Algorithmic Improvements to Sweeping and Multi-Sweeping Volume Mesh Generation

by

Shengyong Cai

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Engineering Mechanics and Astronautics)

at the

UNIVERSITY OF WISCONSIN–MADISON

2015

Date of final oral examination: 04/10/2015

The dissertation is approved by the following members of the Final Oral Committee:

Vadim Shapiro, Professor, Mechanical Engineering and Computer Sciences
Krishnan Suresh, Associate Professor, Mechanical Engineering
James P. Blanchard, Professor, Engineering Physics
Paul P.H. Wilson, Professor, Engineering Physics
Timothy J. Tautges, Adjunct Professor, Engineering Physics and CD-Adapco

For my beloved family - San Zeng, Ruth Cai, Hehua Liu and Jiagun Cai. Above all, for Strom.

ACKNOWLEDGMENTS

I am grateful to a number of extraordinary professors and colleagues at the Universities of Wisconsin and Argonne National Laboratory, who have inspired and encouraged me. My committee members, Dr. Vadim Shapiro, Dr. Krishnan Suresh, Dr. James P. Blanchard, Dr. Paul P.H. Wilson and Dr. Timothy J. Tautges, were especially supportive in this endeavor.

Among them, I am particular grateful to my research advisor, Professor Tautges, for his indispensible technical and practical advising on the sweeping algorithm design and implementation. He has been a dedicated ally, always generous with analytical perspectives, far-ranging enthusiasm, and deep concern for the professional and personal well being of his students. Meanwhile, I am also grateful to him, who has spent a lot efforts on polishing and elaborating my thesis writing and giving me a lot of insightful comments. In addition, I would like to thank my academic advisor, Professor James Blanchard, for his advice and efforts on my academic development.

So too, my colleagues Jiangtao Hu, Jason Kraftcheck, Chaman Singh Verma, Iulian Grindeanu, Rajeev Jain, James Porter, Steve Jackson, Hongjun Kim, Patrick Shriwise, Andrew Davis, Vijay S. Mahadevan, Brandon Smith, Katy Huff, Rachel Slaybaugh and Milad Fatenejad have been essential in the success of my PhD research as well as the growth of my collabborative skills. I would also thank my parents for their unconditional love and support.

This work was carried out in University of Wisconsin-Madison under contract with Scalable Interfaces for Geometry and Mesh-based Application team (*SIGMA* team) at Argonne National Laboratory. It 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.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Due to numerical properties, hexahedral meshes are preferred and widely used for numerical simulations in several engineering domains. One of the most robust and widely used algorithms for all-hexahedral meshes is the sweeping algorithm which can generate the hexahedral meshes by sweeping the surface meshes on the *source* surfaces to the *target* surfaces. In addition, sweeping is also useful for generating other swept volume meshes besides hexahedral meshes such as tri-prism for alignment. The sweeping algorithm consists of four main steps: surface mesh generation on the *source* surfaces, projection of *source* surface meshes onto the *target* surfaces, structured mesh generation on the *linking* surfaces and interior node placement inside volumes. Current state of the art suffers from either low robustness or poor mesh quality: surface mesh mapping between concave or multiply-connected domains with inverted elements, poor interior node placement inside volumes with complicated internal structures, poor imprinting algorithm for multi-sweeping problems and failed corner assignment for vertices with ambiguous angles on the *linking* surfaces. Therefore, in this work, an improved and robust sweeping tool has been developed, which consists of several things: surface mesh mapping between the s/t surfaces has been developed based on *Harmonic Mapping* which works for convex, concave and multiply-connected surfaces; interior node placement method inside volumes has been developed based on the *Cage-based Morphing* which can deal with local deformation from the *linking* surfaces and relocate interior nodes accordingly; an improved imprinting algorithm for multi-sweeping has been developed where edge patches are imprinted between the *source* and *target* surfaces; an optimal corner assignment method on the *linking* surfaces has been developed based on templates and ***LP***. Finally, the sweepability assessment problems are discussed based on the topological constraints, geometric constraints and some constraints from users' specified matchings. Overall, our improved sweeping algorithm can generate a

swept volume mesh with good mesh quality and $O(nlogn)$ time complexity.

1 INTRODUCTION

Discretization such as *Finite Element Method* (***FEM***)[32], *Finite Volume Method* (***FVM***) or the discontinuous Galerkin method are the most used techniques to simulate physical domains in engineering and applied computational sciences, for example, heat transfer, structural mechanics and *Computational Fluid Dynamics* (***CFD***)[9, 75, 103]. Those methods rely on a spatial discretization to approximate the physical domain for solving the *Partial Differential Equation* (***PDE***)[38, 104]. The discretization is called a mesh which consists of a set of nodes and elements (faces in 2D and cells in 3D). The most common element types of 3D meshes are the tetrahedron and hexahedron. For some types of *Finite Element Analysis* (***FEA***), it is widely believed that hexahedral meshes are more accurate and robust than tetrahedral meshes, especially in the nonlinear regime[8, 113]. Alignment of non all-hexahedral meshes by sweeping is also sometimes desirable.

There has been a great deal of research on fully automatic all-hexahedral mesh generation, but no algorithm has been found with some key requirements such as high robustness, high mesh quality and low element amount. Hence, current existing hexahedral mesh generators count on simpler tools along with geometry decomposition for an adequate mesh generation process. One of the most widely used hexahedral mesh generation algorithms is the sweeping algorithm which can mesh the *Two-and-One-Half* (***2.5D***) dimensional geometries.

1.1 Sweeping

Extrusion of 2D meshes into a general third dimension is called "sweeping", which is a method for generating the swept volume meshes by sweeping a set of mesh faces through volumes subject to both topological and geometrical restrictions. Typically, sweeping

volumes are defined by *source* surfaces, *target* surfaces and a series of *linking* surfaces (see Fig.1.1). Based on characteristics of sweeping volumes, it generates the volume mesh by sweeping mesh faces on the *source* surfaces along volumes until all the *target* surfaces are reached and vice versa. Hence, there is an inherent characteristics that every mesh face on the *source* surfaces should be matched to a specific mesh face on the *target* surface. The traditional procedure to generate the swept volume meshes by sweeping consists of the following four steps:

(1) Generate surface meshes over the *source* surfaces.

(2) Map the *source* surface meshes onto the *target* surfaces.

(3) Generate the structured quadrilateral meshes on the *linking* sides.

(4) Locate interior nodes over the volume interior and generate volume elements inside volumes.

While the basic sweeping algorithm requires only one single *source* and *target* surface, some implementations of the sweeping algorithm can also handle multiple *source* and *target* surfaces. The multi-sweeping algorithm works by imprinting s/t surfaces to guarantee that every *source* surface or patch can be swept to exactly one *target* surface or patch, and vice versa. Therefore, there are three variants of sweeping from the point of number of *source*/*target* surfaces, that is, *One-to-One* (***1-1***), *Many-to-One* (***M-1***) and *Many-to-Many* (***M-N***) sweeping. A ***M-1*** sweeping is an extension of ***1-1*** sweeping by adding M *source* surfaces. It generates the swept volume meshes by sweeping M *source* surface meshes through the volume onto one *target* surface. A ***M-N*** sweeping is an extension of ***M-1*** sweeping by adding N *target* surfaces. It generates the swept volume meshes by first imprinting *source* and *target* surfaces to ensure the mesh continuity through the volume and then sweeping M *source* surface meshes through the volume onto N *target* surfaces.

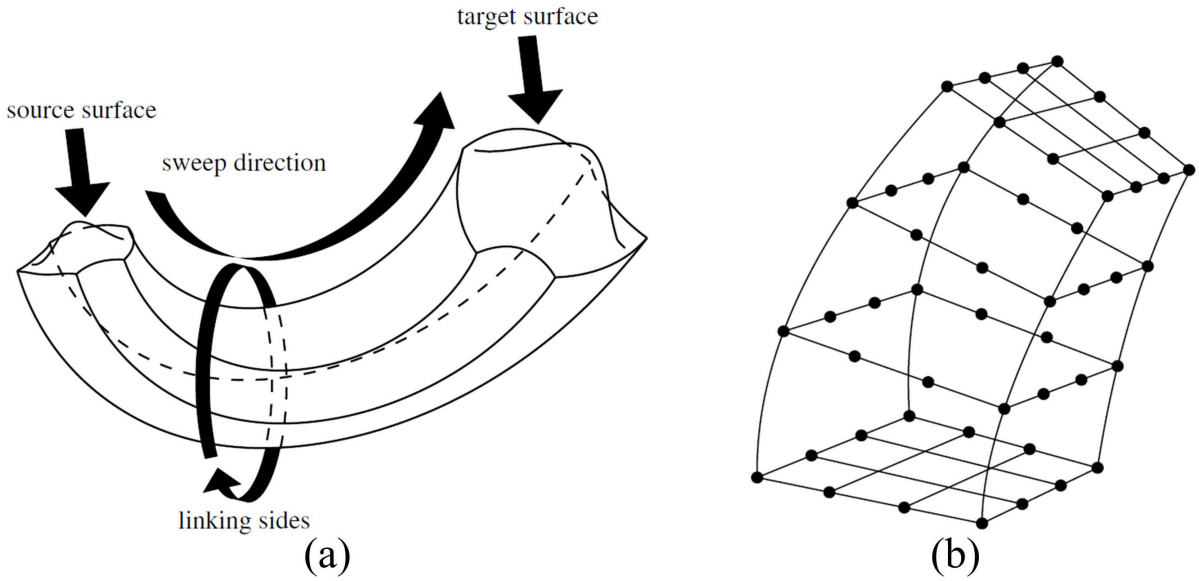

Figure 1.1: The definition of mesh sweeping[69]. (a)Sweeping example of a two and a half dimensional volume; (b)a mesh sweeping example

The multi-sweeping including ***M-1*** and ***M-N*** just relieves users of more work such as volume decomposition and removes constraints due to the geometry-based disconnected *source* or *target* surfaces. Examples for those types of sweeping are shown in Fig.1.2 where Fig.1.2(a) is a ***1-1*** sweeping example, Fig.1.2(b) is a ***M-1*** sweeping example and Fig.1.2(c) is a ***M-N*** sweeping example.

1.2 Mapping/Submapping

Due to the inherent characteristics of sweeping algorithms, structured quadrilateral meshes have to be generated on the *linking* surfaces which connects the *source* and *target* surfaces. This is generally achieved by a *Mapping/Submapping* algorithm. The structured mesh is defined as an all-quadrilateral mesh, where interior nodes are shared by exactly four quadrilaterals in 2D and an all-hexahedral mesh whose interior nodes are shared by

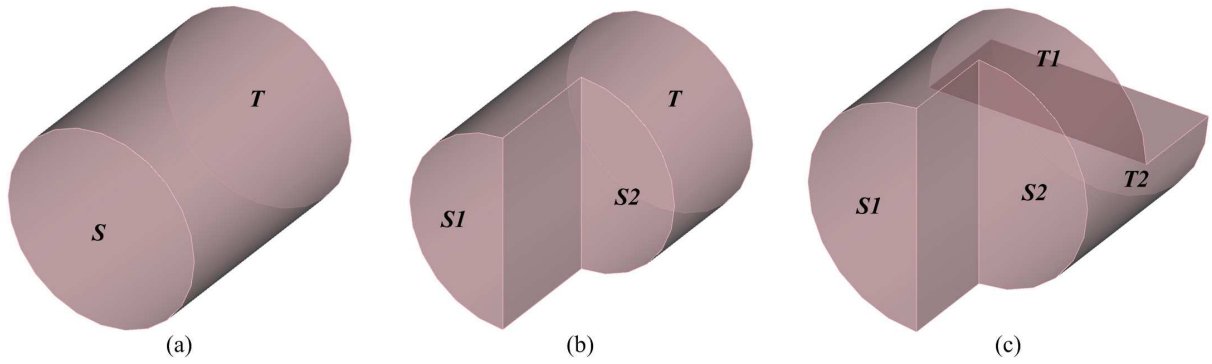

Figure 1.2: Three topological invariants of mesh sweeping. (a)a ***1-1*** sweeping example; (b)a ***M-1*** sweeping example; (c)a ***M-N*** sweeping example

exactly eight hexahedra and bounding surface meshes are structured as well. Structured meshes/grids are widely used in a wide range of simulations such as *linking* surface mesh generation by sweeping and boundary layers in ***CFD*** or ***FEA*** in solid mechanics. There are two kinds of methods for generating structured meshes, namely, *Mapping* and *Submapping*.

For the *Mapping* mesh, geometries must have four logical sides in 2D or six logical sides in 3D. Therefore, if a geometry does not satisfy these constraints, it must either be decomposed into several mappable pieces, or initially created out of mappable pieces. An example for 2D Mapped mesh is shown in Fig.1.3(a). *SubMapping*[114] mesh is defined as a structured mesh with more than four logical sides in 2D and more than six logical sides in 3D. In 2D, it is a meshing tool which extends the surface *Mapping* capabilities for structured all-quadrilateral mesh generation, and is suitable for surfaces which can be decomposed into mappable/submappable subsurfaces. Figure 1.3(b) shows an example of 2D Submapped mesh. Although the *Submapping* method generates high quality structured meshes, it is best suited for surfaces and volumes that are fairly blocky or that contain internal angles which are close to multiples of 0.5π. The same is true for *Mapping*. In

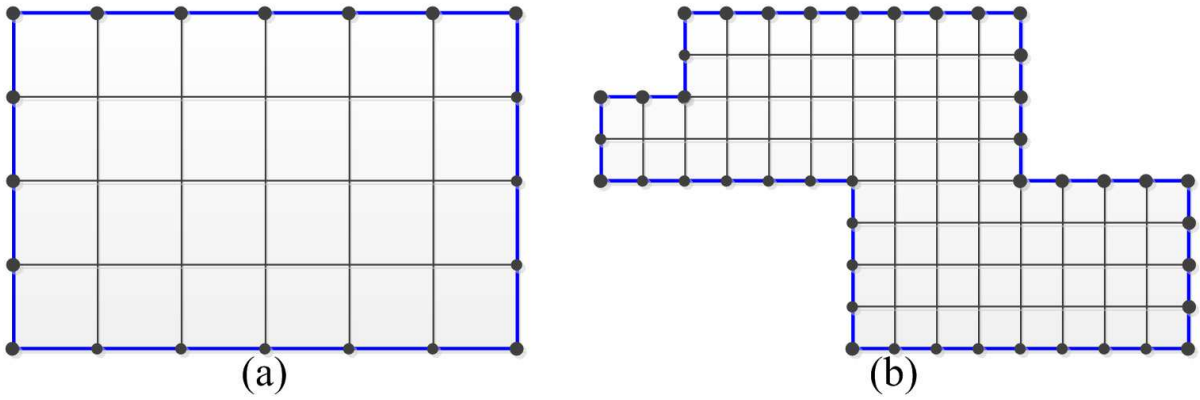

Figure 1.3: An example of 2D Mapped and Submapped mesh. (a)2D Mapped mesh; (b)2D Submapped mesh

both cases, they are more a characteristics of the combined meshing algorithm and corner assignment on surfaces or volumes.

The regular structure of *Mapping* and *Submapping* meshes imposes constraints on the mesh boundaries: two sets of edges are paired and the number of mesh edges called "interval" are constrained to be equal[1, 52]. For example, in Fig.1.4(a), the *Mapping* method requires that $I_1 = I_3, I_2 = I_4$. In Fig.1.4(b), the *Submapping* method requires that $I_1 = I_3 + I_5 + I_7 + I_9 + I_{11}, I_2 + I_6 + I_{10} = I_4 + I_8 + I_{12}$.

1.3 Attributes of Meshing Algorithms

There are some desired attributes for an ideal mesh generation algorithm[11, 74, 94]:

(1) *Geometry Generality* The meshing algorithms should be able to handle the largest number of geometries as possible. In the ideal case, it should be able to generate meshes for an arbitrary geometry.

(2) *Topological Matching* The generated meshes should resolve the topological fea-

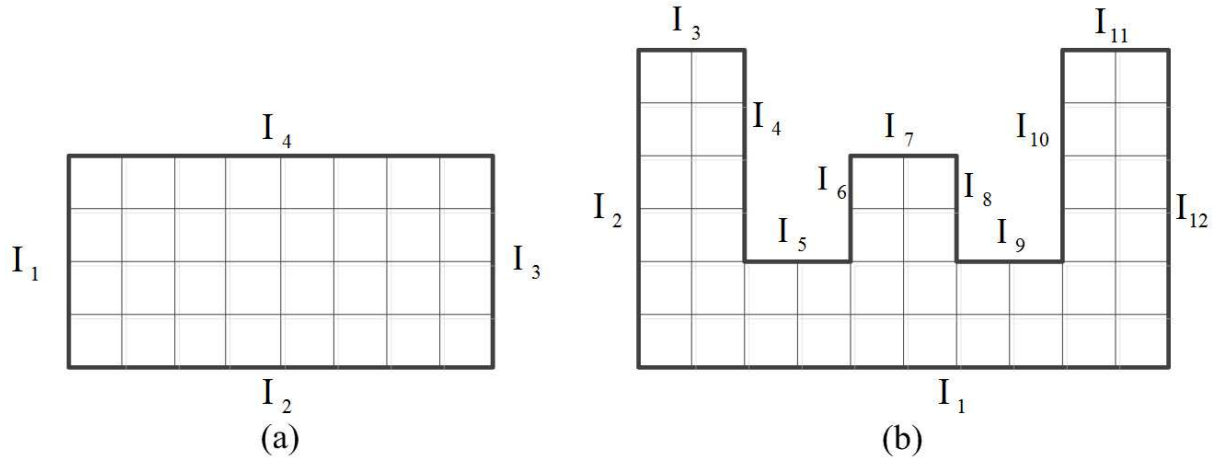

Figure 1.4: Interval assignment problem for *Mapping* and *Submapping*: (a)Interval assignment of *Mapping* method; (b)Interval assignment of *Submapping* method

tures on geometries. For instance, edges on a geometry should be resolved by mesh edges. Therefore, users can influence the final mesh by modifying the volume topology (for instance, by adding edges). In addition, users may add marks in the topological entities of the domain that are inherited to the final mesh. For example, users may assign boundary conditions or the solver to be used on different volumes

(3) *Mesh Quality* The resulting meshes should contain reasonable quality elements. While the mesh quality can be improved by smoothing algorithms, these techniques increase the computational cost to obtain a valid mesh.

(4) *Mesh Structure* The different mesh generation algorithms generate different types of meshes. The resulting mesh could be structured(usually by *Mapping* or *Submapping*), semi-structured(usually by *sweeping*) or unstructured. It is well known that structured meshes usually contain better mesh quality than unstructured meshes. Unstructured meshing algorithms have full automation, even though none are known for hexes or they are somewhat constrained such as Grid-based algorithm[77, 119,

120].

(5) *Boundary Sensitivity* Generally, the most important part during meshing is to generate meshes at boundaries of the domain with good mesh quality. In addition, in ***CFD***, it is also desired that elements smoothly follow shapes of the boundary in order to better capture the flow of fluids.

(6) *Orientation Insensitivity* The orientation of geometries in the space should not affect the meshing algorithms. Therefore, any dependencies on volume location and orientation should be removed from the meshing algorithms.

(7) *Size Control* The mesh should match the desired element size prescribed by users. This is particularly important in adaptive analysis.

(8) *Computational Cost* A meshing algorithm should be able to generate large meshes using a reasonable amount of computational resources.

1.4 Motivation

The common element types in the finite element methods are triangle or quadrilateral elements in 2D and tetrahedral or hexahedral elements in 3D. For triangular, quadrilateral and tetrahedral meshing, there exists robust fully automatic algorithms[59]. However, automatic all-hexahedral mesh generation algorithms with good mesh quality are available for a more limited class of geometries, with the only alternative being geometry decomposition. Because of the limited class of geometries for which hexahedral meshes can be built, the amount of time devoted to decomposing a model into pieces for which a known hexahedral mesh generation algorithm will succeed is significant. The geometry processing

for creating a hexahedral mesh can take several months for a generalized model, whereas tetrahedral meshes can often be created in a matter of hours or days[110, 111].

Therefore, it is often difficult and time-consuming to generate all-hexahedral meshes and even more challenging to obtain a hexahedral mesh with good mesh quality. In spite of the unavailability of an automatic & robust all-hexahedral mesh generation algorithm with good mesh quality, hexahedral meshes are sometimes preferred over tetrahedral meshes in certain applications and situations for the following reasons:

(1). Tetrahedral meshes typically require 4-10 times more elements than a hexahedral mesh to obtain the same level of accuracy[19, 68, 107].

(2). In some types of numerical approximations such as high deformation structural ***FEA*** with linear elements, tetrahedral elements will be mathematically stiff due to a reduced number of degrees of freedom associated with a tetrahedral element. The stiffness matrix eigenvalues for linear tetrahedrons are generally larger than those for linear hexahedrons thus linear hexahedrons can generally deform in a lower strain energy state and make them more accurate than linear tetrahedrons[7]. This kind of characteristics is known as "tet-locking"[81].

(3). In the boundary layer problem of ***CFD***, hexahedral meshes perform better than tetrahedral meshes by capturing the anisotropy of the flow field over viscous regions along the streamwise direction[68].

(4). In the structural analysis of composite materials, hexahedral meshes perform better than tetrahedral meshes with respect to the anisotropic properties of composite materials[68, 91].

Since there is no fully automatic all-hexahedral meshing algorithm, practical algorithms for generating swept meshes are often limited to a subclass of geometric solids to which the

sweeping algorithm can be applied. That is usually a large amount. Note that sweeping algorithm is not only used for all-hexahedral mesh generation, but also used for generating polyhedral or triangular prisms since in ***CFD***, swept meshes of polyhedral and triangular prisms are also desirable[100]. The sweeping algorithm for all-hexahedral, polyhedral and triangular prism mesh generation is attractive in a lot of industrial application due to the following advantages.

(1). No geometric decomposition is needed for the class of problems handled by the respective primitives. As long as the geometry is sweepable, no volume decomposition is necessary. This can save a lot of user intervention. For some unsweepable geometries due to the topological or geometric constraints, the geometry decomposition is needed such that the original geometry topology is changed to be sweepable for all-hexahedral, polyhedral or triangular prism mesh generation

(2). Meshing is fast. The hexahedral, polyhedral and triangular prism meshes are generated by sweeping the *source* surface meshes through volumes onto the *target* surface. Meanwhile, the sweeping process can be parallelized.

(3). The Element quality is relatively high as long as the mesh matches the geometry well.

(4). The swept mesh is boundary sensitive and orientation insensitive.

Therefore, the objective of this dissertation is to improve the sweeping algorithm by extending the sweeping capabilities to allow more geometries and improving the mesh quality, which is illustrated in the following five goals in Section.1.5.

1.5 Research Goals

For the past two decades, all-hexahedral mesh generation has been a very important and active research area. However, there does not exist a fully automatic hexahedral mesh generation algorithm with good mesh quality for a wider set of models. Hence, further efforts have to be put in generating high-quality hexahedral meshes for an arbitrary geometry by sweeping.

It is obvious that the sweeping techniques have been indispensable to semi-automatic swept volume mesh generation in the industrial applications. Therefore, the purpose of this dissertation is to improve the overall sweeping algorithm by improving specific important pieces, demonstrating their impacts in an united implementation, which can be described in the following goals.

- **Goal I: Surface Mesh Mapping** Given that the *source* surfaces are meshed with face elements, the sweeper requires a robust method to generate the *target* surface meshes with the same mesh connectivity as the *source* surface and vice versa. The main challenge of surface mesh mapping is to map surface meshes between two different shapes with good mesh quality, especially, between two concave or multiply-connected domains. Often a single transformation matrix is insufficient for mapping all parts of a surface to a different shape with good quality. From the mesh quality point of view, the most competitive technique to locate the interior nodes on the *target* surface is to perform *harmonic mapping* for both the *source* and *target* surfaces onto the common domain so that the *source* surface mesh can be embedded and mapped onto the *target* surfaces. Therefore, the first goal is: *develop an algorithm which can map one surface mesh onto another surface with the same mesh connectivity and with good mesh quality and that works for concave or multiply-connected surfaces.*

- **Goal II: Interior Nodes' Placement During Sweeping** Current existing

sweepers locate interior nodes inside volumes based on *affine transformation* or other ad hoc methods. However, they are incapable to distribute influences from *linking* surfaces on the interior nodes if there is a geometry with complicated internal structures. This is due to facts that a layer-by-layer mechanics is used and interior node placement is affected not only by current layer, but also its adjacent layers. In addition, they are incapable to handle the concave/multiply-connected domains since *affine transformation* is used. The key point for interior node placement inside volumes is to accommodate the shape of bounding surfaces, namely, redistribute interior nodes inside volumes when there is local deformation from bounding surfaces. In this work, a cage-based technique is developed and demonstrated to place interior nodes constrained by the *source*, *target* and *linking* surfaces. What is more, it can be applied to ***M-1*** or ***M-N*** sweeping. Therefore, the second goal is: *develop an algorithm which can place interior nodes inside volumes with good mesh quality.*

• **Goal III: Imprinting between *sources* and *targets* in multi-sweeping** Imprinting is useful for multi-sweeping problems. For a swept mesh to be possible, every mesh face element on the *source* surfaces must be matched to a specific mesh face element on the *target* surfaces and vice versa. During ***M-N*** sweeping, edges on *source* and *target* surfaces are very difficult to match since topologies between them are different, namely, it is difficult to resolve geometric curves on the *source* and *target* surfaces. In order to deal with this kind of challenge, most existing methods either use geometry decomposition or use imprinting between surfaces to guide the geometry decomposition. Another challenge during ***M-N*** sweeping is how to generate specific sweeping scheme: which area on the *source* surfaces will be swept onto a specific area on the *target* surfaces. In this dissertation, an edge patch imprinting algorithm is developed and demonstrated to resolve curves between *source* and *target* surfaces so that vertices and edges and s/t surface patches can be matched. The

resulting advantage is that a specific sweeping scheme by multi-sweeping is produced by imprinting: which *source* surface or regions of a *source* surface is swept toward a specific *target* surface. Therefore, the third goal is: *develop an imprinting algorithm based on cage-based morphing to determine the edge patch and surface patch matching between the source and target surfaces during* ***M-N*** *sweeping and generate specific sweeping schemes.*

• **Goal IV: Surface vertex classification during *Submapping*** *Submapping* is used to generate the structured grids on the *linking* surfaces. The main challenge of *Submapping* is to classify vertices so that corners can be determined for the structured grids. Current algorithms fail to find corners for *Submapping* in an ambiguous situation and users have to interact to adjust surface vertex types. However, the automation for placing corners during *Submapping* is critical to the general goal of widening the scope of models to be swept. Hence, in this dissertation, *Linear Programming* (***LP***) is used to correct the invalid corner assignment on the *linking* surfaces purely based on angles. The fourth goal is: *develop an algorithm which can automatically classify surface vertices on the linking surfaces and place corners for structured grids.*

• **Goal V: Meshability during Multi-Sweeping** The meshability or exactly sweepability in this dissertation is used to assess solids by multi-sweeping before running the sweeping algorithms themselves. There are two kinds of constraints which determine the sweepability for geometries, namely, topological constraints and geometric constraints. In addition, there are some cases that volumes with specified sweeping schemes are not sweepable due to users' specified matchings even though volumes are sweepable in themselves. They are due to topological constraints, geometric constraints or a combination of them. The understandings of sweepability constraints are improved to catch unfeasible sweeps due to user's identification error. Therefore, the fifth goal is to *assessing the sweepability problems for solids by sweeping.*

Implicit in all those goals described above is not only to improve specific individual parts of current sweeping algorithms, but also to demonstrate their implementation, not only individually but also together in an integrated fashion. This is the only way to truly assess whether progress has been made toward reducing the time to mesh by sweeping while also improving the overall robustness of the algorithm and quality of the resulting mesh.

2 LITERATURE REVIEW

This chapter presents a literature review for all-hexahedral mesh generation, sweeping algorithms and methods relevant to this work(e.g. conformal mapping techniques). It is structured in the following sections: Section 2.1 briefly describes the background of all-hexahedral mesh generation. In Section.2.2, a semi-structured swept volume mesh generation is described, which includes ***1-1***, ***M-1*** and ***M-N*** sweeping. Section.2.3 details some related works including mesh morphing, *Mapping/SubMapping* for *linking* surface mesh generation, *Cage-based Deformation* and homeomorphism for assessing sweepability. Finally, summary of existing sweeping algorithms is made.

2.1 General Hexahedral Meshing

Over the last two decades, several algorithms for fully automatic hexahedral mesh generation have been proposed, which are described in the surveys[5, 11, 59, 94]. However, none of the existing algorithms are robust, automatic or can generate high-quality meshes for an arbitrary geometry.

There are two families of automatic hexahedral mesh generation algorithms, that is, *Grid-based/Octree* methods[76, 77, 119, 120] and conversion from the tetrahedral meshes (by dividing tetrahedrons)[60]. For the *Grid-based/Octree* methods, a fitted Cartesian grid of hex elements is generated on the interior of the volume, then hexahedral elements are added at boundaries to fill gaps where the regular grid of hex elements does not conform to the boundary surfaces. However, the mesh quality at the boundary of volumes is very poor and interior hexahedral elements are not aligned with hex elements around boundaries. The resulting hexahedral meshes are highly dependent on the orientation of interior grid of hex elements. It remains an open question whether these meshes are of adequate quality

for analysis, though this has been the subject of recent work by Owen et al.[23, 61]. For the direct conversion from tetrahedral meshes, it generates an all-hex mesh by first generating tetrahedral meshes and then converting the tetrahedral meshes into hexahedral meshes by either combining tetrahedrons into hexahedrons, or decomposing a tetrahedron into hexahedrons[60]. While it is very easy to satisfy the boundary or topological constraints, the decomposition produces a mix of both high-valence and low-valence nodes, resulting in poor mesh quality within solids. Because it is difficult to combine tetrahedrons in such a way to guarantee the formation of all-hexahedrons, the combination approach is not usually robust, that is, it does not result in an all-hexahedral mesh.

The direct methods for all-hexahedral mesh generation include Whisker Weaving algorithm [95], Plastering [12, 87, 88], Medial Axis/Surface method [46, 62, 63], *Submapping/Mapping*[109] and sweeping [39, 66, 69, 86].

The Whisker Weaving algorithm was proposed by Tautges et al.[95] based on a global interpretation of the topological dual of an all-hexahedral mesh. It starts with a closed all-quadrilateral surface mesh bounding a solid geometry[13] and then constructs hexahedral element connectivity advancing into the solid by crossing or intersecting dual entities. However, since this algorithm only takes into account the mesh topology, bad quality elements are usually generated. In addition, the pre-meshed boundaries are also the cause of the poor quality or failure of algorithms. Some success has been found by removing the problematic sheets or placing additional constraints on the quadrilateral meshes [42, 54]. However, these approaches have been either too tedious or not reliable.

Plastering[12, 87, 88] is an application of advancing front algorithm[13] in physical space by projecting and connecting faces into the volume to make hexes. The hexahedral elements are first placed around the boundaries and then advances towards the inside of volumes. Intersecting faces must be detected in order to determine when and how to

connect to pre-existing nodes or seam faces. As *plastering* advances, complex interior voids may occur where it is impossible to fill with all-hex elements in some cases. The Constrained Plastering[89] is another application of advancing front algorithm for all-hexahedral meshes where constraints of prescribing a priori boundary quadrilateral mesh are relaxed while still maintaining the desirable characteristics of advancing-front meshes. The topology of boundary meshes is defined as a consequence of the interior meshing process. However, Current existing *plastering* algorithms have not been proven to be reliable on a large class of problems.

The *Medial Surface* method[46, 62, 63] involves the volume decomposition which is extended of *Medial Axis* method for quadrilateral meshing where the domain is decomposed by a set of medial surfaces. The medial axis of a domain is the locus of centers of spheres that touch the boundary in two or more faces. First the medial surface or axis of the volume is constructed and then the medial surface is used to decompose the volume into meshable pieces. The resulting subvolumes after decomposing are usually meshable by mid-surface subdivision which is hypothesized to be mappable or submappable. However, the medial axis is heavily sensitive to the boundary surface modification and thus it can be problematic. It may be degenerated since it may be locally defined by curves or points. For example, the medial axis of a sphere is a point. In addition, it is a decomposition method whose mesh is not guaranteed to be conformal across multiple volumes.

The rest of hexahedral meshing algorithms focuses on generating high-quality meshes for a subset of geometries with several configurations or settings(manually or automatically). However, they are not robust or fully automatic.

Volume *Submapping*[109] is used to generate structured all-hexahedral meshes. It uses pseudo virtual geometry to decompose complex volumes into "mappable subvolumes". Mappable regions are generally limited to volumes that are parameterized into logical

hexahedron. The local integer "*i*-*j*-*k*" space for the mesh connectivity is used to create the virtual surfaces inside the volume and separate the volume into mappable "sub-volumes". Even though *Submapping* can generate structured hexahedral meshes with good mesh quality, it only works for a very limited number of geometries.

Sweeping, sometimes referred to as two-and-one-half dimensional meshing, is another class of all-hexahedral meshing. The regular layers of hexahedra are formed by sweeping a set of quadrilaterals on surfaces through the volume toward another set of surfaces. Sweeping can be generalized to mesh a subset of volumes by defining *source* and *target* surfaces, which are not restricted to be one *source* and one *target* surface. With this kind of relaxation and extension, a greater subset of geometries may be meshed with high quality volume elements by sweeping. The details of various sweeping algorithms will be reviewed in Section 2.2.

With these algorithms, satisfaction of the boundary fitting is guaranteed if the geometric topology matches a pre-defined geometric topology written for a specific algorithm. Potentially, the sweeping algorithm can be augmented for larger classes of geometric topologies by enabling the algorithm to capture additional boundary details. The most obvious advantages for sweeping algorithm are related to the mesh size and relative speed with which large meshes can be constructed once the sweepable volumes have been found.

Without a reliable all-hexahedral meshing algorithm for arbitrary models, hexahedral meshes are generated using one of two strategies: divide& conquer approach and multi-block approach.

Divide and conquer approach: In the divide and conquer approach for hexahedral meshing, a geometry is decomposed into smaller pieces, each of which is presumably easier to mesh than the original part. The geometry decomposition continues until all sub-

volumes are meshable with existing algorithms such as *Mapping*[20], *Submapping*[73, 114] and sweeping[16, 17, 18, 39, 65, 66, 67, 69, 70, 86]. Generally, a large number of the resulting sub-volumes(2.5D geometries) are extrusion geometries which are meshable with current sweeping techniques. The decomposition is mainly a geometrical and interactive process, where engineers have to decide how to divide a given domain.

Multi-block approach: Multiblock method is an early approach to generate hexahedral meshes and is based on a mapped meshing. The domain to be discretized is manually divided into blocks which are then meshed separately using techniques such as parametric space *Mapping* etc. The blocks with locally structured grids in a three-dimensional region are commonly homeomorphic to a three-dimensional cube, thus having the shape of a curvillinear hexahedron.

2.2 Sweeping

This section reviews current existing sweeping algorithms: Section 2.2.1 gives a brief description of current ***1-1*** sweeping algorithms. The literature reviews of multi-sweeping are given in Section 2.2.2. Finally, a summary is made for the existing sweeping algorithms in Section 2.2.3.

2.2.1 One-To-One Sweeping

It has been demonstrated that geometries that are 2.5D can be meshed with all-hexahedral meshes by ***1-1*** sweeping[39, 66, 69]. The ***1-1*** sweeping is characterized by one *source* surface, one *target* surface and *linking* surfaces between them (see Fig.1.2(a)). The *source* surface should be homeomorphic to the *target* surface. Current methods work very well

for geometries whose *source* and *target* surface are convex and simply-connected. For multiply-connected and concave geometries, they all attempt to deal with them in various ways, but are often not robust for some challenging shapes.

Knupp [39, 40] devised an algorithm to locate interior nodes on the *target* surface and inside the volume based on linear *affine transformation* between two bounding loops of nodes and smoothing. In his approach, a layer of points with bounding loops are given, and a linear transformation is established between two layers if a second bounding loop consisting of the same number of bounding points on the second layer is given. In order to avoid a singular transformation matrix(the transformation matrix is singular if and only if all the boundary points on the *source* surface are coplanar with those on the *target* surface), a set of point vectors on the bounding loops are redefined as follows (the details of *affine transformation* for sweeping mesh nodes between the *source* and *target* surface are described in Appendix A.1).

$$x_k = x_k - (2x_c - x_{c'}) \tag{2.1}$$

$$x'_k = x'_k - x_c \tag{2.2}$$

The *affine transformation* between successive loops is computed using an advancing front method based on consecutive boundary loops derived from the *linking* surfaces. After locating interior nodes and connecting points based on mesh face element connectivity from the *source* surface mesh, the mesh on a specific layer is smoothed independently of connections to nodes on the neighbouring layers. However, this approach fails for moderately concave or some multiply-connected *source*/*target* surfaces, and often does not produce smooth transitions between curved *source*/*target* surfaces in the sweeping direction. The reasons are: linear transformation takes all the boundary nodes as the

whole. It is not able to accommodate changes both from outmost boundaries and interior boundaries and will take an intermediate stage for changes of boundaries if there is a multiply-connected geometry. The same thing happens if there is a concave geometry because it fails in accommodating changes from both concave areas and other areas.

Roca et al.[65, 66, 67, 69] used the *Least-Square* approximation of an *affine mapping* for projections of the *source* surface mesh onto the *target* surface. In order to avoid the root-finding problem such as an orthogonal projection of nodes on the *target* surface[31], the mapping through *Least-Square* algorithm is done in the parametric spaces between the *source* and *target* surface by using only boundary nodes. All the boundary nodes are used to compute the *affine transformation* matrix. In order to avoid the skewness and flattening effects when projecting interior nodes from the curved *source* surface onto the curved *target* surface[65], several functions are introduced to perform the *Least-Square* approximation. The details of *Least-Square* approximation method for sweeping mesh nodes between the *source* and *target* surface are depicted in Appendix A.2.

$$F(\mathbf{A}) = \sum_{i=1}^{m} ||\mathbf{y}^i - \mathbf{c}^Y - \mathbf{A}(\mathbf{x}^i - \mathbf{c}^X)||^2 \tag{2.3}$$

$$G(\mathbf{A}) = \sum_{i=1}^{m} ||\mathbf{y}^i - \mathbf{c}^X - \mathbf{A}(\mathbf{x}^i - \mathbf{c}^X + \mathbf{c}^Y - \mathbf{c}^X)||^2 \tag{2.4}$$

$$H(A, u^X, u^Y) = \sum_{i=1}^{m} ||\mathbf{y}^i - \mathbf{c}^Y - \mathbf{A}(\mathbf{x}^i - \mathbf{c}^X)||^2 + ||\mathbf{u}^Y - \mathbf{A}\mathbf{u}^X||^2 \tag{2.5}$$

The *Least-Square* approximation functions are also used to avoid singularities in the transformation matrix that sometimes arise for particular arrangements of *source*/*target* surfaces (see discussions in Knupp's method described above). Different *Least-Square* approximation functions will have different effects. For example, if function $F(\mathbf{A})$ is used for an example model as Fig.2.1(a), the flattening effect happens since the boundary of the *source* surface is planar. If function $G(\mathbf{A})$ is applied for the example model as Fig.2.1(a),

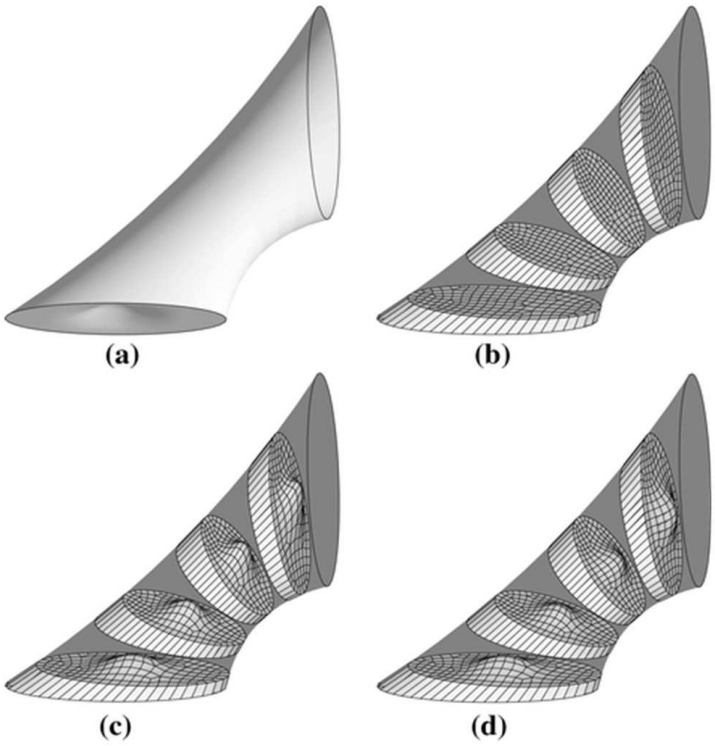

Figure 2.1: An example of skewness and flattening effects of sweeping[65]: (a)the geometry model with a curved *source* surface but flat *source* boundary and flat *target* surface; (b)Flattening effects of sweeping; (c)Skewness effects of sweeping; (d)Sweeping without flattening and skewness effects

the *source* surface shape is not well preserved during sweeping as Fig.2.1(c), namely, the skewness effect. There is no skewness or flattening effects as Fig.2.1(d) when function $H(\mathbf{A})$ is used for an example in Fig.2.1(a).

Roca's method has solved the problem of smooth transition between the curved *source* and *target* surfaces during sweeping. However, this approach still suffers from poor mesh quality on the *source*/*target* surfaces with no parameterization because mapping from a *source* surface to its *target* surfaces is computed with the help of parametric space. It fails as well when meshing discrete(i.e. facet-based) geometry because there is no parametric space for facet-based geometry. The same thing happens when there is a multiply-connected or concave geometry in that it is a linear method.

The reason why linear *affine transformation* is incapable to deal with concave or multiply-connected geometries can be illustrated as follows. Suppose there is a multiply-connected geometry as Fig.2.2 where the outmost boundary is kept constant while interior hole boundary is enlarged during sweeping from the *source* surface to the *target* surface. If a linear *affine transformation* is applied, a transformation matrix will be obtained by using all the boundary nodes between the *source* and *target* surfaces.

$$Matrix = \begin{pmatrix} 1.0367 & 0.0 & 0.0 \\ 0.0 & 1.0367 & 0.0 \\ 0.0 & 0.0 & 1.0367 \end{pmatrix}$$

The obtained transformation matrix is pretty close to and a bit bigger than the identity matrix. This is due to the fact that the outmost boundary has more influences (more boundary nodes) on the transformation matrix than interior hole boundary and the interior hole is enlarged. If boundaries on the *source* surface are mapped onto the *target* surface as well, the resulting mesh can be depicted in Fig.2.2(c) with blue colour and real boundaries on the *target* surface are denoted with red colour. From Figure 2.2(c), it is known that the mapped *source* outmost boundary onto the *target* surface is enlarged a little bit because of the enlarged hole from the *source* surface to the *target* surface; the mapped interior boundary on the *target* surface is enlarged a little bit but far away from the real interior boundary on the *target* surface because, on one hand, the outmost boundary between the s/t surfaces is almost kept constant and has more influences on the transformation matrix; on the other hand, interior boundary between the s/t surfaces is enlarged a bit but it has less influence on the transformation matrix than the outmost boundary. Therefore, one transformation matrix for mapping all the interior nodes cannot deal with multiply-connected geometries. The resulting *target* surface mesh with physical *target* boundaries is depicted in Fig.2.2(d) where inverted quadrilateral elements are produced. Neither does

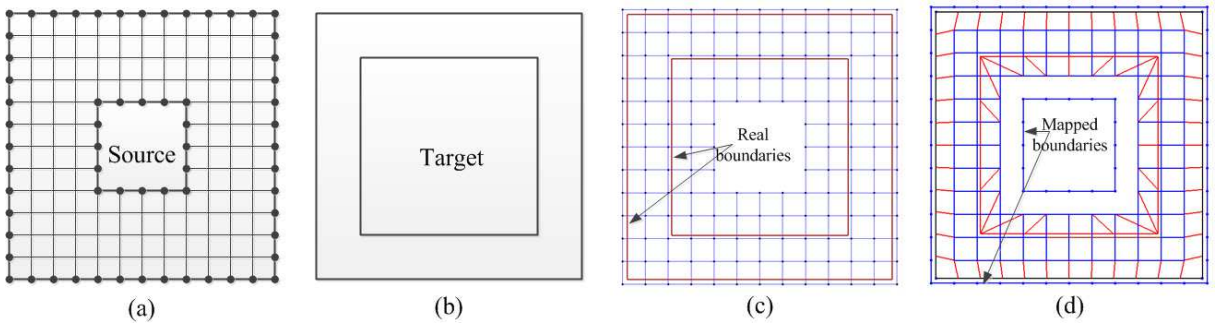

Figure 2.2: An example why the affine transformation fails: (a)*source* surface mesh; (b)*target* surface; (c)mapped target surface mesh with free boundaries; (d)mapped target surface mesh with real boundaries

one transformation matrix deal with the concave geometries since the boundary nodes on the concave area are a small portion of all the boundary nodes and thus have less influence on the transformation matrix.

Staten et al.[86] developed an algorithm called *Background Mesh Sweeping* (***BMSweep***) to determine locations of interior nodes while volume sweeping. The background mesh generated by tessellating the boundary nodes on the *source* surface in the parametric space is needed to provide a framework for computing nodes' locations on each layer. During sweeping, the background mesh would be elevated to each layer. With the background mesh in 2D parametric space, every interior node is located in a specific triangle where it is located. Then those parametric coordinates for each interior node on the *source* surface which are used for calculating barycentric coordinates are created. In order to handle the case that *source* and *target* surface are not flat, the offset distances are used since the background mesh is generated from boundary nodes without including surface curvatures. Finally, interior nodes are located by using the barycentric coordinates evaluated with real space coordinates of the background triangle nodes on the corresponding layer, which are always on the *linking* surfaces. However, the background triangular meshes can invert if

holes move or rotate relative to the body, which can produce an inverted map. For example, Figure 2.3 describes a failed case of ***BMSweep***: suppose there is a geometry like Fig.2.3(a) with specified *source* and *target* surfaces. Edges on the *source* surface are discretized as Fig.2.3(b). With the help of *linking* surfaces, edges on the *target* surface are discretized as Fig.2.3(c), which is mapped from Fig.2.3(b). The constrained Delaunay Triangulation is used to generate the background mesh on the *source* surface by using edge meshes. ***BMSweep*** uses the same background mesh connectivity for all the layers including the *target* surface. Hence, inverted triangular elements in the background mesh on the *target* surface are produced as Fig.2.3(e). The subsequent *source* and *target* surface meshes are shown in Fig.2.3(f) and Fig.2.3(g), respectively, where there are inverted elements on the *target* surface. In addition, there are assumptions that the *source* and *target* surfaces are meshed with face elements and all the *linking* surfaces are meshed with structured grids. For sweeping, the most difficult thing is to map a surface mesh on the *source* surface to the *target* surface, which has not been addressed by their approaches.

