morphing to propagate edge patches, partition the *source* and *target* surfaces and match edges between the *source* and *target* surfaces. Note: volumes are not decomposed and only

Algorithm 5. Surface matching between *S-T* surfaces.
Input: $F'^{src}_i, i = 1, \cdots, m$, $F'^{tgt}_i, i = 1, \cdots, n$
Output: Src_Tgt_Map

```
function surfMatching(F'^src_i, F'^tgt_i)
    Src_Tgt_Map = ∅
    WHILE F'^tgt_i, the stack of target surfaces, is not empty
        LET T be the first target surface in F'^tgt_i and remove it from F'^tgt_i.
        LET ∂T_k be edge patching bounding the target surface T.
        FOR EACH source surface S in the list of source surfaces, F'^src_i.
            LET ∂S_j be edge patches for the source surface S.
            IF ∂S_j is able to propagate to ∂T_k.
                PUSH {S, T} into Src_Tgt_Map
                REMOVE S from F'^src_i.
                BREAK FOR EACH
            ELSE
                CONTINUE
            ENDIF
        ENDFOR
    ENDWHILE
```

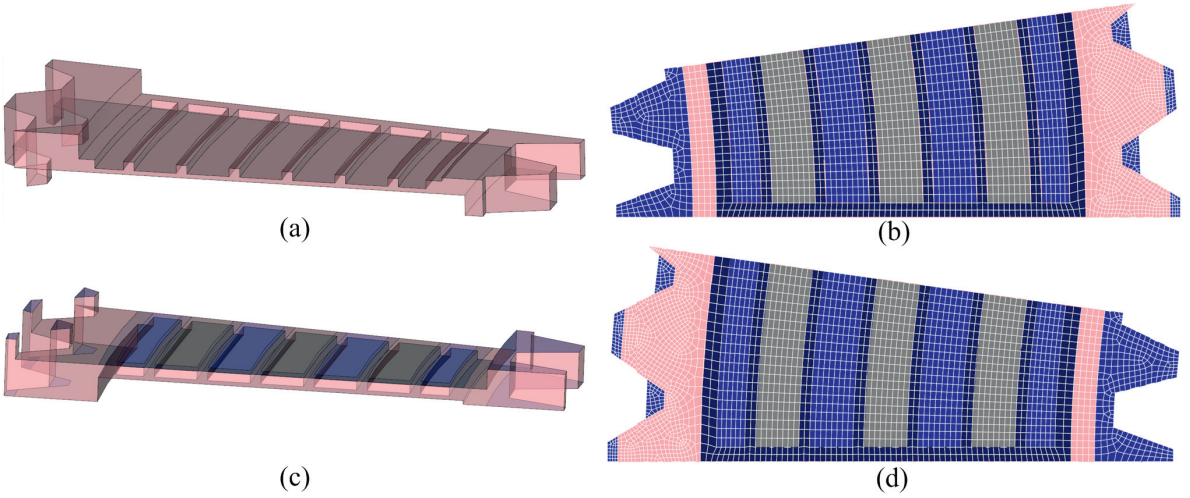

Fig. 6. A real part from caterpillar: (a)a geometric model; (b)the *source* surface meshes; ()partitioned *source* and *target* surfaces by imprinting edge patches; (d)the *target* surface meshes by morphing[19]

surfaces are partitioned in order to match the *source* and *target* surfaces. This is due to the inherent characteristics of the sweeping algorithm: every quad element on the *source* surfaces has its corresponding quad element on the *target* surfaces. The great disadvantage for decomposing volumes is that interior nodes can not be moved from one subvolume to another subvolume if poor volume mesh quality is produced.

6.1. Examples

The first example presents the surface mesh generation for a real part from caterpillar with multiple *source* and *target* surfaces by sweeping. Even though the volume(shown in Fig.6(a)) is pretty simple, it is very difficult to match *source* and *target* surfaces: which *source* or parts of *source* surfaces will be swept onto a specific *target* surface. Meanwhile, the edge matching problem between the *source* and *target* surfaces during multi-sweeping is difficult to solve as well. By using our proposed imprinting algorithm, the *source* and *target* surfaces are partitioned as Fig.6(c): every new *source* surface has its corresponding new *target* surface. The *source* surface meshes are shown in Fig.6(b). By mapping *source* surface meshes onto the *target* surfaces, the resulting *target* surface meshes are represented in Fig.6(d).