The *BoundaryError* method was proposed by Blacker[10, 79, 109]. It was introduced to place nodes by using the linear *affine transformation* and a subsequent *residual error* correction. In order to successfully capture curvatures of the *source* surface and *target* surface, the *BoundaryError* method calculates the *residual error* twice, once sweeping from the *source* surface and terminating at the *target* surface, and then sweeping from the *target* surface and terminating at the *source* surface. Two error distances are then interpolated to compute the final location of interior nodes. This method is pretty useful for placing interior nodes between the *source* and *target* surface. However, this approach assumes that the *target* surface is meshed with the same mesh connectivity as the *source* surface while the *source* surface mesh needs to be projected onto the *target* surface with good mesh quality during sweeping. In addition, the *BoundaryError* method locates

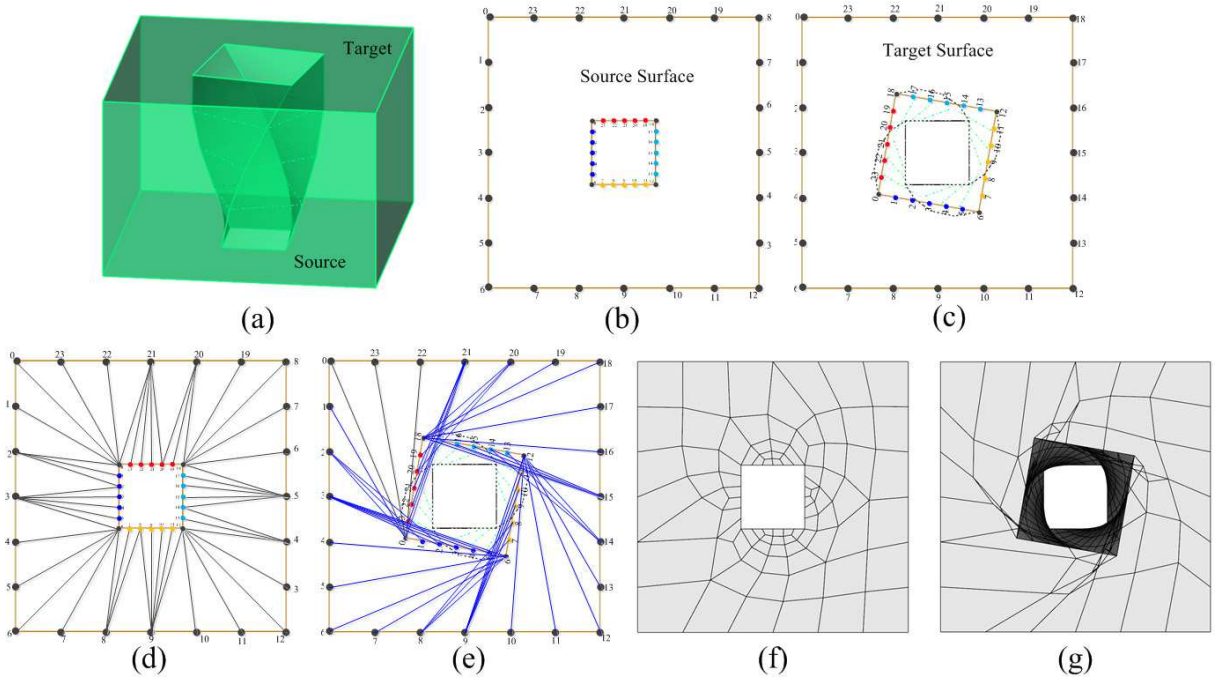

Figure 2.3: ***BMSweep*** fail case: (a)Geometry model; (b)edge mesh on the *source* surface; (c)edge mesh on the *target* surface; (d)Background mesh on the *source* surface; (e)Background mesh on the *target* surface; (f)*source* surface mesh; (g)*target* surface mesh

interior nodes inside the volume by using the boundary nodes only on one layer instead of neighbouring regions on the *linking* surfaces and does not have the property of locality(a node only affects those node locations in their neighbouring area).

2.2.2 Multi-Sweeping

Sweeping algorithms[16, 17, 39, 65, 67, 86] can be used in extrusion geometries defined one *source* and *target* surface, namely, ***1-1*** sweepable geometries. However, in the industrial application, real CAD extrusion solids are not just ***1-1*** sweepable, but also ***M-1*** and ***M-N*** sweepable which are more general cases.

A ***M-N*** sweepable geometry is characterized by multiple *source* surfaces, multiple

target surfaces and *linking* surfaces between them. Multi-sweeping including ***M-N*** and ***M-1*** sweeping is difficult because of the need to resolve the bounding edges of all the *target* surfaces into the *source* surface meshes, even though those edges are not represented on the *source* surface meshes, and vice versa. This is due to facts that every *target* surface and its bounding edges must be swept towards the *source* surfaces and those edges must be reasonably placed and imprinted onto the *source* surfaces and vice versa so that there are no distortions or degenerated elements when sweeping the *source* surface meshes onto the *target* surfaces. In effect, this requires the bounding edges on all the *target* surfaces to be imprinted onto the *source* surfaces before the *source* surfaces are meshed. This kind of imprinting process can be quite complex in practice. In addition, interior node placement inside a multi-sweepable volume with complicated internal becomes problematic by current multi-sweepers since they rely on the volume decomposition while internal surfaces used for physical or logical decomposition are not correctly determined. What is more, the layer-by-layer approach used by current multi-sweepers has an inherent disadvantage: interior node placement is only affected by one layer of boundary nodes instead of boundary nodes in the neighbouring regions, which usually produces poor volume meshes.

In order to solve the multi-sweeping problems, the most intuitive way that one can come up with is to decompose solids such that each decomposed pieces are ***1-1*** sweepable. This is why volume decomposition has been prevalent for multi-sweeping problems since the last two decades. There are several disadvantages for the volume decomposition: (1)poor mesh quality produced by volume decomposition since interior nodes are separately placed in each decomposed pieces; (2)difficulties to generate the internal surfaces for volume decomposition when there are complicated internal structures inside volumes; (3)extra constraints introduced during decomposing solids since the structured quadrilateral meshes on the *linking* surfaces are used to guide volume decomposition but it is difficult to do the

interval matching for edges on the *source* and *target* surfaces; (4)poor imprinting algorithm used for volume decomposition in that loops of edge mesh are propagated based on the *affine transformation* or *Least-square* method and loops are not correctly placed on the concave or multiply-connected domain.

Scott et al.[78] presented a *Polymorphic Many-To-One Sweeping Tool* (*PMOST*) which combined the ***BMSweep*** and BoundaryError method. First volumes are decomposed into ***1-1*** sweepable blocks with each *source* surface mesh swept to the *target* surface, respectively. It is a generalized ***1-1*** sweeper for ***M-1*** sweeping where *PMOST* is used for each block sweeping. However, it fails for highly concave or multiply-connected geometries. In addition, it is a decomposition method which suffers from poor mesh quality inside volumes since interior nodes are placed separately in each decomposed sweeping portion.

White et al.[109] introduced two CUBIT meshing facilitators: volume *Submapping* and n-surface sweeping(***M-1*** sweeping), which had been used to reduce the need for manual decomposition and propagation of these cuts throughout the model and generate high-quality hexahedral meshes. The n-surface sweeping is used to project multiple *source* surfaces on a single *target* surface. However, poor mesh quality may be generated due to the fact that interior nodes are placed separately in each decomposed pieces. What is more, it is difficult to generate internal surfaces for volume decomposition when there is a geometry with complicated internal structures. The ***M-1*** sweeping is mostly a slightly more general version of ***1-1*** sweeping, but with slightly less user interaction for decomposition.

Lai et al.[41, 51] presented an algorithm to rebuild the boundary representation of *source* surfaces by Boolean operations between boundary loops of both *target* and *source* surfaces to accomplish s/t imprinting described earlier. They enhanced the traditional ***1-1*** sweeping techniques by developing a projection technique that minimized mesh distortion and allowed sweeping multiple connected surfaces to a single *target*, multiple unconnected

surfaces to a single *target*, and multiple unconnected surfaces to multiple unconnected *target* surface sweeping. However, it is still a geometry decomposition which suffers from poor mesh quality since interior nodes are placed separately in each decomposed sweeping portions. What is more, the structured quadrilateral meshes on the *linking* surfaces are used to guide sweeping loops of boundary nodes and interval assignment problem for different chains of *linking* surfaces has to be solved, which creates extra constraints for the s/t imprinting process.

Blacker[10] introduced the Cooper tool for a general subset of geometries. It recognizes applicable geometries and decomposes geometries into logically single axis swept subvolumes. Then it enforces strict compatibility constraints by imprinting, intersecting, and matching all the bounding pieces to generate a well-formed continuous mesh. However, poor mesh quality can be produced in that for each subvolume, internal surfaces used for decomposing volumes constrain the volume mesh. The *linking* surface meshes are used for volume decomposition, which will create extra constraints for imprinting since interval assignment problem on the different chains of *linking* surfaces is needed to be solved. In addition, the ***1-1*** sweeping engine based on the *affine transformation* is used, which produces poor virtual volume decomposition.

Miyoshi et al.[53] presented a multi-axis cooper tool: a tool for creating all-hexahedral meshes by using multi-axis imprinting sweeping. It recognizes applicable geometries and divides them into hierarchy subvolumes, which are then meshed by existing single-axis sweeping tools. The resulting meshed volumes, called inlay volumes, contain the individual mesh which is non-conformal or discontinuous at their interfaces. The non-conformal sections with meshes are then removed and replaced with a conformal mesh using the cooper tool[10]. However, this approach still involves tedious geometry decomposition, which suffers from the same drawbacks as the Cooper Tool[10].

White et al.[112] proposed the *CCSweep* which decomposed multi-sweepable volumes into ***M-1*** sweepable volumes by projecting the *target* surfaces through the volume onto the corresponding *source* surfaces. First, the *linking* surfaces are discretized with structured quadrilateral meshes, which provides a layering system to traverse the volume vertically. The *target* faces approximated by the loop face data consisting of the ordered boundary nodes are then pushed through the volume. The nodal projections by following the exterior layering and projecting new interior nodes are calculated through the *affine transformation* and corrected locally by the *residual errors*. The projected surfaces are imprinted and merged with the *source* surfaces to determine the decomposition of solids, which enables the *source* faces to topologically match *target* faces. The interior faces are created to decompose the volume into separate new subvolumes. In summary, it is a decomposition method for multi-sweeping problems and poor mesh quality will be produced inside volumes if there is a volume with complex internal structures. In addition, the linear *affine transformation* is used and it fails for geometries with concave or multiply-connected surfaces. What is more, intersecting points produced during imprinting are only allowed to occur at existing nodes on the bounding loops, which over-constrain the imprinting and also prevent interval adjustment to make the resulting surface patches meshable.

Ruiz-Girones et al.[70, 71] proposed to use a decomposition method for multi-sweeping problems based on a computational domain and three-stage procedure. It is stated that the mesh element quality generated by using the multi-sweeping method is heavily affected by positions of inner nodes created during the decomposition process. Hence the quality of the decomposition process is improved by two processes: on one hand, the robustness of imprinting is improved by using the computational domain(project non-planar loops onto planar domain), which improves several geometric operations(such as determination whether a point is inside a loop or not) involved in the imprinting

process. On the other hand, a three-stage procedure is proposed to improve the location of interior nodes created during the decomposition process: advancing nodes from the *source* surfaces to the *target* surfaces; advancing nodes from *target* surfaces to the *source* surfaces; the final location of interior nodes is computed as a weighted average of two projections. In summary, Ruiz-Girones's method applies Roca's methods(***1-1*** sweeping by the *Least-Square* approximation[65]) and decomposition to multiple sweeping problems and it suffers from the same disadvantages as their previous methods: it fails for concave or multiply-connected surfaces when projecting one surface mesh onto another surface. In addition, it is a decomposition method: interior nodes are placed separately in each subvolume and thus poor mesh quality is produced.

2.2.3 Summary

Current existing sweepers including ***1-1***, ***M-1*** and ***M-N*** sweeping are not robust in the following ways:

(a). **Interior Node Placement on the *Target* Surfaces** For surface mesh mapping between two physical surfaces, the key problem is to locate interior nodes such that the boundary constraints and surface features are taken into account. The linear *affine transformation* works for the convex/simply-connected domain while it fails for concave or multiply-connected domain (Knupp's method[39] and Roca's method[65, 69]). ***BMSweep*** fails for geometries with twisted holes between the *source* and *target* surfaces since the same background mesh connectivity is used. In addition, it is incapable to do the surface mapping by using only boundary nodes when there is a curved s/t surface and parametric space has to be used.

(b). **Interior Node Placement Inside Volumes** The interior node placement is

constrained by the *source*, *target* and *linking* surfaces. Current existing methods place interior nodes based on the *affine transformation* or *Least-square approximation* method by using the layered structured quadrilateral meshes on the *linking* surfaces. The subsequent error correction (BoundaryError) is used to improve the mesh quality. However, they fail in one way or another since a single transformation matrix can not work for all parts of a volume. Moreover, transformation plus ad-hoc correction such as BoundaryError method fails often. In addition, those methods are difficult to be extended for multi-sweeping problems unless volumes are decomposed.

(c). **Volume Decomposition during Multi-sweeping** The key problem of multi-sweeping is to resolve every curves from the *source* surfaces onto the *target* surfaces and vice versa, combined with the *source* and *target* surface patches needing to be matched. Most existing algorithms use the resulting imprints based on the *affine transformation* or *Least-square Approximation* to decompose the volume into ***1-1*** meshable pieces, producing extra geometric constraints for the interior of original volume. These decompositions are difficult to compute for complex volumes. Before s/t imprinting and volume decomposition, they generate the *linking* surface meshes in order to guide the imprinting operation and decomposition, which creates extra constraints for both these processes.

2.3 Related Works

Current existing sweepers are not robust for reasons described in the previous section. Fortunately, there are existing works in the computer graphics field that can be applied to solve these problems. Therefore, we describe some specific techniques/technologies that will be used later in this dissertation to solve problems described earlier and review them

as follows: surface mesh morphing methods between two surfaces in Section 2.3.1; surface vertex classification methods during *Submapping* in Section 2.3.3; the binding methods of cage-based deformation for interior node placement in Section 2.3.2. In the end, the sweepability problem is reviewed in Section 2.3.4.

2.3.1 Surface Mesh Morphing

Mapping a surface mesh from the *source* surface to the *target* surface in a sweeping algorithm is quite similar to the general problem of mesh morphing, which has been developed for animation and other applications [30, 40]. During ***1-1*** sweeping, the structured meshes on the *linking* surfaces provide correspondences for boundary nodes between *source* surfaces and *target* surfaces. The key problem is to locate interior nodes on the *target* surfaces by mapping the *source* surface mesh. The linear transformation methods(*affine mapping* or *Least-Square* approximation) are widely used. However, they fail for concave and non-simply connected geometries in that they take all the boundary nodes as a whole into account and ignore the deformation of boundary nodes in a local domain. Morphing techniques, on the other hand, can redistribute influences of boundaries onto interior nodes in a nice way when performing the surface mesh mapping. Therefore, techniques of surface mesh morphing can be utilized during sweeping in several ways.

In the area of computer graphics, there are a lot of ways to morph from one object to another and mesh morphing is a active and popular research area. However, they are mainly used to produce the smooth shape deformation where a *source* and *target* shape are provided. For example, Lerios et al.[45] used fields of influence of 3D primitives to warp volumes where the *source* and *target* shape were given and a sequence of intermediate shapes were produced; Alexa et al.[2, 3] used the differential coordinates for local mesh morphing and deformation; The mesh geometry was described in the differential way so

that insertions of local features from one shape to another didn't suffer from difference in the absolute coordinates. Blanding[14], Kosuke[36] and Nealen et al.[55] used the skeleton-based approach for morphing; The vertex correspondence was determined based on the skeleton of the *source* and *target* shapes. This was used to solve morphing when the *source* and *target* shapes have different topologies. Yan et al.[115] proposed to use the strain field interpolation for 3D morphing. First the strain fields relating *source* and *target* shapes were extracted and then a list of intermediate shapes were obtained by interpolating the strain field between zero and a final desired value. Yoo et al.[117] proposed to morph from a *source* shape to a *target* shape by using the template mesh which was mapped directly to the *target* mesh based on a shape deformation method by using an implicit function and mesh smoothing. The intermediate shapes were obtained by linear interpolation of the modified Laplacian coordinates of the *source* and *target* meshes. Yang et al.[116] used the T-spline level sets for 3D shape metamorphosis where in-between objects were constructed by using T-spline scalar functions. First a global coordinate transformation was used to approximately align two objects. The T-spline control grid was adaptively generated and initial T-spline level was found by approximating the signed distance function of the *source* shape. Then a evolution process by combining the signed distance function of the *target* shape and using curvature-dependent speed function was used until it converged to the *target* shape. Sheffer et al.[80] proposed the pyramid coordinates for morphing and deformation based on a small number of user-specified control vertices. Pyramid coordinates captured the local shape of the mesh around each vertex and helped maintain this shape under various shape editing operations. They were based on a set of angles and lengths relating a vertex to its immediate neighbors. Lin et al.[47] described a metamorphosis between two closed manifold genus-0 polyhedral models where spherical parameterizations of the *source* model and *target* model were created first.

Then feature vertex pairs were specified and a fold-over free warping method was used to align two spherical embeddings. However, those techniques described above are mainly used to create a smooth deformation between two given shapes, which are different from the surface mesh mapping during sweeping: the *source* surface mesh is given and a good mapping of *source* surface mesh onto the *target* surface needs to be computed.

Fortunately, there are some techniques which can possibly be used for mapping surface meshes between two shapes during sweeping. For example, during morphing, the key problem in morphing from one shape to another is to establish the correspondence map[43] which is used to match nodes between two shapes. There is only one requirement for dense correspondences between two irregular connectivity meshes, that is, topologically equivalent. This involves the construction of mappings from the fine meshes to their coarse base domains and a mapping between the base domains. These mappings are realized through the metamesh, a topologically and geometrically merged version of *source* and *target* meshes. Recently, a lot of scholars have solved this problem from their perspectives. Kanai et al.[35] used *harmonic maps* for morphing arbitrary triangle meshes with the same topology. The basic idea was to define reference shape by using vertex-to-vertex correspondences between two meshes. The partition of mesh was defined by the reference shape and partitioned meshes were then embedded into a polygonal region in the plane through the *harmonic map*. By overlapping two embedded meshes, the correspondence was established between them. Lee et al.[43] presented a new method for morphing two homeomorphic triangle meshes of arbitrary topology controlled by users. The MAPS algorithm[44, 106] was employed to parameterize both meshes over simple base domains and an additional *harmonic map* would bring the latter into correspondence. Users were required to specify the feature pairs of points. Fan et al.[25] applied the polycube-based cross-parameterization for mesh morphing (Polycube map[92] was a generalization of cube-map mechanics for

seamless texture mapping with low distortion. Polycube-based cross parameterization was to parameterize meshes on their polycubes and construct a correspondence through their polycubes). Michikawa et al.[50] proposed a new multiresolution-based shape representation for 3D mesh morphing. Two types of fitting schemes had been used to interpolate the mesh. The common method described above is the mapping method, namely, *harmonic mapping*, which inspires us to solve the surface mesh mapping problem during sweeping.

Therefore, based on the concept of morphing and inspired by the previous work mentioned above, a new surface mesh mapping during ***1-1*** sweeping which will be described in Chapter 3 is proposed in this work. *Harmonic Mapping* which belongs to conformal mapping has many merits which are valuable for surface mapping[33, 105]:

(1) *Harmonic Mapping* is computed through the global optimization and takes into account the surface topology. Therefore, local minimum, folding and clustering can be avoided.

(2) *Harmonic Mapping* is insensitive to the surface resolution and noises on surfaces.

(3) *Harmonic mapping* does not require the surface to be smooth. Even there is a sharp feature on the surface, it can be accurately computed.

(4) Under certain circumstances such as mapping both *source* and *target* surfaces onto an unit disk, *Harmonic Mapping* exists and is diffeomorphism.

(5) *Harmonic Mapping* is determined by the metric, not the embedding. This indicates that *Harmonic Mapping* is invariant for the same surface with different orientations. If there is not too much stretching between two surfaces with different mathematical expression, they will induce similar *harmonic maps*. In addition, it preserves angles and lengths in the metric.

Special cares should be taken for *harmonic mapping* of multiply-connected surfaces. According to Koebe's uniformization theory[118], all genus zero multiply-connected surfaces can be mapped to a planar disk with multiply circular holes. Based on this kind of idea, first we can map both the *source* and *target* surface faceting onto a common domain so that the *source* surface meshes can be embedded on the common domain and mapped back to the *target* surface.

2.3.2 Cage-based Deformation

2.3.2.1 Introduction

Cage-based deformation is a general class of methods where a parameterization of nodal positions with respect to a set of bounding nodes(the cage) around the mesh is evaluated against a deformed cage to produce a good-quality deformed mesh. During a cage-based deformation, the undeformed stage is the start point before a deformation where an undeformed cage is created to bound objects and binding relationship between objects and cage vertices is solved (we call it the binding function), that is, objects are a function with cage vertices as arguments and inputs; the deformed stage is the final point of a deformation where cages are deformed by users or algorithms and deformed objects under the deformed cage are interpolated by using the same binding function as the undeformed stage but with the deformed cage as inputs. Objects can be any shape while there is only one requirement for the bounding mesh cage that it could be any shape of mesh (convex or concave) but it must be closed.

Figure 2.4(a-b) shows that a triangle is deformed when a cage vertex x_1 on its bounding cage moves to x_1'. In Fig.2.4(c), it is a sphere that has been altered using proportional editing, as a result, the deformed object alters its shape in response. The object(gray color) is bound with its cage mesh. When the cage mesh is deformed, the object is deformed in

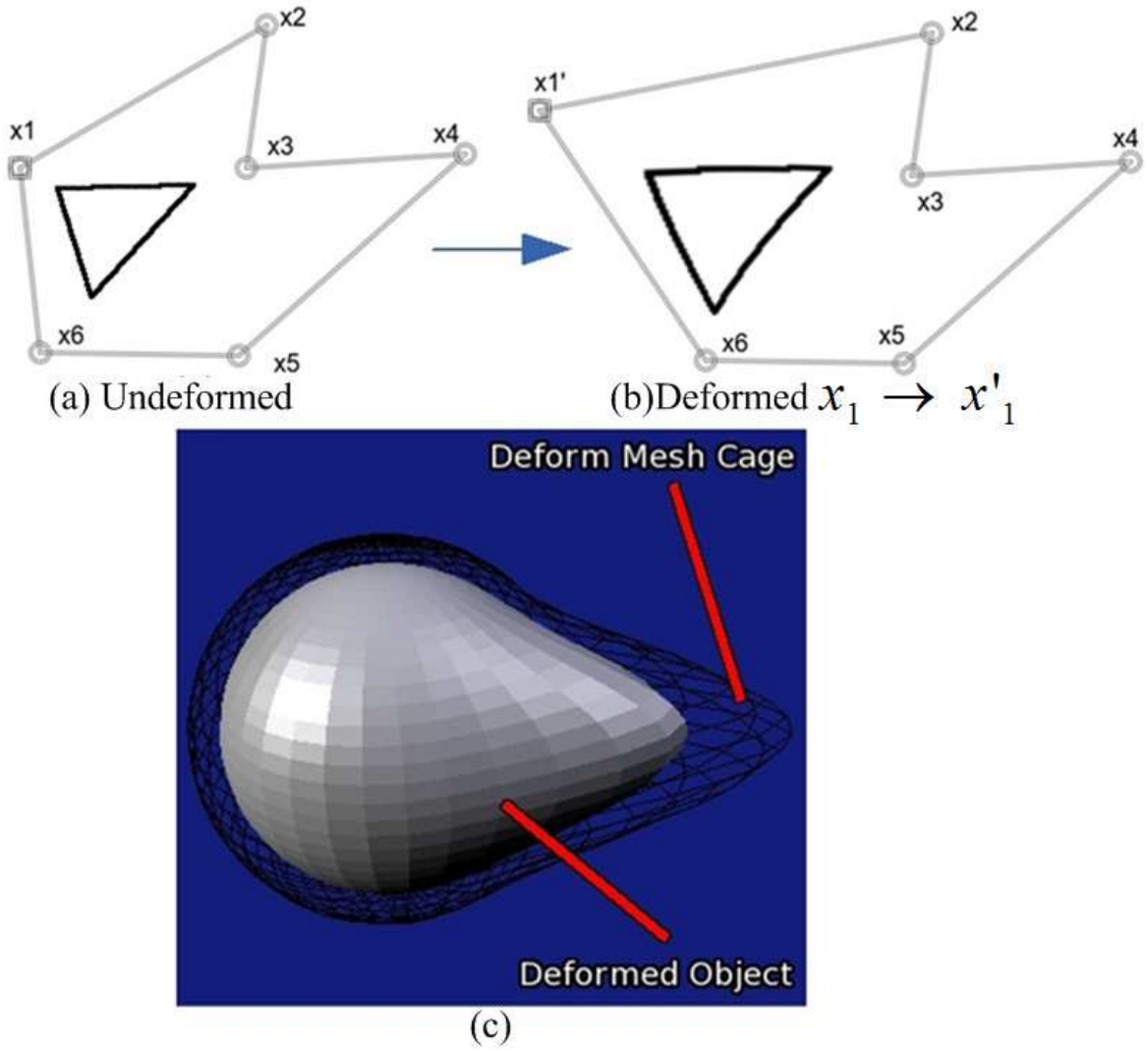

Figure 2.4: Examples of cage-based deformation[16, 18]: (a-b)an example of 2D deformed triangle; (c)an example of 3D deformed sphere[15]

response. Figure 2.5 shows an example with local deformation by using cages: when the bounding cage for the index finger and middle finger is bent to the right side, only index and middle finger are deviated to the right side and the remaining regions are kept fixed.

In the cage-based deformation, a cage can be any triangle mesh in 3D, or more generally a polyhedral mesh in 3D. The deformation from the cage will affect its inside volumes,

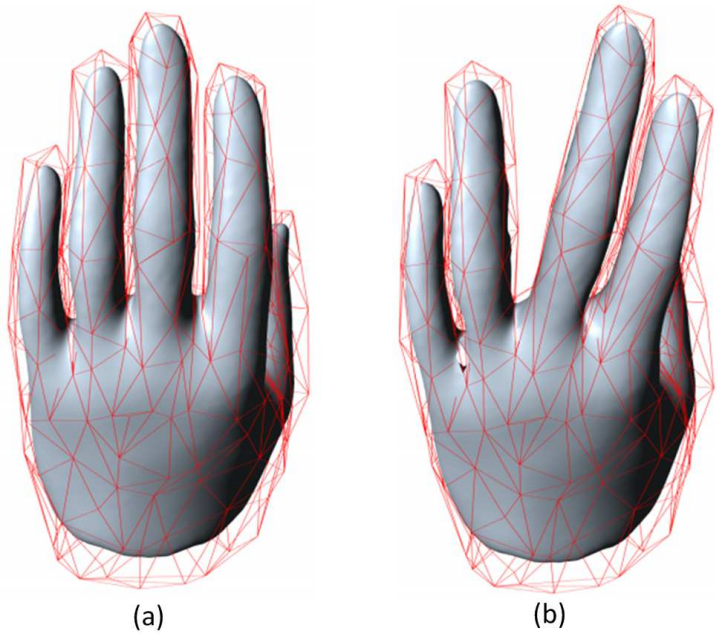

Figure 2.5: A deformed finger example with a cage deformation[56]. (a)undeformed finger model; (b)deformed finger model

and therefore any object it contains as long as there is a well-defined binding relationship between the cage surfaces and its inside volume. Generally, there are four steps for a cage-based deformation.

(1) Automatically or manually create a cage to enclose an object to be deformed;

(2) Bind an object with its bounding cage (cage vertices). Suppose in a cage P, there is a point η whose location is a function of 3D positions 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 I_V be a set of cage vertices, then step (2) can be summarized as

$$\eta = F(\eta; P) = \sum_{i \in I_V} \phi_i(\eta) v_i \tag{2.6}$$

where $\phi_i(\eta)$ is the weight for representing the deformation influence and often referred

as "coordinates". Equation (2.6) is an implicit equation where ϕ is unknown with the given η and v_i.

(3) Deform a cage in order to deform an enclosed object;

(4) Interpolate the new object in response to the deformed cage. The deformation defined by a deformed cage P' can be formalized as

$$\eta' = F(\eta; P') = \sum_{i \in I_V} \phi_i(\eta) v_i' \tag{2.7}$$

where η, $\phi_i(\eta)$ and v_i' are known and η' is unknown. Equation (2.7) is an explicit function which can be used to interpolate its deformed object inside a cage.

There are some properties for the cage-based deformation, which makes it more attractive compared to other deformation methods.

(1). ***Deformation Domain*** This is the space region influenced by the cage. The deformation is well defined inside the bounding cage. Therefore, we constrain an object to be totally inside the undeformed cage. If we only deform a part of an object, the deformation portion of an object will remain inside the deformed cage.

(2). ***Smoothness*** Coordinates(they are functions defined on cage vertices and are used to interpolate interior nodes with the deformed cage vertices as inputs) can be managed as a function and smoothness at first derivatives (with respect to xyz) should be guaranteed.

(3). ***Affine Invariance*** Since an affine interpolation is used, it should satisfy that $\varphi(v) = 1(\sum_{i=1}^{n} \lambda_i(v) = 1)$ for all points v over the entire domain. This means that the deformed mesh will be unaffected by rigid body transformation of all the cage vertices, that is, it is orientation insensitive.

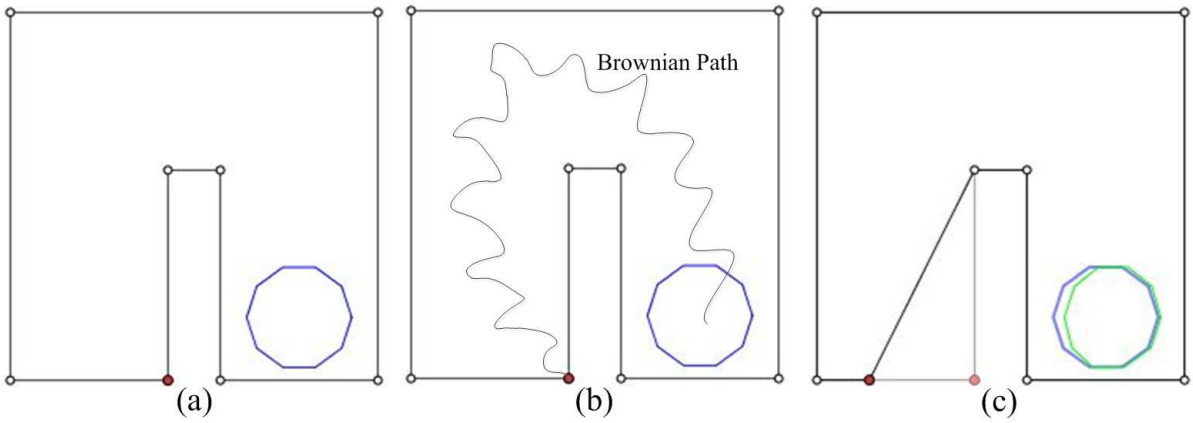

Figure 2.6: Properties of local deformation and positiveness[4]: (a)undeformed geometry; (b)influence of cage vertex on interior circle based on the Brownian path; (c)deformed geometry

(4). ***Local Deformation*** Cage vertex influence is restricted to its neighbourhood only at local region. This means that the cage vertex will only influence interior nodes based on distances from Brownian path (see Fig.2.6(b)) corresponding to *harmonic interpolation*(not straight-line distances). A cage vertex must cut off the influence of other cage vertices. This means that they are independent. For example, in Fig.2.6, a circle is invisible to the left part and should not deform if the left part is deformed. On the contrary, the circle is deformed by *Mean Value Coordinates* (***MVC***)[26] to the right when the left part is deformed to the left (see Fig.2.6(c)) because of negative ***MVC*** coordinates. In one word, an arbitrary interior node is affected by all the cage vertices on the bounding surfaces while most of cage vertices have almost zero influences on its location and only those cage vertices in its neighbouring regions have strong influences on its location.

(5). ***Conformality*** The deformation needs to have conformality(such as preserving surface details).

(6). ***Positiveness*** The coordinates should be non-negative over the entire domain. This guarantees that an object deforms in the same direction as its bounding cage's movement. In Fig.2.4(e) where there is the outward deformation from a sphere, a sphere becomes sinked if coordinates are negative and protruded if coordinates are non-negative. In Fig.2.6, the circle deforms by ***MVC***[26] to the right because coordinates are negative.

2.3.2.2 Binding methods

For the binding methods in the cage-based method, there are *Mean Value Coordinates*[26], *Harmonic Coordinates*[22], *Green Coordinates*[48] and *Radial Basis Function*[83].

Floater et al.[26] derived a generalization of barycentric coordinates which allowed an interior vertex in a planar triangulation or a tetrahedron to be expressed as a convex combination of its neighbouring vertices. The coordinates were motivated by the *Mean Value Theorem* for *harmonic functions* and could be used to simplify and improve methods for parameterization and morphing. They could be generalized to any convex polyhedra or any planar polygons. In order to accommodate for 3D cases, Floater et al.[27] generalized the barycentric coordinates to the convex polyhedra and kernels of star-shaped polyhedra. Ju et al.[90] generalized ***MVC*** from closed 2D polygons to closed triangular meshes. Given such a mesh $\mathbf{M}$, it was shown that these coordinates were continuous everywhere and smooth on the interior of $\mathbf{M}$. The coordinates were linear on the triangles of $\mathbf{M}$ and could reproduce linear functions on the interior of $\mathbf{M}$. However, ***MVC*** are based on *Euclidean*(straight line) distances between points of control polygon or polyhedral and points of the object. Therefore, the movement of points in one area might influence the movement of points in another area because ***MVC*** are not zero. On the contrary, an object moves in the opposite direction of its cage's movement because of negative ***MVC***,

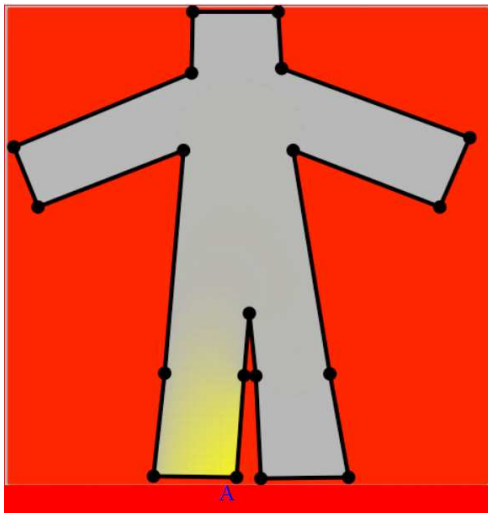

Figure 2.7: An example of influence of cage vertices on the interior nodes by Harmonic Coordinates[33]

which is contrary to users' requirements.

Meyer et al.[22, 33] presented a new form of d dimensional generalized barycentric coordinates called *Harmonic Coordinates* (***HC***). The new coordinates were defined as solutions to *Laplace's Equation* subject to carefully chosen boundary conditions (usually *Dirichlet* boundary conditions). ***HC*** possesses non-negativity and interior locality that make them more attractive than ***MVC*** as for defining two and three dimensional deformation. An appealing example which can explain why ***HC*** succeeds to place interior inside volumes is shown in Fig.2.7[22]. When the cage vertex A is set to 1, its influence regions are only restricted to the neighbouring area of A. The influence of the cage vertex A decreases with respect to the visible distances, which is denoted by the decreasing intensities of yellow color. The right leg which is blocked by the cage mesh (line segments in 2D and face meshes in 3D) is not affected at all where the gray color means 0, namely, almost zero influences. Therefore, ***HC*** can produce good deformation, which can be used to interpolate interior nodes with good mesh quality inside volumes during sweeping.

Lipman et al.[48] introduced *Green Coordinates* (***GC***) for closest polyhedral cages

which respected both cage vertices' position and cage faces' orientation. It was shown that ***GC*** lead to space deformation with a shape-preserving property. In particular, they induce conformal mapping in 2D, and extend naturally to quasi-conformal mappings in 3D. However, in order to preserve the shape, object points may come outside of cages if its cage deformation is very large, which means that if the bounding surfaces are used as cage points, interior nodes will come out of the bounding surfaces.

A *Radial Basis Function* (***RBF***)[49] is a real-valued function whose value only depends on distances from some point c(called a center), so that $\phi(\mathbf{X}, \mathbf{c}) = \phi(\parallel \mathbf{X} - \mathbf{c} \parallel)$. Any function ϕ that satisfies the property $\phi(\mathbf{X}) = \phi(\parallel \mathbf{X} \parallel)$ is a radial function. A list of common ***RBF*** are shown in Table.2.1 .

Table 2.1: Common radial basis functions[83]

Radial Basis Functions	$\phi(r)$
Spline Type (R_n)	$\|r\|^n, n$ odd
Thin Plate Spline (TPS_n)	$\|r\|^n log\|r\|, n$ even
Multiquadric (MQ)	$\sqrt{1+r^2}$
Inverse Multiquadric (IMQ)	$\frac{1}{\sqrt{1+r^2}}$
Gaussian (GS)	e^{-r^2}

Sieger et al.[83] used triharmonic ***RBF*** for high quality mesh morphing. By relying on triharmonic ***RBF***, the shape deformation distortions were minimized and thereby the shape quality was implicitly preserved. The details are shown in Appendix A.5. However, since ***RBF*** is based the Euclidean distances between points, it can cause unexpected movement of interior nodes. For instance, suppose there is an example as Fig.2.8 where a blue circle is very close to the bounding cage, the cage vertex with solid dot moves to the left. Since the blue circle is blocked by the cage mesh, it should almost keep fixed ideally while as a matter fact, the blue circle is deformed into the red ellipse which comes out of the cage due to the short Euclidean distance between the cage vertex with red solid dot

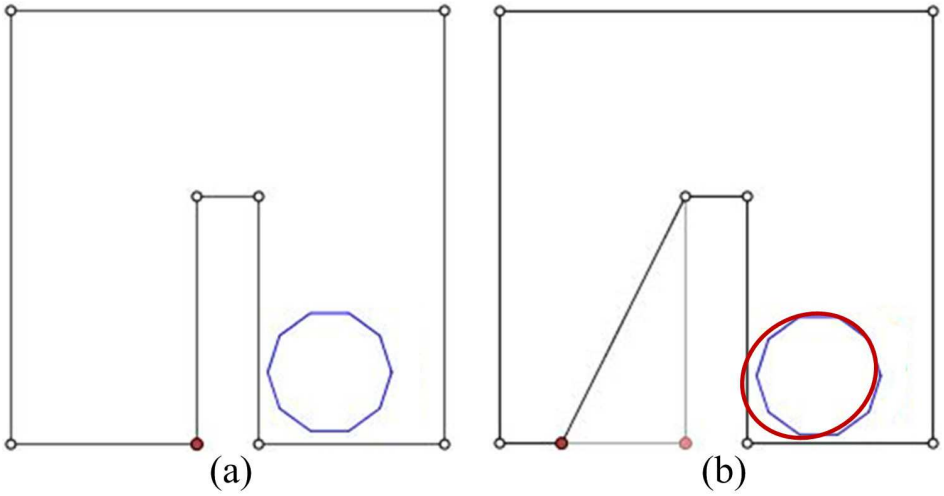

Figure 2.8: An example of ***RBF*** deformation: (a)undeformed model; (b)deformed model where the red ellipse is deformed from blue circle

and blue circle. This may be fine in the computer graphics area while it is unacceptable for locating interior nodes inside volumes by sweeping.

2.3.3 Mapping/SubMapping

The *Submapping* is a generalized *Mapping*, which is used to generate structured quadrilateral meshes in 2D and structured hexahedral meshes in 3D for a given geometry. During sweeping, *Mapping/Submapping* is used to generate structured quadrilateral meshes on the *linking* surfaces. The basic and traditional idea of *Submapping* method is to divide a geometry into patches logically equivalent to a four-side patch and then mesh each patch separately preserving the mesh compatibility via a linear integer problem. The *Submapping* method consists of four steps: vertex classification, edge parameterization($-i, +i, -j$ and $+j$), edge discretization and interior node interpolation. An example of four steps are described in Fig.2.9. Of those steps, the most challenging one is corner assignment for *Submapping*. In Section 2.3.3.1 and Section 2.3.3.2, the introduction and literature reviews

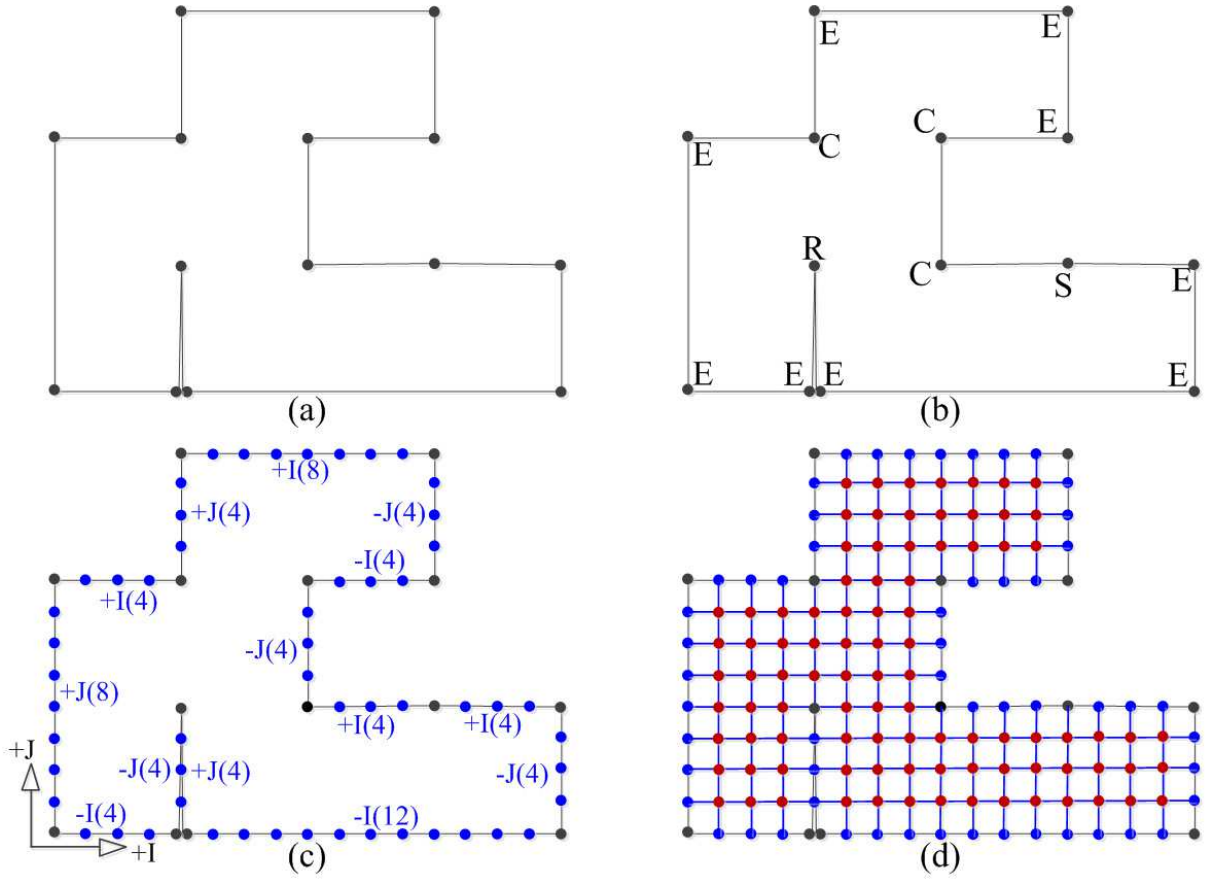

Figure 2.9: Pipeline for generating structured grids by *Submapping*: (a)geometry; (b)vertex classification; (c)edge classification and interval assignment on edges; (d)interior node interpolation and structured quadrilateral mesh generation.

for surface vertex classification by *Submapping* are given.

2.3.3.1 Surface vertex definition by *Submapping*

The surface vertex type during vertex classification is defined as the classification of the topology of a vertex bounding a structured all-quadrilateral surface mesh by the number of quadrilaterals sharing the vertex[113]. There are four surface vertex types possible in a *mapped* or *submapped* mesh[114], that is, ***END***, ***SIDE***, ***CORNER*** and ***REVERSAL***.

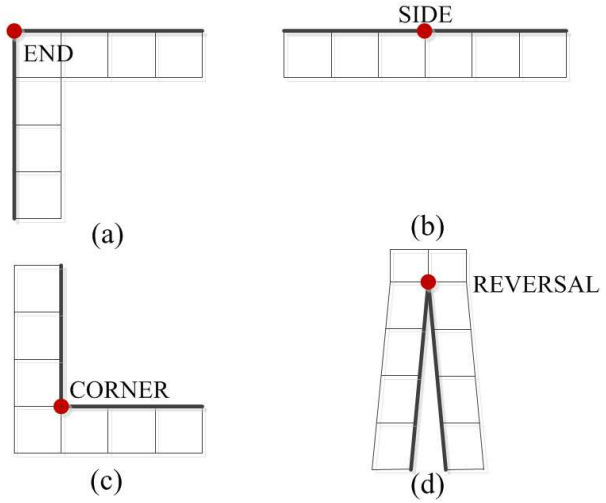

Figure 2.10: Surface vertex types by *Submapping*: (a)an ***END*** vertex; (b)a ***SIDE*** vertex; (c)a ***CORNER*** vertex; (d)a ***REVERSAL*** vertex

The number of quadrilateral elements sharing the vertex is one, two, three and four, respectively. The mesh topology of four surface vertex types is shown in Fig.2.10. For convenience, vertex types are assigned with integer values which are used to evaluate the submappable problem. They are assigned by $+1$,0,-1 and -2, respectively. These integers can also be interpreted as the ideal angle for each corner type, with the formula $(1 - \frac{VtxType}{2})\pi$. It is easily verified that vertex types for vertices bounding a *mapped* or *submapped* simply-connected surface sum to four [73, 114].

Therefore, in 2D, based on surface vertex types, a *Mapped* mesh is redefined as a structured mesh whose boundary consists of exactly four ***END***-type vertices or nodes, and any remaining boundary vertices or nodes are of type ***SIDE***. The boundary nodes and edges between a pair of non-***SIDE*** vertices are parts of a "side" of the map. Since a *mapped* mesh is constrained by four ***END***-type vertices, it also contains exactly four logical sides. One side can be made up of one or more geometric edges. In 2D, based on the surface vertex types, a *submapped* mesh can also be redefined as a structured mesh whose boundary vertices and nodes are each type ***END***, ***SIDE***, ***CORNER*** and ***REVERSAL***,

such that the sum of vertex types is $4 - 4 * g$ where g is the number of holes on a surface (the detail is described in Section 6.1 of Chapter 6). It has more capabilities than *Mapping* in that it admits surfaces with more than four logical sides. Note that determination of vertex types is a geometric operation local to vertices on the surfaces. The vertex type on a surface uniquely determines the mesh topology sharing that vertex and therefore can be used to determine the topology of the overall mesh. There is also an analogous property of edge types in volume meshes which is relevant for auto sweeping detection and idealized cage building(see Section 4.2.1 of Chapter 4), even though it is not directly relevant in this dissertation.

2.3.3.2 Corner classification

Whiteley et al.[114] presented a heuristic method for the structured quadrilateral mesh generation called *Submapping*. First, surface vertices are classified purely based on angles. This method relies on the condition that every angle between two consecutive edges on a surface is approximately an integer multiple of 0.5π, which is used to classify vertices and place corners on vertices successfully. However, it is tedious to require users' interaction to adjust surface vertex types. In one word, its main drawback is that it totally relies on geometric surface internal angles to automatically determine corners for *Submapping* and subsequent fixups.

The main limitation of implementations of *SubMapping* algorithms is the heuristic corner identification and manual fixup. Ruiz-Girones et al.[73] presented an algorithm to improve the vertex classification during *Submapping*. In order to overcome drawbacks of the heuristic *Submapping* algorithm, ***LP*** is a tool that is used in their methods to optimize the initial classification of surface vertices in such a way that necessary conditions are satisfied. The objective is to minimize the differences between new vertex classification

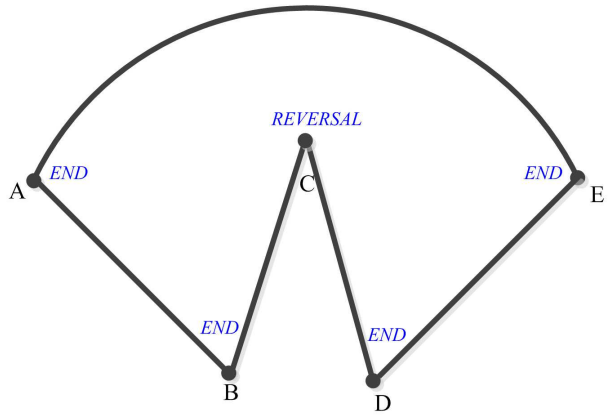

Figure 2.11: An un-submappable example by Ruiz's method

and heuristic vertex classification purely based on angles. In order to efficiently mesh geometries with holes, a new objective function that better distributed the number of intervals on edges has been proposed in the references[1, 52, 72, 82]. However, some heuristic constraints are added into ***LP***, which makes some surfaces un-*submappable* in some cases. For example, in Fig.2.11, it is impossible to convert the vertex type at the vertex C from the type-***REVERSAL*** to the type-***SIDE*** because of the heuristic variation constraint: the absolute value of vertex type variation is restricted to be one.

2.3.3.3 Summary

The structured meshes on the *linking* surfaces are used to constrain the interior node placement inside volumes where *Submapping* is widely used. One of the most difficult tasks is to classify the surface vertices, which is usually calculated based on angles. It fails in one way or another to satisfy the basic *Submapping* constraint (sum of vertex types is equal to four) since surface vertex angles are not exactly integers of 0.5π. Users have to interact and manually adjust surface vertex types. In order to improve the *Submapping* automation, ***LP*** is a good tool to correct invalid surface vertex classification. However, current existing methods formulate a ***LP*** by adding some heuristic constraints, which

makes some surfaces *unsubmappable*.

2.3.4 Sweepability assessment

The sweepability assessment algorithm is useful for sweeping in that it can tell users whether a solid is sweepable or not before meshing and thus extra computational costs can be avoided. White et al.[113] addressed some sweepability problems for auto sweeping detection based on edge types between *source/target* surfaces and *linking* surfaces and ij parameter traversing on the *linking* surfaces.

It is stated in the reference[113] that a volume is sweepable only if: (a)the volume has one or more non-intersecting chains; (b)for each set of contiguous source/target surfaces, traversing over all ***END***-type or ***CORNER***-type edges bounding the set results in traversing in the same global ij parameter and direction on the *linking* surfaces; (b)For each set of contiguous source/target surfaces bounded by both ***END***-type and ***CORNER***-type edges, traversing over any ***END***-type edge bounding the set results in traversing in the same global ij parameter and opposite direction on the *linking* surfaces as resulting from traversing over any ***CORNER***-type edge bounding the set.

However, it is stated in the reference[113] that three conditions described above are necessary but not sufficient. What is more, it fails for the radial sweeping where there are no *linking* surfaces and chains connecting the *source* and *target* surfaces. A typical failed example is shown in Fig.2.12 where all three conditions as described in the previous paragraph are satisfied while the geometry is unsweepable.

During a sweepability problem, the key problem is to determine whether there exists mappings between the *source* surface group and *target* surface group so that volume meshes can be generated by sweeping mesh faces from one group to the other group. In order to evaluate the sweepability problem more effectively, the homeomorphism theory[28] can be

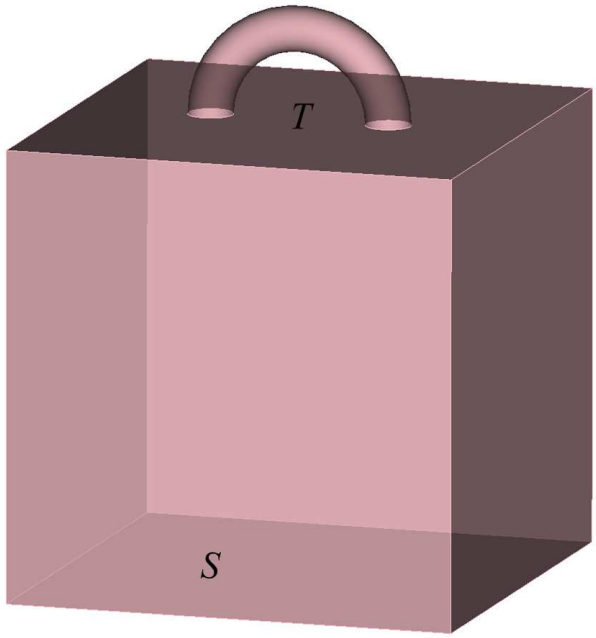

Figure 2.12: A typical failed example of sweepability assessment by White's method[113]

used since its purpose is to determine whether there are mappings between two shapes with preserving all the topological properties of a given space.

2.3.5 Homeomorphism

Homeomorphism is very useful for sweeping in that it can determine whether there exists the continuous maps between two different shapes or not from the topological point view. That can be used to determine whether sets of *source* and *target* surfaces are *mappable* or not.

In the mathematical field of topology, homeomorphism are the isomorphisms in the category of topological spaces, that is, they are the mappings that preserve all the topological properties of a given space. Two spaces with a homeomorphism between them are called homeomorphic and they are the same from a topological viewpoint. Therefore, some terms[28] are defined with respect to relations between the topological spaces:

Definition 7.1. A homeomorphism is a function $f : S \to T$ between two topological space S and T that

- is a continuous bijection, and
- has a continuous inverse function f^{-1}

Definition 7.2. Two topological spaces S and T are said to be homeomorphic if there are continuous map $f : S \to T$ and $g : T \to S$ such that

$$f \circ g = I_Y \qquad \text{and} \qquad g \circ f = I_X$$

Moreoever, the maps f and g are homeomorphisms and are inverses of each other, so we may write f^{-1} in place of g and g^{-1} in place of f. I_X and I_Y denote the identity maps. Two examples are described in Fig.2.13 where two topological spaces in Fig.2.13(a) and Fig.2.13(b) are homeomorphic while two topological spaces in Fig.2.13(c) and Fig.2.13(d) are not homeomorphic.

Translations, *reflections*, *rotations*, *enlargements*, *contractions* and *twisting* and *bending* shown in Fig.2.14 are homeomorphisms (in terms of the usual topology of R^3). *Projections* are not, because they do not have inverses. *Tearing* and *joining*(or *gluing*) have inverses but, because *tearing* is not continuous, neither *tearing* nor *joining* are homeomorphisms. *Tearing* is not a continuous operation. Points that are close to together before *tearing* are no longer close together if two pieces are pulled apart after tearing. The inverse of *tearing* is joining and in this direction it is continuous. But for many purposes it requires functions to be continuous in both directions. In other words, the homeomorphism requires a function that has an inverse, namely, one that is ***1-1*** and onto (bijective).

Figure 2.15 shows two homeomorphic examples between surfaces: a circle in Fig.2.15(a) is homeomorphic to a square in Fig.2.15(b) since there exists continuous mapping and inverse mapping between them. The random disk in Fig.2.15(c) is not homeomorphic to

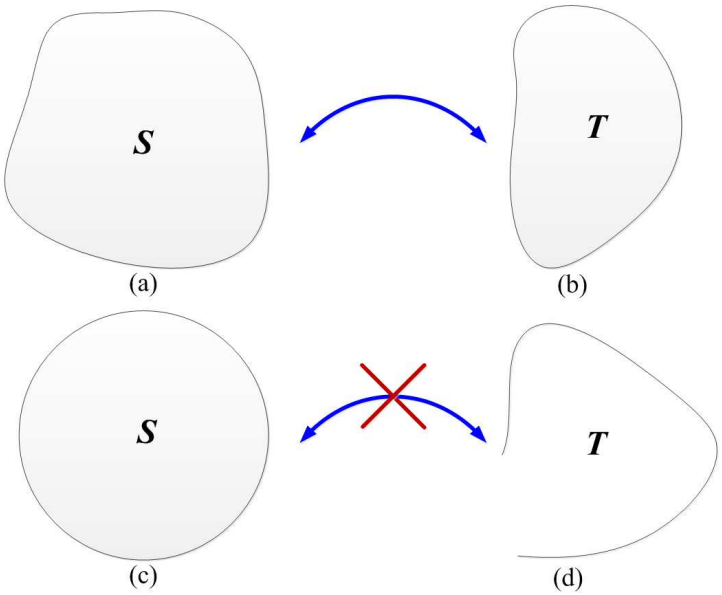

Figure 2.13: Two homeomorphic examples: (a) and (b) are homeomorphic; (c) and (d) are not homeomorphic

a circular disk in Fig.2.15(d) since there does not exist continuous mapping or inverse mapping between them.

2.4 Summary

Current existing sweep implementations fail in a variety of ways. In order to guarantee the robustness and mesh quality by sweeping, several things have to be done such as morphing *source* meshes to the *target* meshes, placing interior nodes within volumes, assigning corners by *submapping* for *linking* surfaces and imprinting *source/target* surfaces to ensure comparability in the swept mesh. These steps are illustrated in Fig.2.16.

Knupp's affine transformation [39] is fast while it fails for concave or multiply-connected surfaces; *Roca's Least-Square Approximation* [65, 66, 69] is fast and can work for curved surfaces while it fails for concave or multiply-connected surfaces; *BMSweep* [86] is mod-

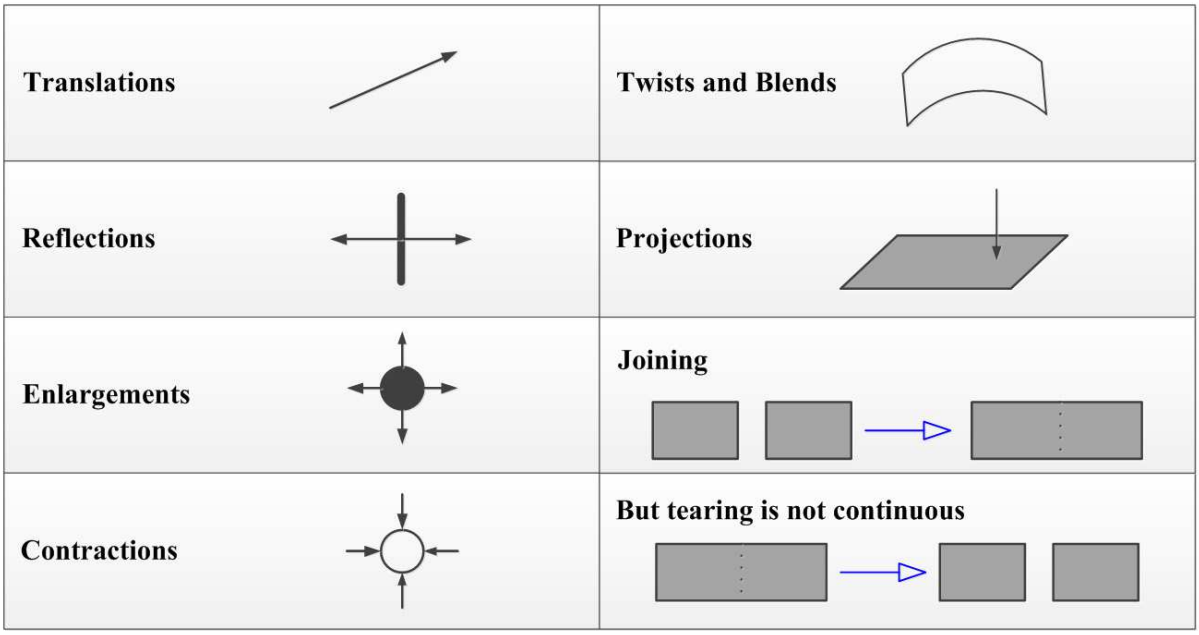

Figure 2.14: Geometric examples of continuous functions

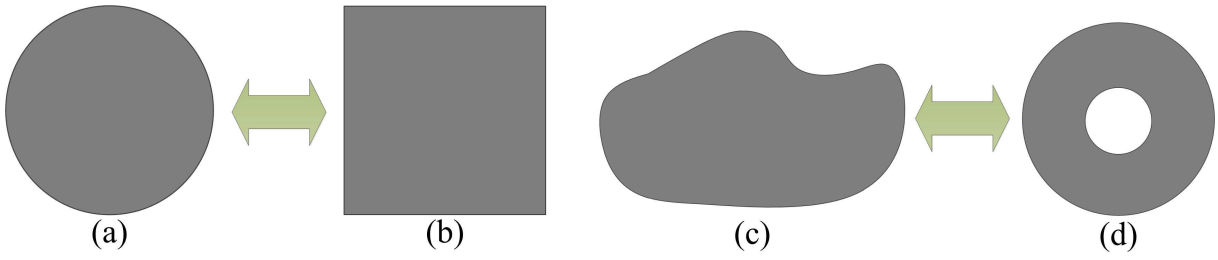

Figure 2.15: Examples of homeomorphism between two surfaces: (a)a circle; (b)a square; (c)a random disk; (d)a circular disk

erately fast and can work for concave or multiply-connected surfaces while it fails for volumes with twisted features and curved surfaces. Fortunately, there exists morphing techniques which can be used to map from one shape to another shape, which is useful for surface mesh mapping between the *source* and *target* surfaces. Therefore, *harmonic mapping* of surface faceting instead of only using boundary nodes is used to embed *source* and *target* surface faceting over the common domain and map the surface meshes between the *source* and *target* surface, which is described in Chapter 3. Of course, it works for convex domain as well.

Current existing methods including *Knupp's affine transformation* [39], *Roca's Least-Square Approximation* [65, 66, 69] and *BMSweep* [86] locate interior nodes inside volumes by a layer-by-layer approach. However, interior node placement is affected not only by one layer where interior nodes are, but also by the adjacent layers on the *linking* surfaces. They fail in one way or another when there is a volume with complicated internal structure. The cage-based method interpolates interior objects by using the bounding surfaces as constraints. Therefore, in Chapter 4, the cage-based method is used to place interior nodes inside volumes by sweeping where all the *source*, *target* and *linking* surface meshes are used as constraints.

Volume decomposition including physical and logical decomposition is a popular method for multi-sweeping problems by current existing methods including White's method [109], Lai's Method [41, 51], Cooper Tool [86], Multi-Axis Cooper Tool [86], *PMOST*[78], *CCSweep* [112] and Ruiz-Girones's Method [70, 71]. The most advantage for those methods is low computational cost. However, there are some serious disadvantages: it is difficult to generate reasonable internal surfaces used for volume decomposition when there is a complicated solid. Generally, the *linking* surface meshes are used to guide the volume decomposition, which creates extra constraints for multi-sweeping problems and make

a volume unsweepable due to constraints from *linking* surface meshes. What is more, interior nodes are placed separately in each decomposed ***1-1*** sweeping portion and poor mesh quality is created. Also, poor ***1-1*** sweeping schemes for each divided sweeping portion are used. In summary, it is not necessary to use the volume decomposition for multi-sweeping problems: if solids are sweepable by multi-sweeping, volume decomposition is not needed at all; volume decomposition is only needed when solids are not sweepable and their topologies need to be broken such that the decomposed pieces are sweepable by multi-sweeping. However, surface partition is indeed needed for multi-sweeping problems such that edges on the *source* and *target* surfaces can be resolved: which *source* surface patch will be swept onto a specific *target* surface patch. These methods also have proven to not be very robust, and therefore are not used in current production meshing tools. Hence, in Chapter 5, the imprinting algorithm based on cage-based morphing is proposed to be used for solving multi-sweeping problems without volume decomposition but with *source*/*target* surface partition.

Mapping/*Submapping* is used to generate structured mesh on the *linking* surfaces. One of the most difficult task is the corner classification for *Submapping*. Whiteley's method [114] classifies surface vertices purely based on angles. This method heavily relies on an assumption that the surface internal angle is an integer mulple of 0.5π. Users have to manually correct the surface vertex types if an invalid vertex classification is obtained. In order to improve automation, Ruiz-Girones's method [73] uses ***LP*** to correct the invalid vertex classification produced by Whiteley's method [114]. However, a heuristic constraint is added, which makes some geometries *unsubmappable*. We solve this problem by optimizing vertex types using ***LP*** with conditions that the *Submapping* constraint is satisfied and some constraints of impossible surface vertex type variation are added. We also formulate templates to deal with a geometry with the fillet or chamfer features by

relaxing or changing the angle-based constraints. The details are described in Chapter 6.

Current existing algorithms use some criteria to assess the sweepability problem while it is not sufficient. Thus they sometimes determine whether a solid is sweepable or not by attempting to actually generate the mesh. However, that is too expensive. Therefore, in order to better assess the sweepability problems, in Chapter 7, we address three constraints, that is, topological constraint, geometrical constraints and prescribed constraint. The prescribed constraint is used to identify the specified error, which is mainly from user's specified matching. Those three constraints are sufficient to evaluate the sweepability problem so that expensive computational costs can be saved.

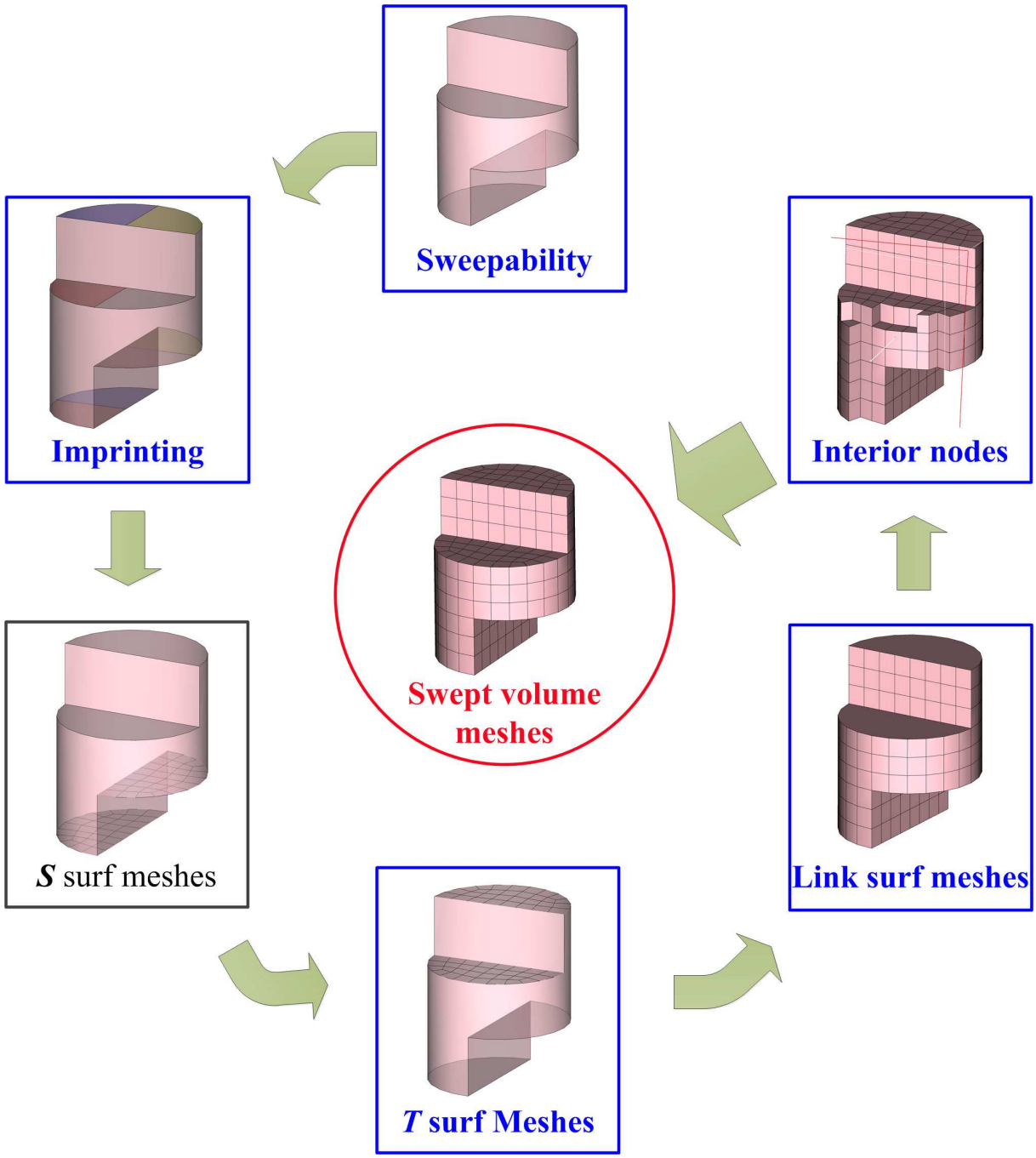

Figure 2.16: Pipeline of our multi-sweeping approach

3 TARGET SURFACE MESH GENERATION BASED ON HARMONIC MAPPING

3.1 Problem Statement

Current existing ***1-1*** sweepers [39, 66, 69] generate the *target* surface mesh by mapping the *source* surface mesh based on *affine transformation*. In computational geometry, an *affine transformation* (or *affine mapping*) is a function between affine spaces which preserve points, straight lines and planes. Examples of *affine transformations* include translation, scaling, rotation and compositions of them in any combination. If $\mathbf{X}$ and $\mathbf{Y}$ are affine spaces, then every *affine transformation* $f : \mathbf{X} \rightarrow \mathbf{Y}$ is of the form $f(\mathbf{X}) = \mathbf{M}\mathbf{X} + \mathbf{b}$ where $\mathbf{M}$ is a linear transformation on $\mathbf{X}$ and $\mathbf{b}$ is a vector in $\mathbf{Y}$.

Current existing ***1-1*** sweepers use all the boundary mesh nodes on the *source* and *target* surface to compute the transformation matrix $\mathbf{M}$ and vector $\mathbf{b}$. However, problems arise that the *target* surface is not a pure *affine transformation* of the *source* surface when there is a geometry with multiply-connected or concave *source* and *target* surfaces such as Fig.3.1 and Fig.3.2(see discussions in Section 2.2.1 of Chapter 2).

Sweeping requires that the *source* surface mesh should be mapped onto the *target* surface with good mesh quality preserved and the same mesh connectivity is kept between the *source* and *target* surface. This is due to the inherent charactistics of sweeping that every mesh face on the *source* surface must match exactly a specific mesh face on the *target* surface. In particular, the surface mesh mapping algorithm during sweeping should work between two concave or multiply-connected surfaces.

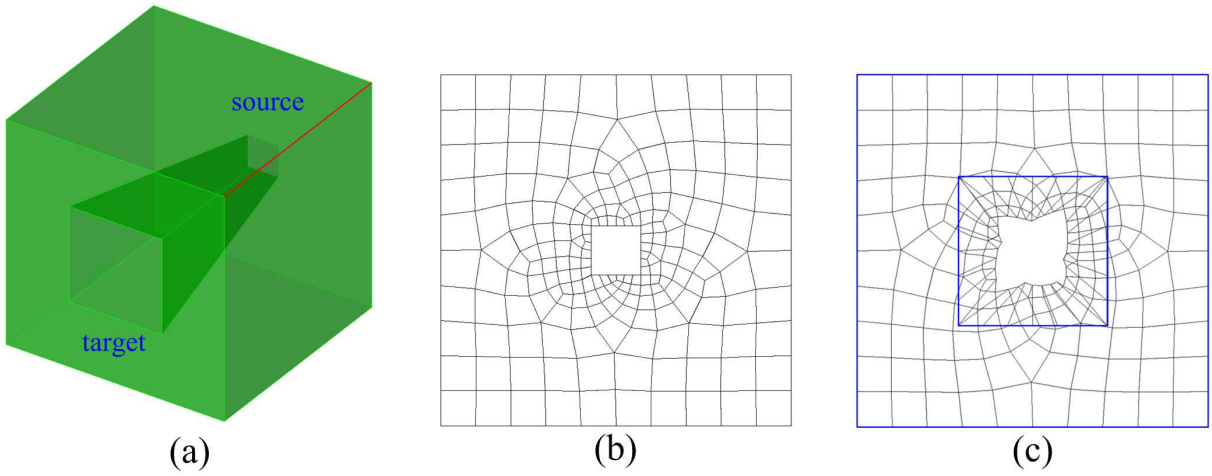

Figure 3.1: A failed sweeping example for a geometry with multiply-connected *source* and *target* surfaces: (a)a geometry model; (2)the *source* surface mesh; (3)the *target* surface mesh projected from the *source* surface based on *affine transformation*

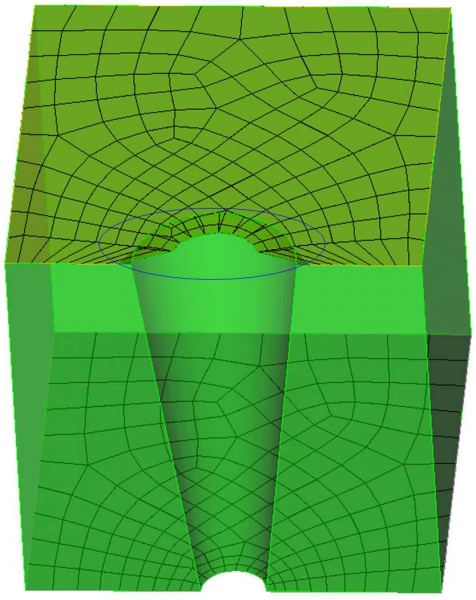

Figure 3.2: A failed sweeping example for a geometry with concave *source* and *target* surfaces

3.2 Overview

In order to solve problems mentioned in Section 3.1, an application of *harmonic* function on projecting a *source* surface mesh onto its *target* surface is described in this section. In computer graphics, surfaces are represented with graphical triangle meshes. For ***1-1*** sweeping, our general approach is to map the *source* and *target* faceting triangle meshes first onto separate unit disks, then associate the disks together. Afterwards, the *source* surface mesh can be located in its graphical triangle meshes from that surface. Because the unit disks from the *source* and *target* surface are associated together, any node from the *source* surface meshes can be located in the faceting triangle meshes of *target* surface on the common unit disk as well through a series of transformations. Therefore, the *source* surface mesh can be mapped onto the *target* surface in the physical space.

The roadmap for mapping the *source* surface mesh onto the *target* surface can be illustrated with an example such as Fig.3.3 where the *source* and *target* surface are multiply-connected with a hole. A *harmonic mapping* algorithm maps the graphical triangle meshes F_1 and F_2 onto 2D unit disks, H_1 and H_2, respectively. In order to establish the correspondence between H_1 and H_2, a new 2D common unit disk H_c (which is replaced by H_2 later) needs to be created by adjusting point locations from faceting triangle meshes on boundaries and combining both H_1 and H_2. For the sake of simplicity, we keep 2D unit disk H_2 fixed and map F_1 to H_1' on the H_2 domain, instead of mapping both H_1 and H_2 to H_c. Without creating a new H_c and mapping from H_1 and H_2 to H_c, the boundary points on H_1' are adjusted in order to make the boundary points between H_1' and H_2 correspond. At this point, since both F_1 and F_2 are mapped onto the H_2 domain, the correspondence between F_1 and F_2 can be established as well. Finally, by embedding M_1 onto F_1, the *source* surface mesh M_1 can be mapped onto the H_2 domain and then back to the *target* surface F_2.

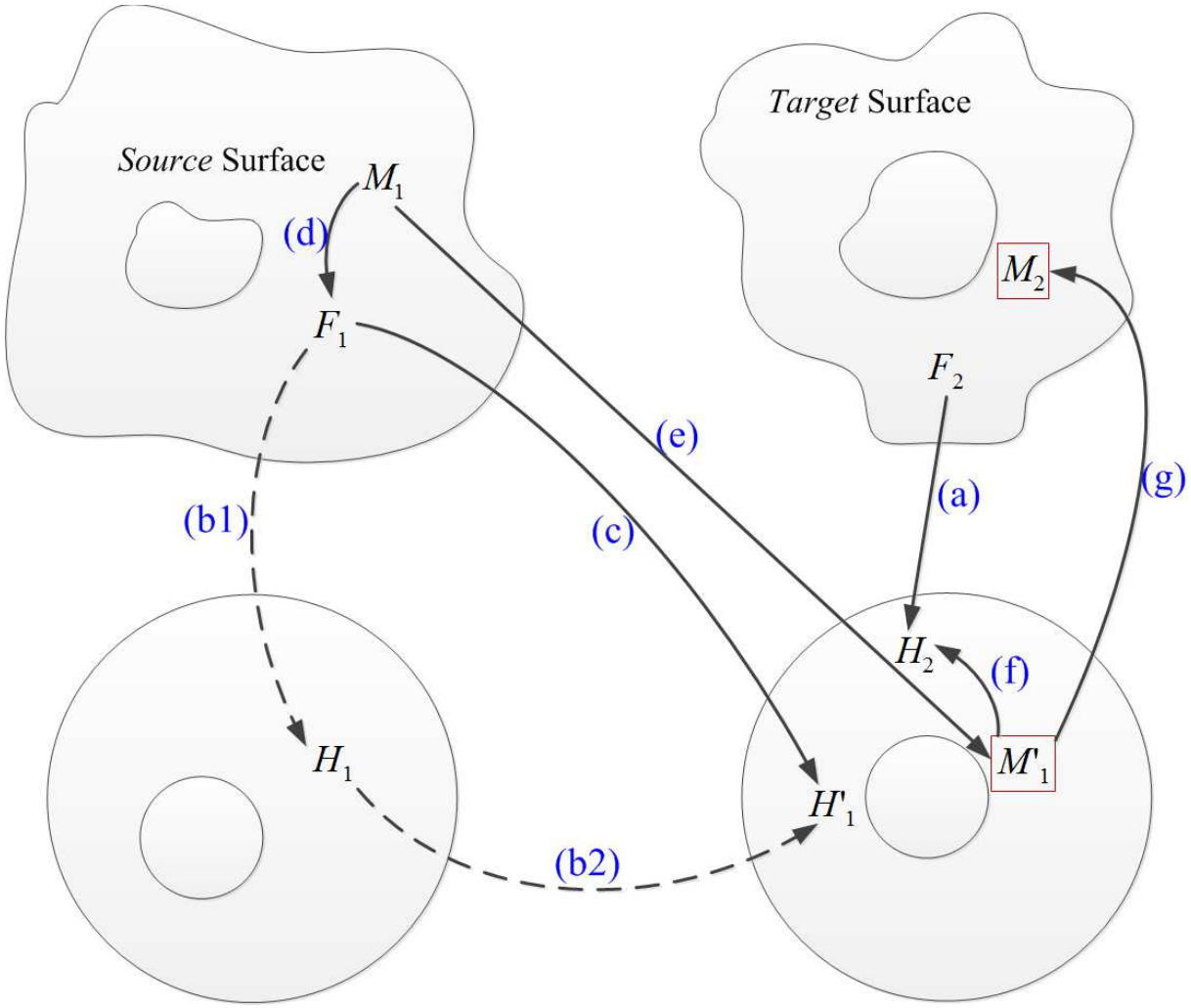

Figure 3.3: Road map for mapping surface meshes from a *source* surface M_1 to its *target* surface M_2: (a)F_2 is mapped onto H_2; (c)F_1 is mapped onto H'_1 on the H_2 domain. This is a simplication of both (b1) and (b2); (d)Embed M_1 into F_1; (e)Map M_1 onto $H'_1 \rightarrow M'_1$; (f)Embed M'_1 into H_2; (g)Map M'_1 onto $F_2 \rightarrow M_2$

3.3 Harmonic Mapping

Harmonic Map φ, $F \rightarrow H$ is mapping between two Riemannian manifolds. M and H are *harmonic* if the *Dirichlet* energy is minimized. *Harmonic Map* performs a mapping from a topological disk to a 2D unit disk. To construct the *source* and *target* surface's embeddings where the *source* and *target* surfaces are mapped onto 2D disks, the piecewise linear approximation method for mapping from F to H is used [21,28], which is established as follows: n vertices on the outmost boundary are distributed on the boundary of 2D disk. This is based on the edge lengths between two adjacent boundary nodes. When vertices on the outmost boundary of *source* surface have been distributed onto the unit disk, vertices on the outmost boundary of *target* surface should be fixed as well because they correspond with those of *source* surface and are located through the *linking* surfaces in ***1-1*** sweeping. For a multiply-connected surface, the mapping remains one-to-one even when considering holes in the domain[64]. Based on the Ref.[34, 118], if there is a genus zero surface S with multiple boundaries, and a *Riemannian metric* g, then there exists a *conformal map* $f : F \rightarrow H$, where H is a 2D unit disk with circular holes.

The *harmonic mapping* function is formulated and solved as follows: let $[v_i, v_j]$ be an interior edge on the graphical triangular meshes as Fig.3.4, connecting two triangle faces $[v_i, v_j, v_k]$ and $[v_i, v_j, v_l]$, the corner angle in $[v_i, v_j, v_k]$ against $[v_i, v_j]$ is θ_k^{ij}, the corner angle in $[v_i, v_j, v_l]$ against $[v_i, v_j]$ is θ_l^{ij}, the edge weight is defined as

$$\omega_{ij} = cot\theta_k^{ij} + cot\theta_l^{ij} \tag{3.1}$$

The discrete *harmonic* energy is defined as

$$E(f) = \sum_{[v_i,v_j]} \omega_{ij}(f(v_i) - f(v_j))^2 \tag{3.2}$$

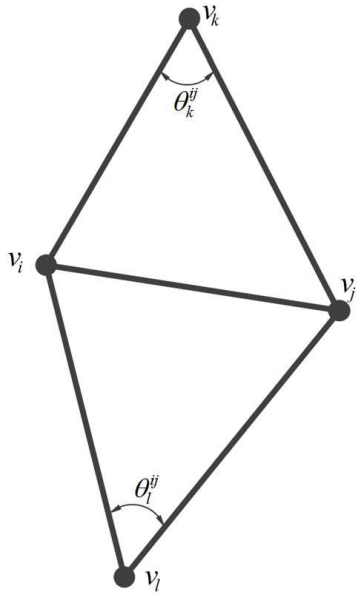

Figure 3.4: Triangle faceting for computing *harmonic map*

The discrete *harmonic* function is the critical point of the *harmonic* energy, which satisfies the following discrete *harmonic* 1-form.

$$\delta f(v_i) = \sum_{[v_i,v_j]\in E} \omega_{ij}(f(v_i) - f(v_j)) = 0, \ \forall v_i \in V \tag{3.3}$$

where V is a set of all the vertices. Let ξ and η be components in 2D unit disk $\mathbf{H}$.

$$\mathbf{V} = (v_{1\xi}, v_{1\eta}, v_{2\xi}, v_{2\eta}, \cdots, v_{N\xi}, v_{N\eta}) \tag{3.4}$$

where N is the total number of vertices and $\mathbf{E}_{harmonic}$ is a quadratic form with respect to $\mathbf{V}$. Therefore, $\mathbf{E}_{harmonic}$ can be rewritten as

$$\mathbf{E}_{harmonic} = \mathbf{V^T H V} \tag{3.5}$$

During the *harmonic mapping*, boundary nodes are fixed. Therefore, the variable $\mathbf{V}$ can

be divided into 2 parts: interior part $\mathbf{V}_i$ and boundary part $\mathbf{V}_b$. The constant matrix H will be divided accordingly.

$$E_{harmonic} = \begin{bmatrix} (\mathbf{V}^i)^T & (\mathbf{V}^b)^T \end{bmatrix} \begin{bmatrix} \mathbf{H}^{ii} & \mathbf{H}^{ib} \\ \mathbf{H}^{bi} & \mathbf{H}^{bb} \end{bmatrix} \begin{bmatrix} \mathbf{V}^i \\ \mathbf{V}^b \end{bmatrix} \tag{3.6}$$

Then the gradient of $E_{harmonic}$ (over the variable $\mathbf{V}^i$) is defined as

$$\bigtriangledown E_{harmonic} = 2\mathbf{H}^{ii}\mathbf{V}^i + 2\mathbf{H}^{ib}\mathbf{V}^b = 0 \tag{3.7}$$

The linear equation can be solved with direct method (large sparse linear equations)[21] or iterative method such as Preconditioned Conjugate Gradient Method[57]. The solutions are the ideal coordinates for vertices in the parametric domain H.

3.4 Multiply Connected Domains

In this section, the *harmonic mapping* of physical multiply-connected *source* and *target* surfaces onto the disk is described where interior hole boundaries are mapped onto the circular curves. This corresponds to the step (c) in Fig.3.3.

If a function is *harmonic* (that means it satisfies *Laplace's Equation* over a particular space) and transformed via a *conformal map* to another space, the transformation is also *harmonic*. If there is a genus zero surface with multiple holes, the generalized *Koebe*'s method is applied to compute the canonical *conformal mappings* [118] where the *harmonic mapping* is a particular *conformal mapping*. By keeping one hole and temporarily filling the remaining holes, the *conformal mapping* of a multiply connected domain is equivalent to compute the *conformal mapping* of a topological annulus, which is reduced to compute a pair of conjugated *harmonic* 1-forms. This can be computed recursively until all the

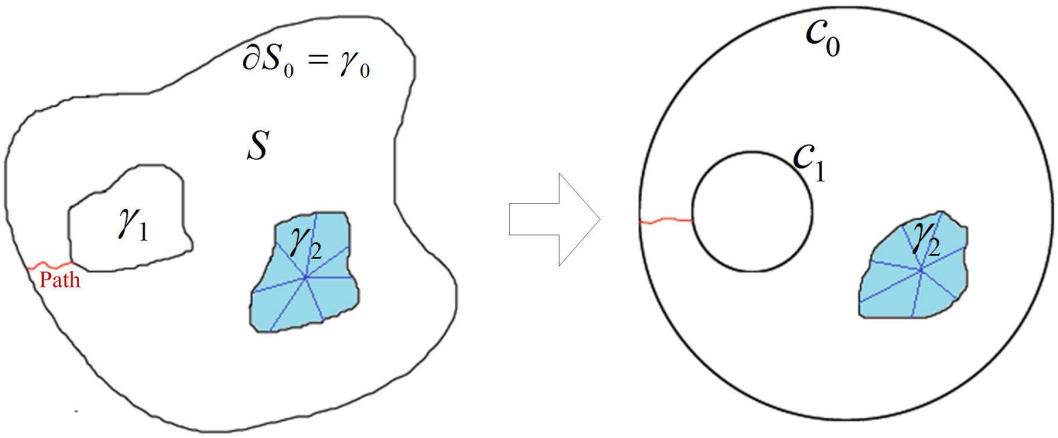

Figure 3.5: An example of conformal mapping of S to the canonical annulus ($\gamma_0 \to c_0$, $\gamma_1 \to c_1$)

holes are processed.

The method of *harmonic mapping* of multiply-connected surfaces works as follows: suppose there is a surface S with n holes γ_i, $i = 1, \cdots, n$ and boundaries D_i, $i = 1, \cdots, n$. The outmost boundary which bounds the surface S can be denoted as ∂S_0. The *conformal mapping* can be computed in the following steps [24, 118].

(1) Remove a hole from a surface S by computing a path which connects its boundary D_k and ∂S_0. Then the boundary of a hole becomes a part of ∂S_0. All other holes are temporarily filled by generating constrained triangle meshes for holes. An example is shown in Fig.3.5.

(2) Conformally map the surface S(annulus) to the canonical annulus, such that the boundary (D_k) of γ_k is mapped to a circle c_k.

$$\varphi_k : S - D_k \to \textit{unit disk}$$

such that $\varphi_k(\gamma_k) = c_k$

(3) Compute a *harmonic mapping* of D_k, with the boundary condition that D_k is mapped to c_k.

$$f_k : D_k \rightarrow a\ disk\ inside\ 2D\ unit\ disk,\ \Delta f_k = 0, f_k|_{\gamma_k} = c_k$$

(4) Update the whole mesh S

$$S \leftarrow \varphi_k(S - D_k) \cup f_k(D_k) \tag{3.8}$$

(5) Process the remaining holes $D_i(i = 1, \cdots, n)$ individually using the above steps by restoring one of the remaining filled holes to the real hole. Then the boundary of γ_k is mapped to a specific circular curve with its center and radii as (c_k,r_k).

(6) Repeat step 5 until it converges.

$$\sum_{k=1}^{3} |c_k^0 - c_k^1|^2 + |r_k^0 - r_k^1|^2 < \epsilon \tag{3.9}$$

where (c_k^0,r_k^0) and (c_k^1,r_k^1) are the center and radius of D_i of two consecutive iterations.

3.5 *Target* Surface Mesh Generation

In this section, the *source* F_1 and *target* faceting F_2 are mapped and associated together on the common domain (H_2 is fixed, kept constant and used as the common domain and F_1 is mapped to H_1' on the H_2 domain directly), which has the combined mesh connectivity from both F_1 and F_2. After establishing the correspondence between F_1 and F_2 onto the H_2 domain, any point from the *source* surface faceting F_1 corresponds with a specified position on the *target* surface faceting F_2. Since the mesh nodes from *source* surface mesh

M_1 can be located in their respective facetings F_1 and F_2, the net result is a mapping from M_1 onto F_2, this is, M_2.

3.5.1 Barycentric Coordinates

In order to better formulate the mapping from M_1 to M_2, two kinds of evaluation mapping based on barycentric coordinates are used in this work: forward evaluation mapping and reverse evaluation mapping. The forward evaluation mapping is defined as computing a real space point $\mathbf{P}$ inside a triangle based on an input triangle and barycentric coordinates. Suppose there is a triangle $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$ and barycentric coordinates $\{i_1, i_2, i_3\}$ of an interior point, then $\mathbf{P}$ can be interpolated as

$$\mathbf{P} = i_1\mathbf{v}_1 + i_2\mathbf{v}_2 + i_3\mathbf{v}_3 \tag{3.10}$$

The reverse evaluation mapping is defined as: when a set of triangles as well as a point $\mathbf{P}$ in real space are provided, a specific triangle where $\mathbf{P}$ is located needs to be found and barycentric coordinates of $\mathbf{P}$ with respect to that triangle needs to be computed. Suppose that a specific triangle $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$ is found from a set of triangles, the barycentric coordinates of $\mathbf{P}$ with respect to the triangle $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$ can be computed as

$$i_1\mathbf{v}_1 + i_2\mathbf{v}_2 + i_3\mathbf{v}_3 = \mathbf{P} \tag{3.11}$$

$$i_1 + i_2 + i_3 = 1 \tag{3.12}$$

$$i_1 \geq 0, i_2 \geq 0, i_3 \geq 0 \tag{3.13}$$

3.5.2 Interior Node Interpolation

To summarize, the *target* surface mesh generation mapped from a *source* surface mesh consists of the following steps where the *source* and *target* faceting are used to guide the surface mesh mapping.

(a) Map F_2 to H_2: map the *target* surface faceting F_2 onto H_2 by *harmonic mapping* described in Section 3.3 such that 3D surface faceting can be embedded onto a 2D disk. Points of surface faceting on the outmost boundary can be distributed as follows before *harmonic mapping*.

$$\xi_{P_i} = cos(2\pi * l_i/L) \tag{3.14}$$

$$\eta_{P_i} = sin(2\pi * l_i/L) \tag{3.15}$$

where L is the total length of the outmost boundary and l_i is the sum distance of line segments from point P_i to the starting point along the outmost boundary. This corresponds to the step (a) in Fig.3.3 of Section 3.2.

(c) Map F_1 to H_1' on the H_2 domain directly (this is a simplification of step (b1) and step (b2) in Fig.3.3).

(c1) Establish correspondences of boundary faceting points: since the *linking* surfaces connect the *source* and *target* surface, the boundary mesh nodes on the *source* surface boundaries can be mapped onto the *target* surface boundaries based on the global consistent parametric space of *linking* surfaces. That means every boundary mesh node from the *source* surface is associated with a specific boundary mesh node on the *target* surface. Since boundary points from surface faceting are located among the boundary mesh nodes, boundary points from the

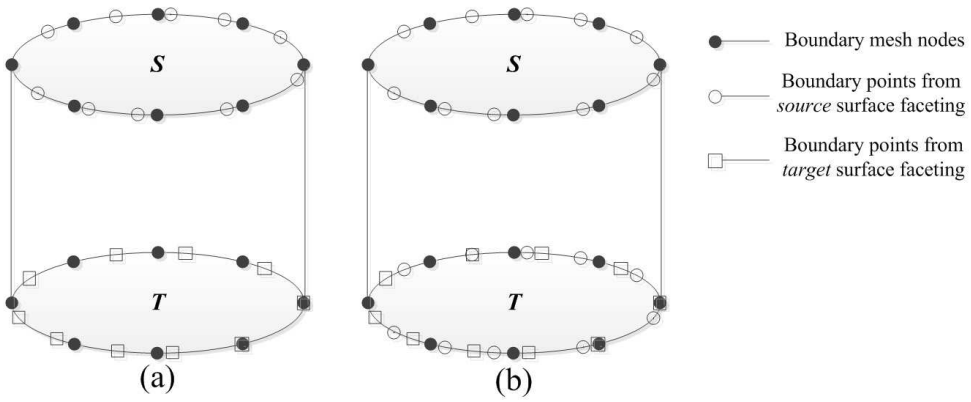

Figure 3.6: An example of establishing correspondences of boundary points from the *source* and *target* surface faceting: (a)mesh nodes and faceting points on the *source* and *target* boundaries; (b)Correspondences of boundary points from the *source* surface onto boundaries of *target* surface

source surface faceting can be located on their own positions on the boundaries of *target* surface. In this way, the correspondence of boundary points between the *source* and *target* surface faceting is made. Note that F_1 and F_2 may have different triangle mesh points and connectivity, it is not necessary to make all the points on boundaries overlap on the H_2 domain. As a matter of fact, they do not overlap in most of cases. However, they should be corresponded. In other words, those boundary points from F_1 should have fixed positions on the H_2 domain once H_2 is fixed because the *linking* surfaces guide the correspondence between F_1 and F_2. An example of correspondences of boundary points from the *source* and *target* surface is shown in Fig.3.6 where correspondences of boundary points from *source* surface faceting to the *target* surface faceting is made. Note that this corresponds to a part of step (c) in Fig.3.3 of Section 3.2.

(c2) Map interior points of F_1 to H_1' on the H_2 domain: since correspondences of boundary points between the *source* surface faceting F_1 and *target* surface faceting F_2 have been made, boundary points on the *source* surface faceting

F_1 can be embedded on boundaries of the disk H_2: every boundary point from *source* surface faceting F_1 is between two boundary nodes on M_1 and it can be located on H_2 based on its relative position with respect to two boundary nodes. After boundary points from F_1 are redistributed on the H_2 domain, interior points from the *source* faceting F_1 can be embedded on the H_2 domain by our approach described in Section 3.4. If F_1 or F_2 is a multiply-connected domain, our algorithm described in Section.3.4 for the multiply-connected domain should be used to map F_1 to H_1' where holes are processed recursively. This corresponds to the step (c) in Fig.3.3 of Section 3.2.

(d) Embed M_1 onto F_1 (see Fig.3.7(a)): since the *source* surface mesh M_1 and *source* surface faceting F_1 are both on the same surface, namely, sharing the common domain, every mesh node from M_1 is located inside a specific triangle from F_1. Therefore, barycentric coordinates can be used to represent mesh nodes of M_1 with F_1: the reverse evaluation mapping as Section 3.5.1 is used where a set of triangles F_1 as well as the mesh nodes from M_1 are provided. The output is a specific triangle $\{v_1^{F_1}, v_2^{F_1}, v_3^{F_1}\}$ and barycentric coordinates $\{i_1^{F_1}, i_2^{F_1}, i_3^{F_1}\}$ of a mesh node $n_m^{F_1}$. Note that this corresponds to the step (d) in Fig.3.3 of Section 3.2.

(e) Map M_1 into H_1' on the H_2 domain$\rightarrow M_1'$: since F_1 has already been embedded into H_1' on the H_2 domain, every mesh node from the *source* surface mesh M_1 can be embedded on H_1' as well by using the same barycentric coordinates. The mapping of F_1 onto H_1' results in that the triangle $\{v_1^{F_1}, v_2^{F_1}, v_3^{F_1}\}$ is changed into $\{v_1^{H_1'}, v_2^{H_1'}, v_3^{H_1'}\}$ where $v_1^{H_1'}$,$v_2^{H_1'}$ and $v_3^{H_1'}$ are 2D positions of three points of a *source* faceting triangle on H_1'. The forward evaluation mapping is used: inputs are $\{v_1^{H_1'}, v_2^{H_1'}, v_3^{H_1'}\}$ and $\{i_1^{H_1'}, i_2^{H_1'}, i_3^{H_1'}\}$ which is equal to $\{i_1^{F_1}, i_2^{F_1}, i_3^{F_1}\}$; outputs are mesh nodes $n_m^{H_1'}$ on the H_2 domain from M_1. This corresponds to the step (e) in Fig.3.3 of Section 3.2.

(f) Embed M_1' onto H_2 (see Fig.3.7(b)): since the *target* surface faceting F_2 and *source* surface mesh M_1 have already been embedded on the common domain H_2 and H_1', respectively, the *source* surface mesh M_1 can also be embedded onto H_2 through the common domain. This can be computed through the reverse evaluation mapping: inputs are a set of triangles H_2 and mesh nodes $n_m^{H_2}$ which are located at the same position as $n_m^{H_1'}$ in the step (e); outputs are $\{i_1^{H_2}, i_2^{H_2}, i_3^{H_2}\}$ of mesh nodes $n_m^{H_2}$ and the triangle $\{v_1^{H_2}, v_2^{H_2}, v_3^{H_2}\}$ where $n_m^{H_2}$ are located. This corresponds to the step (f) in Fig.3.3 of Section 3.2.

(g) Map M_1' to M_2: calculate 3D positions of mesh node M_1 on the *target* surface F_2. Since every mesh node from M_1' is represented with barycentric coordinates with respect to the faceting triangles from F_2 at H_2, M_1' can be mapped onto the physical *target* surface. By keeping the same barycentric coordinates and *target* faceting triangle connectivity, the forward evaluation mapping of the mesh node $n_m^{F_2}$ from H_2 to F_2 can be computed: inputs are $\{v_1^{F_2}, v_2^{F_2}, v_3^{F_2}\}$ and $\{i_1^{F_2}, i_2^{F_2}, i_3^{F_2}\}$ which is equal to $\{i_1^{H_2}, i_2^{H_2}, i_3^{H_2}\}$; outputs are mesh nodes $n_m^{F_2}$ which are originally from M_1. Note that this corresponds to the step (g) in Fig.3.3 of Section 3.2.