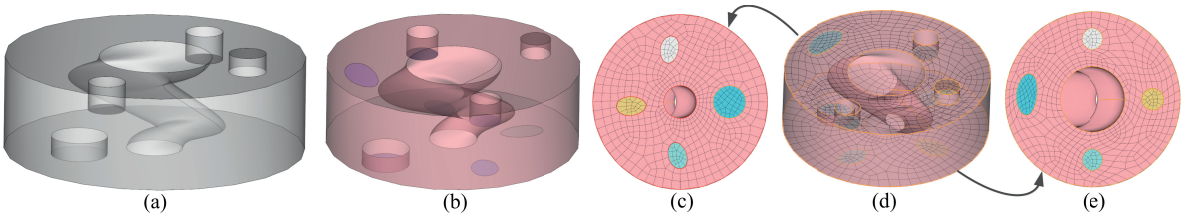

Fig. 7. An example of generating surface meshes and matching the *source* and *target* surfaces by imprinting edge patches: (a)the geometric model; (b)the partitioned *source* and *target* surfaces; (c)the *source* surface meshes; (d) 3D-view surface meshes; (e)the *target* surface meshes by morphing[19]

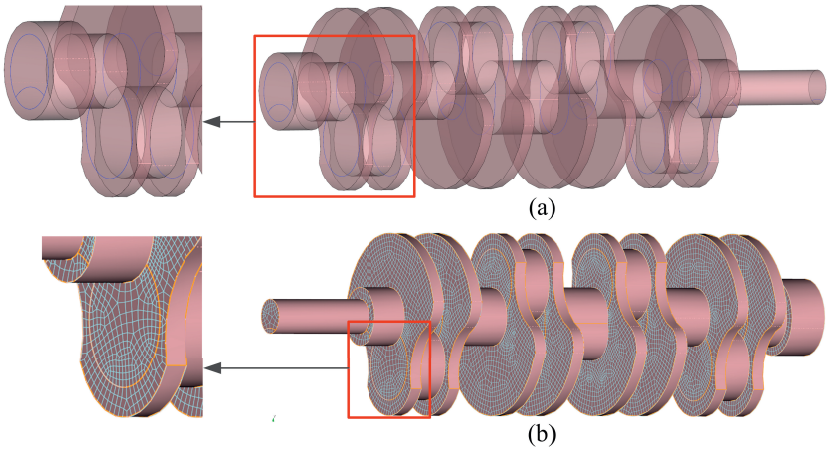

Fig. 8. Surface mesh generation for crankshaft by imprinting: (a)the geometric model with imprinted edge patches(denoted by blue curves); (b)the *target* surface meshes by morphing[19]

The second example shows the edge patch imprinting and surface mesh generation for a solid with a varying hole and complicate internal structure(shown in Fig.7(a)). If an affine transformation method is used to propagate edge patches between the *source* and *target* surfaces during imprinting, edge patches may not be appropriately located(the propagated circle may intersect with the circle-like hole on the *target* surface while there is no intersection between them on the bottom *source* surfaces). If the volume is decomposed by connecting edges between *source* and *target* surfaces, the cutting path may touch the through hole between *source* and *target* surfaces which is not what we want. When our imprinting algorithm based on the cage-based morphing technique is used, edge patches are propagated correctly and constrained by their bounding *linking* surfaces, see Fig.7(b) for details. Figure 7(c) presents the *source* surface meshes and the resulting *target* surface meshes are generated by our morphing methods[19](shown in Fig.7(d) and Fig.7(e)).

An crankshaft example from the automobile engine is presented in Fig.8. This example contains many cylinders of which some have the same size. The resulting new model with partitioned *source* and *target* surfaces is presented in Fig.8(a) where the partitioned edge patches are represented with blue curves: every new *source* surface has its corresponding *target* surface. The *source* surfaces are meshed with quads and the resulting *target* surface meshes are generated by mapping *source* surface meshes onto the *target* surfaces(Fig.8(b)).

In order to assess the mesh quality, we plot the mesh quality histogram for Fig.6, Fig.7 and Fig.8 in Fig.9. The results show that the proposed algorithm produces quadrilateral surface meshes with good mesh quality.

6.2. Discussion

Prior to imprinting, layers can be built in the linear time $o(c1)$ where $c1$ is the number of surfaces. During imprinting, there are two imprintings between the *source* and *target* surfaces. In each imprinting, detection of bounding