After the above steps, a surface mesh M_2 on the *target* surface can be generated by mapping the *source* surface mesh M_1. The above steps can be simply expressed in the equation form as Equation (3.16) where inputs of our approach are the *source* surface mesh M_1, *source* surface faceting F_1, *target* surface faceting F_2 and the parametric space of *linking* surfaces $\boldsymbol{F}_{link}\{i,k\}$; the output of our approach is the *target* surface M_2.

$$M_2 = targetSurfMesh(M_1, F_1, F_2, F_{link}(i,k)) \tag{3.16}$$

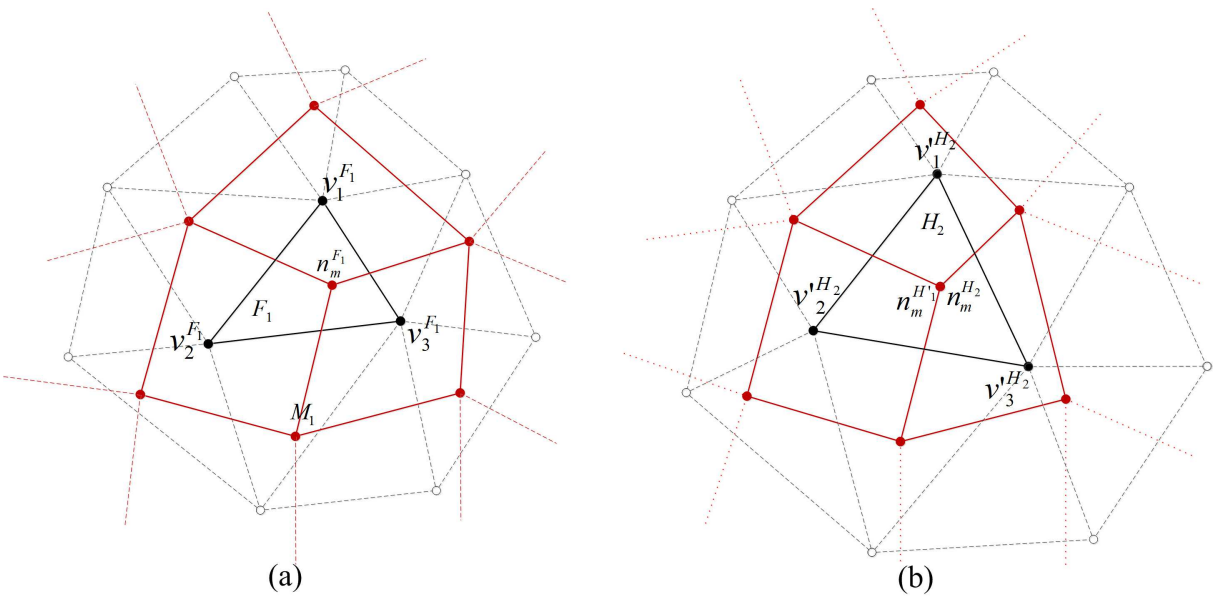

Figure 3.7: Mapping mesh node $n_m^{F_1}$ from F_1 to H_2 and F_2: (a)*source* surface mesh M_1 is embedded with F_1; (b)*source* surface mesh M_1 is mapped onto H_2

An example of surface mesh mapping between two multiply-connected domains is shown in Fig.3.8 where the *target* surface is meshed by mapping the *source* surface mesh M_1.

Although at first glance, the algorithm described above looks similar to ***BMSweep***[86], it is different from ***BMSweep***[86] in the following ways:

(a). ***BMSweep*** uses only boundary mesh nodes while our approach uses surface faceting including triangles and points on boundaries and surfaces, thus our approach can not only capture the boundary changes, but also surface variation. This is especially important for curved surfaces. It is also ideal and well bounded, in that the surface faceting is guaranteed to capture any geometric curvature that is there, with as few or as many as needed.

(b). ***BMSweep*** uses the same background mesh connectivity on all the layers including

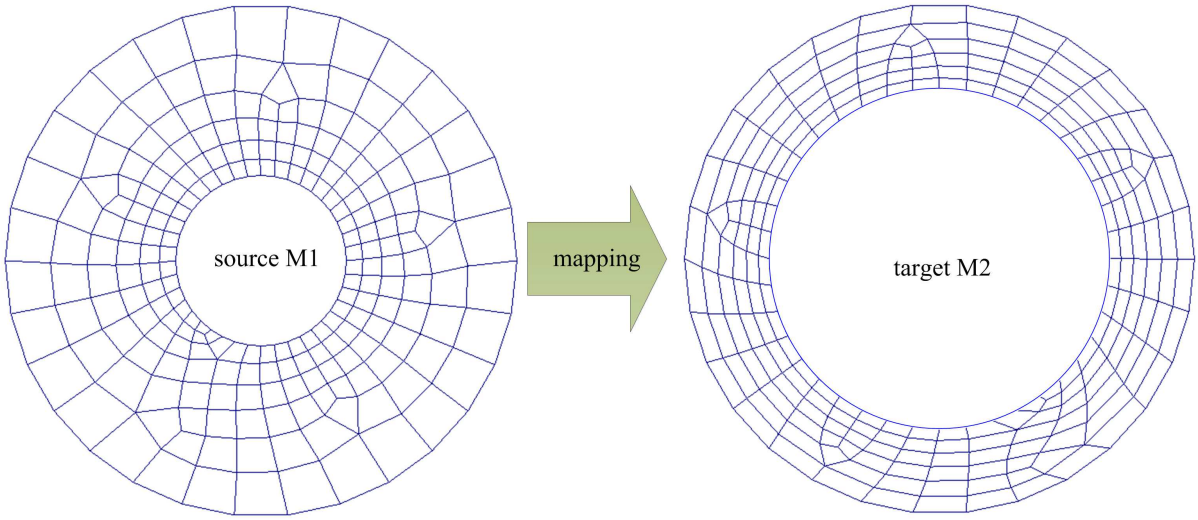

Figure 3.8: An all-quadrilateral mesh mapping from a *source* surface M_1 to its *target* surface M_2

the *target* surface while our approach uses the different surface faceting thus we have the same guarantee of the *target* surface faceting resolving well the *target* surface;

(c). ***BMSweep*** simply uses barycentric coordinates and the same background mesh connectivity to place interior nodes on the *target* surface while our approach uses *harmonic mapping* to map surface faceting onto common domain where interior nodes from the *source* surface can be embedded and mapped onto the *target* surface by barycentric coordinates, thus our approach can generate the *target* surface mesh mapped from the *source* surface with good mesh quality.

3.6 Examples and Results

In this section, four examples are presented to demonstrate capabilities and advantages of the algorithm described above. To highlight the analyzed capabilities, we have selected simple geometries with specified concave or multiply-connected features. For all the

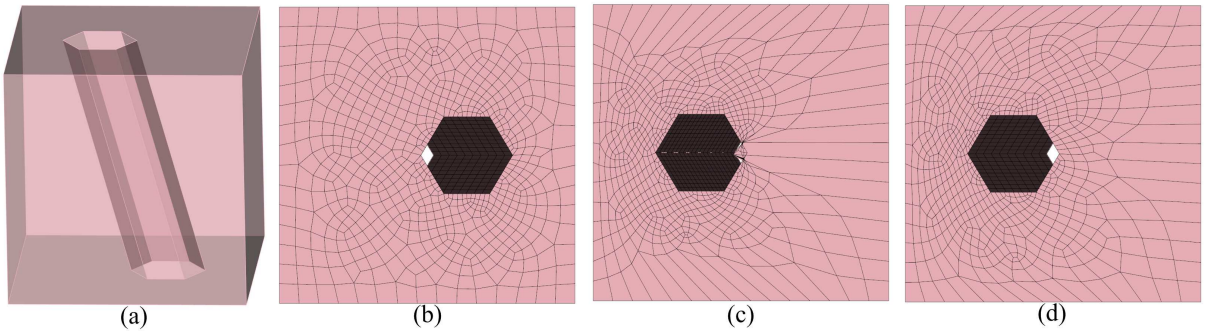

Figure 3.9: A volume with a moving hole between the *source* and *target* surface: (a)geometry model; (b)an all-quad *source* surface mesh; (c)an all-quad *target* surface mesh generated by *Cubit 13.2*; (d)all-quadrilateral *target* surface mesh generated by our method

examples, first an all-quad mesh from *source* surface is projected onto the *target* surface based on the *harmonic mapping* of graphical faceting(triangular meshes). Note that in these examples, it is not necessary to apply any smoothing procedure even though it can improve the mesh quality a little bit, because our algorithm can generate surface meshes with adequate mesh quality as long as the *source* surface meshes with good quality are provided. In these examples, we analyze capabilities of the proposed algorithm for *target* surface mesh generation, that is, we test *target* surface mesh generation on the concave or multi-connected surfaces. In order to measure the quality of mapped surface meshes, mesh quality histograms are plotted.

Multiply-Connected Surface Test: The goal of the first example in Fig.3.9 is to illustrate capabilities of mapping one surface mesh onto another surface when they are multiply-connected. In this example, there is a moving hole between the *source* and *target* surface where the interior boundary is deformed while the outmost boundary is constant during sweeping. By mapping the *source* and *target* surface facetings(triangle meshes) onto a common domain based on *harmonic mapping*, the *source* surface mesh shown in Fig.3.9(b) can be projected onto the *target* surface(see Fig.3.9(d) for details) while Cubit 13.2

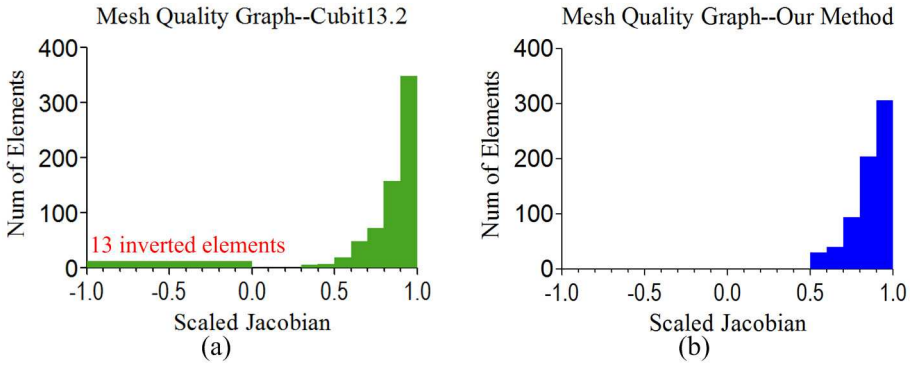

Figure 3.10: Mesh quality histogram for Fig.3.9: (a)mesh quality histogram for Fig.3.9(c); (b)mesh quality histogram for Fig.3.9(d)

generates the *target* surface mesh with inverted elements. The mesh quality histogram is plotted in Fig.3.10 where our method can generate surface meshes with better mesh quality than *Cubit13.2* shown in Fig.3.10(a) and Fig.3.10(b).

The goal of the second example in Fig.3.11 is to illustrate capabilities for generating surface mesh on the concave and multi-connected surface. In this example, there is a blocky volume with a concave feature and an increasing twisted hole between the *source* and *target* surface. Our algorithm based on *harmonic mapping* is able to project the *source* surface mesh (shown in Fig.3.11(b)) onto the *target* surface in Fig.3.11(d). For comparison, we run this example in *Cubit13.2* and results are shown in Fig.3.11(c). In order to measure the mesh quality, we plot the mesh quality histogram in Fig.3.12. The result shows that our algorithm can produce *target* surface meshes mapped from *source* surface with good mesh quality(see Fig.3.12(b) for details) while inverted elements are generated by *Cubit13.2* in Fig.3.12(a).

In the third example shown in Fig.3.13, it is a blocky volume where there is a twisted hole between the *source* and *target* surface. If the linear transformation is applied to generate *target* surface mesh, 170 inverted elements are created(see Fig.3.13(d)). After the surface mesh projection based on *harmonic mapping* is used, inverted elements are

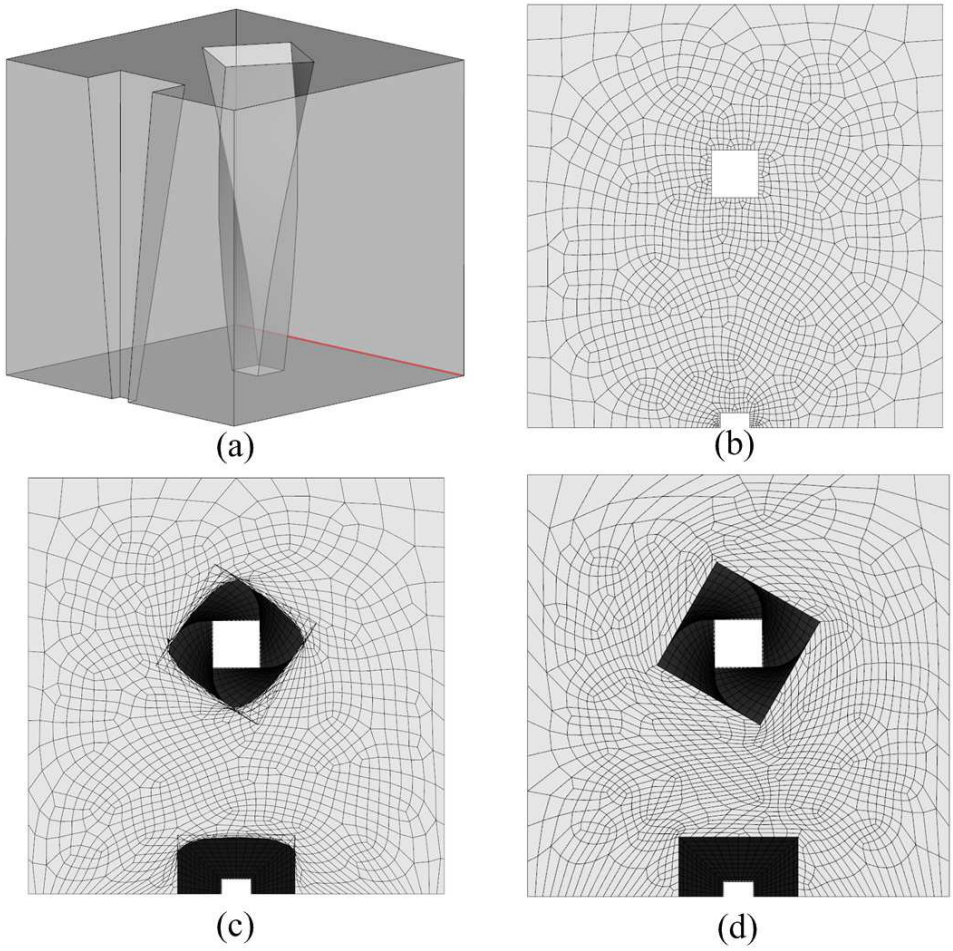

Figure 3.11: Volumes with concavities and an increasing twisted hole: (a)geometry model; (b)an all-quad *source* surface mesh; (c)an all-quad *target* surface mesh generated by *Cubit 13.2*; (d)an all-quad *target* surface mesh generated by our method

eliminated(see Fig.3.13(c) for details). In order to compare mesh quality, histograms are plotted in Fig.3.14 where we can conclude that our algorithm can produce *target* surface mesh with better mesh quality.

Curved Cap Surface Test: The goal of the fourth example in Fig.3.15 is to illustrate how our algorithm projects a *source* surface mesh onto its corresponding non-flat *target* surface with many internal holes. There is a spongecake with a lot of varying holes inside a volume shown in Fig.3.15. There are 709 inverted elements if simple linear *affine transformation*

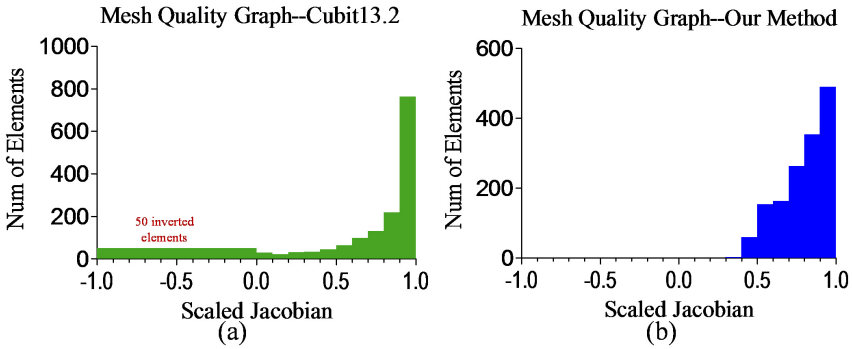

Figure 3.12: Mesh quality histogram for Fig.3.11. (a) mesh quality histogram for Fig.3.11(c); (b) mesh quality histogram for Fig.3.11(d)

is applied. After our algorithm based on *harmonic mapping* of graphical triangular meshes is applied, no inverted element are created. In addition, Figure.3.15(d) shows that good mesh quality on the *target* surface has been achieved.

3.7 Summary

In this chapter, a new surface mesh mapping algorithm based on *harmonic mapping* have been developed. It can map surface meshes between two convex, concave or multiply-connected domains by first mapping the *source* and *target* surface faceting onto the common domain so that the *source* surface mesh can be embedded onto the common domain and mapped back to the *target* surface in the physical space. The correspondence of boundary points from s/t surface faceting is made through the *linking* surfaces and interior faceting points are mapped through *harmonic mapping*. It has been demonstrated that our method can achieve good mesh quality when mapping the *source* surface mesh onto the *target* surface, which is crucial for getting high quality swept volume meshes. What is more, our method described in this chapter can be easily extended to generate *target* surface meshes for multi-sweeping problems as long as *source* and *target* surfaces are partitioned into a

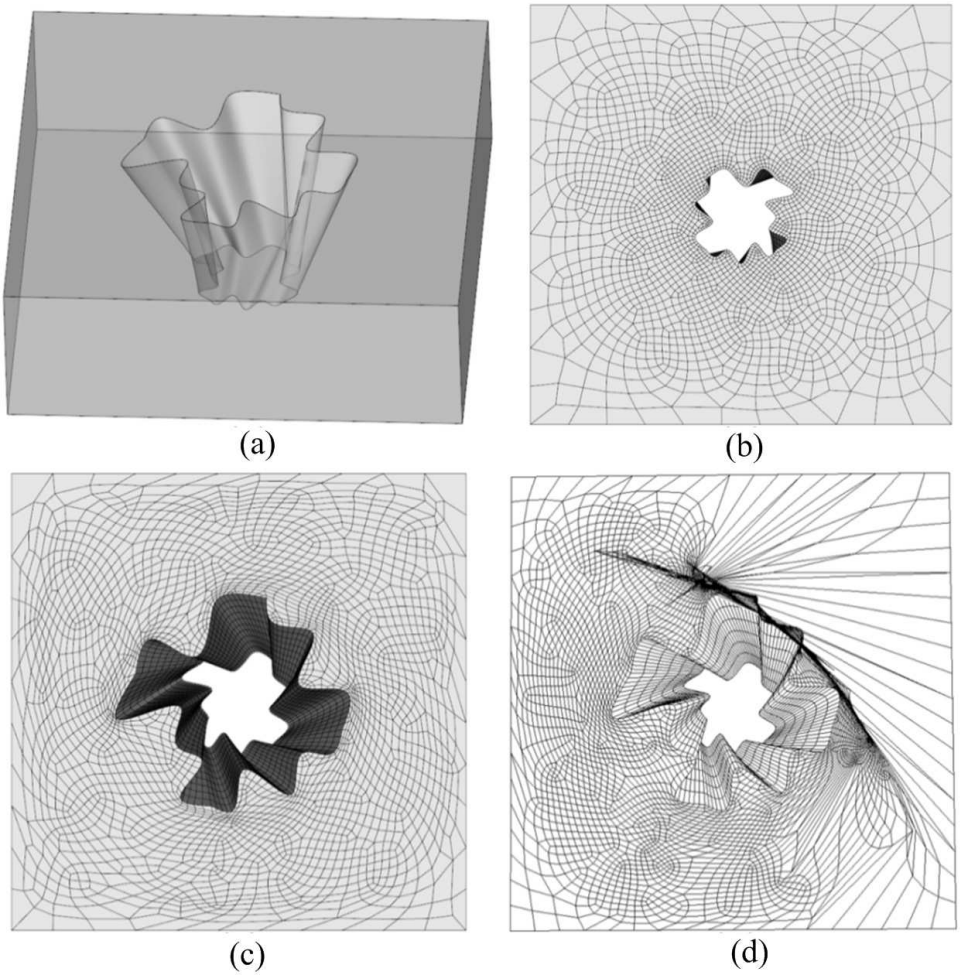

Figure 3.13: Swept volume with a varying hole of random boundary. (a)geometry model; (b)an all-quad mesh on the source surface; (c)quadrilateral meshes on the *target* surface generated by our method; (d)quadrilateral meshes on the *target* surface generated by *Cubit13.2*

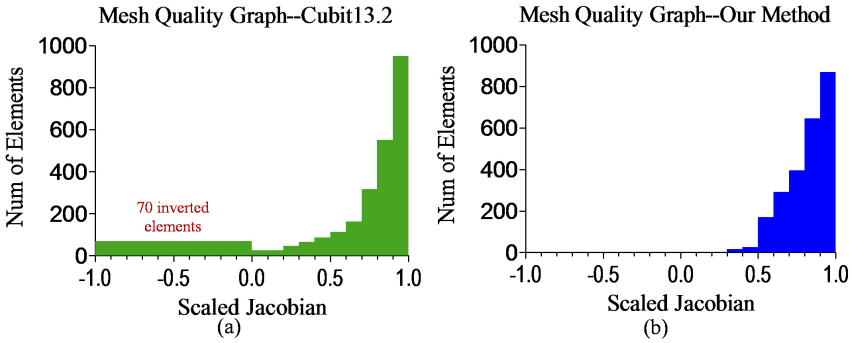

Figure 3.14: Mesh quality histogram for Fig.3.13: (a)mesh quality histogram for Fig.3.13(d); (b)mesh quality histogram for Fig.3.13(c)

list of surface patches so that they are paired with each other, which will be described in Chapter 5.

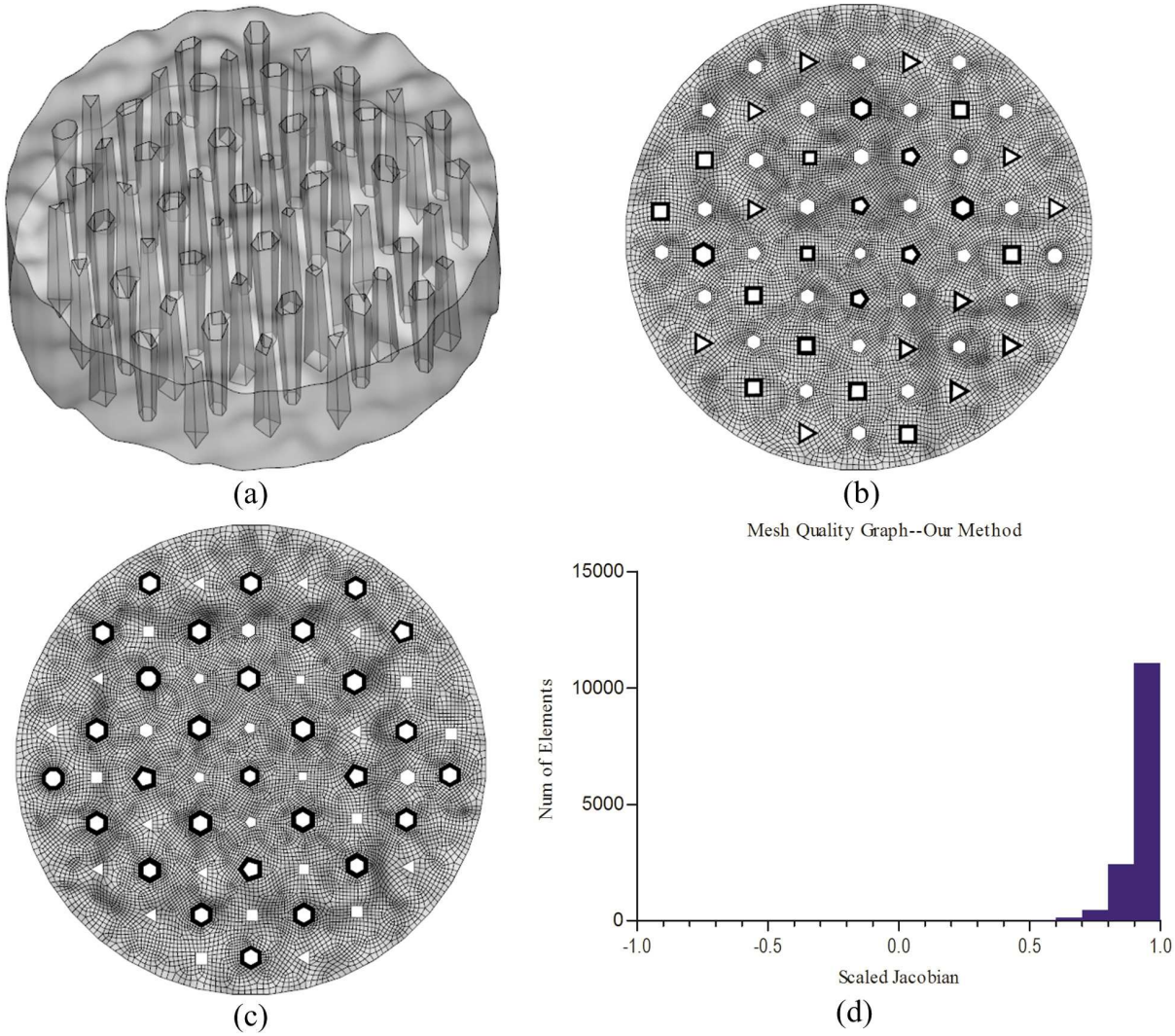

Figure 3.15: A spongecake meshed by sweeping. (a)geometry model; (b)quadrilateral meshes on the *source* surface; (c)quadrilateral meshes on the *target* surface generated by our method; (d)mesh quality histogram

4 INTERIOR NODES' PLACEMENT INSIDE VOLUMES

The interior nodes in a swept mesh are bound by the *source*, *target* and *linking* surfaces. In Section 4.1, problems of placing interior nodes by current existing methods are discussed, which results in an alternative method for determining locations of interior nodes via the cage-based deformation in this dissertation. The remainder is structured as follows: the application of cage-based method on how to locate interior nodes inside volumes is described in Section 4.2 and subsequent Section 4.3 demonstrates examples of placing interior nodes inside volumes by sweeping.

4.1 Problem Statement

Interior mesh *connectivity* for swept volume meshes is entirely determined once those bounding surfaces have been identified and meshed. However, spatial embedding of the interior nodes inside volumes is also a crucial step in generating a quality mesh. For more general geometries, there is no uniquely correct location for interior nodes. Therefore, the guiding principle in the general case is that nodes are placed so that volume elements of a swept mesh are of good quality.

Inspired by the concept of sweeping, current existing methods[39, 65, 66, 67, 69, 86] place interior nodes layer by layer. There is an obvious disadvantage that it is difficult for those methods to be extended to the multi-sweeping problem: new surface meshes need to be added when a new *source* surface is met or another part of surface meshes needs to be terminated when a new *target* surface is met during multi-sweeping, namely, the boundary meshes are not fixed like ***1-1*** sweeping and keep changing during multi-sweeping. In addition, there are some other disadvantages: they fail for concave or multiply-connected domain in that the *affine transformation* is used; they can not deal with boundary changes

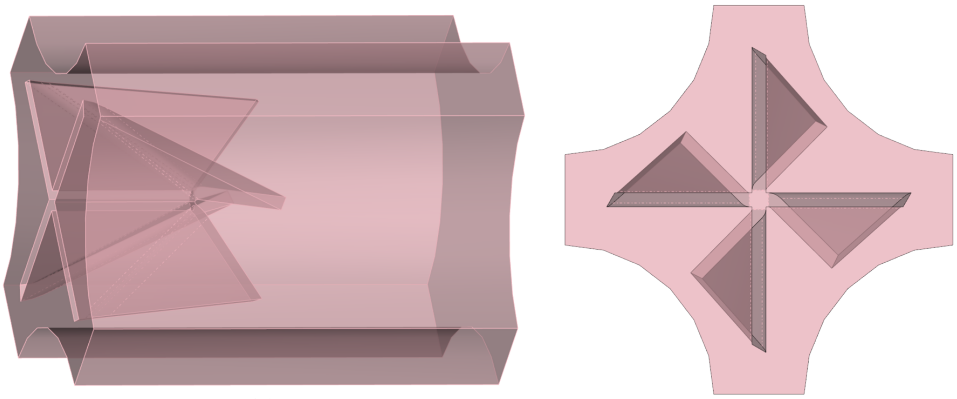

Figure 4.1: An example problem of locating interior nodes with a blade-like blind hole: (a)geometry model; (b)top view

in a small area where the number of boundary nodes is a small percentage compared to the number of all the boundary nodes since a simple *transformation matrix* is used for all the interior nodes on one layer, namely, locality described in Section 2.3.2 of Chapter 2. This is especially important when there is a volume with complicated internal structure from the *linking* surfaces. For example, Figure 4.1 shows a volume with blade-like blind hole inside volume, which is the flow region around a nuclear reactor assembly grid spacer. Since the cross-like hole is twisted gradually from the left to the right, current existing methods fail to correctly place interior nodes since a transformation matrix can not capture the boundary changes of blade-like hole during sweeping: the number of boundary nodes on the cross-like hole is very small compared to all the boundary nodes; if dense boundary mesh on the cross-like hole compared to the outmost boundary is used, interior nodes around the outmost boundary will be pushed outside the volume due to the twisted changes of the cross-like hole.

Hence, a new algorithm for locating interior nodes inside volumes by sweeping based on the cage-based method is developed in this work: inputs are the bounding surface meshes (*source*, *target* and *linking* surface meshes); outputs are locations of interior nodes

inside volumes by sweeping with good volume mesh quality.

4.2 Interior Nodes' Placement Inside Volumes

At this point, all the bounding surfaces have been meshed. Interior nodes are constrained by their bounding surfaces including *source*, *target* and *linking* surfaces, which can be used to guide the interior node placement inside volumes. Hence, the application of cage-based deformation on locating interior nodes inside volumes by sweeping is structured as follows: *idealized model* defined in Section 4.2.1, pipeline of interior node placement in Section 4.2.2, the binding function interpolation in Section 4.2.3 and interior node interpolation in Section 4.2.4.

4.2.1 Idealized Model

When used for animation, the cage-based method works by deforming one 3D mesh into another 3D mesh, using the closed bounding surfaces as constraints. For surface meshes, mapping methods described in Chapter 3 are effective for mapping a real space mesh to a reference space (2D disk), then back to another deformed, real space mesh. In order to locate interior nodes within volumes by sweeping, we have derived an algorithm that is a mix of these approaches, where we take a 3D mesh on a reference space called idealized model, and map it to the actual, real space model. The 3D mesh in both the reference space and real space should have the same mesh connectivity. In the reference space, interior nodes inside volumes are placed by simple *affine transformation* such as translation. Then constrained by the bounding surfaces in the real space, interior nodes within volumes in the reference space are mapped into the real space. Therefore, we have the **Definition 4.1.** for the *idealized model*

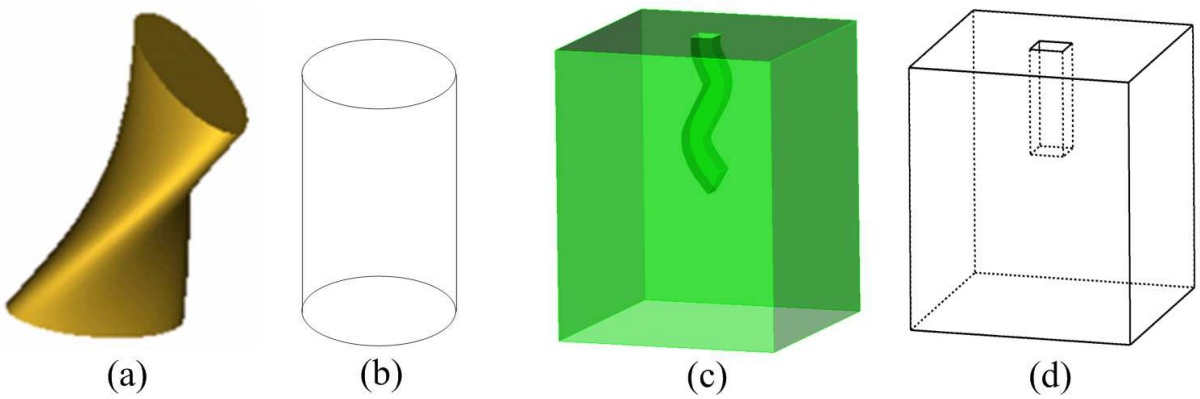

Figure 4.2: An example of idealized model:(a)a physical model for ***1-1*** sweeping (b)the idealized model for (a); (c)a physical model for multi-sweeping; (d)the idealized model for (c)

Definition 4.1. *Idealized model*: it is defined as a simplified model which is topologically equivalent to the physical model. Interior nodes in the *idealized model* can be located by the *affine transformation* such as translation while the same mesh connectivity both on surfaces and inside volumes is kept as the physical model. The bounding surfaces of an idealized model are called *idealized cage*.

An example of a real model and its idealized model for ***1-1*** sweeping is shown in Fig.4.2(a-b) and Fig.4.2(c-d) shows an example of idealized model for multi-sweeping. One of great advantages for the idealized model is that interior nodes can be quickly placed with good mesh quality and binding relationship between the bounding surface mesh and interior nodes can be interpolated.

In order to apply the cage-based deformation for locating interior nodes inside volumes, there are several requirements for creating the idealized model:

(1). The idealized model should be meshable by *affine transformation*: this is used to avoid extra expensive computational costs when locating interior nodes inside the

idealized model.

(2). The idealized model should be topologically equivalent to the physical model: if they are not equal, it is almost impossible to perform mappings between them.

(3). The idealized model should have the same mesh connectivity as the physical model: this is used to guarantee that the ***1-1*** correspondence between the physical model and idealized model is kept. During multi-sweeping, surface meshes on the *linking* and s/t surface patches are transferred onto the idealized model where interior nodes are placed with the *affine transformation*. Then interior nodes in the idealized model are mapped back to the physical model.

In this dissertation, automatic determination or creation of the idealized cage for an arbitrary sweepable volume is not included, which will be described in Section 9.2 of Chapter 9.

4.2.2 Framework for locating interior nodes

The cage-based deformation can be applied to place interior nodes inside volumes because interior nodes are enclosed by the bounding surfaces(*source*, *target* and *linking* surfaces), which constrains the interior node placement inside volumes. Our method for locating interior nodes is derived from the cage-based deformation, but it is different from cage-based deformation since the binding relationship between the bounding surface meshes and interior nodes is unknown while the deformed cage from the bounding surface meshes in physical space is already known. Therefore, an example which illustrates the general framework for locating interior nodes by sweeping is shown in Fig.4.3. Note that there is a prerequisite that all the *source*, *target* and *linking* surfaces should be meshed before placing interior nodes inside volumes such as an example in Fig4.3(b). Our method for

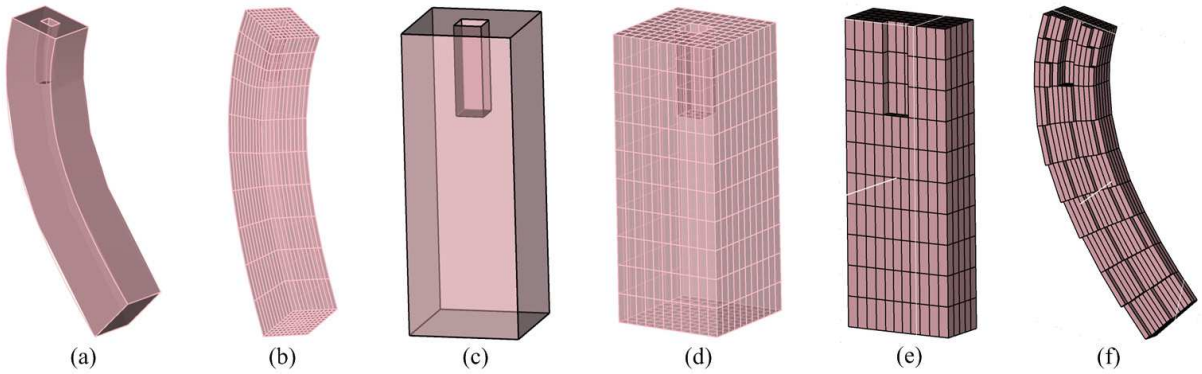

Figure 4.3: Framework for locating interior nodes inside volumes by sweeping: (a)a physical Model; (b)the bounding surface meshes on the physical model; (c)the idealized model; (d)surface mesh on the idealized model; (e)interior nodes inside the idealized model; (f)interior nodes inside the physical model

placing interior nodes inside volumes by sweeping which is derived from the cage-based method is described as follows:

(1) **Create the *idealized model*** The idealized model creation should satisfy requirements described in Section 4.2.1. The reason why we should create idealized model is that: since a physical model may contain very complicated internal structures such as Fig.4.1, it is difficult to directly locate interior nodes; however, if we can create an idealized model and construct mapping between the idealized model and physical model, interior nodes within a physical model can be placed with good mesh quality. How to develop a robust algorithm for creating idealized models is one of our future work described in Section 9.2 of Chapter 9. Figure 4.3(c) shows an example of creating the idealized model. Since there is ***1-1*** correspondence between the idealized model and physical model, the bounding surface meshes in the idealized model for Fig.4.3(c) can be derived from the physical model, which is shown in Fig.4.3(d).

(2) **Locating interior nodes in the idealized model** Since the idealized model is a simplified model derived from the physical model, interior nodes can be placed with the *affine transformation* such as previous methods described in References [39, 66, 69]. Note that in the idealized model, the mesh connectivity including surface meshes and interior volume meshes keeps the same as the physical model. Hence, there is ***1-1*** correspondence of every mesh node between the idealized model and physical model. An example of placing interior nodes inside the idealized model is shown in Fig.4.3(d) where the translation is used.

(3) **Binding** In the idealized model, positions of interior nodes have already been placed by the *affine transformation*(translation, rotation, scaling or combinations of them) as step (2). Because all the mesh nodes on the bounding surfaces in the idealized model have already been located as well, Equation (2.6) is solved for binding interior nodes with respect to cage vertices on the bounding surfaces in the idealized model. That means: every interior node location is a function of cage vertices on the bounding surfaces.

(4) **Interpolation inside the deformed cage** The surface meshes on the bounding surfaces(*source*, *target* and *linking* surfaces) in the physical model are used as the deformed cage mesh. Because the binding function has already been computed in step (3), Equation (2.7) with the deformed cage as inputs is used to interpolate final location of interior nodes inside the deformed cage in the physical space. An example of locating interior nodes in the physical space is shown in Fig.4.3(f).

4.2.3 Binding

Of the four steps described in Section 4.2.2, the most expensive part is to solve the binding function of interior nodes with respect to the cage vertices in the idealized model. The main problem to solve desired cage-object relationship (binding) starts with a more theoretical problem. The inputs of cage-object binding process are all the surface nodes(*source*, *target* and *linking* surfaces) $\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m\}$ and interior nodes $\{\boldsymbol{\eta}_1, \cdots, \boldsymbol{\eta}_n\}$ of the mesh M in the idealized model so that ϕ_i can be computed in Equation.2.6.

For the binding process, there exists ***MVC***, ***HC***, ***GC*** and ***RBF***. From Section 2.3.2 of Chapter 2, it is known that ***MVC***, ***GC*** and ***RBF*** are not suitable for binding interior nodes with the bounding surface meshes. Therefore, ***HC*** is used to compute the binding function of interior nodes with respect to the cage vertices in the idealized model. Based on the Reference [22], ϕ_i in Equation (2.6) can be computed as follows.

(1). Allocate a regular grid of cells that is large enough to enclose the cage. Each grid cell contains a value and a tag where the tag could be one of *UNTYPED*, *BOUNDARY*, *INTERIOR*, or *EXTERIOR*.

(2). Initialize the grid by

  (a). Tag all cells as UNTYPED.

  (b). Scan-convert boundary conditions into the grid, marking each scan converted cell with the BOUNDARY tag when a cell intersects the boundary cage.

  (c). Starting with one of corner cells, flood fill the exterior, marking each visited cell with the EXTERIOR tag. The flood fill recursion stops when BOUNDARY tags are reached. Since the cage is closed, only the exterior cells are visited during this stage. The remaining cells are INTERIOR ones.

(d). Mark the INTERIOR cells with *harmonic coordinate* value equal to 0.

(3). Apply Laplacian's operator: for each INTERIOR cell, replace the value of the cell with the average of the value of its neighbors.

$$\bigtriangledown^2 \varphi_i(p) = 0, \qquad p \in Interior(C) \tag{4.1}$$

where C are the bounding surfaces. For any point ∂p on the boundary ∂C,

$$h_i(\partial p) = \phi_i(\partial p) \tag{4.2}$$

where $\phi_i(\partial p)$ is the piecewise linear function so that $\phi_i(C_j) = \delta_{i,j}$, that is, $\phi_i(\partial p)$ is the B-spline basis function commonly known as the "hat function" basis (φ on each boundary node is equal to 1) in the numeric analysis area. After solving Equation (4.1), φ on each interior node is a combination of influences from all the bounary nodes, while most of boundary nodes which are far away from the interior node have almost zero influence on it and only its close neighbouring boundary nodes have strong influence on its location. In the meantime, $\sum_i \varphi_i(p_j) = 1$ where p_j is an interior node, which is proved in Reference [22].

The binding process can be accelerated by Multigrid method which uses a hierarchy of discretization of grids for solving the *Laplacian Equation*. Note that the binding process could be done by using the volume mesh (interior nodes and bounding surface meshes) within the idealized model. However, the accuracy is lost since some interior nodes are directly connected to the mesh nodes on the bounding surfaces. For example, the interior nodes closest to the bounding surfaces are directly connected to the bounding surfaces and there are only one mesh edge between them. This can be avoided by adding more

nodes between those interior nodes and nodes on the bounding surfaces.

4.2.4 Interior nodes' placement by interpolation

In the idealized model, interior nodes are bound with cage vertices, which means: any interior node can be placed with a function with arguments of bounding cage vertices as inputs. Given the real space cage coordinates and the binding function computed earlier, the interior vertex positions can be computed by evaluation of that binding function. The mesh quality is improved by the use of ***HC*** approach for the binding function. Equation.(2.7) can be used to interpolate interior nodes in the physical model after ϕ_i has been solved during the binding process. From Equation (2.7), it is known that interior nodes are affected by influences from all the cage vertices while most of them have close to zero influence of cage vertices which are far away. In other words, The cage vertices only have strong influence on their near neighbouring regions.

4.3 Examples and results

In this section, several examples are demonstrated in order to show the robustness of our interior node placement algorithm by sweeping. Before placing interior nodes inside volumes, all the bounding surfaces should be meshed including the *source*, *target* and *linking* surfaces. If there is a multi-sweeping problem, the s/t surfaces are split into a list of s/t surface patches after imprinting and their new bounding surfaces should be meshed so that those meshes can be transferred onto the idealized model. For all the examples described below, their idealized models are created by starting from the bottommost *target* surface, translating its surface meshes in the inverse sweeping direction, adding new *target* surfaces if meeting a new one, subtracting corresponding parts of surfaces when meeting a

new *source* surface until all the *source* surfaces are reached. Since the idealized model is meshable by *affine transformation* described in references [39, 65, 69] and meshes on all the s/t surfaces or surface patches can be transferred onto the idealized model directly, interior nodes within the idealized models can be located by *affine transformation* as well. By constructing mapping between idealized model and physical model, interior nodes inside volumes in the physical space can be placed with good mesh quality.

The first example in Fig.4.4 presents the interior node placement for a block with a twisted blind hole by cage-based deformation. Because of a twisted hole, the linear *affine transformation* fails to generate the swept volume meshes without introducing any inverted element. This is due to the fact that linear *affine transformation* takes all the boundary nodes as a whole and lacks a property of local deformation. By applying our method where the idealized model for Fig.4.4(a) is shown in Fig.4.4(b), The results with correctly locating interior nodes are shown in Fig.4.4(c) while *Cubit 13.2* produces poor volume mesh which is shown in Fig.4.4(d). The mesh quality histograms for Fig.4.4(c) and Fig.4.4(d) are shown in Fig.4.4(e) and Fig.4.4(f), respectively.

The second example in Fig.4.5 describes interior node placement for a blade-like solid where there are four blades and one blade is bended straightly in order to test the robustness of our method. The blades have the twisted features which make interior node placement difficult. Due to the fact that the cage-based deformation has a property of local deformation, the resulting interior node placement without any inverted element by our method is shown in Fig.4.5(c) and the corresponding idealized model is shown in Fig.4.5(b). However, *Cubit 13.2* generates poor volume mesh with inverted elements shown in Fig.4.5(d). The corresponding mesh quality histograms are shown in Fig.4.6(a) and Fig.4.6(b), respectively.

The third example in Fig.4.7 depicts interior node placement for a volume with a

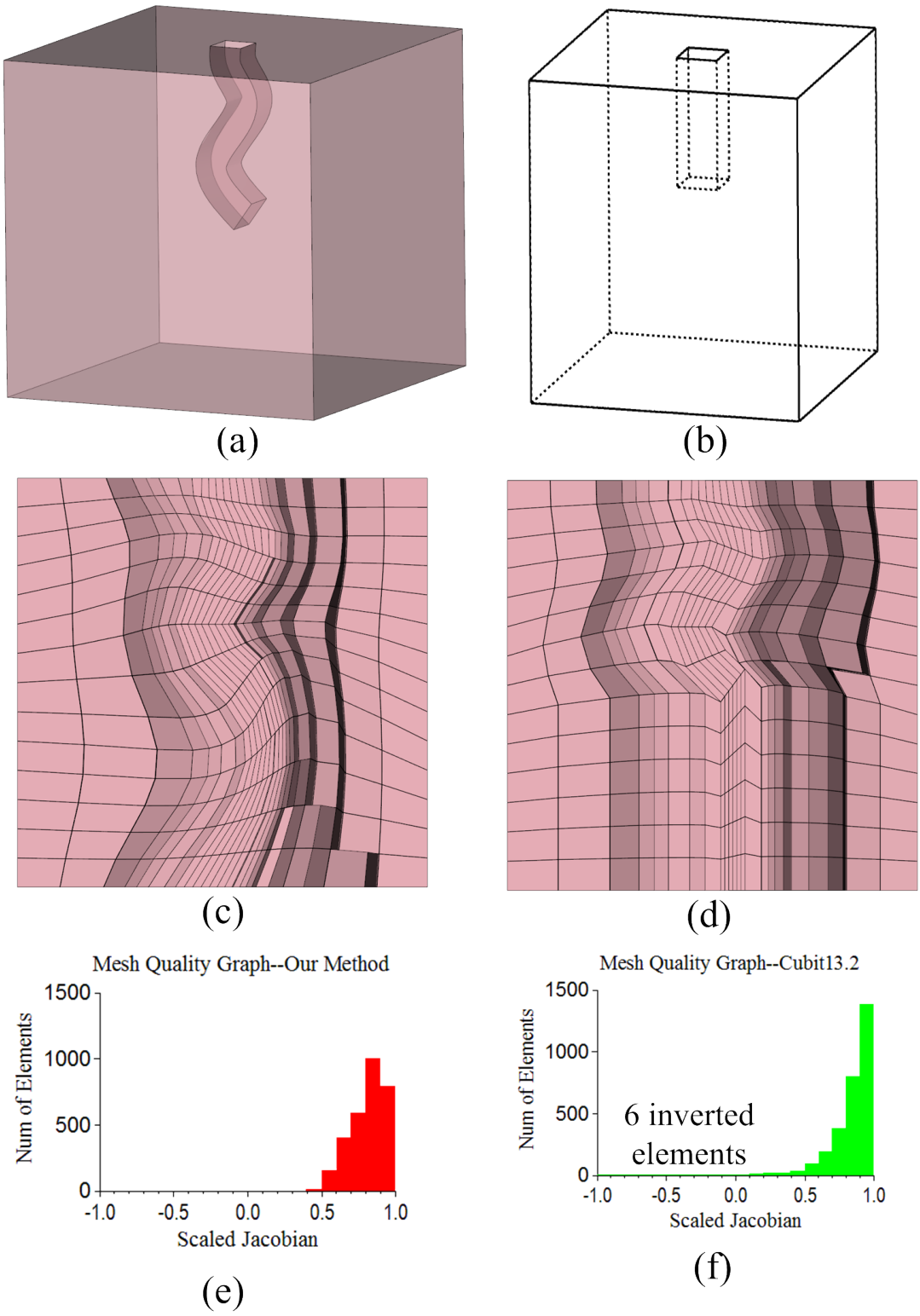

Figure 4.4: An example of interior node placement: (a)geometry Model; (b)idealized model for (a); (c)all-hexahedral mesh with located interior nodes by our method; (d)all-hexahedral mesh by *Cubit13.2*

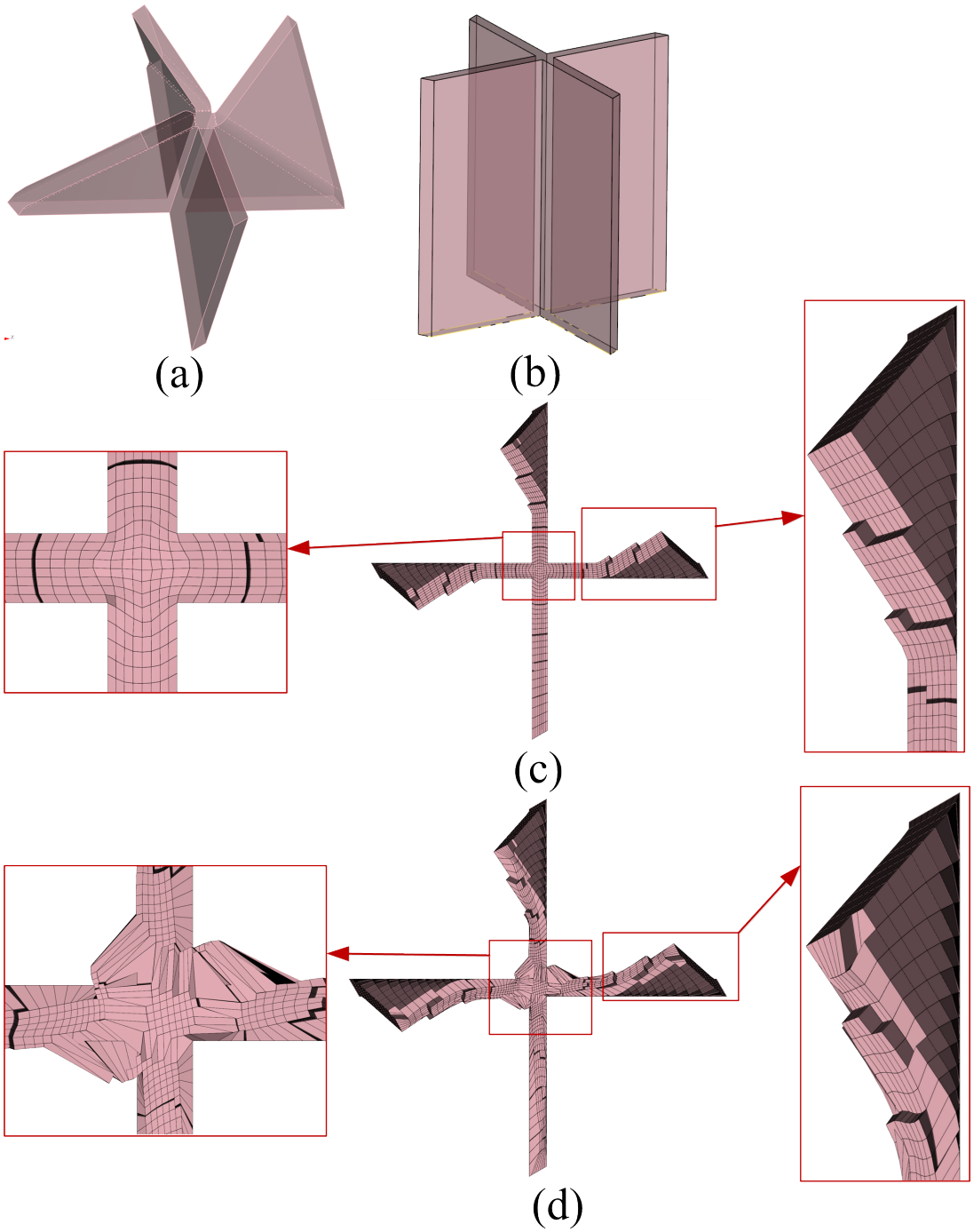

Figure 4.5: An example of interior node placement inside a blade-like solid: (a)geometry Model with four blades while one blade is bended straightly; (b)idealized model for (a); (c)all-hexahedral mesh with located interior nodes by our method with two zoom-in views; (d)interior node placement by *Cubit 13.2* with inverted elements

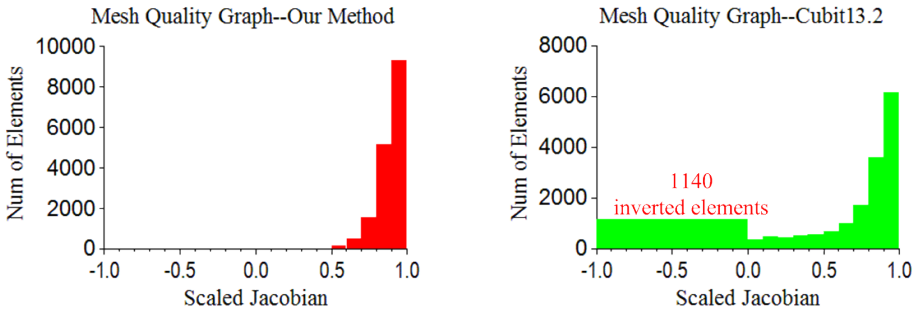

Figure 4.6: Mesh quality histogram for Fig.4.5: (a)Mesh quality histogram for Fig.4.5(c); (b)Mesh quality histogram for Fig.4.5(d)

twisted through hole. If the linear *affine transformation* is applied, inverted volume elements are created as Fig.4.7(b) since the linear *affine transformation* only works for convex cases and lacks the property of local deformation. However, our method correctly interpolates locations of interior nodes and the resulting hexahedral meshes with a local view of interior nodes are shown in Fig.4.7(c) and Fig.4.7(e). The corresponding idealized model is shown in Fig.4.7(f). Good mesh quality has been achieved as Fig.4.7(h) while poor mesh quality are generated by *Cubit 13.2* as Fig.4.7(g).

For the time complexity of placing interior nodes inside volumes by sweeping, there are two main parts: binding of interior nodes with boundary nodes and interpolation of interior nodes; The binding process can be done in $O(nlogn)$, which is related to solve a ***PDE*** equation and n is the number of interior nodes inside volume. The interpolation proces can be done in $O(m * n)$ where m is the number of boundary nodes and n is the number of interior nodes inside volumes.

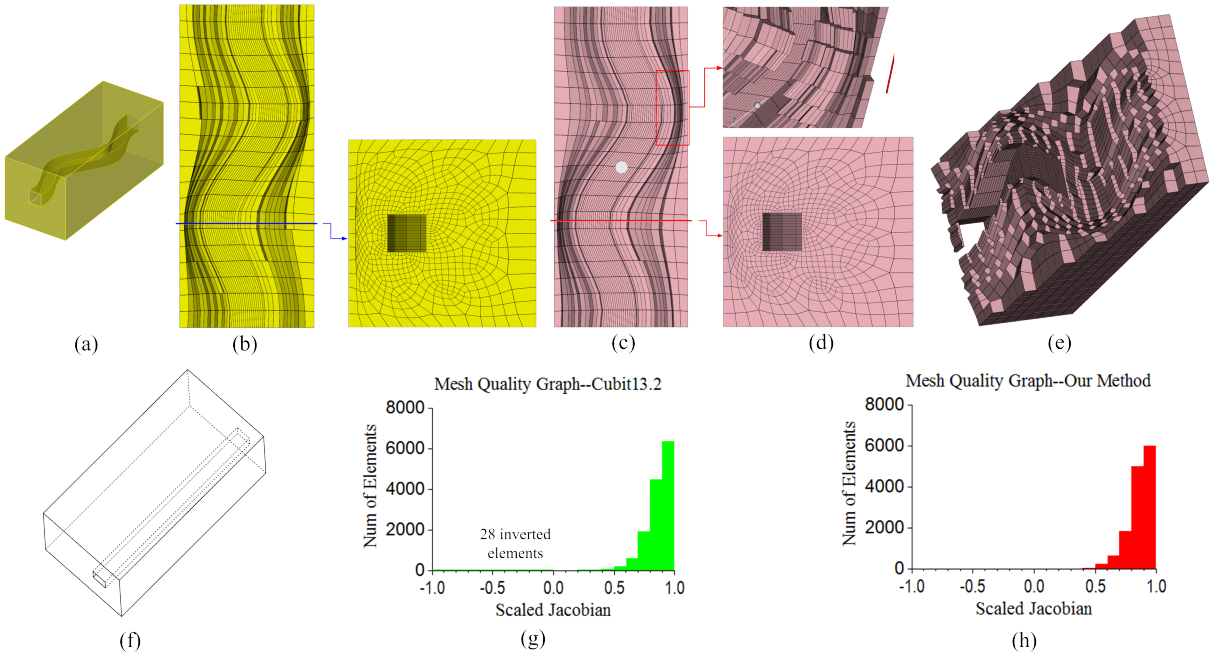

Figure 4.7: An example of interior node placement: (a)geometry Model; (b)all-hexahedral mesh generated by *Cubit13.2* with a local view on the right; (c)all-hexahedral mesh generated by cage-based method; (d)a local view of interior node placement; (e)3D local view of interior node placement; (f) idealized model for (a); (g)mesh quality histogram for (b); (h)mesh quality histogram for (c)

4.4 Summary

In this chapter, a new interior node placement algorithm for sweeping has been developed, which consists of several steps: idealized model creation, interior node placement in the idealized model, binding and interior node interpolation in the physical model. *Harmonic coordinates* are used to bind interior nodes with cage vertices on the bounding surfaces in the idealized model. Interior nodes are interpolated with the bounding surface mesh nodes in the physical model as inputs. Our method uses all the bounding surfaces as constraints to locate interior nodes inside volumes instead of the layer-by-layer approach. Good mesh quality has been achieved, which is critical for getting high quality swept meshes.

5 IMPRINTING OF s-t EDGE PATCHES DURING MULTI-SWEEPING

In this chapter, edge patch imprinting for multi-sweeping problem is described. To begin with, problems of multi-sweeping are discussed in Section 5.1, which leads to the subsequent sections: Section (5.2-5.10) describes the edge patch imprinting between the *source* and *target* surfaces including pipeline of edge patch imprinting, edge patch identification and edge patch propagation along the *linking* surfaces, edge patch projection to s/t surfaces, interior edge patch location within other loops, edge patch intersection/overlapping, edge patch decomposition and s/t surface decomposition. After imprinting, ***1-1*** s/t surface patches are matched as Section 5.12 and Section 5.13 briefly depicts how to generate surface meshes on the *target* surface patches resulting from the s-t imprinting, which is described in detail in Chapter 3. In the end, examples are provided as Section 5.14 and summary is made in Section 5.15.

5.1 Motivation and Problem Statement

Sweeping algorithms[16, 17, 39, 65, 67, 86] have been successfully applied to extrusion geometries for many years. Users want a hexahedral mesh or aligned non-hexahedral mesh by sweeping, and they decompose until the resulting portions can be swept with only ***1-1*** sweeping. This requires more decomposition, while a ***M-N*** sweeping algorithm requires less decomposition. Since the volume decomposition can be very tedious and requires lots of efforts, ***M-N*** sweeping greatly reduces time to mesh.

The ***1-1*** sweeping is a special case of multi-sweeping which includes ***M-1*** and ***M-N*** sweeping. Unlike ***1-1*** sweeping where the *source* and *target* surface has the same topology

and they are directly connected by the *linking* surfaces, ***M*-*N*** sweeping is characterized by M *sources* and N *targets*, which generally have different topologies. The key problem during multi-sweeping is to resolve curves on the *source* and *target* surfaces, combined with the resulting *source* and *target* surface patches needing to match: more specifically, multi-sweeping requires to solve which *source* surface or area of a specific *source* surface will be swept onto a specific *target* surface or a specific area of a *target* surface. This is due to the inherent characteristics that each mesh face element on the *source* surfaces should match only one mesh face element on the *target* surfaces. This subsequently results in that the interval assignment problem for edges on the *source* and *target* surfaces should solved.

Figure 5.3 depicts two multi-sweeping problems. In Fig.5.1(a), a typical multi-sweeping problem is shown where the edge e^1_{src} and e^1_{tgt} need to be resolved by imprinting and partitioning S_1, S_2, T_1 and T_2 so that surface patch matching can be done between the *source* and *target* surfaces. One may argue that Fig.5.1(a) can be solved by the existing volume decomposition approaches. Let's make the multi-sweeping problem in Fig.5.1(a) a bit complicated: Figure 5.1(b) shows a multi-sweeping problem with an increasing hole from the *source* surfaces (S_1 and S_2) to the *target* surfaces (T_1, T_2 and T_3). In Fig.5.1(b), the edge e^1_{src}, e^1_{tgt}, e^2_{tgt} and e^3_{tgt} need to be resolved by imprinting them on the *source* and *target* surfaces so that surface patches can be generated and matched. More specifically, the curve e^1_{src} needs to be resolved by imprinting it on the appropriate *target* surfaces so that a specific region swept to S_1 is computed. So do the curves e^1_{tgt}, e^2_{tgt}, and e^3_{tgt} on the *target* surfaces.

Therefore, a new edge patch imprinting algorithm for multi-sweeping problems has been developed in this work: inputs are the *source* surfaces, *target* surfaces and *linking* surfaces with their parametric spaces; outputs are a list of *source* and *target* surface patches with each s surface patch matching an exactly t surface patch. After imprinting, curves

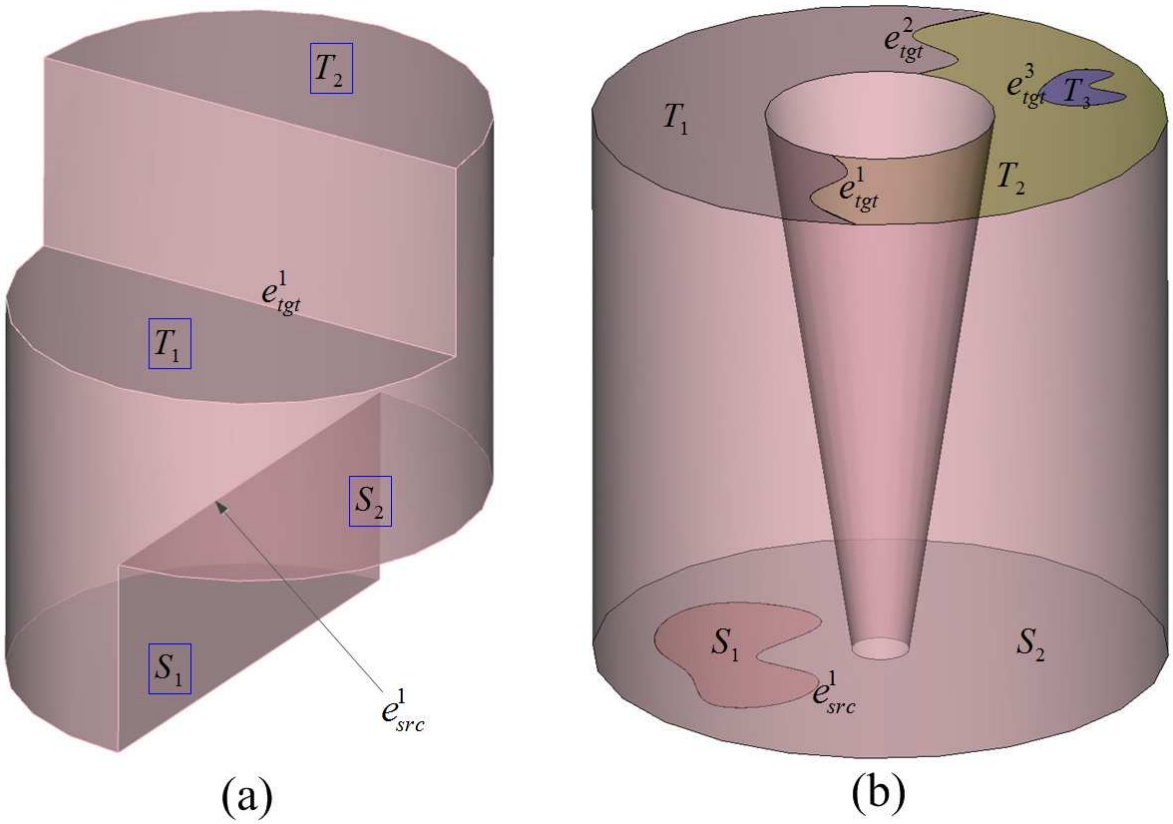

Figure 5.1: An example of problems from current existing multi-sweepers: (a)a multi-sweeping problem with *sources* $\{S_1, S_2\}$ and *targets* $\{T_1, T_2\}$; (b)a multi-sweeping problem with *sources* $\{S_1, S_2\}$ and *targets* $\{T_1, T_2, T_3\}$

and vertices on the *target* surfaces with *source* surfaces are resolved by embedding them on the *source* surface meshes and vice versa.

5.2 Pipeline for *s-t* imprinting

In order to solve problems described in Section 5.1, a new edge patch imprinting algorithm based on cage-based morphing for resolving edges and vertices between s/t surfaces is described in this section. For multi-sweeping, our general approach is to propagate edge patches on the s/t surfaces through the volume. Afterwards, edge patches as well as s/t

surfaces are partitioned so that those edge patches can be resolved by embedding them into the s/t surface meshes.

Figure 5.2 describes the flowchart of our new edge patch imprinting 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, which is used to guide the edge patch propagation along the *linking* surfaces. In this work, edge patches are represented with a list of faceting points and thus edge patch propagation is simplified to propagate those points through volumes: this does not imply that all the boundary edges have to be meshed; there is no restriction on the number of faceting points on the boundary edge patches as the surface facetings in Chapter 3. These faceting points are as coarse or fine as necessary to resolve the underlying geometric curves within some tolerance. Prior to imprinting, sweeping level should be built. Then edge patch imprinting between the *source* and *target* surfaces is performed, which includes: edge patch identification, propagation of bounding edge patches, determination of interior edge patches within loops, projection of interior edge patches, edge patch intersection/overlapping, edge patch decomposition and s/t surface decomposition. After imprinting, a list of new *source* and *target* surface patches have been generated and they have to be matched with ***1-1*** pairing. Finally, surface meshes are mapped between ***1-1*** paired *source* and *target* surface patches by the morphing described in Chapter 3.

5.3 Build sweeping levels

The sweeping *level* is different from the sweeping *layer* in the current existing sweepers where the sweeping layers are simply representation of k parameters on the *linking* surfaces. The sweeping level is defined as any layer where there exists *source* or *target* surfaces in the sweeping direction. The s/t surfaces should be always on one of sweeping levels

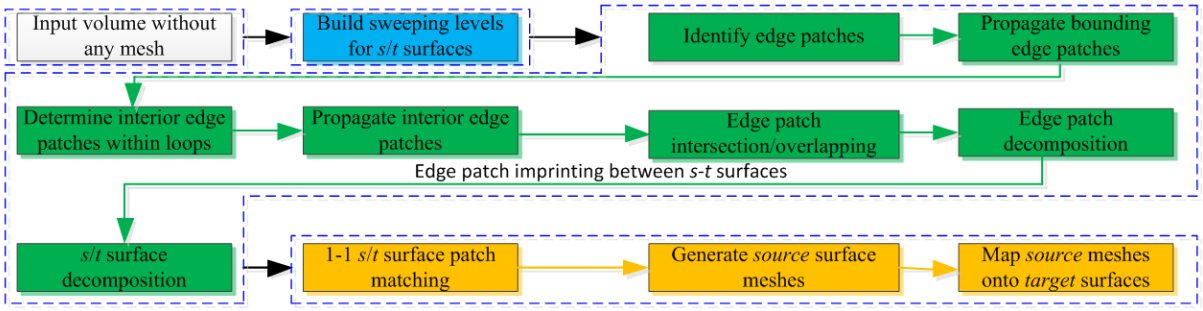

Figure 5.2: Flowchart of s-t edge patch imprinting

while they are not necessarily on every sweeping layer. An example of differences between the sweeping levels and sweeping layers is shown in Fig.5.3. No *linking* surface mesh is required when building the sweeping levels. The sweeping level and layer are important for imprinting between s-t surfaces since it guides where to propagate edge patches during imprinting. Hence, before the edge patch imprinting, the sweeping levels should be built between the *source* and *target* surfaces in the direction of sweeping ($l = 0, \cdots, L-1$).

The sweeping level is built by relying on the parametric space $\{i, k\}$ of different chains of *linking* surfaces. The connectivity of *source* and *target* surfaces on a volume can be represented with a directed graph. For example, Figure 5.4(a) shows a volume with s/t surfaces and the corresponding graph is shown in Fig.5.4(b). In the directed graph, each node represents a *source* or *target* surface. The edge weight is defined as the $\triangle k_n$ on the parametric space $\{i, k\}$ of n-th *linking* surface: absolute value of range of k values. In this work, we define the sweeping level which starts from the topmost *target* surfaces to the bottommost *source* surfaces. Hence, the sign of edge weight is determined based on whether it flows in that direction or in the reverse of that direction, which indirectly relies on the edge type (***END*** and ***CORNER***) between the s/t surfaces and *linking* surfaces.

Since there exists the directed graph for every multi-sweepable volume with edge weights for two arbitrary s/t surfaces directly connected by the *linking* surfaces, the graph

algorithm (*breadth-first search* or *depth-first search*) can be used to build sweeping levels between the *source* and *target* surfaces in the following ways:

(1). **PICK** a *target* surface node T_1 and **ASSIGN** its node value with 0.

$$Level(T_1) = 0$$

(2). **USE** the *breadth-first search* algorithm to visit all the nodes in the directed graph and **ASSIGN** each visited node with a level value.

$$Level(T_i/S_i) = Level(pre\_node) + weight(pre\_node \rightarrow T_i/S_i)$$

(3). **SORT** all the node values in the increasing order.

(4). **ADJUST** all the node values such that the minimum node value is 0 and the remaining ones are assigned with one of $\{1, \cdots, L-1\}$ and keeping the increasing order.

The detail of pseudo code for building the sweeping levels before multi-sweeping is described in **Algorithm**.1. The worst case for the *breadth-first search* algorithm is $O(n)$ where n is the number of nodes in the directed graph, namely, the number of *source* and *target* surfaces. Therefore, the sweeping level for a multi-sweeping problem can be built with $O(n)$.

5.4 Edge Patch Identification

There are two kinds of edge patches during imprinting. The first one is the bounding edge patches defined as: those edge patches are able to propagate along the *linking* surfaces to a specific sweeping level. Hence, all the faceting points on the bounding edge patches lies

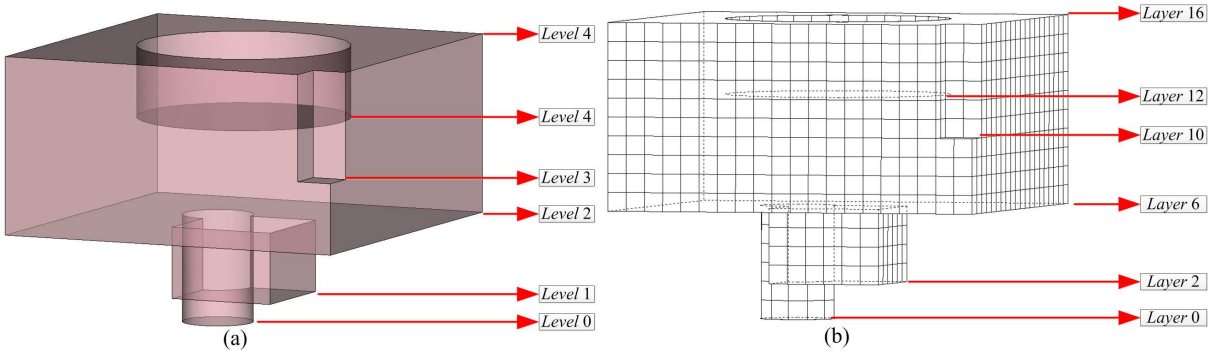

Figure 5.3: An example of differences between the sweeping levels and sweeping layers: (a)sweeping levels; (b)sweeping layers

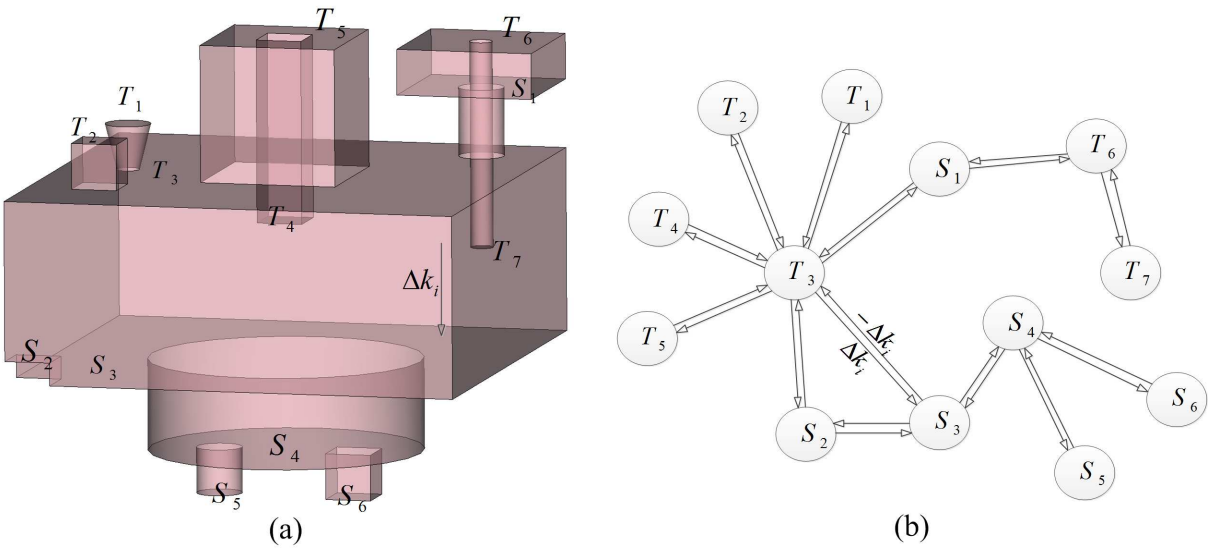

Figure 5.4: An example of directed graph for representing the connectivity between s/t surfaces

Algorithm 1 Pseudo Code of Building Sweeping Levels during Multi-sweeping

Input:

(1). $\boldsymbol{F}_{src} = \{\boldsymbol{F}_{geom}^{i_1}, i_1 = 1, \cdots, m_1\}$

(2). $\boldsymbol{F}_{tgt} = \{\boldsymbol{F}_{geom}^{i_2}, i_2 = 1, \cdots, m_2\}$

(3). $\boldsymbol{F}_{link} = \{\boldsymbol{F}_{geom}^{i_3}, i_3 = 1, \cdots, m_3\}$ with global consistent $\{i, k\}$ space.

Output: $Level(\boldsymbol{F}) \in \{0, 1, \cdots, L-1\}$

---

function buildSweepingLevel($\boldsymbol{F}_{src}, \boldsymbol{F}_{tgt}, \boldsymbol{F}_{link}(i,k)$)

1. **INITIALIZE** the sweeping *Level* for all the s/t surface

$$Level(\boldsymbol{F}_{src}/\boldsymbol{F}_{tgt}) = -1$$

2. **POP** the first *target* surface from $F_{tgt} \rightarrow \boldsymbol{F}$
3. **PUSH** $\boldsymbol{F}$ into the stack $\boldsymbol{\lambda}(\mathbf{F}_{src}, \mathbf{F}_{tgt})$ and **ASSIGN** the sweeping level for $\boldsymbol{F}$

$$Level(\boldsymbol{F}) = 0$$

4. **WHILE** $\boldsymbol{\lambda}(\mathbf{F}_{src}, \mathbf{F}_{tgt}) \neq \emptyset$

4.1. **POP** the first surface from the stack $\boldsymbol{\lambda}(\mathbf{F}_{src}, \mathbf{F}_{tgt}) \rightarrow \boldsymbol{F}_j$

4.2. **GET** the *linking* surfaces which are connected to $\boldsymbol{F}_j$

$$\boldsymbol{\omega}(\mathbf{F}_{link}) = getLinkingSurfs(\boldsymbol{F}_j)$$

4.3. **GET** the *source/target* surfaces which are connected to $\boldsymbol{\omega}(\mathbf{F}_{link})$

$$\boldsymbol{\phi}(\mathbf{F}_{src}, \mathbf{F}_{tgt}) = getSrcTgt(\boldsymbol{\omega}(\mathbf{F}_{link}))$$

4.4. **FOR EACH** $\boldsymbol{F}_n$ in the $\boldsymbol{\phi}(\mathbf{F}_{src}, \mathbf{F}_{tgt})$

4.4.1. **IF** $(Level(\boldsymbol{F}_n)! = -1)$

continue

4.4.2. **IF** $((edgeType(\boldsymbol{F}_j, \boldsymbol{\omega}(\mathbf{F}_{link})) == \boldsymbol{END}$ && $\boldsymbol{F}_j \in \boldsymbol{F}_{tgt})$ || $((edgeType(\boldsymbol{F}_j, \boldsymbol{\omega}(\mathbf{F}_{link})) == \boldsymbol{CORNER}$ && $\boldsymbol{F}_j \in \boldsymbol{F}_{src}))$

$$Level(\boldsymbol{F}_n) = Level(\boldsymbol{F}_j) + \triangle k\{\boldsymbol{F}_j, \boldsymbol{F}_n\}$$

4.4.3. **ELSE**

$$Level(\boldsymbol{F}_n) = Level(\boldsymbol{F}_j) - \triangle k\{\boldsymbol{F}_j, \boldsymbol{F}_n\}$$

4.4.4. **PUSH** $\boldsymbol{F}_n$ into $\boldsymbol{\lambda}(\mathbf{F}_{src}, \mathbf{F}_{tgt})$.

5. **SORT** the *Level* values in the increasing order.
6. **ADJUST** the sweeping *Level* values so that *Level* numbering $\in \{0, \cdots, L-1\}$

on the *linking* surfaces. The other one is the interior edge patches defined as: those edge patches are not able to propagate along the *linking* surfaces to a specific sweeping level. So they have to rely on the cage-based morphing with the help of their bounding edge patches to be propagated to the next sweeping level. Generally, faceting points on the interior edge patches do not always lie on the *linking* surfaces. A special case of interior edge patches is that two end points of a curve are located on the *linking* surfaces while the remaining ones do not lie on the *linking* surfaces at all since edge patches must be interpolated with the help of its bounding edge patches, even though two end points can propagate by themselves on the *linking* surfaces. There is one point to be noted that the bounding edge patches can be converted to be interior edge patches on different sweeping levels and vice versa. For example, in Fig.5.5(a), between the sweeping level $l+1$ and $l+2$, E_0^{l+1}, E_1^{l+1}, E_0^{l+2} and E_1^{l+2} represented with black dots are the bounding edge patches; E_2^{l+1}, E_3^{l+1} and E_3^{l+2} are the interior edge patches. During the propagation from the l level to the $l+1$ level, E_0^l and E_2^{l+1} are the bounding edge patches, while E_2^{l+1} is converted into the interior edge patches when propagating from the $l+1$ sweeping level to the $l+2$ sweeping level.

The edge patch identification algorithm is primarily based on the *linking* surfaces: inputs are edge patches on a specific sweeping level, sweeping levels for s/t surfaces, *linking* surfaces $\boldsymbol{F}_{link}$; outputs are a list of bounding edge patches and a list of interior edge patches. The bounding edge patches are identified based on whether they are directly connected to the next sweeping level by the *linking* surfaces. When the bounding edge patches have already been identified, the remaining edge patches are the interior edge patches.

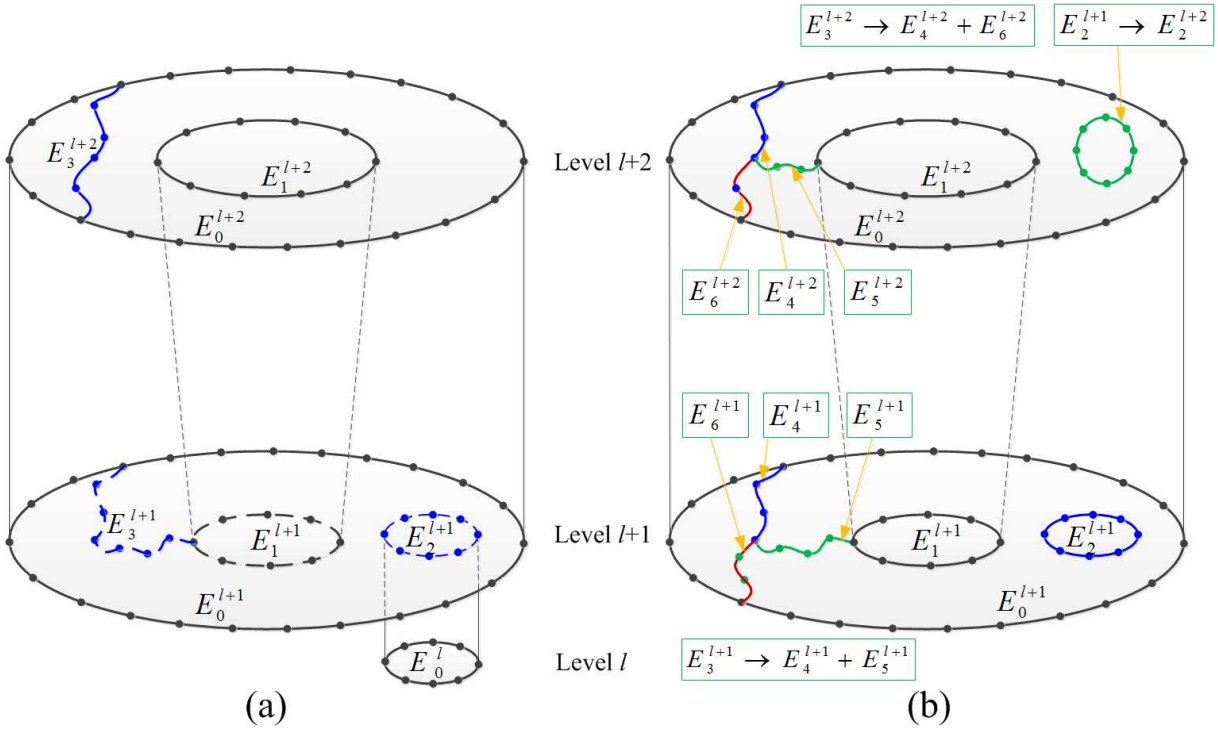

Figure 5.5: Propagation of different types of edge patches: (a)different types of edge patches; (b)edge patch propagation between $l+1$ and $l+2$ sweeping level

5.5 Edge Patch Propagation along *Linking* Surfaces

As Section 5.4 describes, the *linking* surfaces connected those bounding edge patches on the current sweeping level to the next sweeping level. The bounding edge patches are represented with a list of faceting points from surface faceting such as Fig.5.6(a) where edge patches e_a, e_b and e_c are represented with blue circle dots, purple diamond dots and red box dots, respectively.

As it is known from the reference [93], the structured mesh is one with an implied d-parametric space, where d is the topological dimension of the mesh. In this work, we assume that the parametric space of *linking* surfaces should be provided prior to s-t edge patch imprinting. However, that does not mean that structured quadrilateral meshes

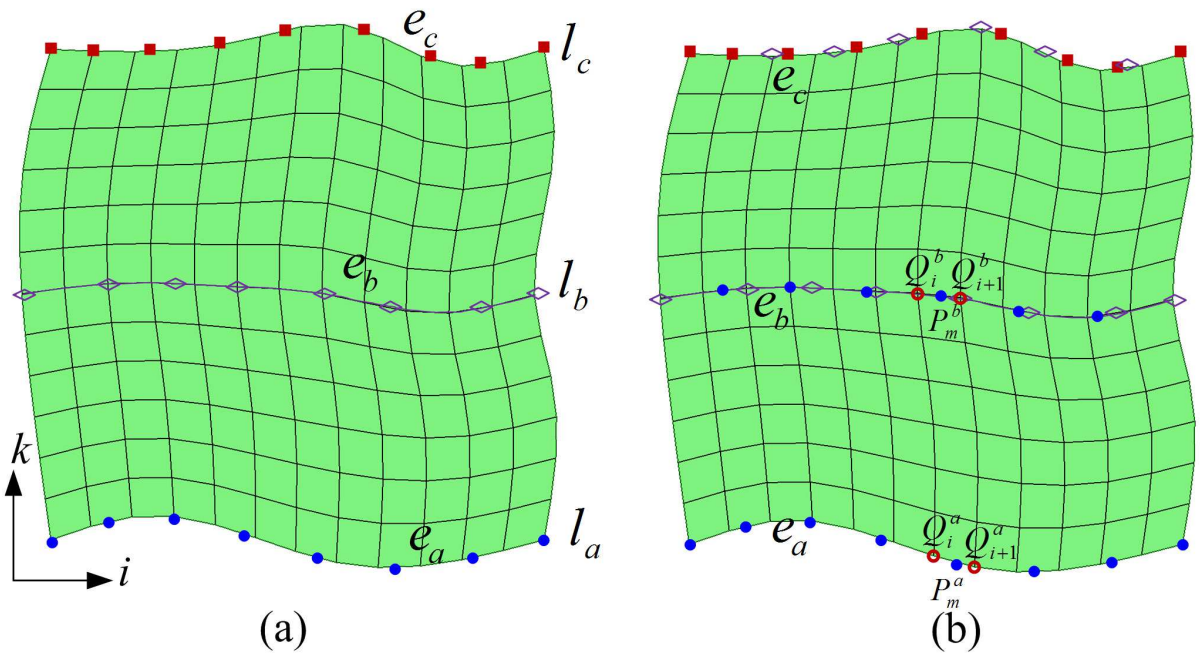

Figure 5.6: Propagation of bounding edge patches on the *linking* surfaces: (a)bounding edge patches represented with points; (b)bounding edge patch propagation between $l_a \rightarrow l_b$ and $l_b \rightarrow l_c$

should be generated on the *linking* surfaces prior to s-t edge patch imprinting. It is only necessary to use points and connectivity to represent the parametric space on the *linking* surfaces, which is used to guide the propagation of bounding edge patches between the sweeping levels.

The bounding edge patch propagation algorithm relies on the parametric space of *linking* surfaces: inputs are the bounding edge patches on a specific sweeping level, sweeping levels for s/t surfaces, *linking* surface parametric space $\boldsymbol{F}_{link}(i,k)$; outputs are propagated edge patches on the next sweeping level. For an arbitrary point $\boldsymbol{P}_m^a$ on an edge patch as Fig.5.6(b), it should be located between $\{i, k_a\}$ and $\{i+1, k_a\}$, namely, $\boldsymbol{P}_m^a$ can be

represented with $\boldsymbol{Q}_i^a$ and $\boldsymbol{Q}_{i+1}^a$.

$$\alpha = \frac{||\boldsymbol{P}_m^a - \boldsymbol{Q}_i^a||}{||\boldsymbol{Q}_{i+1}^a - \boldsymbol{Q}_i^a||} \tag{5.1}$$

$$\boldsymbol{P}_m^a = (1-\alpha) * \boldsymbol{Q}_i^a + \alpha * \boldsymbol{Q}_{i+1}^a \tag{5.2}$$

On the next sweeping level l_b, the parametric coordinates for $\boldsymbol{Q}_i^a$ and $\boldsymbol{Q}_{i+1}^a$ are changed into $\{i, k_b\}$ and $\{i+1, k_b\}$, respectively. The point $\boldsymbol{P}_m^b$ propagated from $\boldsymbol{P}_m^a$ can be computed as

$$\boldsymbol{P}_m^b = (1-\alpha) * \boldsymbol{Q}_i^b + \alpha * \boldsymbol{Q}_{i+1}^b \tag{5.3}$$

In this way, Figure 5.6(b) shows the propagation of bounding edge patches $e_a \rightarrow e_b$ between $l_a \rightarrow l_b$ and $e_b \rightarrow e_c$ between $l_b \rightarrow l_c$: faceting points denoted by blue circle dots on e_a are propagated onto e_b where the propagated edge patch of e_a at l_b sweeping level are denoted by blue circle dots. Blue circle dots and purple diamond dots on e_b represent the same edge patch. During the next propagation from l_b level to l_c level, purple diamond dots are used to represent the edge patch e_b and propagate e_b onto e_c where the propagated edge patch of e_b at l_c is represented with the purple diamond dots.

5.6 Interior Edge Patch Location within Other Loops

Since interior edge patches rely on the bounding edge patches in order to propagate to the next sweeping level, problems arise that which loop of bounding edge patches contain a specific interior edge patch. This happens when the propagation of bounding edge patches terminates since they reach the s/t surfaces, and their bound interior edge patches still

need to be propagated to the next sweeping level. The interior patch containment problem can be solved by the Ray Casting algorithm [58] as Section 5.6.2. In order to increase robustness for solving the patch containment problem, 3D edge patches are mapped onto the planar domain as Section 5.6.1.

5.6.1 Mapping of 3D patches onto planar domain

Often, loops of faceting points on geometric edges in the real applications are non-planar. In order to solve problems such as determining whether a loop of edge patches contains another edge patch or not during imprinting, 3D edge patches are projected onto the planar domain[70], but not necessarily the XY Cartesian plane. Note that edge patch imprinting can be done in 3D and mapping of 3D edge patches onto planar domain is only used to determine whether one loop of edge patches contains another loop.

The pseudo-area vector, $\mathbf{a}$, of a 3D loop of points $\{\mathbf{x}_i\}_{i=1,\cdots,n}$ is defined as

$$\mathbf{a} = \sum_{i=1}^{n} \mathbf{x}_i \times \mathbf{x}_{i+1} \tag{5.4}$$

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

$$\mathbf{n} = \frac{\mathbf{a}}{||\mathbf{a}||} \tag{5.5}$$

The planar domain of a 3D loop can be constructed as follows: first, the pseudo-normal of loop points $\mathbf{n}$ is computed; second,the computational position of $\mathbf{x}_i$ is defined as

$$\bar{\mathbf{x}}_i = \mathbf{x}_i - \langle \mathbf{x}_i, \mathbf{n} \rangle \mathbf{n} \tag{5.6}$$

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

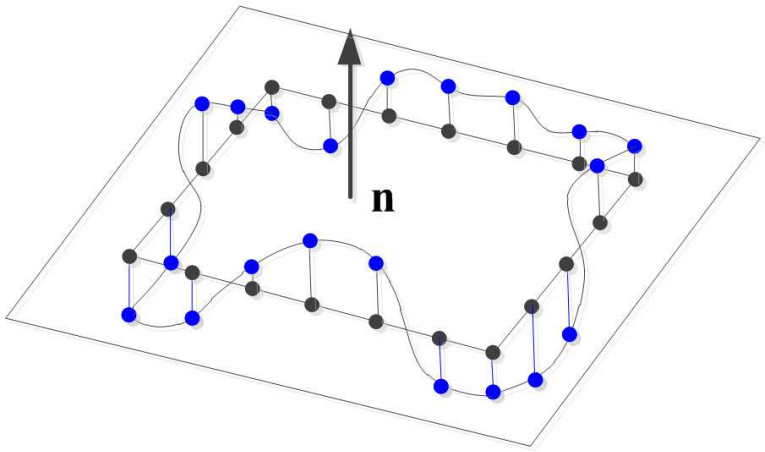

Figure 5.7: Mapping of 3D Edge Patches onto 2D planar Domain: 3D edge patches are represented with blue dots and 2D planar domain is denoted with black dots

the plane defined by the pseudo-normal. An example of mapping 3D edge patches onto 2D planar domain is shown in Fig.5.7.

5.6.2 Interior patch containment problem

The Ray Casting algorithm is originally used to determine whether a point is inside a polygon or not by casting a ray from the point and computing number of intersecting points. First whether the point in question is on the boundary or not needs to be checked. If it is not on the boundary, then cast a ray and compute the number of intersecting points: if the number of intersecting points is even, the point in question is outside the polygon. Otherwise, it is inside the polygon.

During multi-sweeping, there may exist multiple loops of bounding edge patches and it is required to determine which loop of bounding edge patches contains a specific interior edge patches so that interior edge patches can be correctly placed and there are no degenerated elements when sweeping the surface meshes through the volume. For example, Figure 5.8 shows that there are loops of bounding edge patches $\{E_0, E_1\}$ and $\{E_3\}$ and it

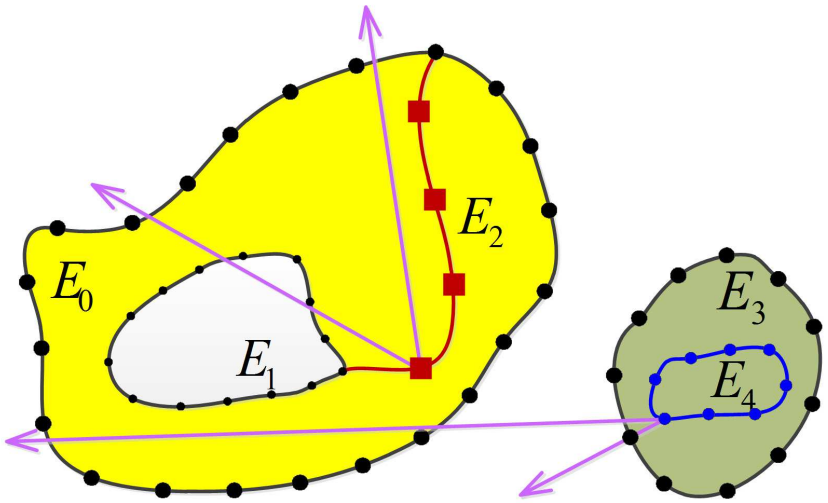

Figure 5.8: An example of patch containment problem with the bounding edge patches E_0, E_1, E_3 and undetermined edge patches E_2, E_4

needs to determine which loop of bounding edge patches contain the interior edge patch E_2 and E_4. By the Ray Casting algorithm, $\{E_0, E_1\}$ contains E_2 and $\{E_3\}$ contains E_4. Note that there is an impossible case that there are interior edge patches without any bounding edge patches: since an interior edge patch on one sweeping level has a bounding edge patch on that sweeping level and bounding edge patches always propagate between sweeping levels, interior edge patches will always have the bounding ones too.

5.7 Projection of interior edge patches onto another Surface

Since the bounding edge patches have already propagated to the next sweeping level along the *linking* surfaces as Section 5.5, their enclosed interior edge patches are needed to be propagated to next sweeping level so that curves on the s/t surfaces can be resolved. As matter of fact, those edges on the *source* and *target* surfaces must be reasonably placed

and imprinted on surfaces so that there are no distortions or degenerated elements when sweeping the surface meshes through volumes.

As it is known in Section 5.4, interior edge patches are not directly connected to the next sweeping level by the *linking* surfaces. They have to rely on the cage-based morphing and their loops of bounding edge patches in order to propagate to the next sweeping level: inputs are the interior edge patches on the current sweeping level, sweeping levels, their loops of bounding edge patches on the current sweeping level as well as the next sweeping level; outputs are the propagated interior edge patches on the next sweeping level.

In order to illustrate how to place those interior edge patches, we use Fig.5.5(a) as an example. Suppose it is required to propagate from the $l+1$ sweeping level to the $l+2$ sweeping level, inputs of the cage-based morphing problem is the bounding edge patch points $\{E_0^{l+1}, E_1^{l+1}\}$ and interior edge patch points $\{E_2^{l+1}, E_3^{l+1}\}$ on the sweeping level $l+1$, as well as the propagated bounding edge patch points $\{E_0^{l+2}, E_1^{l+2}\}$ on the sweeping level $l+2$, where E_0^{l+1} and E_1^{l+1} propagate to E_0^{l+2} and E_1^{l+2} respectively through the *linking* surfaces as Section 5.5. The goal is to find updated interior edge patch point positions E_2^{l+1} and E_3^{l+1} at new sweeping level $l+2$, namely, E_2^{l+2} and $\{E_4^{l+2}, E_5^{l+2}\}$ in Fig.5.5(b), respectively.

(1) **Binding:** The binding process of faceting points on interior edge patch E_2^{l+1} and E_3^{l+1} with faceting points on the bounding edge patches E_0^{l+1} and E_1^{l+1} can be achieved by using Equation (2.6) and *Harmonic Coordinates* as Refence.[22, 33]. It is used to solve φ_i with E_0^{l+1}, E_1^{l+1}, E_2^{l+1} and E_3^{l+1} as inputs, which are already known on the sweeping level $l+1$.

$$E_p^{l+1} = \sum_{i=1}^{m} \varphi_i(E_p^{l+1}) E_j^{l+1}, p = \{2,3\}, j = \{0,1\} \tag{5.7}$$

(2) **Interpolation:** With faceting points on E_0^{l+2} and E_1^{l+2} as inputs, faceting points of interior edge patch E_2^{l+1} and E_3^{l+1} on new sweeping level $l+2$ can be interpolated by using Equation (2.7) as follows

$$E_p^{l+2} = \sum_{i=1}^{m} \varphi_i(E_q^{l+1}) E_j^{l+2}, p = \{2,4,5\}, q = \{2,3\}, j = \{0,1\} \tag{5.8}$$

where E_p^{l+2} is the interior edge patches on the sweeping level $l+2$ which is propagated from E_q^{l+1} on the sweeping level $l+1$.

It is important to point out that the mapping function φ is solved by interpolating binding relationships between the bounding edge patches which can directly propagate by themselves and those interior edge patches which can not propagate 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 current sweeping level as inputs.