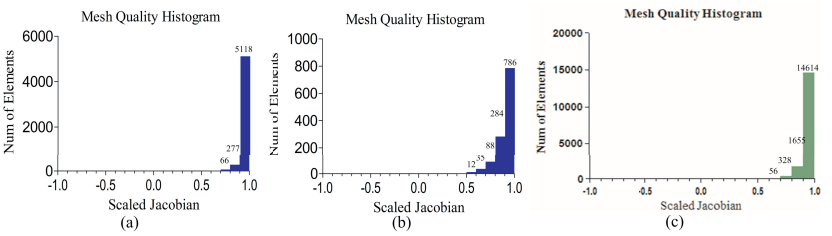

Fig. 9. Mesh quality histogram. (a)mesh quality histogram for Fig.6; (b)mesh quality histogram for Fig.7; (c)mesh quality histogram for Fig.8

and interior edge patches can be done in linear time $o(c2)$ by using the *linking* surfaces where $c2$ is the number of patches. The layer propagation can take longer time. There are three parts, namely, edge patch propagation on the *linking* surfaces, the solution to the mapping function φ and interpolation of interior points $o(n) + o(nlogn) + o(n)$ where n is the number of points on the edge patches. The surface partition can be done in linear time as well $o(n)$ where n is the number of points on the edge patches. The last part is the intersection processing of edge patches which takes $o(n_1) + o(n_2)$ where n_1 and n_2 are the number of points on two edge patches. The surface matching can be done in $o(n)$ as well by using tags to record where an edge patch is from. Therefore, the total running time for imprinting edge patches is $2 * K * o(nlogn)$. Even though the proposed imprinting algorithm is more expensive than the affine transformation method, it can generate good surface partition and surface meshes with good mesh quality for multisweeping.

7. Conclusion

It is well known that two significant problems for multi-sweeping are: surface matching and edge matching problem between the *source* and *target* surfaces. Current existing methods of multi-sweeping use the volume decomposition for all-hex meshes. There are significant disadvantages for decomposing volumes during multi-sweeping: interior nodes inside a volume can not be moved from one subvolume to another subvolume if poor volume mesh quality is produced; it is difficult to compute the cutting path when there is complicate internal structure inside a volume and most decomposition methods use the affine transformation method to cut volumes directly. In order to avoid volume decomposition during imprinting, an imprinting algorithm based on the cage-based morphing is proposed in this paper which is key to robust multi-sweeping since good quality for interior edge patch placement on the surfaces has already been demonstrated.

First, layers should be identified prior to imprinting. Second, *target* surfaces are imprinted onto the *source* surfaces layer by layer. This includes: (1)propagate edge patches of *target* surfaces from k-1 layer to k layer by the cage-based morphing; (2)partition the *source* surfaces on the layer k if any imprinted edge patches on them; (3)process intersections of propagated edge patches and new edge patches on the layer k. At this stage, a *target* surface matches one or multiple *source* surfaces. Third, *source* surfaces are imprinted onto the *target* surfaces again: (1)propagae edge patches of *source* surfaces from k+1 layer to k layer by using the cage-based morphing; (2)partition the *target* surfaces on the layer k; (3)process the intersection of propagated edge patches and new edge patches on the layer k. Fourth, new *source* and *target* surfaces are matched with eath other: every new *source* surface has a specific corresponding *target* surface. Finally, generate quadrilateral meshes on the *source* surfaces and map them onto the *target* surface by morphing: interior mesh node placement on the *target* surfaces can proceed as single ***S*-*T*** sweeping[19] since matching between the *source* and *target* surfaces is made.

The proposed algorithm for multi-sweeping has been been applied to several cases with great success. High quality surface meshes on the *source* and *target* surfaces are produced by our proposed algorithm. What is more, it has $o(nlogn)$ time complexity where n is the number of facet points on the edge patches.

Acknowledgement

This work was funded under the auspices of the Nuclear Energy Advanced Modeling and Simulation (NEAMS) program of the Office of Nuclear Energy, and the Scientific Discovery through Advanced Computing (SciDAC) program funded by the Office of Science, Advanced Scientific Computing Research, both for the U.S. Department of Energy, under Contract DE-AC02-06CH11357. We also thank all the Fathom members (from both Univ. of Wisconsin-Madison and Argonne National Lab) for their efforts on the CGM[14], MOAB[15], Lasso[16] and MeshKit[17] libraries, which were used heavily to support this work.