For the time complexity of edge patch propagation, the most expensive part is the binding process which is related to solving a ***PDE*** and can be done in $O(nlogn)$, while the remaining part is to interpolate interior edge patches, which can be done in $O(m * n)$, where m is the number of faceting points on the bounding edge patches and n is the number of faceting points on the interior edge patches.

There is one point to be noticed that: the approach of surface mesh mapping between two shapes described in Chapter 3 can not be employed to propagate interior edge patches between the sweeping levels since there does not exist the surface faceting for surface patches during edge patch propagation. The cage-based morphing, as a matter of fact, uses the bounding edge patches to guide the propagation of interior edge patches and those bounding edge patches can rely on the parametric spaces of *linking* surfaces to be

propagated between different sweeping levels.

5.8 Edge Patch Intersection/Overlap

When two edge patches are projected to the same sweeping level and are in the same neighbourhood on that sweeping level, they can either intersect cleanly in one or more points, or they can be made to overlap for part or all of their lengths on that sweeping level. Intersection or overlapping can also occur between an edge patch and a bounding edge patch on that sweeping level. Since edge patches are represented with a list of line segments, first intersection problem between two line segments is discussed in Section 5.8.1. The intersecting problem of edge patches happens when a propagated edge patch from previous sweeping levels intersects new edge patches on the current sweeping level, which is described in Section 5.8.2; meanwhile, overlapping of edge patches is discussed in Section 5.8.3. For efficiency, Section 5.8.2 and Section 5.8.3 are generally integrated together during implementation.

5.8.1 Intersection between two line segments

Suppose there are two line segments $\mathbf{e}^a$ whose two end points are $\mathbf{P}_0^a$ and $\mathbf{P}_1^a$, and $\mathbf{e}^b$ where two end points are $\mathbf{P}_0^b$ and $\mathbf{P}_1^b$, two line segments can be expressed as follows

$$\mathbf{e}^a = \mathbf{P}_0^a + t_a(\mathbf{v}_a) \tag{5.9}$$

$$\mathbf{e}^b = \mathbf{P}_0^b + t_b(\mathbf{v}_b) \tag{5.10}$$

where $\mathbf{v}_a = \mathbf{P}_1^a - \mathbf{P}_0^a$, $\mathbf{v}_b = \mathbf{P}_1^b - \mathbf{P}_0^b$, t_a and t_b are parameters for $\mathbf{e}^a$ and $\mathbf{e}^b$, respectively. When $\mathbf{e}^a$ intersects $\mathbf{e}^b$, we have

$$\mathbf{P}_0^a + t_a\mathbf{v}_a = \mathbf{P}_0^b + t_b\mathbf{v}_b \tag{5.11}$$

$$\tag{5.12}$$

By taking the cross product of each side with $\mathbf{v}_b$, the variable t_b can be eliminated and we have

$$t_a(\mathbf{v}_a \times \mathbf{v}_b) = (\mathbf{P}_0^b - \mathbf{P}_0^a) \times \mathbf{v}_b \tag{5.13}$$

There are three cases when processing intersections between $\mathbf{e}^a$ and $\mathbf{e}^b$

(1) Intersection at a single point: $(\mathbf{v}_a \times \mathbf{v}_b)$ on the left side in Equation 5.13 must be parallel to $((\mathbf{P}_0^b - \mathbf{P}_0^a) \times \mathbf{v}_b)$ on the right side, $(\mathbf{v}_a \times \mathbf{v}_b) \neq \mathbf{0}$ and $0 \leq t_a \leq 1$

$$t_a = \frac{|(\mathbf{P}_0^b - \mathbf{P}_0^a) \times \mathbf{v}_b|}{|\mathbf{v}_a \times \mathbf{v}_b|} \tag{5.14}$$

(2) Overlapping: If $\mathbf{v}_a \times \mathbf{v}_b = \mathbf{0}$, $(\mathbf{P}_0^b - \mathbf{P}_0^a) \times \mathbf{v}_b = \mathbf{0}$, then two line segments are collinear. If in addition, either $0 \leq (\mathbf{P}_0^b - \mathbf{P}_0^a) \cdot \mathbf{v}_b \leq \mathbf{v}_b \cdot \mathbf{v}_b$ or $0 \leq (\mathbf{P}_0^a - \mathbf{P}_0^b) \cdot \mathbf{v}_a \leq \mathbf{v}_a \cdot \mathbf{v}_a$, then two line segments are overlapping and two of $\{\mathbf{P}_0^a, \mathbf{P}_1^a, \mathbf{P}_0^b, \mathbf{P}_1^b\}$ are end points for the overlapping part.

(3) No intersection or overlapping: if it does not satisfy (1) and (2) described above, there is no intersection or overlapping.

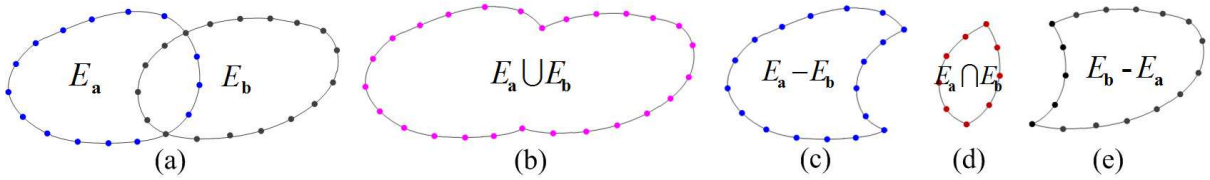

Figure 5.9: Intersection processing for edge patches: (a)patch E_a and patch E_b; (b) *Union* of patch E_a and patch E_b; (c)*Subtraction* of patch E_a and E_b; (d)*Intersection* of patch E_a and E_b; (e)*Subtraction* of patch E_b and E_a

5.8.2 Intersection processing of edge patches

After edge patches propagate from l_a sweeping level to l_b sweeping level, 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 the bounding patches into interior patches. Meanwhile, intersection processing of edge patches mandates that a geometric vertex should be placed there, so that new curves are matched between *source* and *target* surfaces.

While processing intersections of edge patches, the key problem is to resolve intersecting points on edge patches and determine which part of edge patches to be kept for the next edge patch propagation based on the propagation direction. An example for Boolean operations between edge patches is shown in Fig.5.10 where two edge patches E_a and E_b are given: *Union*, *Subtraction* and *Intersection*. Note that new points may be added and created at the intersecting locations between two edge patches.

Suppose there are two edge patches $E_a = \{e_i^a\}$ and $E_b = \{e_i^b\}$ where e_i^a and e_i^b are line segments on the edge patch E_a and E_b, respectively, our algorithm for processing edge patch intersection works as **Algorithm 2**.

For the time complexity of processing intersections of two edge patches, the general case is $O(n_1 + n_2)$ where n_1 and n_2 are the number of faceting points on two edge patches

Algorithm 2 Pseudo Code of processing intersections of two edge patches

Input:

(1). $E_a = \{e_i^a\}, \qquad i \in \{0, 1, \cdots, n_a\}$

(2). $E_b = \{e_i^b\}, \qquad i \in \{0, 1, \cdots, n_b\}$

Output: $\{\mathbf{P}_m, is\_overlap_m, is\_intersect_m\}$

function processIntersections(E_a, E_b)
1. **COMPUTE** the bounding boxes BOX_a and BOX_b for edge patches E_a and E_b, respectively.
2. $set_a = \emptyset$, $set_b = \emptyset$
3. **FOR EACH** line segment e_i^a in E_a
3.1. **IF** ($e_i^a \cap BOX_b \neq \emptyset$)
3.2. **ADD** e_i^a to set_a
4. **FOR EACH** line segment e_i^b in E_b
4.1. **IF** ($e_i^b \cap BOX_a \neq \emptyset$)
4.2. **ADD** e_i^b to set_b
5. **FOR EACH** line segment e_i in set_a
6. **FOR EACH** line segment e_j in set_b
7. **PROCESS** intersection between two line segments e_i and e_j and **ADD** results into $\{P_m, is\_overlap_m, is\_intersect_m\}$

since size(set_a)*size(set_b) $\ll (n_1 + n_2)$. The worst case is O($n_1 * n_2$) when two edge patches are fully overlapping.

5.8.3 Overlapping of edge patches

The overlapping problem of edge patches happens when a propagated edge patch from previous sweeping level is closely or exactly located on the bounding edge patches on current sweeping levels. Due to different topologies of *source* and *target* surfaces during multi-sweeping, there are different types for the edge patch overlapping: full overlapping and partial overlapping. Figure 5.10 shows an example of full overlapping and partial overlapping of edge patches: the propagation of bounding edge patch e_0^{tgt} will fully overlap the edge patch e_0^{src}. The full overlapping of edge patches happens for the edge patch e_1^{src}

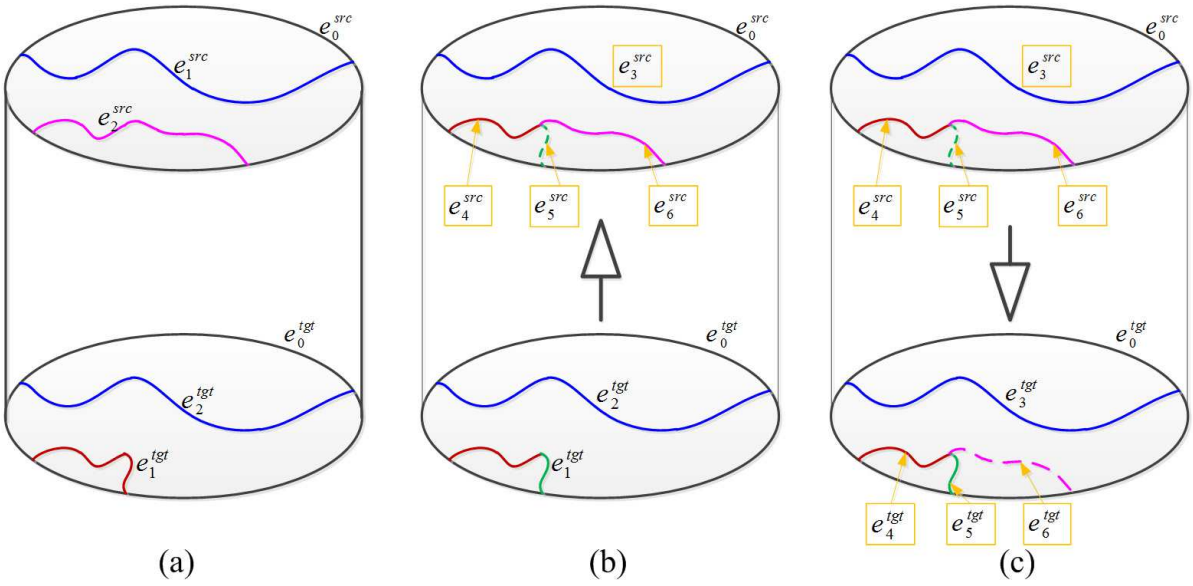

Figure 5.10: An example of different types of edge patch overlapping: (a)a multi-sweeping example; (b)imprinting results from bottom to the top; (c)imprinting results from top to the bottom from (b)

and e_2^{tgt} while the propagation of e_1^{tgt} can only partially overlap e_2^{src} on the *source* surfaces. The imprinting results are shown in Fig.5.10(c) with the imprinted edge patches (shown in dash lines) on the *source* and *target* surfaces.

Since the propagation of bounding edge patches is through the *linking* surfaces, the propagated bounding edge patches from previous sweeping level either exactly overlap the edge patches on current sweeping level or do not overlap those edge patches at all on current sweeping level. However, for the propagated interior edge patches from previous sweeping levels, they can either partially/fully overlap new edge patches on the s/t surfaces of current sweeping level or do not overlap those edge patches at all. In rare cases, the propagated interior edge patches fully overlap new edge patches on the s/t surfaces of current sweeping level even though they overlap ideally. There are two reasons for this case: one is that essentially, one edge patch only partially overlap another edge patch such

as Fig.5.11. In Fig.5.11(a), imprinting from the bottom to the top results in the edge e_1 partially overlap the edge e_4, e_2 intersects e_4 at a point and e_3 intersects e_4 at a point. Therefore, e_4 has to be cut so that edge matching problem between the bottom and the top can be solved. The same thing happens for the other way of imprinting. Hence, after two ways of imprintings, edge patches between the *source* and *target* surface are matched such as Fig.5.11(c). The other one is that one edge patch may not overlap another edge patch due to the numeric error from the calculation or geometric error. This is pretty common during imprinting and can make algorithms fail or bad results. For example, after imprinting, one edge patch may be pretty close to another edge patch on a surface and distorted elements with poor mesh quality have to be created between two edge patches such as Fig.5.12. In order to avoid that, tolerance control based on the mesh element size specified by users is used to improve the quality of imprinting. If the propagated edge patches are closer than some fraction of the user-specified mesh size during imprinting such as Fig.5.11(b), they overlap.

Note that previous methods use edge mesh node for representing edge loops, there is a disadvantage for their methods: there may not exist a mesh node at intersecting location of two edge patches and mesh nodes have to move so that a mesh node is placed at the intersecting location. Also, these methods may constrain the number of allowed intersections on an edge patch due to the number of nodes on the edge patch.

5.9 Edge Patch Decomposition

As Section 5.8 describes, the intersection processing of edge patches mandates that a geometric vertex should be placed there, so that new curves are matched between the *source* and *target* surfaces. In the meantime, edge patches need to be decomposed as well. The edge patch decomposition happens when a propagated edge patch from previous

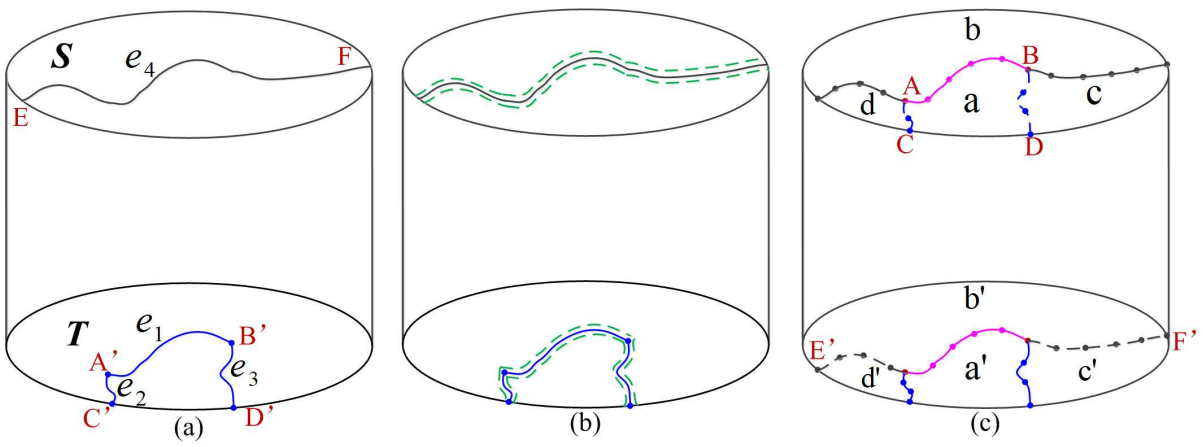

Figure 5.11: An example of partial merging of edge patches: (a)a volume with edges on the *source* and *target* surfaces; (b)ranges of possible overlapping based on mesh element size; (c)imprinting results

sweeping levels intersects or partially overlaps a new edge patch on the current sweeping level.

The algorithm of edge patch decomposition relies on results from intersections of edge patch: inputs are edge patches to be decomposed and intersecting locations between those edge patches; outputs are a list of new edge patches. The edge patch decomposition cuts those edge patches with the intersecting locations. Note that edge patch decomposition does not happen when the intersecting locations meet at the existing vertices on the edge patches.

For example, in Fig.5.11, edge patches e_1, e_2 and e_3 are propagated onto the top surface. They intersect with the existing edge patch e_4 which subsequently needs to be decomposed. Hence, one of decomposed pieces of e_4 matches e_1 on the bottom surface and the intersecting point A' between e_1 and e_2 match new vertex A on e_4. So do vertices B',C',D',E and F. The final effect is that curves and vertices on the *target* surfaces are embedded into the *source* surface meshes and vice versa. Note that in some cases, there does not exist a point at the intersecting location between two edge patches. In this work,

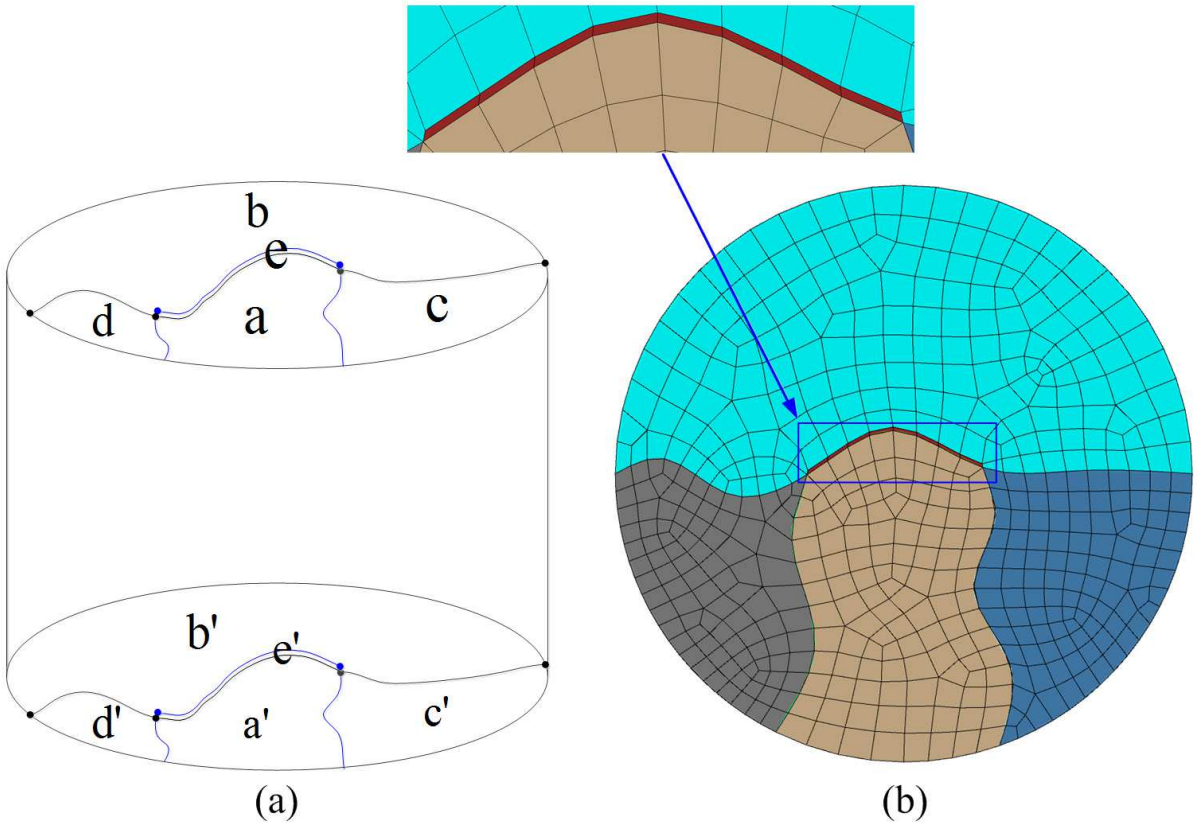

Figure 5.12: An example of Distorted elements due to numeric errors: (a)overlapping problem of edge patches due to numeric errors; (b)distorted mesh element created due to the numeric error with a zoom-in view

a new point is simply inserted at the intersecting location: even though the accuracy for representing an edge patch can be affected by the number of points, the number of points on an edge patch does not need to match that of the edge patch on the next sweeping level and those points are only used to represent the edge patch to be propagated.

Meanwhile, the edge patch correspondence can be recorded and tracked during this process, to aid in matching surface patches later. After edge patch decomposition, new edge patches are created and there exists the correspondences between the split edge patches and those edge patches where the split ones are originally from. For example, in Fig.5.10(b), after imprinting from bottom to the top, e_2^{src} in Fig.5.10(a) is split into e_4^{src} and e_6^{src} in

Fig.5.10(b). Meanwhile, a new edge patch e_5^{src} is created. It is known that e_4^{src} and e_5^{src} in Fig.5.10(b) are originally from e_1^{tgt} and the correspondence $(e_1^{tgt} \leftrightarrow \{e_4^{src}, e_5^{src}\})$ should be recorded. In Fig.5.10(c), after imprinting from top to the bottom from Fig.5.10(b), e_1^{tgt} in Fig.5.10(a) is split into e_4^{tgt} and e_5^{tgt} and there is a new created edge patch e_6^{tgt}. Based on the last tracked correspondence result $(e_1^{tgt} \leftrightarrow \{e_4^{src}, e_5^{src}\})$, the new correspondences are made: $(e_4^{tgt} \leftrightarrow e_4^{src})$ and $(e_5^{tgt} \leftrightarrow e_5^{src})$. Since the new created edge patch e_6^{tgt} is originally from e_6^{src} on the last imprinting in Fig.5.10(b), the new correspondence $(e_6^{tgt} \leftrightarrow e_6^{src})$ is made.

5.10 Surface Decomposition

If there are imprinted edge patches on a specific surface (*source* or *target*), that means curves and surfaces where the imprinted edge patches are originally from can be partially or fully resolved on this surface and new surface patches need to be created on this surface. When imprinting from the *target* surfaces to the *source* surfaces, there are only imprinted edge patches from *target* surfaces onto the *source* surfaces and vice versa. In other words, the *source* surfaces are partitioned only when imprinting from the *target* surfaces to the *source* surfaces and vice versa. Since there are bounding and interior edge patches during propagation between different sweeping levels, surfaces and their propagated edge patches(*source* or *target*) may be partitioned with the bounding edge patches or interior edge patches. Since the bounding edge patches propagate by relying on the *linking* surfaces, they can only overlap partially or fully with the bounding edge patches on the destination sweeping level. Hence, it is impossible for the *source* or *target* surfaces to be partitioned with the bounding edge patches. However, it is totally different for interior edge patches regarding the surface partition: since interior edge patches are not directly connected to the next sweeping level and can only propagate based on the

cage-based morphing between different sweeping levels, which relies on the bounding edge patches as well, there are two cases for interior edge patches, namely, intersection of edge patches and overlapping of edge patches. The surface partition is resolved after resolution of those two things.

(1). *Intersection of edge patches*: surface partition happens when there exists intersections between the propagated edge patches and edge patches on surfaces from the new sweeping levels. Examples are shown in Fig.5.13(e), Fig.5.13(f) and Fig.5.13(g).

(2). *Overlapping of edge patches*: There is no surface partition when there is only overlapping of edge patches including full and partial overlapping of edge patches such as Fig.5.13(c) and Fig.5.13(d). However, if there is a combination of intersection and overlapping of edge patches, surfaces are needed to be partitioned with imprinted edge patches. Examples are shown in Fig.5.13(e) and Fig.5.13(f).

In summary, the *source* and *target* surface partition occurs only when there are imprinted edge patches on the interior of surfaces, which excludes imprinted edge patches on the boundaries of surfaces: inputs are imprinted edge patches from previous sweeping levels, new edge patches and s/t surfaces on the current sweeping level; outputs are new s/t surface patches on the current sweeping level.

The s/t surfaces may be partitioned with a list of faceting points on the imprinted edge patch. However, this results in a curve represented with a list of faceting line segments, which requires that boundary mesh nodes should be inserted on the faceting point locations on the edge patch at the stage of surface patch meshing. More boundary mesh nodes may be inserted between two faceting points on an edge patches. That is too restrictive for surface patch meshing and creates too dense meshes if users only need very coarse meshes since there are at least the same number of mesh nodes as faceting points on the edge

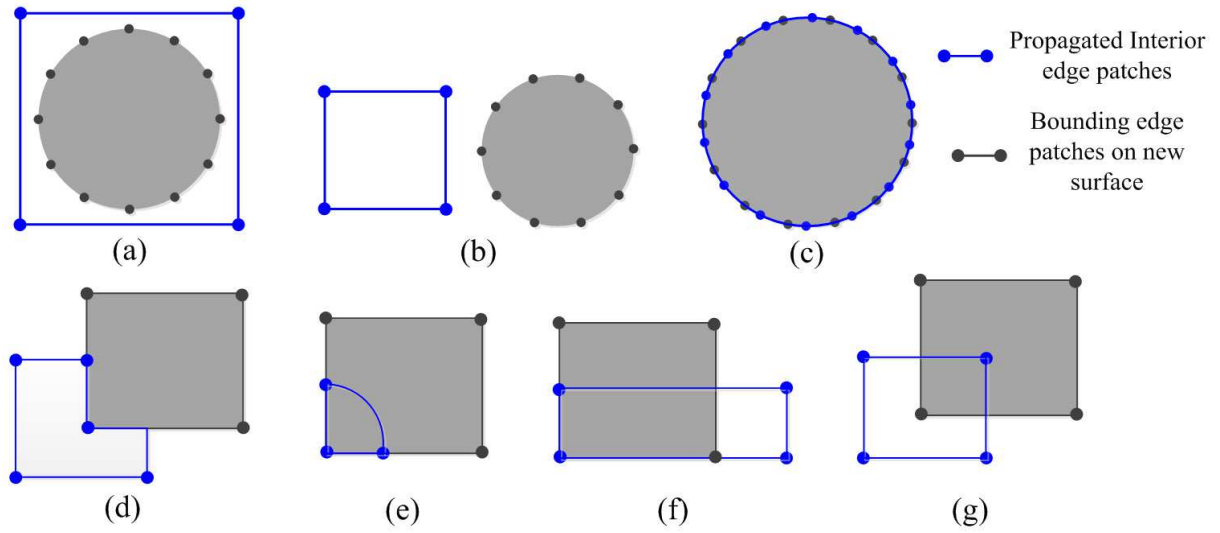

Figure 5.13: Examples of different cases of surface partition: (a)no intersection or overlapping of edge patches: no surface partition; (b)no intersection or overlapping of edge patches: no surface partition; (c)full overlapping of edge patches: no surface partition; (d)partial overlapping of edge patches: no surface partition; (e)a combination of intersection and overlapping of edge patches: partition surface; (f)a combination of intersection and overlapping of edge patches: partition surface; (g)intersection with edge patches: partition surface

patch. Therefore, in this work, the smooth splines are first constructed by using a list of faceting point coordinates on the edge patch. Then s/t surfaces are partitioned with those splines.

5.11 Integration of *s*-*t* imprinting parts

During imprinting, edge patches on the *target* surfaces are imprinted onto the appropriate *source* surfaces and vice versa. Afterwards, edge patch and surface decomposition are needed to resolve intersections/overlaps of vertex and edges between s/t surfaces: the topology of *source* surfaces is modified as a result of imprinting of *target* surfaces onto *source* surfaces and vice versa.

Prior to imprinting, the sweeping levels should be built for each *source* and *target* surface. Then edge patch imprinting is performed between s/t surfaces: edge patch identification, bounding edge patch propagation, determination of interior edge patches within loops, projection of interior edge patches onto s/t surfaces, intersection processing of edge patches, edge patch decomposition and s/t surface partition.

Our algorithm of edge patch imprinting between the *source* and *target* surfaces works as follows: suppose there are the *source* surfaces $\boldsymbol{F}_{src}$, *target* surfaces $\boldsymbol{F}_{tgt}$, *linking* surfaces with parametric space $\boldsymbol{F}_{link}(i,k)$, edge patches of s/t surfaces on all the sweeping levels E_p^l and imprinting direction $ImprintDir$.

$$ImprintDir = \{TgtToSrc, SrcToTgt\} = \{+1, -1\}$$

The outputs of edge patch imprinting are a list of new *source* and *target* surface patches.

(1) Start at the *target* surfaces on the sweeping level 0 and perform imprinting from *target* surfaces towards the *source* surfaces

$$\boldsymbol{F}'_{src} = imprintEdgePatches(\boldsymbol{F}_{src}, \boldsymbol{F}_{tgt}, \boldsymbol{F}_{link}, TgtToSrc, E_p^l)$$

(2) Start at the *source* surfaces on the sweeping level $L-1$ and perform imprinting from *source* surfaces towards the *target* surfaces

$$\boldsymbol{F}'_{tgt} = imprintEdgePatches(\boldsymbol{F}'_{src}, \boldsymbol{F}_{tgt}, \boldsymbol{F}_{link}, SrcToTgt, E_p^l)$$

After imprinting, the surface patch matching and interval assignment problem for edges of *source* and *target* surfaces can be solved since each *source* surface patch matches exactly one *target* surface patch and there exists the matching of edge patches for each paired *source* and *target* surface. Note: volumes are not partitioned at all during imprinting; 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, surface mesh quality on those surfaces is guaranteed to be good since cage-based morphing places the

Algorithm 3 Pseudo code of edge patch imprinting

Input:

(1). $\boldsymbol{F}_{src} = \{\boldsymbol{F}^{i_1}_{geom}, i_1 = 1, \cdots, m_1\}$

(2). $\boldsymbol{F}_{tgt} = \{\boldsymbol{F}^{i_2}_{geom}, i_2 = 1, \cdots, m_2\}$

(3). $\boldsymbol{F}_{link} = \{\boldsymbol{F}^{i_3}_{geom}, i_3 = 1, \cdots, m_3\}$ with global consistent $\{i, k\}$ space.

(4). $\{patch^l_p\}, l = \{0, 1, \cdots, L-1\}$

(5). $\boldsymbol{D} = \{TgtToSrc, SrcToTgt\} = \{+1, -1\}$

Output: $\{\boldsymbol{F}'_{src}/\boldsymbol{F}'_{tgt}\}$

function imprintEdgePatches($\boldsymbol{F}_{src}, \boldsymbol{F}_{tgt}, \boldsymbol{F}_{link}(i,k)$,$D$,$E^l_p$)

1. **IF** $\boldsymbol{D} = TgtToSrc \quad l_{start} = 0, \quad l_{end} = L-1, \quad Dst = src$
2. **ELSE** $\quad l_{start} = L-1, \quad l_{end} = 0, \quad Dst = tgt$
3. **FOR** $l = \{l_{start}, \cdots, l_{end}\}$

3.1. $l_{next} = l_{start} + \boldsymbol{D}$

3.2. **IDENTIFY** bounding patches $\{E^l_{B,m}\}$ and interior patches $\{E^l_{I,n}\}$

$$\{E^l_{B,m}, E^l_{I,n}\} = identifyPatches(E^l_p, \boldsymbol{F}_{link}(i,k))$$

3.3. **PROPAGATE** bounding edge patches to the l_{next} level

$$E'^{l_{next}}_{B,m} = propagateBoundingPatches(E^l_{B,m}, \boldsymbol{F}_{link}(i,k))$$

3.4. **DETERMINE** interior patches within loops on the l sweeping level

$$Loop_n = determineLoops(E^l_{B,m}, E^l_{I,n})$$

3.5. **PROJECT** interior patches to the l_{next} level

$$E'^{l_{next}}_{I,n} = projectInteriorPatches(E^l_{I,n}, Loop_n)$$

3.6. **PROCESS** intersections of edge patches

$$\{Q\} = processIntersections(E^{l_{next}}_{B,m}, E^{l_{next}}_{I,n}, E^{l_{next}}_p)$$

3.7. **DECOMPOSE** edge patches

$$E^{l_{next}}_p = decomposePatches((E^{l_{next}}_{B,m}, E^{l_{next}}_{I,n}, E^{l_{next}}_p, \{Q\})$$

3.8. **DECOMPOSE** surface patches and **ADD** them into $\boldsymbol{F}_{Dst}$

$$\boldsymbol{F}_{Dst} \leftarrow decomposeSurfs(\boldsymbol{F}_{Dst}, E^{l_{next}}_p)$$

4. **RETURN** $\{\boldsymbol{F}'_{Dst}\}$

propagated edge patches appropriately constrained by using the *linking* surfaces and it has an important property: local deformation.

For the time complexity of imprinting of edge patches consisting of edge patch propagation, intersection process of edge patches and surface partition, it can done in $O(nlogn)$ where n is the number of points on the edge patches since the most expensive part is interior edge patch projection which is described in Section 5.7.

5.12 Surface Matching between s/t Surface Patches

For multi-sweeping, the s/t surfaces are reduced to multiple ***1-1*** paired s/t surface patches when two ways of edge patch imprinting have been done. The *source* and *target* surface patches need to be matched (***1-1***) so that our method for surface mesh mapping described in Chapter 3 can be applied to generate meshes for the *target* surface patches mapped from *source* surface patches and vice versa. However, interior nodes inside volumes between two matched *source* and *target* surface patches are not located at this stage. They can be located by using our method described in Chapter 4.

Based on the imprinting results, our method for matching the *source* and *target* surface patches can proceed as follows: suppose there is a set of *source* surface patch $F_i^{src}, (i = 0, \cdots, m)$, a set of *target* surface patch $F_j^{tgt}, (j = 0, \cdots, m)$ and edge patch imprinting results; outputs are ***1-1*** pairs of *source* and *target* surface patches.

(1) Pick and remove a *target* surface patch T from the set of *target* surface patches $\mathbf{F}^{tgt}$

(2) Get new bounding edge patches E_T of *target* surface patch T.

(3) Find the edge patch propagation of E_T such that there exists a *source* surface patch S with its bounding edge patches E_S, which can propagate to E_T. Note that this

can be obtained from the imprinting results in Section 5.11 and thus avoid extra computational costs.

$$E_S \leftrightarrow E_T$$

(4) Update the set of *source* surface patches $\mathbf{F}_i^{src}, (i = 0, \cdots, m)$ and matching result

$$F^{src} = F^{src} - S$$

$$Map(S) = T \text{ and } Map(T) = S$$

(5) Repeat this process until

$$\mathbf{F}^{tgt} = \phi$$

The pseudo code for matching s/t surface patches is described in **Algorithm** 4. Whether one pair of *source* and *target* surface patch is matched or not is based on whether edge patches on one surface propagated to those on the other surface. If so, two surface patches are matched. Otherwise, they are not. For the time complexity of surface patch matching, it can be done in the linear time since imprinting of edge patches has already produced whether two edge patches propagated to each other or not.

5.13 Surface Mesh Generation for *S-T* Surfaces

After *s-t* edge patch imprinting, two groups of surface patches have already been generated, namely, *source* surface patch group and *target* surface patch group. Between s/t surface patch groups, every new *source* surface patch should match exactly one *target* surface patch. Before meshing the surface patches, interval assignment problem for edge patches should be solved: equality constraints for edge patches between ***1-1*** paired surface patches must be enforced, which is critical to generate the valid meshes for all the surface patches. Meanwhile, sweeping requires that either s or t surface patch group must be meshed first

Algorithm 4 Pseudo Code of matching s/t surface patches

Input:

(1). $\mathbf{F}_i^{src}, (i = 0, 1, \cdots, m)$

(2). $\mathbf{F}_i^{tgt}, (i = 0, 1, \cdots, m)$

(3). $Map(\mathbf{E}_{j_1}^{tgt}) = \mathbf{E}_{j_2}^{src}$ from imprinting results

Output: $Map(\mathbf{F}_i^{src}), Map(\mathbf{F}_i^{tgt})$

function matchSurfPatches($\mathbf{F}_i^{src}$, $\mathbf{F}_i^{tgt}$)

1. **FOR EACH** *target* surface patch $\mathbf{F}_i^{tgt}$.
2. **EXTRACT** edges patches $\mathbf{E}_{i_1}^{tgt}$ from $\mathbf{F}_i^{tgt}$.
3. **GET** the corresponding *source* edge patches $\mathbf{E}_{j_1}^{src}$ from imprinting results

$$\mathbf{E}_{j_1}^{src} = Map(\mathbf{E}_{i_1}^{tgt})$$

4. **Find** the *source* surface patch $\mathbf{F}_j^{src}$ with edge patches $\mathbf{E}_{j_1}^{src}$.

$$Map(\mathbf{F}_j^{src}) = \mathbf{F}_i^{tgt}, \qquad Map(\mathbf{F}_i^{tgt}) = \mathbf{F}_j^{src}$$

with an existing surface mesh generator, which are then mapped onto the other surface patch group. The surface mesh mapping method described in Chapter 3 is employed to map the surface meshes between s/t surface patch group. Figure 5.14 shows an example of generating surface meshes on the *target* surface patches mapped from the *source* surface patches.

5.14 Examples and Results

The first example presents *s-t* imprinting and *target* surface mesh generation for a solid model with multiple *source* and *target* surfaces. Even though the volume(shown in Fig.5.15(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. This is due to facts that unlike ***1-1*** sweeping, the bounding edges on one *source* surface are

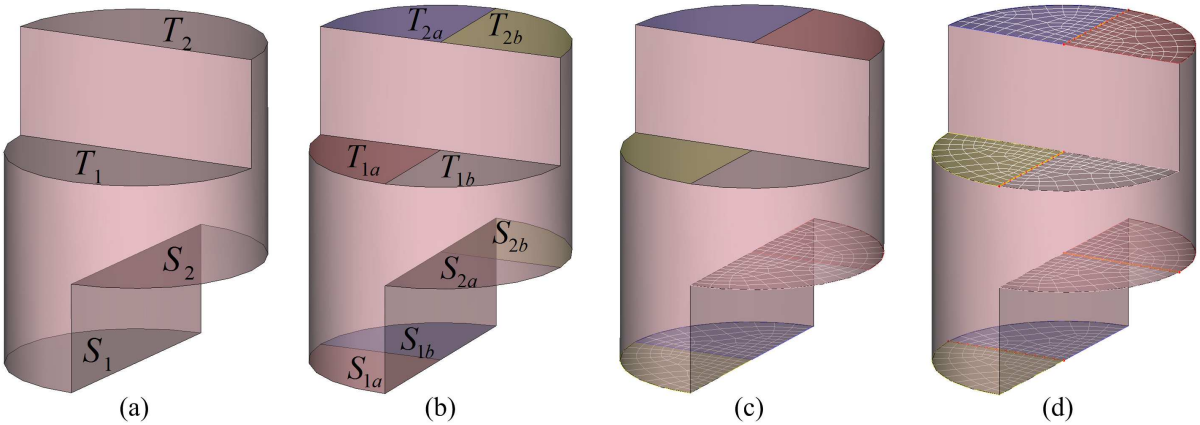

Figure 5.14: An example of mapping *source* meshes onto the *target* surface patches: (a)geometry model; (b)imprinting results: $S_{1a} \rightarrow T_{1a}$, $S_{1b} \rightarrow T_{2a}$, $S_{2a} \rightarrow T_{1b}$ and $S_{2b} \rightarrow T_{2b}$; (c)quadrilateral meshes on the *source* surface patches; (d)quadrilateral meshes on the *target* surface patches mapped from (c)

not necessarily connected to the same *target* surface through *linking* surfaces during multi-sweeping. Meanwhile, the edge patch 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.5.15(b) with two zoom-in views in Fig.5.15(c) and Fig.5.15(d): every new *source* surface patch has its corresponding new *target* surface patch. The *source* surface meshes are shown in Fig.5.15(e). By mapping *source* surface meshes onto the *target* surface patches, the resulting *target* surface meshes are represented in Fig.5.15(f). Note that in Fig.5.15(e) and Fig.5.15(f), the *linking* surfaces are removed for a better view.

The second example shows the edge patch imprinting and surface mesh generation for a solid with a varying hole and complicated internal structure(shown in Fig.5.16(a2)) where the typhoon-like through hole inside the volume is isolated for a better view shown in Fig.5.16(a4). 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

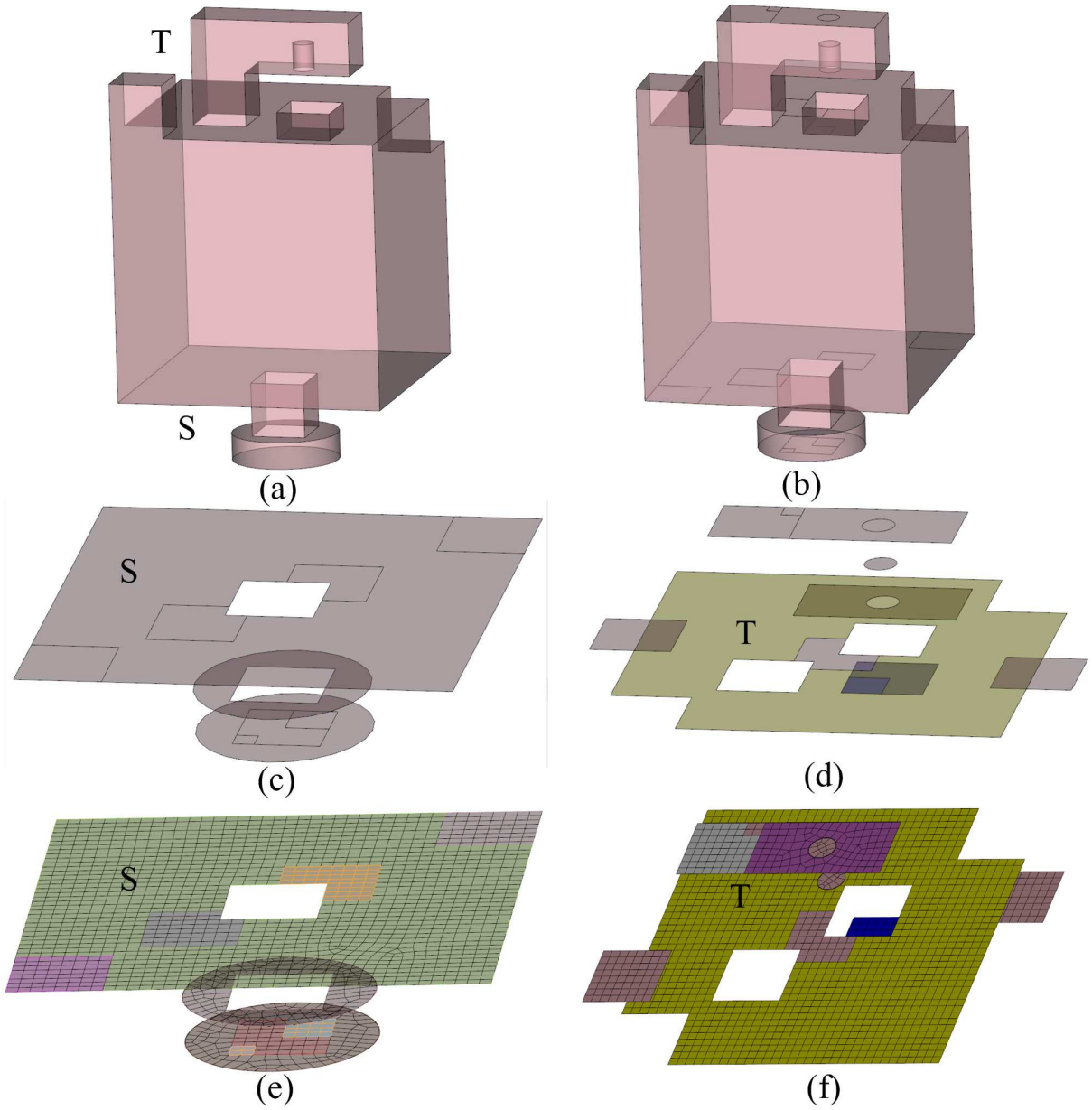

Figure 5.15: An example of matching the *source* and *target* surfaces and resulting surface meshes: (a)a geometric model; (b)imprinting results on the *source* and *target* surfaces; (c)partitioned *source* surface patches; (d)partitioned *target* surface patches; (e)*source* surface meshes; (f)*target* surface meshes mapped from (e) by morphing[16, 17]

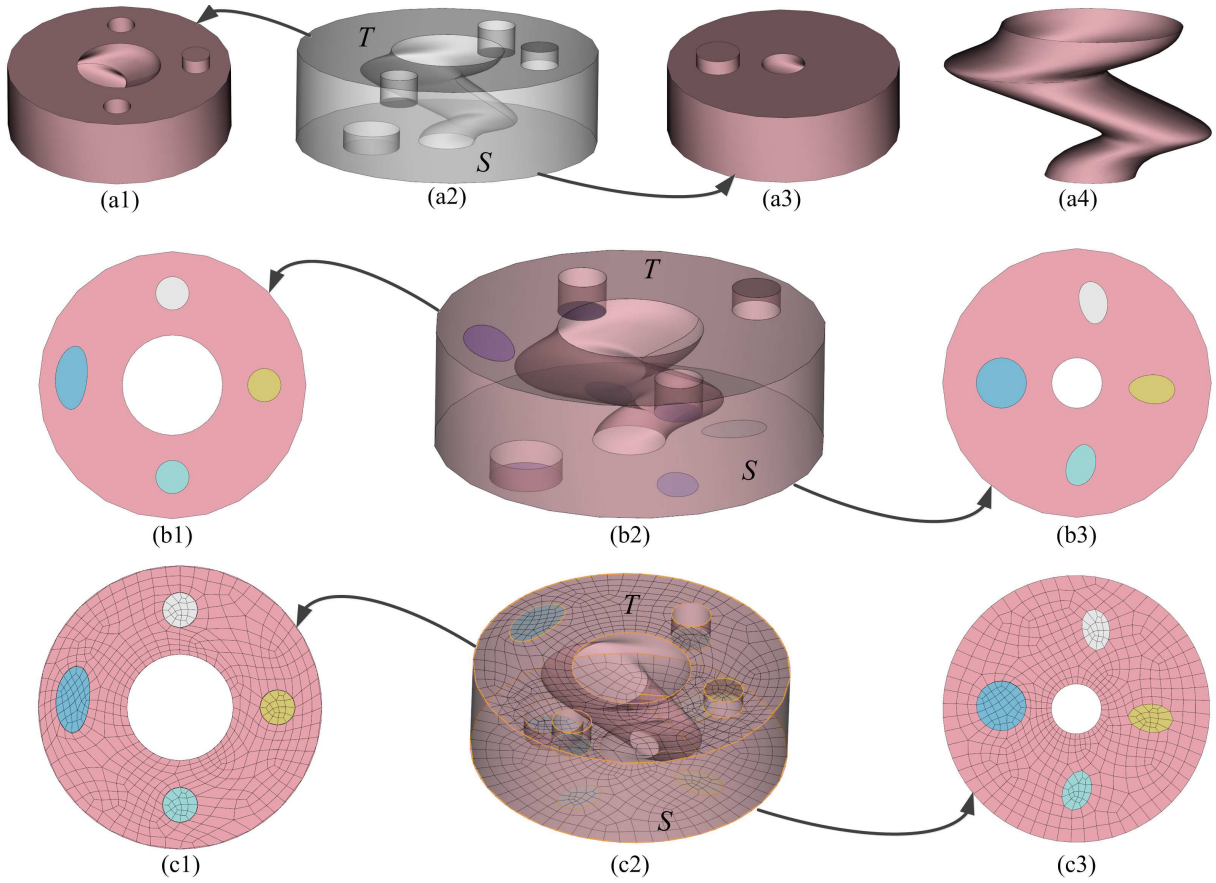

Figure 5.16: An example of generating surface meshes and matching the *source* and *target* surfaces by imprinting edge patches: (a1-a4)a geometric model where a typhoon-like hole inside is isolated and shown in (a4); (b1-b3)the imprinting results where *source* surface patches are shown in (b3) and *target* surface patches are shown in (b1); (c1-c3)surface meshes on the *source* and *target* surfaces where the *source* surface meshes are shown in (c3) and *target* surface meshes shown in (c1) are generated by mapped from (c3) by morphing[16, 17]

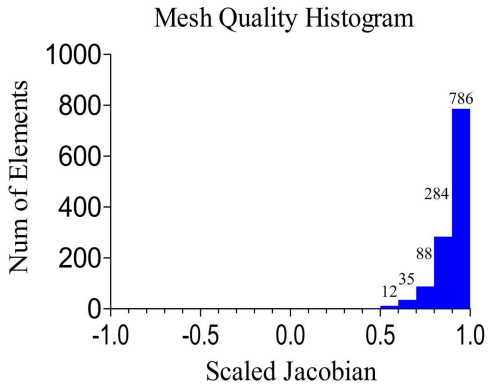

Figure 5.17: Mesh quality histogram for Fig.5.16

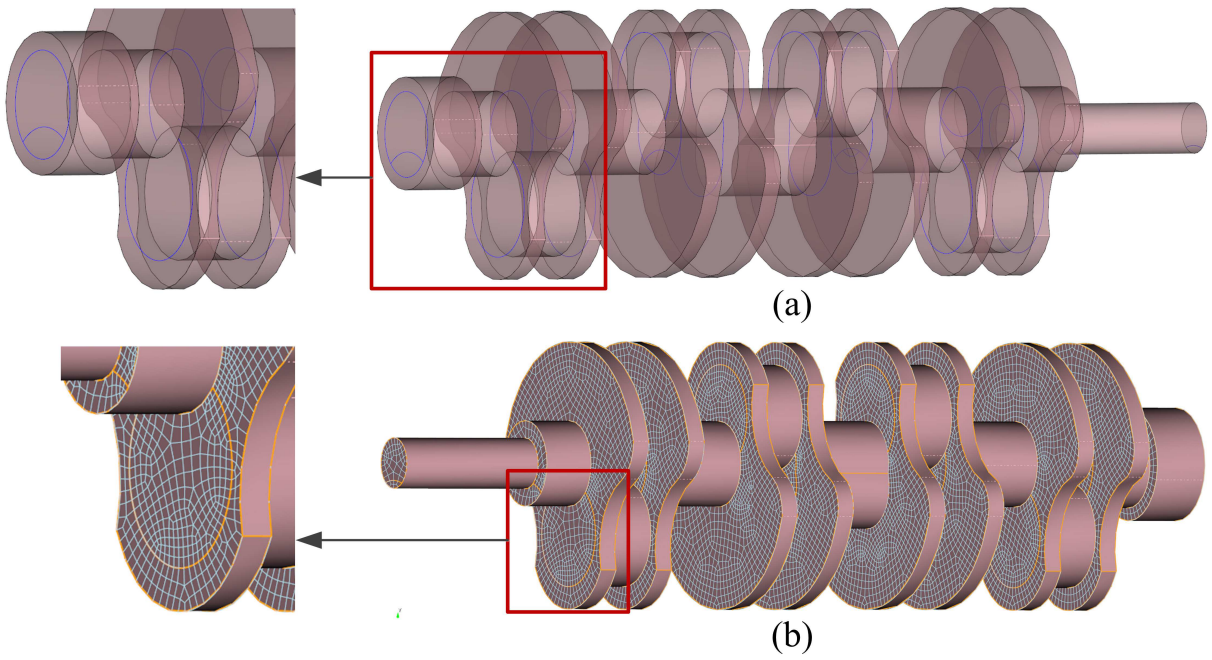

Figure 5.18: Surface mesh generation for crankshaft by imprinting: (a)the geometric model with imprinted edge patches(denoted by blue curves); (b)the *target* surface meshes by morphing[16, 17]

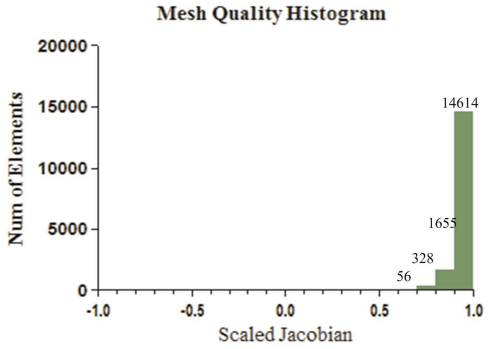

Figure 5.19: Mesh quality histogram for Fig.5.18

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 typhoon-like through hole between *source* and *target* surfaces, which creates inverted volume elements inside volumes during sweeping. 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. The imprinting results by our method is shown in Fig.5.16(b2). Figure 5.16(c3) presents the *source* surface meshes and the resulting *target* surface meshes are generated by our morphing methods[16, 17](shown in Fig.5.16(c1)). The surface mesh quality histogram is shown in Fig.5.17 where good mesh quality has been achieved.