References

[1] R. Biswas, R. C. Strawn: Tetrahedral and hexahedral mesh adaptation for CFD problems. Applied Numerical Mathematics, vol.26, no:1-2: 135-151 (1988).
[2] J. A. Samareh: Geometry and grid/mesh generation issues for CFD and CSM shape optimization. Optimization and Engineering 6.1: 21-32 (2005).
[3] T. Taniguchi, T. Goda, H. Kasper et al.: Hexahedral Mesh Generation of Complex Composite Domain. 5th International Conference on Grid Generation in Computational Field Simmulations: 699-707 (1996)
[4] R. Schneiders: A Grid-based Algorithm for the Generation of Hexahedral Element Meshes. Engineering with Computers, vol.12, no.3-4: 168-177 (1996)
[5] M. A. Price, C. G. Armstrong: Hexahedral Mesh Generation by Medial Surface Subdivision: Part I. IJNME, vol.38, no.19: 3335-3359 (1995)
[6] M. A. Price, C. G. Armstrong: Hexahedral Mesh Generation by Medial Surface Subdivision: Part II. IJNME, vol.40: 111-136 (1997)
[7] T. D. Blacker and R. J. Myers: Seams and Wedgers in Plastering: A 3D Hexahedral Mesh Generation Algorithm. Engineering With Computers, vol.2: 83-93 (1993)
[8] T. J. Tautges, T. D. Blacker and S. A. Mitchell: The Whisker-Weaving Algorithm: A Connectivity Based Method for Constructing All-Hexahedral Finite Element Meshes. IJNME, vol.39: 3327-3349 (1996)
[9] P. M. Knupp: Next-Generation Sweep Tool: A Method for Generating All-Hex Meshes on Two-And-One-Half Dimensional Geometries. 7th IMR: 505-513 (1998)
[10] M. L. Staten, S. A. Canann and S. J. Owen: BMSweep: Locating Interior Nodes During Sweeping. 7th IMR: 7-18 (1998)
[11] X. Roca, J. Sarrate and A. Huerta: Surface Mesh Projection for Hexahedral Mesh Generation by Sweeping. 13th IMR: 169-180 (2004)
[12] X. Roca, J. Sarrate and A. Huerta: A new least-squares approximation of affine mappings for sweep algorithms. 14th IMR: 433-448 (2005)
[13] P. Joshi, M. Meyer, T. DeRose et al.: Harmonic coordinates for character articulation. ACM Transactions on Graphics (TOG). Vol.26, No.3, ACM (2007)
[14] T. J. Tautges: CGM: A geometry interface for mesh generation, analysis and other applications. Engineering with Computers, 17(3):299-314 (2001)
[15] T. J. Tautges: MOAB: a mesh-oriented database. Sandia National Laboratories, (2004)
[16] Lasso: https://trac.mcs.anl.gov/projects/ITAPS/wiki/Lasso
[17] T. J. Tautges and R. Jain: Creating geometry and mesh models for nuclear reactor core geometries using a lattice hierarchy-based approach. Engineering with Computers, 28(4):319-329 (2012)
[18] N. Kowalski, et al.: Fun sheet matching: towards automatic block decomposition for hexahedral meshes. Engineering with Computers 28.3: 241-253 (2012)
[19] S. Y. Cai, and T. J. Tautges: Robust One-to-One Sweeping with Harmonic ST Mappings and Cages. Proceedings of the 22nd International Meshing Roundtable: 1-18 (2014)
[20] E. Ruiz-Girones, X. Roca, and J. Sarrate: A new procedure to compute imprints in multi-sweeping algorithms. Proceedings of the 18th International Meshing Roundtable: 281-299 (2009)
[21] X. Roca, E. Ruiz-Girones, and J. Sarrate: Receding front method: a new approach applied to generate hexahedral meshes of outer domains. Proceedings of the 19th International Meshing Roundtable: 209-225 (2010)
[22] D. White. S. Saigal, and S. J. Owen: CCSweep: automatic decomposition of multi-sweep volumes. Engineering with computers 20.3: 222-236 (2004)
[23] M. W. Lai, S. E. Benzley, et al.: A multiple source and target sweeping method for generating all-hexahedral finite element meshes. Proceedings of the 5th International Meshing Roundtable: 217-225 (1996)
[24] K. Miyoshi, and T. D. Blacker: Hexahedral Mesh Generation Using Multi-Axis Cooper Algorithm. Proceedings of the 9th International Meshing Roundtable:89-97 (2000)
[25] T. D. Blacker: The Cooper Tool. Proceedings of the 5th International Meshing Roundtable:13-29 (1996)
[26] H. Wendland: Scattered Data Approximation. Cambridge University Press, Cambridge, UK (2005)
[27] J. o'Rourke: Computational Geometry in C. Cambridge University Press, Cambridge (1998)
[28] M. S. Floater: Mean Value Coordinates. Computer Aided Geometric Design, 20(1): 19-27 (2003)
[29] Y. Lipman, D. Levin and D. Cohen-Or: Green Coordinates. ACM Transactions on Graphics (TOG), 20(3): 19-27 (2008)
[30] D. Sieger, S. Menzel and M. Botsch: High Quality Mesh Morphing Using Triharmonic Radial Basis Functions. 21st IMR: 1-15 (2013)