An crankshaft example from the automobile engine is presented in Fig.5.18. This example contains many cylinders of which some have the same size but which are not directly connected through the *linking* surfaces. The resulting new model with partitioned *source* and *target* surfaces is presented in Fig.5.18(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.5.18(b)). The surface mesh quality histogram is shown in Fig.5.19 where good mesh quality has

been achieved.

5.15 Summary

In this chapter, an edge patch imprinting method based on the cage-based morphing for multi-sweeping problems has been developed. First, sweeping levels are built based on the parametric space of *linking* surfaces and breadth-first search algorithm of directed graph. Then imprinting of edge patches is performed between the *source* and *target* surfaces. Two ways of edge patch imprinting between s/t surfaces results in that *target* surface mesh generation during multi-sweeping is reduced to multiple ***1-1*** paired surface mesh mapping between the s/t surfaces and intersections/overlaps of curves and vertices between s/t surfaces during multi-sweeping have been resolved. Since a list of *source* and *target* surface patches have been generated, they need to be matched so that our surface mesh mapping algorithm described in Chapter 3 can be applied to generate the *target* surface meshes with good mesh quality, which is crucial for generating high quality swept volume meshes.

6 STRUCTURED MESH GENERATION ON THE LINKING SURFACES

In this chapter, the automatic corner assignment for structured mesh generation on the *linking* surfaces by *Submapping* is described, which is very important to the sweeping since it guides the mesh sweeping between the *source* and *target* surfaces. In Section 6.1, problems of vertex classification in the existing *Submapping* methods are stated, which leads to our improved vertex classification algorithm by *linear programming*($\boldsymbol{LP}$) in Section 6.2. The subsequent steps of *Submapping* are depicted as: the boundary discretization and interior node interpolation in Section 6.3. In the end, examples are provided in Section 6.4 to verify our improved corner assignment for *Submapping*.

6.1 Problem Statement

The surface vertex type for *submapping* is defined as the classification of the topology of a vertex bounding a structured all-quadrilateral surface mesh by the number of quadrilaterals sharing that vertex (see Fig.2.10). Let θ_i be an internal angle between two adjacent geometric curves at a vertex i on a surface. The initial vertex classification $\overline{\alpha_i}$ for a simply-connected geometry can be defined as follows

$$\overline{\alpha_i} = \lfloor 2(1 - \frac{\theta_i}{\pi}) \rceil \tag{6.1}$$

where $\lfloor * \rceil$ is an integer operator which returns the nearest integer to a float value. Then the surface vertex type can be defined as follows

$$\overline{\alpha_i} = \begin{cases} 1 & \text{for } \boldsymbol{END} \text{ vertex} \\ 0 & \text{for } \boldsymbol{SIDE} \text{ vertex} \\ -1 & \text{for } \boldsymbol{CORNER} \text{ vertex} \\ -2 & \text{for } \boldsymbol{REVERSAL} \text{ vertex} \end{cases} \tag{6.2}$$

For a simply-connected polygon with discrete internal vertex angles θ_i, we have

$$\sum_{i=1}^{N} \theta_i = 2\pi \tag{6.3}$$

For a submapped surface which is a simply-connected polygon, the sum of $\overline{\alpha_i}$ value must be equal to 4.

$$0 * S + 1 * E + (-1) * C + (-2) * R = 4 \tag{6.4}$$

where S, E, C and R are the number of ***SIDE***, ***END***, ***CORNER*** and ***REVERSAL*** vertices, respectively. For a submapped surface which is multiply-connected with the g holes, the sum of vertex types on the surface is

$$\sum_{i=1}^{N} \overline{\alpha_i} = 0 * S + 1 * E + (-1) * C + (-2) * R = 4 - 4g \tag{6.5}$$

Equation (6.5) can be interpreted as a constraint on vertex types for a surface which is to be submapped. In practice, automatic corner assignment on a surface may not result in the surface being submapable, because it fails to statisfy the constraint (6.5). For one thing, it is pretty common that angles between two adjacent edges on a surface are not exactly an integer multiple of 0.5π, namely, the actual or continuous angles really are ambiguous.

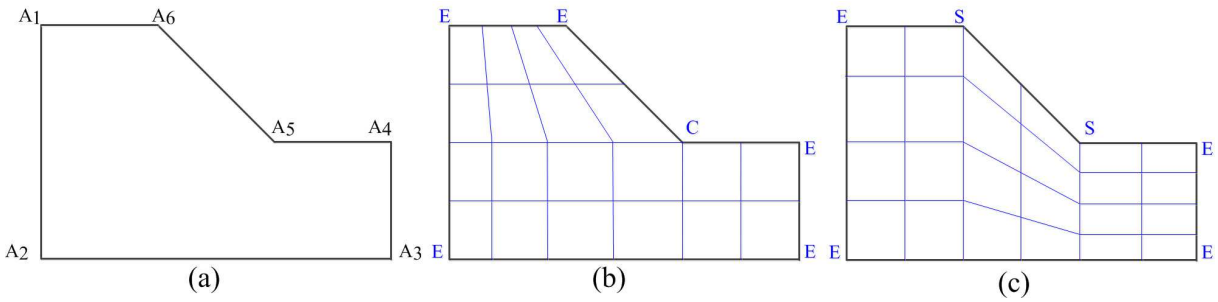

Figure 6.1: A failed example of vertex classification during *Submapping* by the heuristic method: (a)geometry; (b)Correct vertex classification; (c)Another way of correct vertex classification

In this case, the heuristic classification of vertices may lead to unacceptable results which do not satisfy Equation (6.5). For another thing, the continuous angle is one thing (e.g., it may be classified as type-***SIDE*** purely based on angles) while the discrete mesh may be another thing (e.g. it may be converted from type-***SIDE*** to type-***CORNER*** due to the *Submapping* constraint as Equation (6.5)). Hence, current existing methods do not automatically adjust for features on the surface, which can be viewed as an automated fixup step in this work. Therefore, they requires users to manually assign corners for *Submapping*. One example is shown in Fig.6.1 where it is difficult for a heuristic method to classify vertices A_5 and A_6. Users have to manually classify A_5 as ***CORNER*** and A_6 as ***END*** whose results are shown in Fig.6.1(b). There is another way of correctly classifying vertices shown in Fig.6.1(c) and which way is right depends on what users want and what the volume meshing needs to be.

There is another problem for current existing *Submapping* methods when there is a geometry with the fillet, chamfer or round features. For examle, Figure 6.2 shows a polygon with one hole and current existing methods fail to classify those vertices in that rounds and fillets spread discrete angles into continuous angle changes. Users have to interact in order to generate a valid vertex classification for *Submapping*.

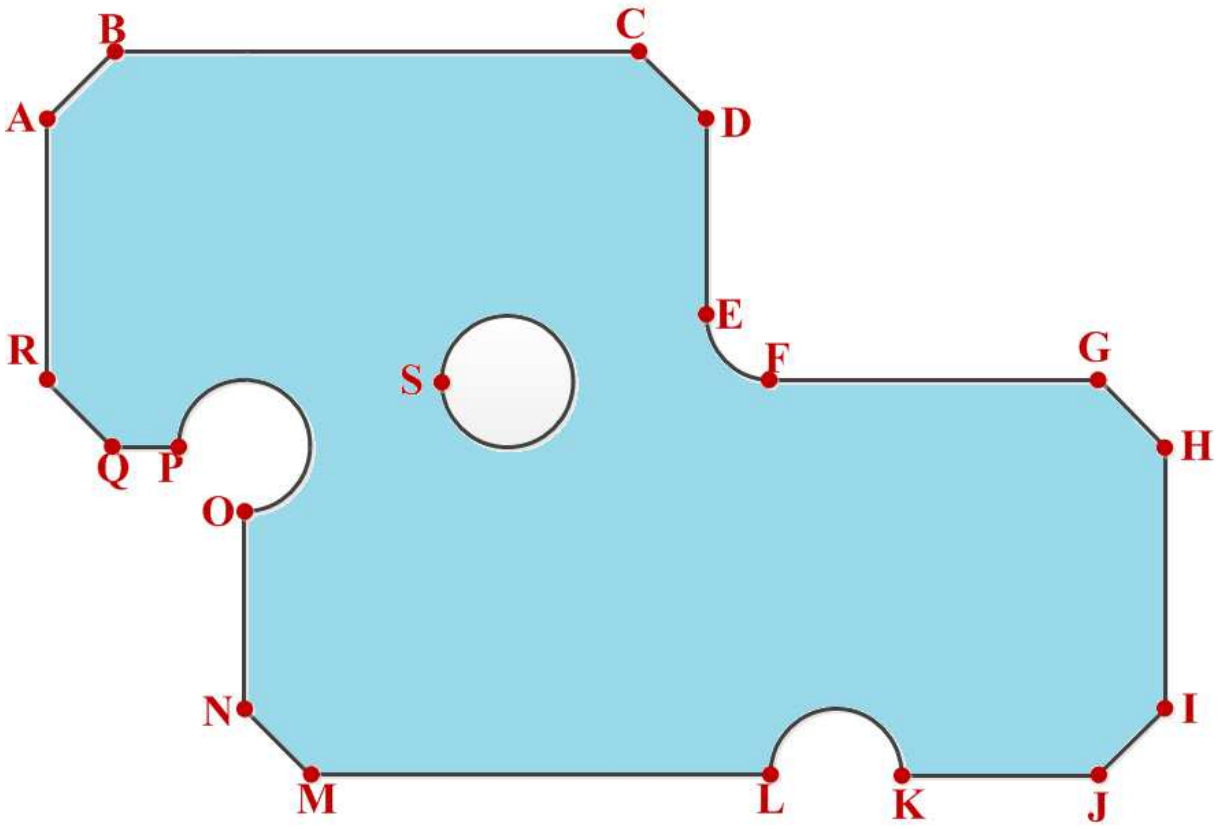

Figure 6.2: A failed example of vertex classification by *Submapping* due to the fillet or chamfer features

6.2 Optimal Corner Assignment

In this section, an optimal corner assignment algorithm for *Submapping* is described. In order to solve the problem of features like rounds, fillets and chamfers on surfaces, those features are relaxed and angle-based constraints are changed by using templates proposed in Section.6.2.1. If angle-based corner assignment is not valid, namely, it does not satisfy the *Submapping* constraint as Equation (6.5), the corner assignment problem is cast as an optimization problem based on the angle-based vertex classification and templates, and solved with a ***LP*** solver. This is described in Section 6.2.2.

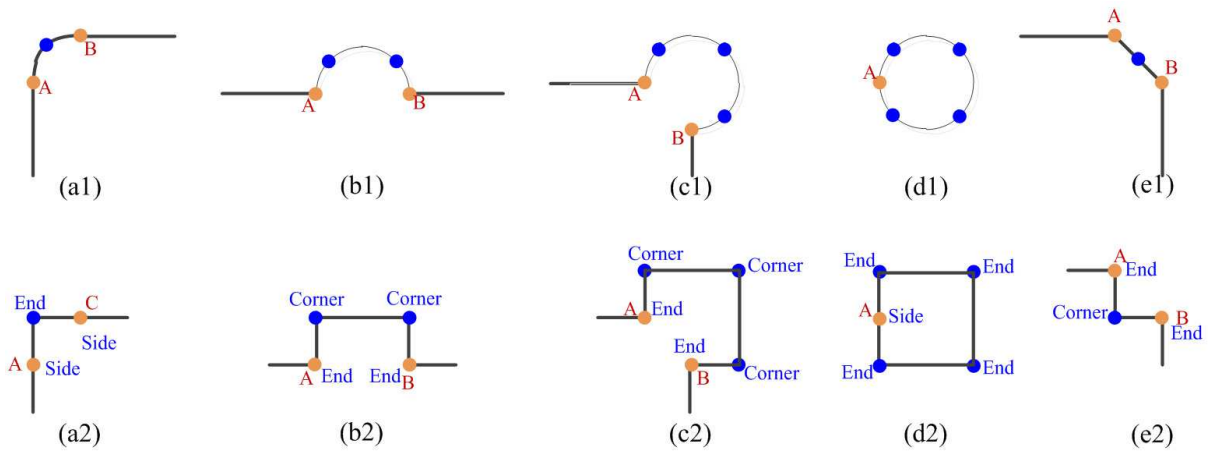

Figure 6.3: Templates for classifying vertices during *Submapping*: (a1)geometry feature with a rounding feature(one-quarter circle) between A and B; (b1)geometry feature with a rounding feature(half an circle) between A and B; (c1)geometry feature with a round feature(three quarters of circle) between A and B; (d1)geometry feature with a rounding feature(full circle); (e1)a chamfer feature(a transitional edge) between A and B; (a2)parametric space for (a1); (b2)parametric space for (b1); (c2)parametric space for (c1); (d2)parametric space for (d1); (e2)parametric space for e(1)

6.2.1 Templates

For many mechanical parts, they have been chamfered and filleted to avoid the stress concentration. However, it is pretty difficult for classifying vertices due to facts that rounds and fillets spread discrete angels into continuous angle changes. For example, in Fig.6.2, segments of DE and FG, instead of meeting at a right angle, meet at a fillet EF. Therefore, extra preprocessing is required to deal with these features by relaxing or changing the angle-based constraints used to determine vertex types before generating structured quadrilateral meshes on these surfaces. Meanwhile, virtual vertices are needed to be added on those features where the optimized vertex types can be put. In order to automate these processes and extend capabilities of *Submapping*, some templates are proposed in this dissertation to get the correct vertex types around these features.

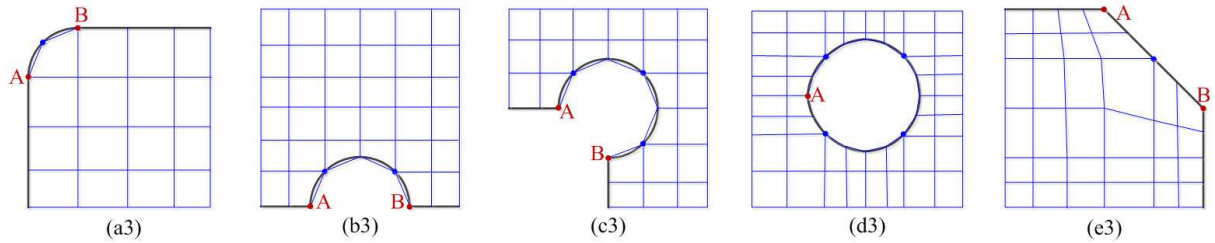

Figure 6.4: Structured quadrilateral mesh for templates during *Submapping*: (a3)structured quad mesh for Fig.6.3(a1) and Fig.6.3(a2); (b3)structured quad mesh for Fig.6.3(b1) and Fig.6.3(b2); (c3)structured quad mesh for Fig.6.3(c1) and Fig.6.3(c2); (d3)structured mesh for Fig.6.3(d1) and Fig.6.3(d2); (e3)structured mesh for Fig.6.3(e1) and Fig.6.3(e2)

Figure 6.3 shows the round, fillet and chamfer feature on the surface where each top row case can be resolved with corner arrangement in the bottom row. The corresponding structured quadrilateral meshes for those templates shown in Fig.6.3 are shown in Fig.6.4. If the heuristic method purely based on angles is applied, the invalid corner assignment will be produced since angles are ambiguous and discrete angles are spread into continuous angles. For example, a purely angle-based method would assign a corner type of ***SIDE*** to all vertices in Fig.6.3(d1), and the surface would not be mappable. For the chamfer feature in Fig.6.3(e1), the vertex A and B can also be assigned as type-***SIDE*** and the middle vertex can be assigned as type-***END***.

6.2.2 Corner assignment optimization based on *LP*

LP is a technique for the optimization of a linear objective function, subject to linear equality and inequality constraints. Generally, ***LP*** are problems that can be expressed in canonical form:

$$Maximize \qquad \mathbf{c^T x} \tag{6.6}$$

$$\text{Subject to} \qquad \mathbf{Ax} \leq \mathbf{b} \tag{6.7}$$

$$\text{Subject to} \qquad \mathbf{x} \geq \mathbf{0} \tag{6.8}$$

where $\mathbf{x}$ represents the vector of variables(to be determined), $\mathbf{c}$ and $\mathbf{b}$ are vectors of (known) coefficients, $\mathbf{A}$ is a (known) matrix of coefficients. The expression to be maximized or minimized such as Equation (6.6) is called the objective function. The inequalities $\mathbf{Ax} \leq \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$ are the constraints which specify a convex polytope over which the objective function is to be optimized.

During classifying vertices purely based on angles, most vertices can get their correct vertex types. However, there are a few vertices whose vertex types need to be adjusted since internal angles between two consecutive edges are not always integers multiple of 0.5π and Equation (6.5) constrains the corner assignment for *Submapping*. Ruiz Girones's method[73] fails to correct the vertex classification in some cases since the variation of vertex types is overly constrained. In addition, the variation constraint of vertex classification is enforced by the heuristic intuition in Ruiz Girones's method[73]. In order to deal with problems addressed above, a new vertex classification based on ***LP*** is described in this dissertation. The key contribution is to generate a valid corner assignment for more geometries based on ***LP*** by including the *Submapping* constraint and excluding impossible vertex type variation due to the geometric constraints.

Therefore, the following improved ***LP*** model is proposed here: the objective is to minimize the number of vertices whose vertex types need to be adjusted. Since some ***REVERSAL*** vertices may be converted to be ***SIDE*** such as Fig.6.6, the constraint(A.29) in Section.A.6 of Chapter A is removed in our improved ***LP*** model.

Our approach changes vertex types based on the Fig.6.5:

(1) . If $eps \leq \theta_i \leq 180^\circ - eps, \qquad \alpha_i \in \{1, 0, -1, -2\}$.

(2) . If $180^\circ - eps \leq \theta_i \leq 360^\circ - eps, \qquad \alpha_i \in \{0, -1, -2\}$.

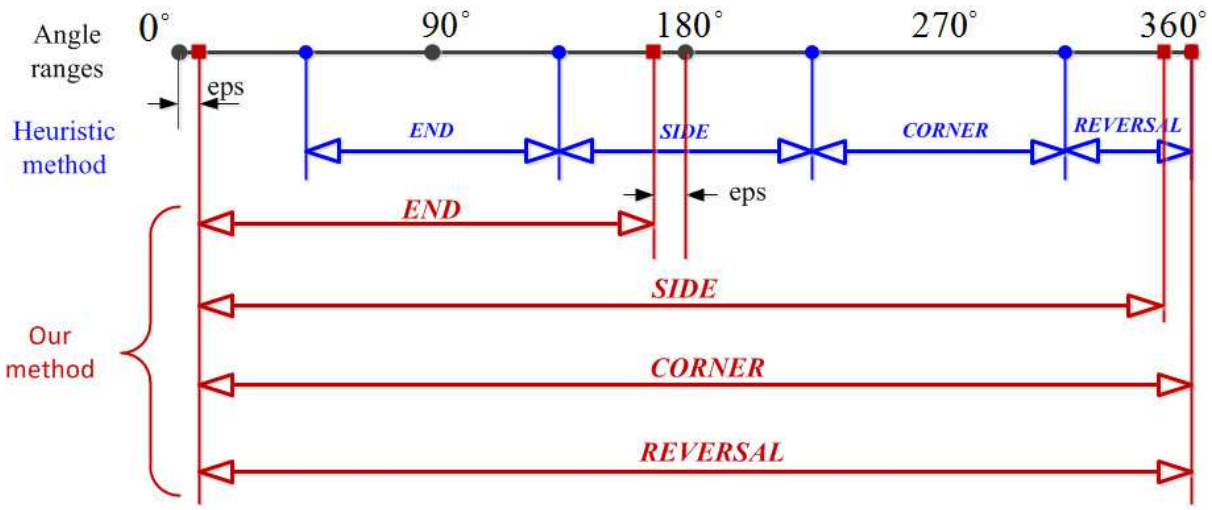

Figure 6.5: Ranges of vertex types with respect to the geometric angles

(3) . If $360\,^\circ - eps \leq \theta_i \leq 360\,^\circ, \qquad \alpha_i \in \{-1, -2\}$.

In order to avoid those impossible cases described, an extra constraint is added. By changing the objective function, removing the constraint(A.29) and adding extra constraints, we have the following ***LP*** model.

$$Objective \qquad min \sum_{i=1}^{N} d_i \tag{6.9}$$

s.t.

$$\sum_{i=1}^{N} \alpha_i = 4 - 4g \qquad \alpha_i \in \{-2, -1, 0, 1\} \qquad \alpha_i \text{ is an integer} \tag{6.10}$$

$$d_i = \begin{cases} 1, & \text{if } |\alpha_i - \overline{\alpha_i}| > 0 \\ 0, & \text{if } |\alpha_i - \overline{\alpha_i}| \equiv 0 \end{cases} \qquad i = 1, \cdots, N \tag{6.11}$$

$$\alpha_i < 1 \qquad \text{if } \theta_i \geq 180\,^\circ - eps \tag{6.12}$$

$$\alpha_i < 0 \qquad \text{if } \theta_i \geq 360^\circ - eps \tag{6.13}$$

where $\overline{\alpha_i}$ is a vertex type at a vertex i based on internal angles θ_i, N is the total number of vertices on the surface and *eps* is the angle tolerance. Figure 6.6 shows an example which can not be solved by ***LP*** model from Section. A.6 of Chapter A (see discussions in Section 2.3.3.2 of Chapter 2).

The conditional constraint (6.11) with absolute value is a nonlinear constraint and can be converted into the constraint (6.14) and constraint (6.15), which can be easily transformed into linear constraints based on the reference[29].

$$\begin{cases} |\alpha_i - \overline{\alpha_i}| - d_i \geq 0 \\ 3d_i - |\alpha_i - \overline{\alpha_i}| \geq 0 \end{cases} \tag{6.14}$$

$$\Rightarrow \begin{cases} \alpha_i - \overline{\alpha_i} \geq d_i \quad \alpha_i - \overline{\alpha_i} \leq -d_i \\ -3d_i \leq \alpha_i - \overline{\alpha_i} \leq 3d_i \end{cases} \quad i = 1, \cdots, N, \quad d_i \in \{0, 1\} \tag{6.15}$$

It is pretty straightforward to verify that the constraint(6.11) is equivalent to the constraint(6.15). If $\alpha_i = \overline{\alpha_i}$, $|\alpha_i - \overline{\alpha_i}| = 0$, the only solution to satisfy $|\alpha_i - \overline{\alpha_i}| - d_i \geq 0$ and $3d_i - |\alpha_i - \overline{\alpha_i}| \geq 0$ is $d_i = 0$. If $\alpha_i \neq \overline{\alpha_i}$, $1 \leq |\alpha_i - \overline{\alpha_i}| \leq 3$, the only solution to satisfy $|\alpha_i - \overline{\alpha_i}| - d_i \geq 0$ and $3d_i - |\alpha_i - \overline{\alpha_i}| \geq 0$ is $d_i = 1$. For the *Either-or-Active* constraint[6], it can be converted into a linear constraint by introducing a new binary variable y_i and a large enough positive M as follows.

$$\alpha_i - \overline{\alpha_i} - My_i \geq d_i \tag{6.16}$$

$$\alpha_i - \overline{\alpha_i} - M(1 - y_i) \leq -d_i \tag{6.17}$$

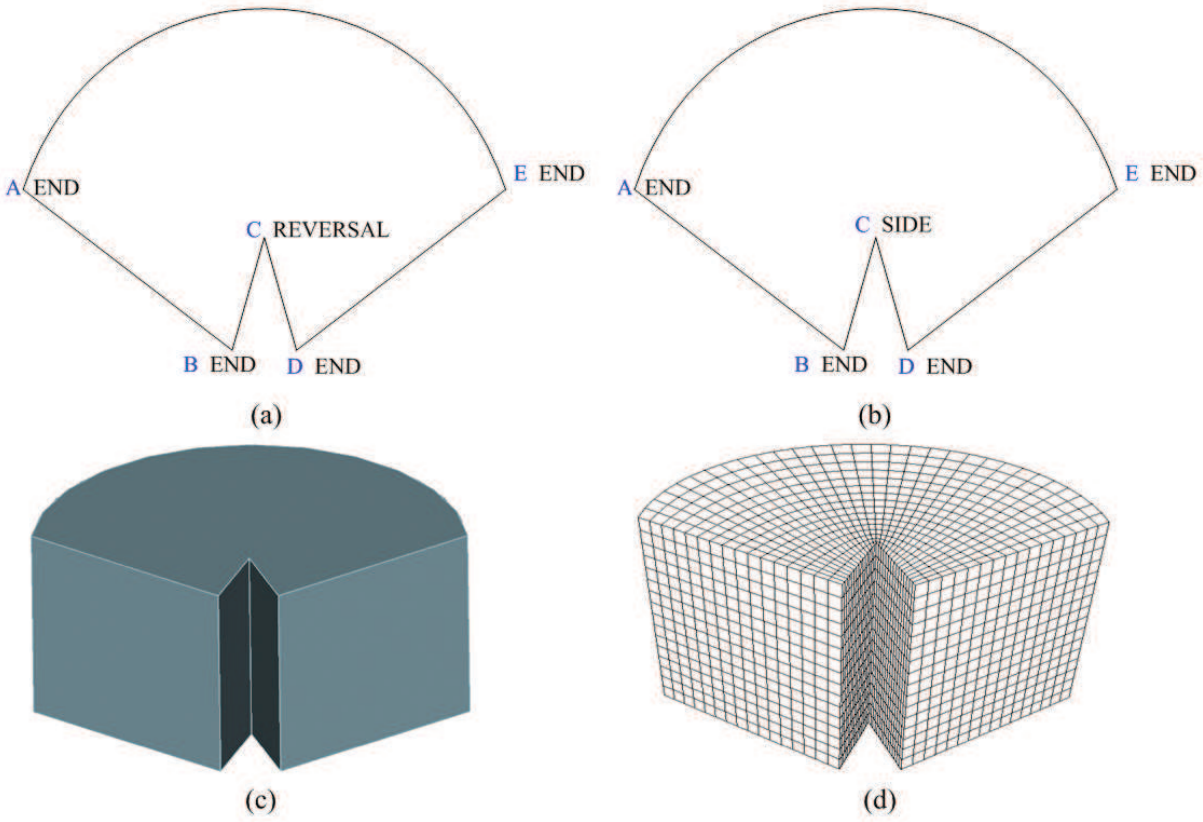

Figure 6.6: An example of vertex classification by our method: (a)an invalid vertex classification purely based on angles; (b)new valid vertex classification based on the improved *LP* model: a *REVERSAL* vertex is converted to be *SIDE*; (c)geometry model; (d)all-quad mesh on all the surfaces generated by *Submapping*

In the same way, the conditional constraint (6.12) and 6.13 can be converted into the linear constraints

$$-\alpha_i + M * x_i \geq 0 \tag{6.18}$$

$$180 - eps - \theta_i + M * (1 - x_i) \geq 0 \tag{6.19}$$

$$360 - eps - \theta_i + M * (1 - z_i) \geq 0 \tag{6.20}$$

$$1 - \alpha_i + M * z_i \geq 0 \tag{6.21}$$

Hence, the above LP model can be summarized as follows and LP can be solved by a tool: *lpsolve*[37].

$$objective \qquad min \sum_{i=1}^{N} d_i \tag{6.22}$$

s.t.

$$\sum_{i=1}^{N} \alpha_i = 4 \quad \alpha_i \in \{-2,-1,0,1\}, \quad \alpha_i \text{ is an integer} \tag{6.23}$$

$$\alpha_i - \overline{\alpha_i} - My_i \geq d_i \qquad i = 1, \cdots, N \tag{6.24}$$

$$\alpha_i - \overline{\alpha_i} - M(1-y_i) \leq -d_i \qquad i = 1, \cdots, N \tag{6.25}$$

$$\alpha_i - \overline{\alpha_i} \leq 3d_i \qquad i = 1, \cdots, N \tag{6.26}$$

$$\alpha_i - \overline{\alpha_i} \geq -3d_i \qquad i = 1, \cdots, N \tag{6.27}$$

$$-\alpha_i + M * x_i \geq 0 \tag{6.28}$$

$$180 - eps - \theta_i + M * (1 - x_i) \geq 0 \tag{6.29}$$

$$360 - eps - \theta_i + M * (1 - z_i) \geq 0 \tag{6.30}$$

$$1 - \alpha_i + M * z_i \geq 0 \tag{6.31}$$

$$d_i \in \{0,1\} \quad x_i, y_i, z_i \in \{0,1\} \qquad i = 1, \cdots, N \tag{6.32}$$

$$M \qquad \text{a large enough integer} \tag{6.33}$$

6.3 Boundary Discretization and Interior Node's Placement

Since surface vertices has already been correctly classified at this point, the edge parameterization, edge discretization and interior node's placement can be done as the reference

[73, 114]. Note that generally, interior nodes are embedded in 3D space based on the transfinite interpolation and 3D positions of boundary nodes. For the curved surfaces, the interpolated nodes may not be located on surfaces. Hence, extra geometric processing is needed by projecting the interpolated interior nodes onto surfaces based on the closest points/distances.

6.4 Examples and *submapping* results

In order to assess the mesh quality of structured quadrilateral meshes by *Submapping* with an improved corner assignment algorithm, several examples are provided. Users specify the mesh element size and the algorithm will classify vertices automatically. If there is a multi-connected geometry, it should be decomposed into several simply-connected geometries by using the *Constrained Delaunay Triangulation* (***CDT***)[85] so that the consistent edge parameterization between the outmost boundary and internal holes can be generated. Here, we will skip the details for geometry decomposition in this dissertation(see Reference[73] for details). Note that all the bounding surfaces in all the examples shown below are meshed with structured quadrilateral meshes by *Submapping*. In the meantime, the running time by *submapping* is much less than that of sweeping for the same models and mesh densities.

The first example in Fig.6.7 shows a surface with fillets, rounds and chamfers which put difficulties for current existing *Submapping* methods. Our method can apply templates for those fillets, rounds and chamfers on geometries and the result of corner assignment is shown in Fig.6.7(a). The resulting structured grids are shown in Fig.6.7(b).

In the second example shown in Fig.6.8, the heuristic method purely based on angles fails to put corners for *Submapping*, but our improved algorithm can classify vertices successfully. The structured quadrilateral meshes on all the bounding surfaces are shown

in Fig.6.8(b). Figure 6.9 shows an example of gear which is multiply-connected. Geometry decomposition for converting a multiply-connected surface into several simply-connected geometries is required before *Submapping*. Otherwise, it will fail to generate the consistent edge parameterization between holes and outmost boundary. Figure 6.10 and Fig.6.11 are complicated examples which are mechanical parts. Templates and geometry decomposition for converting the multiply-connected geometry to several simply-connected geometries should be used during *Submapping*.

6.5 Summary

In this chapter, an optimal corner assignment by *Submapping* for the structured grid generation on the *linking* surfaces has been developed. First, templates are applied if there exists rounds, fillets or chamfers on geometries. Then surface vertices are classified based on a combination of angles and templates. If the *Submapping* constraint is not satisfied, ***LP*** is used to correct the invalid vertex classification and generate an optimal corner assignment for *Submapping*. The structured grids have been successfully generated on the *linking* surfaces, which is crucial for generating high quality swept volume meshes.

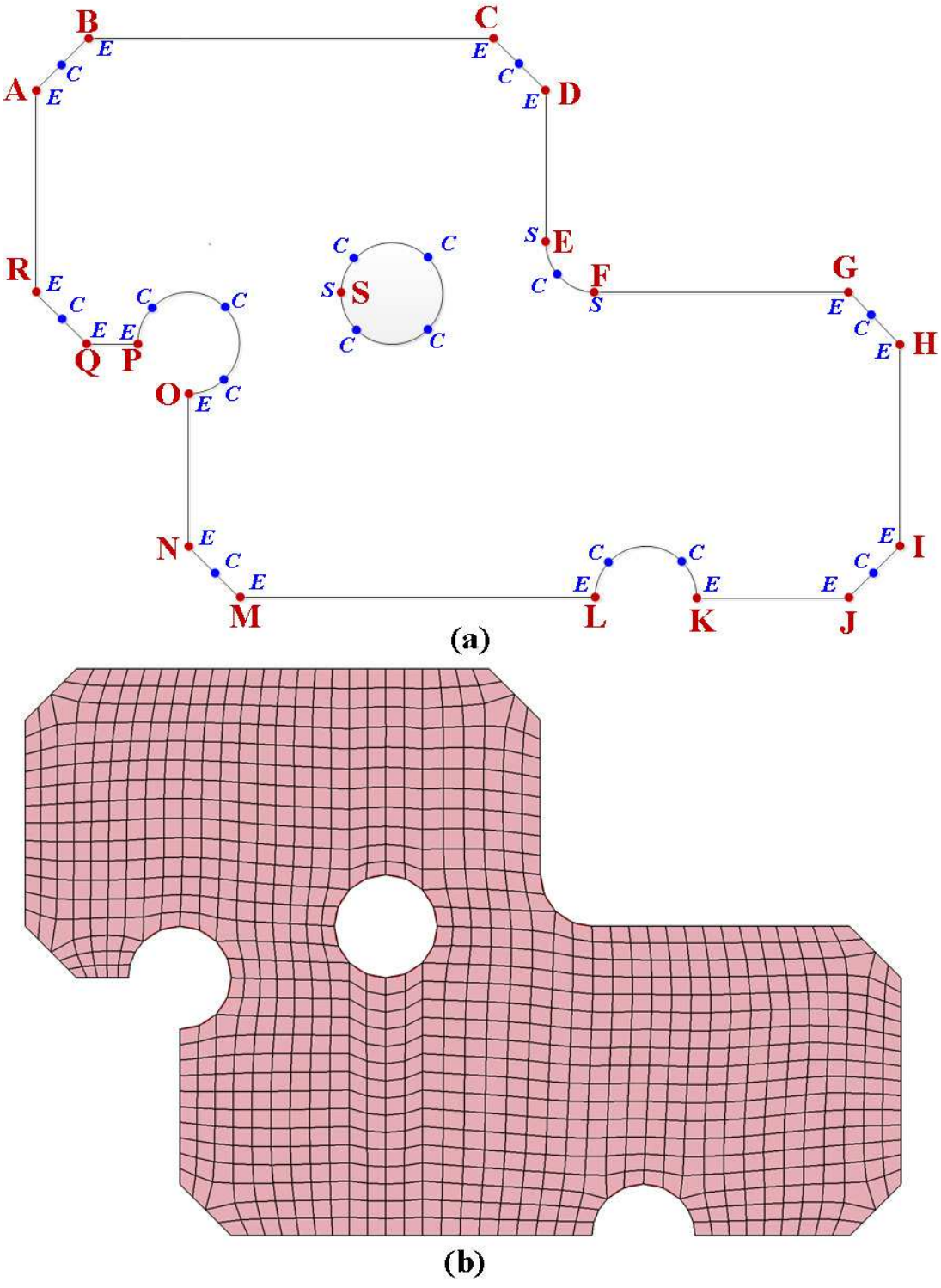

Figure 6.7: Structured grid generation of a surface with fillets, rounds and chamfers by *submapping* through optimized ***LP*** and templates: (a)corner assignment; (b)structured quadrilateral meshes

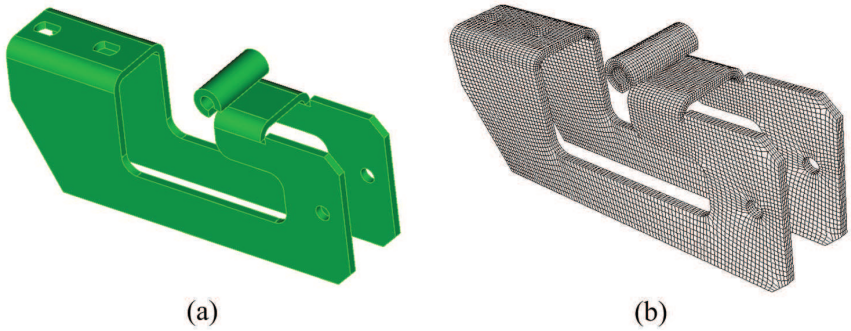

Figure 6.8: Structured all-quad mesh of a mechanical part generated by *submapping*: (a)geometry Model; (b) all-quad mesh for all the surfaces on a mechanical part generated by *submapping*

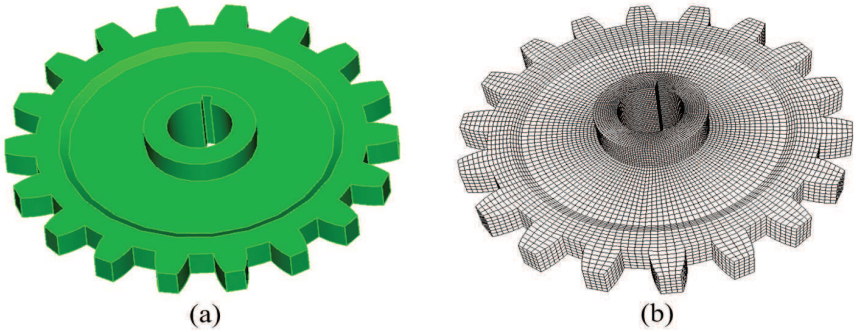

Figure 6.9: Structured quadrilateral meshes of a gear generated by *submapping*: (a)geometry Model; (b) all-quad mesh for gears

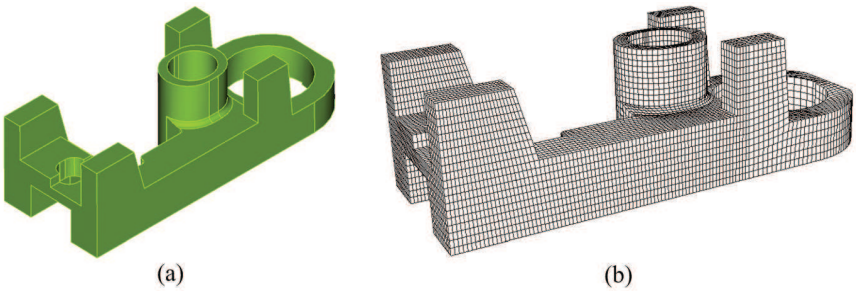

Figure 6.10: Structured all-quad mesh of a mechanical part generated by *submapping*: (a)geometry Model; (b) structured quadrilateral meshes for all the surfaces on a mechanical part by *submapping*

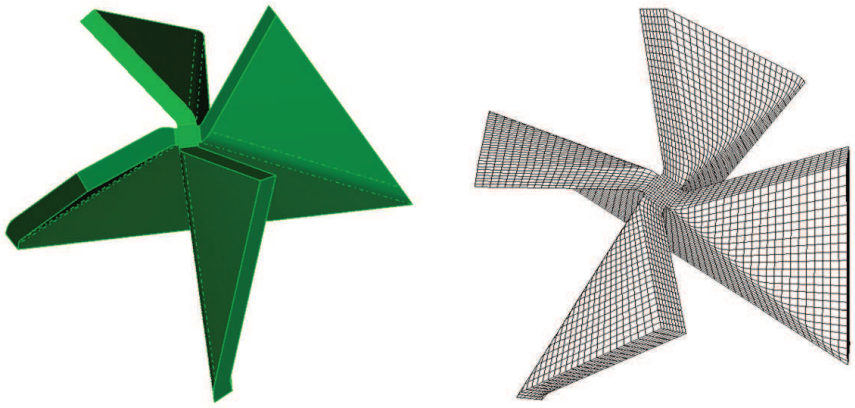

Figure 6.11: Structured all-quad mesh of a mechanical part generated by *submapping* through optimized ***LP***: blade part (a)geometry Model; (b)structured quadrilateral meshes for curved surfaces

7 SWEEPABILITY ASSESSMENT

A sweepabilty assessment algorithm is for evaluating whether a model is sweepable or not. Compared to the multi-sweeping algorithms, the sweepability assessment method should be very cheap from the computational point view. This is very important in that expensive computational cost can be avoided by the sweepability assessment instead of running multi-sweeping algorithms if a model is unsweepable in itself. Sweepability assessment is also of interest from a theoretical point of view, since it helps us to better understand swept meshes and geometries. There are three kinds of constraints affecting sweepability: topological constraints, geometrical constraints and constraints from user's specified matchings (called prescribed constraints in this work). White et al.[113] proposed some necessary constraints for assessing sweepability. As we show in Chapter 2, they are not sufficient to guarantee sweepability in several cases.

7.1 New Sweepability Assessment Discussion

In the Section.A.7 of Chapter A, the list of ***Lemma A.1-A.7*** and ***Theorem A.1-A.3*** from White's method[113] only works for regular sweeping problems and they fail for radial sweeping problems because there are no *linking* surfaces connecting the *source* and *target* surfaces. We posit in this section that sweepability problems also depend on topological aspects of the *source* and *target* surfaces. These aspects must be accounted for before sweepability problems can be assessed.

7.1.1 Topological Constraints

In ***1-1*** sweeping, a swept volume consists of one *source* surface and one *target* surface. As described in White's method[113], sweeping extrudes a collection of mesh faces into a third

dimension, generating prism elements in the process. By construction, the connectivity of all *target* faces is the same as the corresponding *source* faces, and therefore the collections of *source* and *target* faces are homeomorphic in ***1-1*** sweeping.

For ***M-1*** or ***M-N*** sweeping, *source* and *target* faces may be in disconnected sets, separated by *linking* surfaces, with subsets of *source* faces being added and *target faces* being removed at various sweeping levels. In this case, the sets of *source* and *target* faces are not homeomorphic. However, we can define a special operation that accounts for addition and removal of *source*/*target* faces to/from the sweeping, and this operation allows assertion of homeomorphism on these sets. Therefore, some special terms are defined in this dissertation as follows.

Source surface set: it is defined as connected sum of all the *source* surfaces on all the sweeping levels. The connections between different *source* surfaces in the set are made through the *linking* surface parameterization.

Target surface set: it is defined as connected sum of all the *target* surfaces on all the sweeping levels. The connections between different *target* surfaces in the set are made through the *linking* surface parameterization.

Joining: it is defined as a connected set of *source* or *target* surfaces from previous sweeping levels and new *source* or *target* surfaces on a specific sweeping level in this work (see Fig.7.1(a)).

Cutting: it is defined as an subtraction from a connected set of s/t surfaces from previous sweeping levels with new s/t surfaces on a specific sweeping level in this work (see

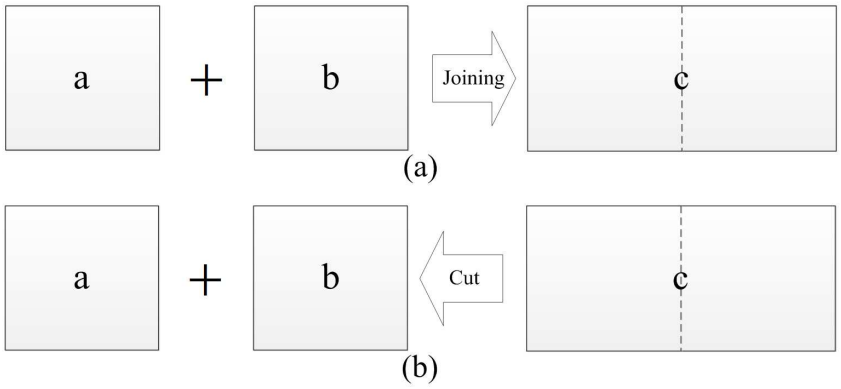

Figure 7.1: Joining and cutting operator during regular sweeping

Fig.7.1(b)).

Go-through Operation: The *source* and *target* surfaces traverse through sweeping levels from 0 to $L-1$ in the following ways: perform *joining* when meeting *target* surfaces, perform *cutting* when meeting *source* surfaces, and then at final sweeping level, the *source* surfaces should exactly match *target* surfaces. Qualitatively, *source* surface collection is homeomorphic to the *target* surface collection if collections combined into connected sets using *Go-through Operation*.

Figure 7.2 shows examples of *Go-through Operation*: in all the examples, start with the bottommost surfaces and perform the *Go-through Operation*. The deformed surfaces from previous sweeping levels at $L-1$ sweeping level are homeomorphic to the existing *source*/*target* surfaces at $L-1$ sweeping level.

Based on this observation, then, we assert that if a volume is sweepable ($s \leftrightarrow t$) with specified s/t surfaces, then those *source* and *target* surface sets must be homeomorphic under the *Go-through Operation*.

For instance, examples in Fig.7.3 are unsweepable, since there is only one surface

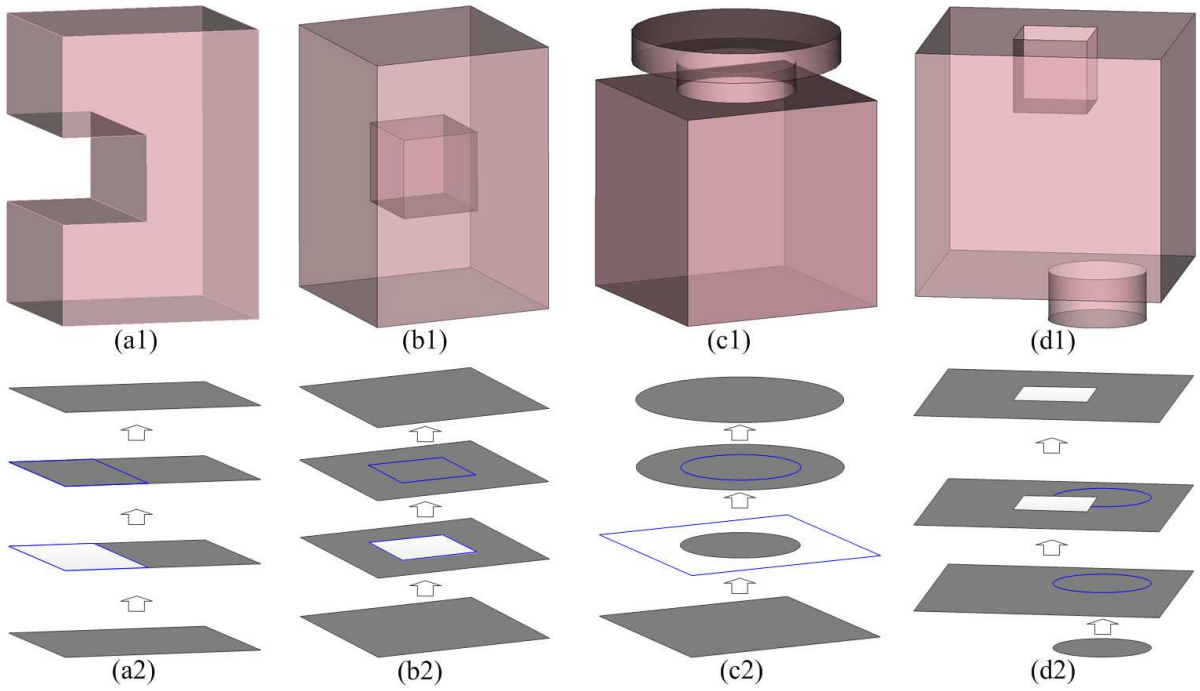

Figure 7.2: Examples of *Go-through Operation* for regular sweeping

on each model and they are not homeomorphic to two shells. In Fig.7.4(a), there is a sphere with two hollow shells which is not homeomorphic to two shells (outer shells are not homeomorphic to interior shells). Therefore, it is unsweepable in itself. Figure.7.4(c) shows an example of a sphere with three shells inside. It is not homeomorphic to two shells either and regular sweeping can not be applied since there are no *linking* surfaces. Therefore, it is unsweepable. However, by decomposition and changing topology of model in Fig.7.4(d), it becomes sweepable. So does Fig.7.4(a) and Fig.7.4(b). In Fig.7.5(a), the solid is unsweepable either by radial sweeping or regular sweeping. By breaking its topological constraints as Fig.7.5(b), the geometry becomes sweepable.

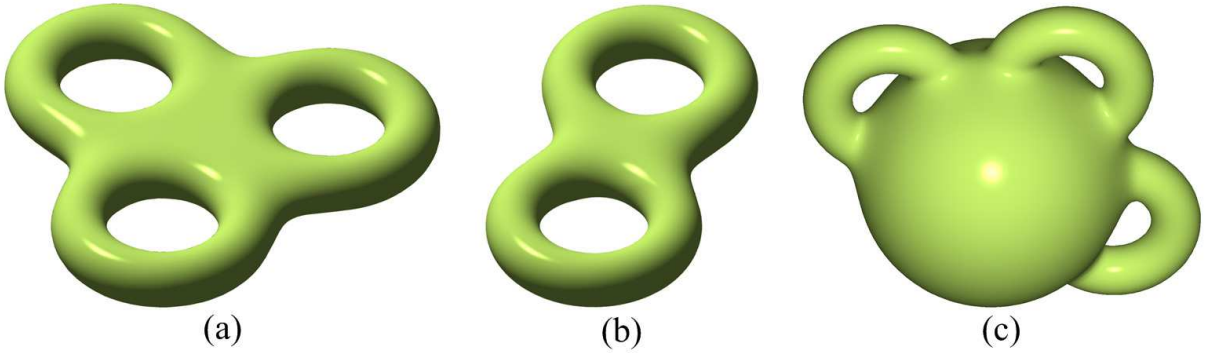

Figure 7.3: Unsweepable example of topological constraints: (a)a geometry with genus three; (b)a geometry with genus two; (c)a geometry with genus three

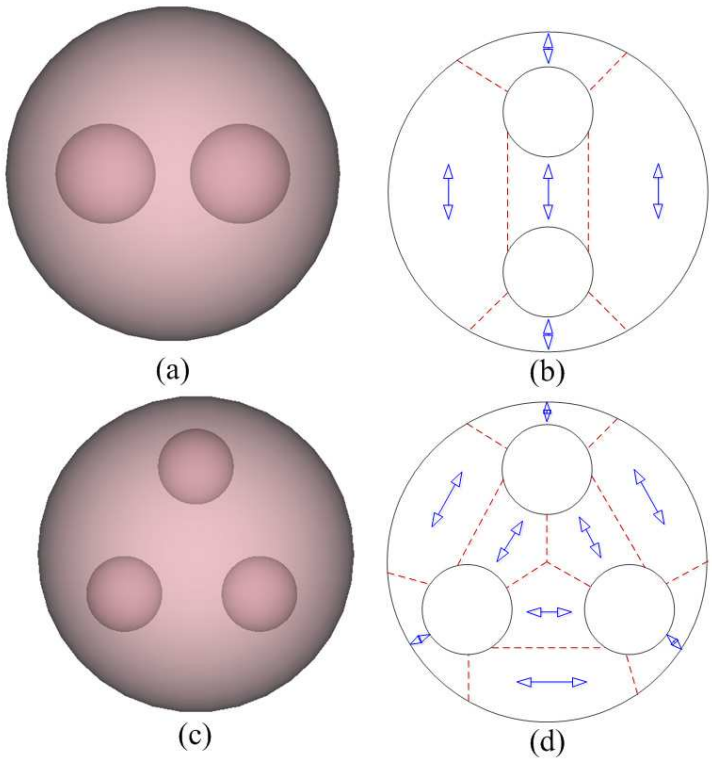

Figure 7.4: Sweepability example of topological constraints: (a)a sphere with two interior shells; (b)decomposed model for (a); (c)a sphere with three interior shells; (d)decomposed model for (c)

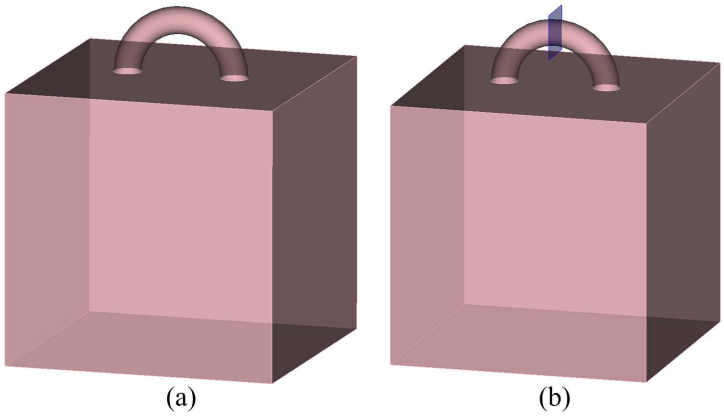

Figure 7.5: Sweepability example of topological constraint by regular sweeping: (a)geometry model; (b)geometry model becomes sweepable by splitting

7.1.2 Geometric Constraints

The geometric constraints define acceptable element shapes in the swept volume meshes. The particular constraints are typically imposed by the *linking* surfaces, namely, dihedral angles between the *linking* surfaces and *source*/*target* surfaces and the shape of *linking* surfaces.

Based on this observation, then, we assert that for a swept volume ($s \leftrightarrow t$) with specified s/t surfaces, to ensure element angles less than 180° and non-overlapping elements, ***END***-type edges must be less than 180°, and ***SIDE***, ***CORNER*** and ***REVERSAL*** edges must be less than 360°. All angles must be greater than 0°.

In some cases, the shape of *linking* surfaces can affect the sweepability problem as well. For example, the geometry in Fig.7.6 is topologically sweepable while it is geometrically unsweepable and a lot of inverted swept volume mesh elements will be created by sweeping inside the volume in that there exists a curved through hole between the *source* and *target* surface.

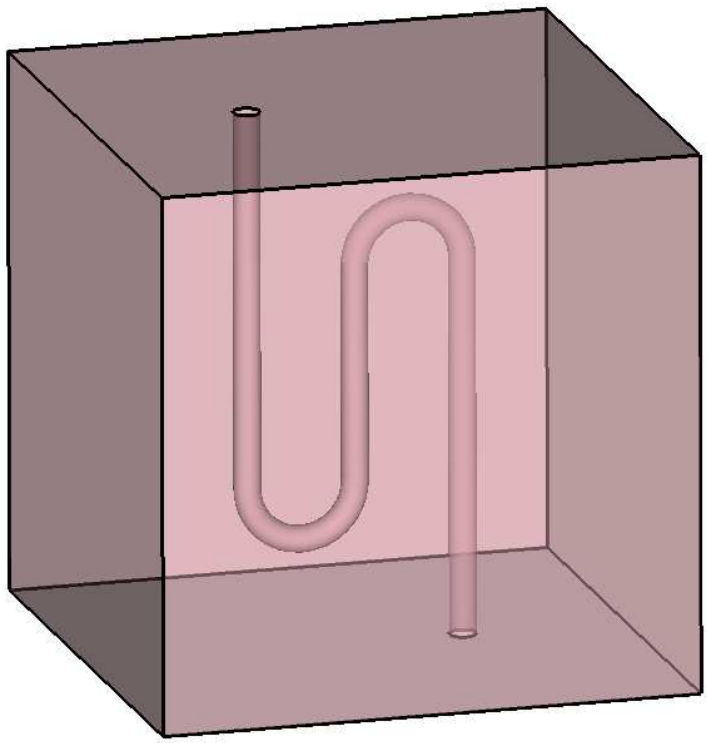

Figure 7.6: An unsweepable example due to the geometric constraint

This type of constraint differs from the topological constraints discussed above and White et al[113], since it depends by definition on the algorithms available to generate interior vertex positions. It would be difficult to prove a priori which volumes could or could not be swept with adequate mesh quality without considering those algorithms themselves. On the other hand, one can assess qualitatively that the volume shown in Fig.7.6 would be almost impossible to sweep successfully. Without further work on that subject, the sweepability problem with respect to this kind of geometric constraint is assessed by relying on our sweeping algorithm.

7.1.3 Prescribed Constraints

In some cases, geometries in themselves are sweepable. However, due to specified matchings from users, geometries may become unsweepable due to the topological constraints and geometrical constraints, which we name it as prescribed constraints. For example, Figure 7.7(a) shows that users specify the matching between a surface A and a, $B \leftrightarrow b$, $C \leftrightarrow c$ and $D \leftrightarrow d$. However, it is impossible to generate the surface mapping from A to a: the vertex v_1 on A is directly connected to the vertex v_1' on c by the *linking* side, which constrains the vertex mapping from v_1 to v_5'; the *linking* surfaces also constrain vertex mappings of v_2/v_6' and v_8/v_4'. The same failure occurs for $B \leftrightarrow b$, $C \leftrightarrow c$ and $D \leftrightarrow d$ due to the topological constraints. Therefore, the geometry in Fig.7.7(a) is unsweepable due to the topological constraints under the user-specified matchings even though it is sweepable in itself without any specified matching.

In Fig.7.7(b), it is required that $A \leftrightarrow a$ and $B \leftrightarrow b$. However, due to the geometrical constraint, it is difficult to generate a swept mesh with good mesh quality that maps A to a and B to b. Therefore, the geometry in Fig.7.7(b) may be unsweepable due to the geometrical constraints even though the geometry is sweepable without the user-specified matchings.

7.2 Summary

In this chapter, the sweepability problem is discussed, which is assessed based on three constraints, namely, topological constraints, geometrical constraints and prescribed constraints. The topological constraints are evaluated based on the homeomorphic theory between the *source* and *target* surfaces, which determines whether there exists mappings between the s/t surfaces. The geometric constraints are evaluated based on edge types

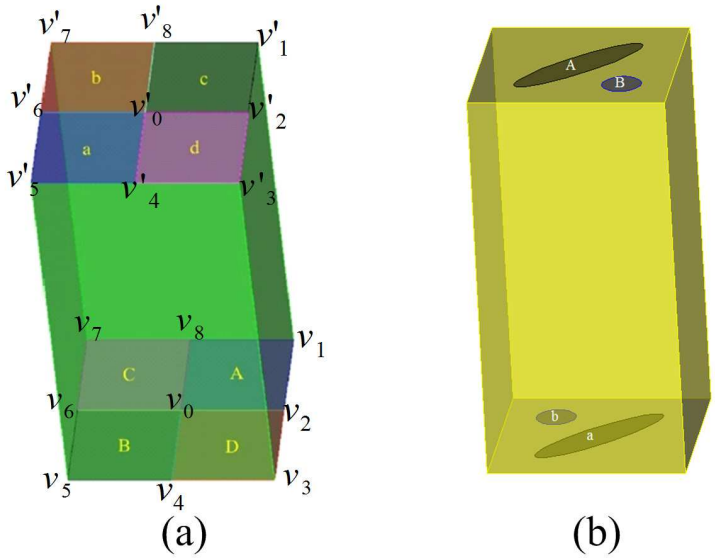

Figure 7.7: Two examples of constraints from users: (a)unsweepable due to the topological constraints; (b)unsweepable due to the geometrical constraints

between the *linking* and s/t surfaces, which determines whether there exists degenerated swept volume elements even though there exists mappings between the *source* and *target* surfaces. The prescribed constraints are mainly used to detect users' specified errors. In summary, the sweepability assessment described in this chapter is important for sweeping in that it can tell users whether geometries are sweepable or not before executing the sweeping algorithms and thus a lot of efforts can be saved.

8 IMPLEMENTATION AND INTEGRATION

Since a sweeping algorithm consists of several pieces, in this chapter, Section 8.1 describes how to integrate those sweeping pieces. In order to verify the sweeping methods in this work, some examples from the real application are shown in Section 8.2.

8.1 Overview

In this section, we present an overview of integration of sweeping pieces. Generally, a sweeping process consists of four main steps: *source* surface mesh generation, *target* surface mesh generation, *linking* surface mesh generation and interior node placement inside volumes. Therefore, the overall swept volume mesh quality is affected by those important pieces.

8.1.1 Integration of sweeping pieces

Figure 8.1 shows the integration of sweeping pieces with details of our multi-sweeping algorithm where there are five main steps for swept volume meshes. In this dissertation, we only cover some important pieces in the five main steps since there exists pretty robust surface mesh generators such as Paving algorithm[13] and Chaman's Jaal quad mesh generator[102]. Our sweeping method in this work starts with an input volume without any mesh but with the identified *source* and *target* surfaces.

(1) . *Sweepability*(blue color in Fig.8.1): Before sweeping, the sweepability for a specific should be assessed so that extra computational costs can be avoided. In this work, whether a solid is sweepable or not is analyzed based on the sweepability assessment discussions, which are described in Chapter 7.

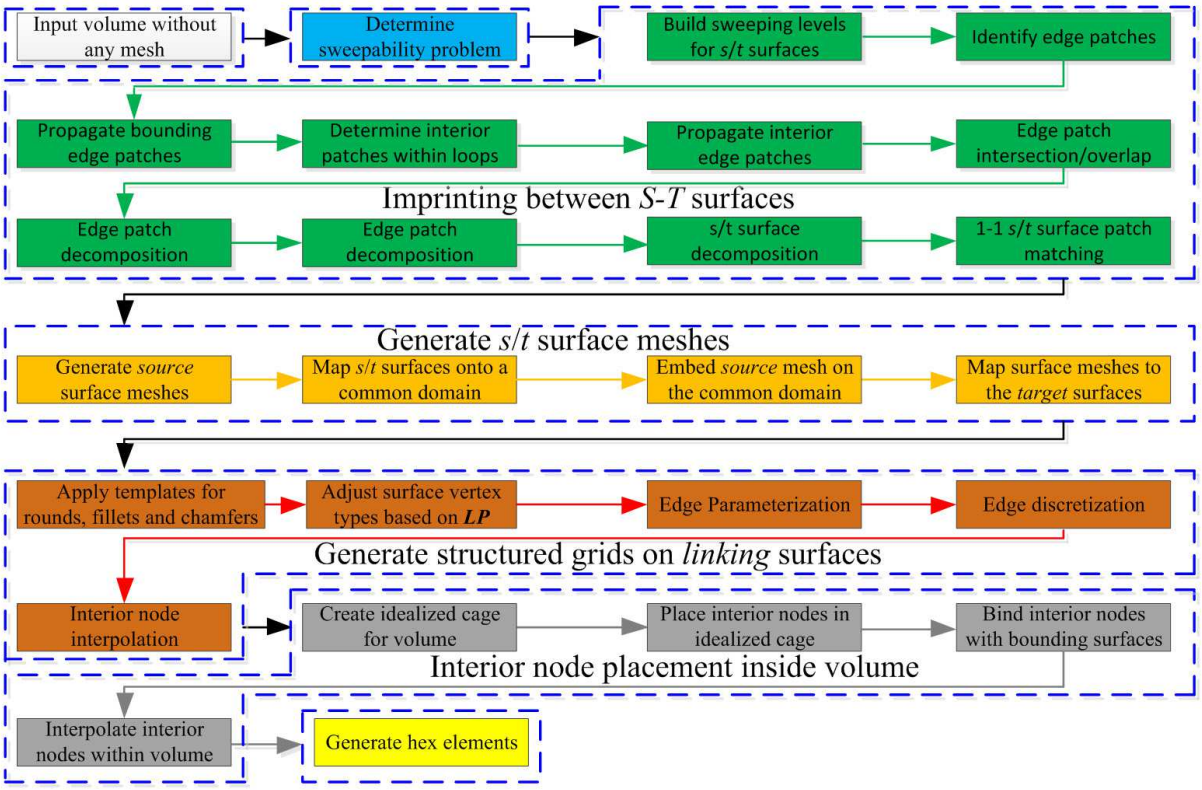

Figure 8.1: Details of multi-sweeping algorithm for swept volume meshes

(2) . *Imprinting between s/t surfaces*(green color in Fig.8.1): During ***1-1*** sweeping, this can be avoided since the *source* and *target* surface are paired and edge patches between them are matched as well. During ***M-1*** or ***M-N*** sweeping, imprinting has to be used so that *source* edge patches can be embedded on the *target* surface meshes and vice versa. The imprinting results in that edge patches and s/t surfaces are partitioned. The direct consequence of edge patch imprinting is that a list of ***1-1*** paired s/t surface patches have been generated and new edge patches between s/t surface patches are paired as well.

(3) . *Mapping surface meshes between s/t surface patches*(orange color in Fig.8.1): First either *source* or *target* surface patches have to be meshed with an existing

surface mesh generator. Then *source* and *target* surface facetings are mapped onto a common domain where the surface meshes are embedded onto the common domain at the same time. In the end, the surface meshes are mapped back to the physical s/t surfaces with the help of their surface faceting. This results in that the surface meshes between s/t surface patches have the same mesh connectivity.

(4) . *Structured grid generation on the linking surfaces*(red color in Fig.8.1): The *submapping* is usually used to generate the structured grids on the *linking* surfaces, which consists of four main steps: corner assignement, edge parameterization, edge discretization and interior node placement. In order to relax the angle-based constraints on the rounds or fillets or chamfers, templates are applied first and then surface corners are assigned. ***LP*** is a tool to generate an optimal corner assignment for *submapping* so that structured grids with good mesh quality on the *linking* surfaces can be generated.

(5) . *Interior node placement by sweeping*(gray color in Fig.8.1): At this stage, all the bounding surfaces have been already meshed, which includes the *source*, *target* and *linking* surfaces. Since the sweeping in itself requires that surface meshes are swept between s/t surfaces to generate the swept volume meshes, interior nodes can be easily placed in the idealized model by sweeping either *source* or *target* surface meshes through volumes based on the *affine transformation*. By mapping those interior nodes in the idealized model to the physical model, interior nodes can be placed within volumes, which results in the swept volume mesh generation.

8.1.2 Prerequisites

Our goal is to develop a volume mesh generation by sweeping which minimizes the user input effort and expertise while the swept mesh quality is improved as much as possible. In this dissertation, however, there are some prerequisites which have to be provided or met during the sweeping process.

(1) . *Surface mesh generator*: After imprinting during sweeping, there are a list of ***1-1*** paired *source* and *target* surface patches. During ***1-1*** sweeping, the *source* and *target* surface has already been paired. Either *source* or *target* surface patches have to be meshed so that the surface mesh can be mapped onto the other ones. In this work, we do not cover the surface mesh generator and assume that it should be provided

(2) . *Parametric space of linking surfaces*: It is important for sweeping in that on one hand, it can make correspondence for boundary nodes between s/t surfaces which are connected by the *linking* surfaces. On the other hand, it guides the propagation of faceting points from the bounding edge patches. However, in this work, it is assumed that the parametric space of chains of *linking* surfaces should be provided before sweeping.

(3) . *Idealized model creation*: In this work, interior nodes in the swept mesh are placed by mapping interior nodes in the idealized model to the physical model. Their locations are constrainted by their bounding surfaces. In the idealized model, interior nodes can be located based on the *affine transformation* such as translation. However, how to robustly create an idealized model from the physical model is not addressed in this work.

(4) . *Source and target surface identification*: During sweeping, the surface meshes are swept between the *source* and *target* surfaces and thus the swept volume meshes are created within volumes. Hence, they have to be identified manually or automatically before sweeping. In this work, it is assumed that the *source* and *target* surfaces have already been identified.

8.2 Real sweeping applications

Our approach is implemented in *MeshKit*[96] which depends on several tools described in Section 8.2.1. In order to verify the integration of our approach, we demonstrate several examples from real applications in order to show our improved sweeping method described in this work. In the meantime, the swept mesh quality with respect to the scaled Jacobian is analyzed. Note that we have not applied any smoothing algorithm for swept volume meshes in all the examples described below.

8.2.1 Implementation and tools

The entire software of our multi-sweeping algorithm is written in C++ and implemented in *MeshKit*[96], which relies on the geometry library *CGM*[98], mesh library *MOAB*[97] and relation tool between geometries and meshes *Lasso*[84].

(1) *CGM*[98]: This open-source software provides all the geometric functionalities that are needed for mesh generation. We used this software for geometry read, queries, creation and processing such as surface partition after s/t edge patch imprinting described in Chapter 5.

(2) *MOAB*[97]: This open-source library is a component for representing and evaluating mesh data. With this software, various relationships among the mesh entities can be

stored, modified and queried every efficiently.

(3) *LASSO*[84]: This open-source software provides relations as pairwise associations between geometric entities and mesh entity sets, which is useful for mesh generation.

(4) *MeshKit*[96]: It is an open-source library for mesh generation functionality, which provides a collection of meshing algorithms for use in real meshing problems, along with other tools commonly needed to support mesh generation. The integration of our multi-sweeping algorithm described in this work is implemented in this library.

8.2.2 Pipe junction model

Pipe networks are mainly used for transportation and supply of fluids and gases. During the meshing stage of fluid flow simulation for pipe networks, the most difficult part is to generate meshes at the intersecting locations, namely, pipe junction model. One of the typical examples is shown in Fig.8.2(a).

The most popular method for generating the swept volume meshes for pipe junction model is to partition the pipe junction model as Fig.8.2(f) where the geometry is split into two halves. After the volume decomposition, the pipe junction model becomes ***1-1*** sweepable and thus the swept volume meshes can be generated by *Cubit 13.2* as Fig.8.2(g). By taking a close look at locations around the horizontal through hole, the hex elements are distorted. The most obvious cut-view is shown in Fig.8.2(h) which shows how the interior hex elements from the swept meshes in Fig.8.2(g) are poorly shaped. The resulting mesh quality with respect to *scaled jacobian* is shown in Fig.8.3(a) where the poor mesh quality has been produced.

In this work, we can generate another sweeping scheme as Fig.8.2(b) where it is a multi-sweeping problem. Our optimal corner assignment approach classifies surface vertices

on the horizontal hole as type-***CORNER***. Figure 8.2(c) shows the imprinting results by our approach described in Chapter 5. After imprinting, a list of new ***1-1*** paired *source* and *target* surface patches have been generated and thus the *source* meshes can be mapped onto the *target* surface patches as Chapter 3. At this stage, all the bounding surfaces including *source*, *target* and *linking* surfaces have been meshed and interior nodes inside the pipe junction model by sweeping are placed as Chapter 4. The detail of interior node distribution is portrayed in Fig.8.2(e). Afterwards, the swept hexahedral meshes are generated as Fig.8.2(d). The corresponding mesh quality of swept meshes by our approach with respect to *scaled jacobian* is shown in 8.3(b) where better mesh quality has been produced.

Therefore, our approach can generate the swept volume meshes with better mesh quality and full automation compared to the large manual efforts required for geometry decomposition.

8.2.3 Grid spacer reactor model

The numerical simulations such as heat transfer (predictions of peak subassembly temperature) and ***CFD*** are needed to assess a designed reactor. One of the typical example is the grid spacer reactor model shown in Fig.8.4(a) and Fig.8.4(b). Generally, the swept volume meshes are preferred for the flow simulation[101].

The only other feasible method for generating the swept volume meshes for the grid spacer model is to partition the geometry until all the decomposed pieces are ***1-1*** sweepable. However, there are some shortcomings for the volume decomposition method: generally it is difficult to generate the internal surfaces which are used to guide the volume decomposition; meanwhile, interior nodes are placed separately (logically or physically) in each decomposed subvolumes and poor mesh quality may be produced inside volume.

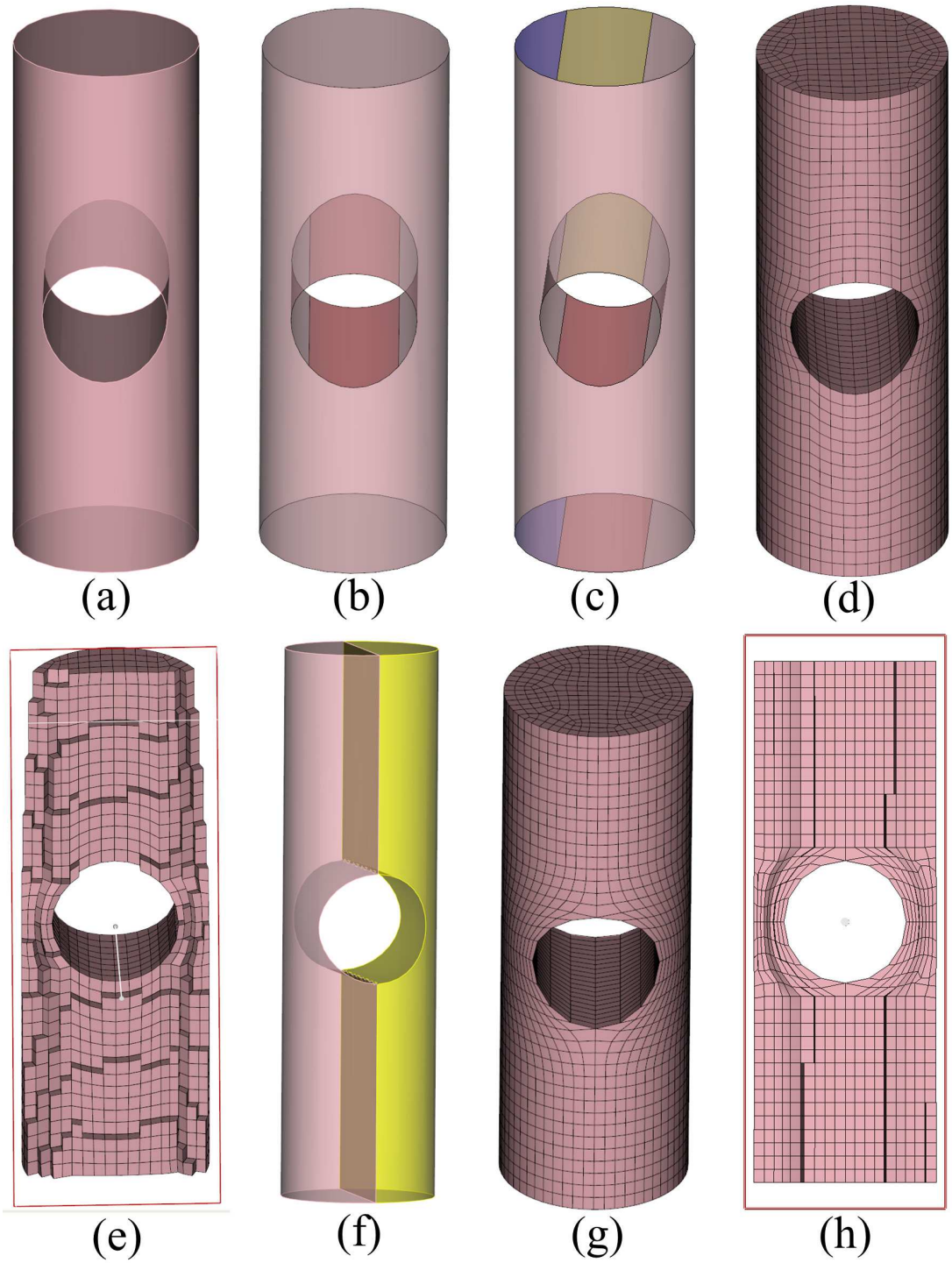

Figure 8.2: All-hexahedral mesh generation for pipe junction model: (a)geometry model; (b)sweeping schemes by our approach; (c)imprinting results; (d)all-hex meshes by our approach; (e)cut-view of all-hex meshes by our approach; (f)another sweeping scheme by volume partition; (g)all-hex meshes by volume partition; (h)cut-view of all-hex meshes by our approach

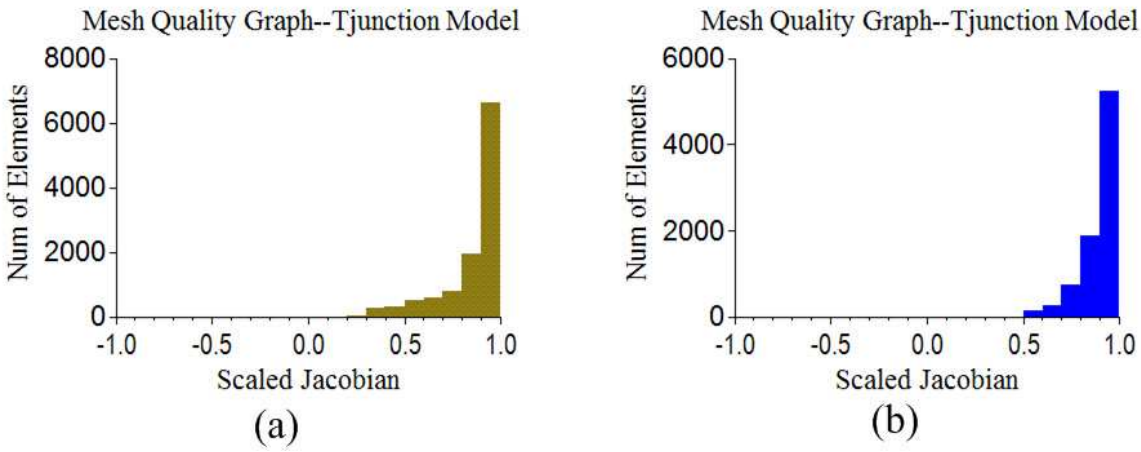

Figure 8.3: Mesh quality histogram for pipe junction model in Fig.8.2: (a)mesh quality for Fig.8.2(g); (b)mesh quality for Fig.8.2(d)

Those disadvantages become very obvious when there is a model with complicated internal structures such as the grid spacer model in Fig.8.4(b) where the blind twisted blade-like holes are located inside the geometry. A more close look can be taken at Fig.8.4(c) and Fig.8.4(d). Therefore, it requires a lot of experiences and takes a lot of efforts to decompose the grid spacer model in a reaonable way so that the acceptable mesh quality of the swept volume meshes can be achieved.

The grid spacer model shown in Fig.8.4(c) is a ***M-1*** sweeping problem where the most difficult part is to locate interior nodes inside the swept volume meshes constrained the bounding surfaces. Instead of using the volume decomposition, our approach uses the edge patch imprinting and generate the *target* surface meshes as Fig8.4(g) which are mapped from the *source* surface meshes in Fig.8.4(e). The interior nodes are placed by mapping those interior nodes from the idealized model to the physical model where interior nodes in the idealized model are simply placed based on *affine transformation*. The resulting interior node distribution is shown in Fig.8.4(h) and Fig.8.4(i). Its hexahedral mesh quality with respect to the *scaled jacobian* metric is shown in Fig.8.5 where there is no inverted

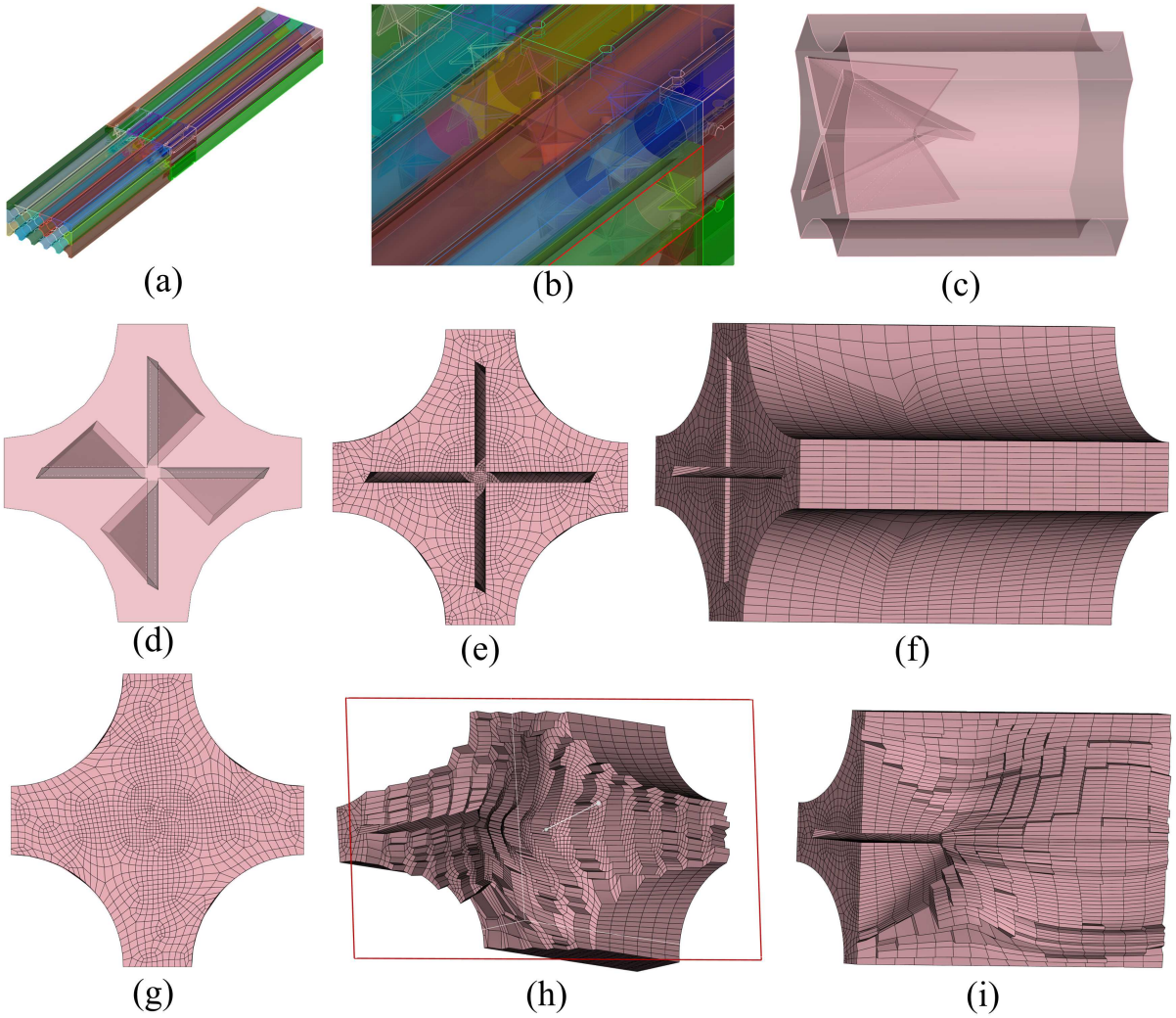

Figure 8.4: All-hexahedral mesh generation for grid spacer reactor model: (a)assembly geometry model; (b)local zoom-in view of assembly model; (c)single part of grid spacer reactor model; (d)top view for (c); (e)*source* surface meshes for (c); (f)all-hex meshes for (c); (g)*target* surface meshes for (c); (h)cut-view of interior node distribution inside the grid spacer model; (i)another cut-view of interior node distribution

element by our approach.

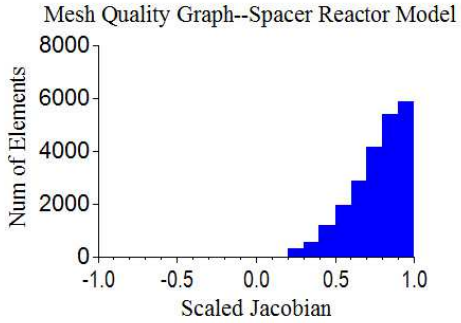

Figure 8.5: Mesh quality histogram for grid spacer reactor model in Fig.8.4

8.2.4 Inlet reactor model

The inlet model shown in Fig.8.6 is taken from the full assembly nuclear reactor model in order to assess our approch for the swept volume mesh generation. It consists of many different parts plotted in different colors in Fig.8.6(a) and Fig.8.6(b). Since all the parts are ***M*-*N*** sweepable, our approach can be directly applied to generate the swept volume meshes for the ***CFD*** simulation. Note that s/t surfaces should be identified before running our approach. Meanwhile, the parametric spaces for *linking* surfaces should be provided to guide edge patch propagation between s/t surfaces.

Figure 8.6(a) shows the full view of inlet assembly model with a zoom-in view shown in Fig.8.6. By applying our approaches, especially the imprinting algorithm described in Chapter 5 in order to avoid too much tedious volume decomposition, the resulting swept all-hexahedral meshes have been generated as Fig.8.6(c) with a zoom-in view as Fig.8.6(d). The swept mesh quality with respect to the *scaled jacobian* is plotted as Fig.8.7 where the good mesh quality has been achieved by our approach.

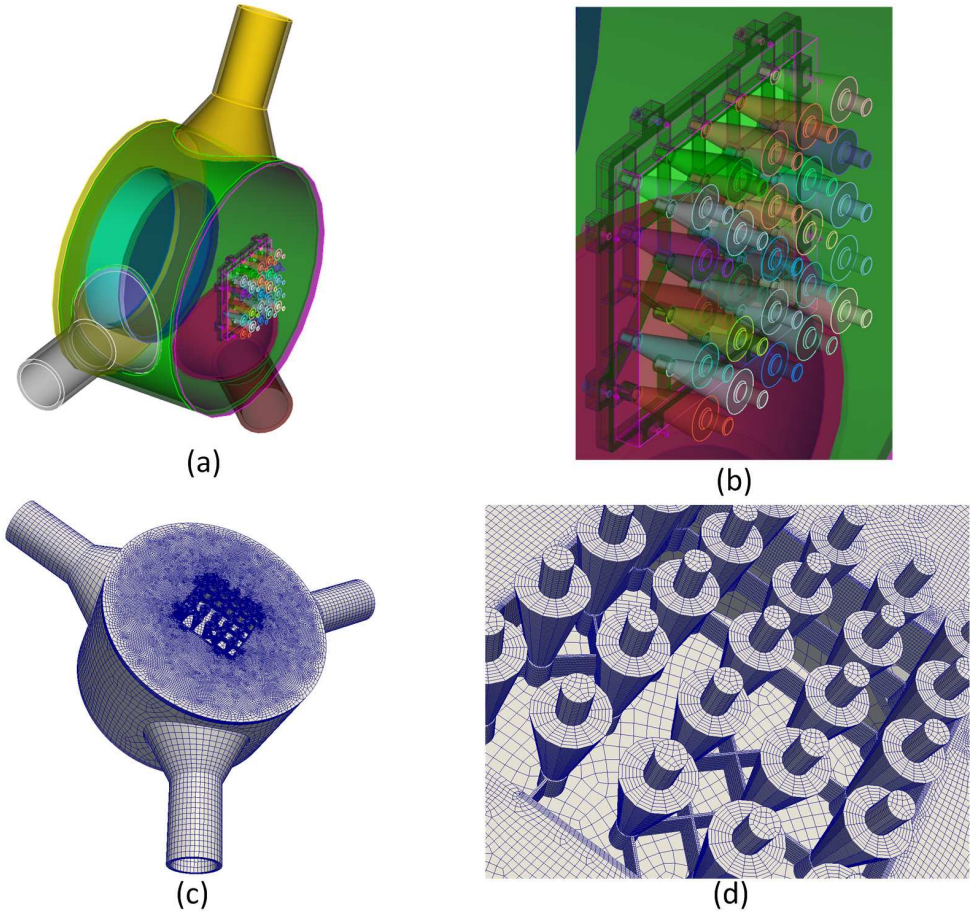

Figure 8.6: All-hexahedral mesh generation for the inlet reactor model: (a)assembly geometry model; (b)local zoom-in view of assembly model; (c)all-hexahedral meshes by sweeping; (d)a local zoom-in view of hexahedral meshes

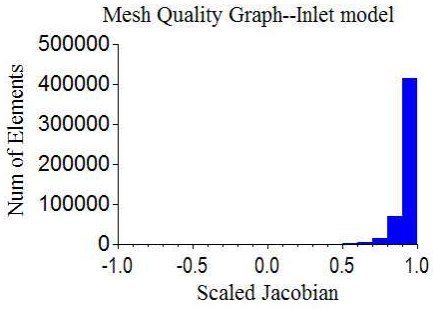

Figure 8.7: Mesh quality histogram for the inlet reactor model in Fig.8.6

8.2.5 Caterpillar part model from CD-adapco

The structure analysis is very popular in the mechanical engineering area. One of typical examples is to perform the numeric analysis for the mechanical parts such as caterpillar parts in order to determine whether the mechanical parts are strong enough to support loads. Figure 8.8 shows an example of caterpillar parts which is a $\boldsymbol{M}$-$\boldsymbol{N}$ sweeping problem.

There are many s/t surfaces. For current existing sweepers, how to reasonably imprint edge patches from s/t surfaces onto the appropriate s/t surfaces becomes problematic, which is important as it affects the overall swept hexhaedral mesh quality by sweeping. Our approach starts with the edge patch imprinting between s/t surfaces as Chapter 5 and imprinting result with split s/t surfaces is portrayed in Fig.8.8(b). After imprinting, *target* surface meshes in Fig.8.8(d) are generated by mapping from the *source* surface in Fig.8.8(c) based on *harmonic mapping*. The remaining steps, that is, structured quarilateral meshes by *Submapping* on the *linking* surfaces and interior node placement by mapping from idealized model to the physical model, can be performed as Chapter 6 and Chapter 4, respectively. The resulting swept hexahedral meshes are portrayed in Fig.8.8(e) with a zoom-in view shown in Fig.8.8(f). The mesh quality for the swept hexahedral meshes on the caterpillar part is shown in Fig.8.9, which indicates that good mesh quality has been achieved.

8.2.6 Crankshaft model from automobile

The crankshaft is mainly used to perform the conversion between reciprocating motion and rotational motion in the automobile engine and ship engine. One of the typical examples is shown in Fig.8.10(a). During the meshing stage of numeric simulation, the most difficult task is how to correctly embed edges from the *target* surfaces onto the *source* surface meshes, which is related to the edge patch intersection and overlapping problem discussed

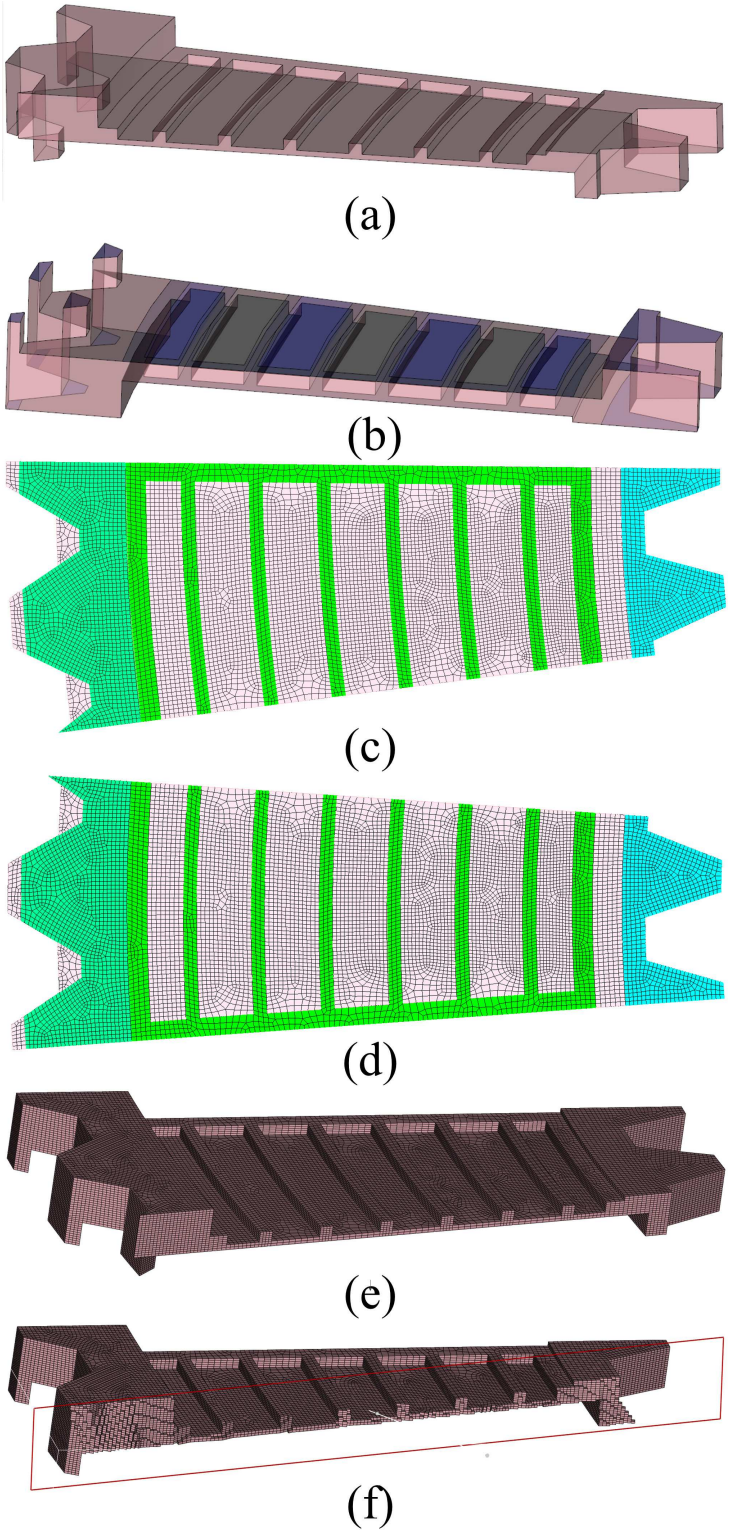

Figure 8.8: All-hexahedral mesh generation for caterpillar part model by sweeping: (a)geometry model; (b)imprinting results; (c)*source* surface meshes; (d)*target* surface meshes mapped from (c); (d)all-hexahedral meshes by sweeping; (e)cut-view of all-hex meshes for (d)

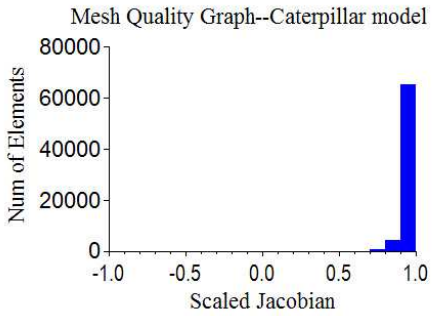

Figure 8.9: Mesh quality histogram for caterpillar part model in Fig.8.8

in Chapter 5.

The current existing multi-sweepers for generating the swept volume meshes for the crankshaft model is to partition the crankshaft model so that the divided pieces are ***1-1*** sweepable. This causes problems in one way or another. For one thing, linear *affine transformation* is used to propagate edge patches between s/t surfaces, which is incapable to correctly project edge patches on the appropriate s/t surfaces. This becomes obvious when projecting edge patches between concave or multi-connected domains. For another thing, since the premeshed *linking* surface meshes are used to guide the propagation of edge patches between s/t surfaces, it is very difficult to guarantee that there exists mesh nodes at the intersecting locations between edge patches. Then additional operation for adjusting edge mesh nodes is needed. In addition, interior nodes in the swept volume meshes are poorly placed since they are done separately in each subvolumes.

In order to avoid those problems mentioned above, our approach described in Chapter 5 uses the faceting points instead of the mesh nodes to propagate edge patches. Meanwhile, the mapping method derived from the cage-based method is used to project edge patches between s/t surfaces, which works for convex, concave and multiply-connected domains. In our approach, the crankshaft volume is not split and only s/t surfaces are split so that edges on the s/t surfaces can be embedded into the *target* and *source* surface meshes,

respectively. The imprinting results with series of new ***1-1*** paired s/t surface patches are shown in Fig.8.10(a). Then surface mesh mapping method described in Chapter 3 is applied to map surface meshes between s/t surfaces and the resulting s/t surface meshes are shown in Fig.8.10(c). The *linking* surface mesh generation and interior node placement can be done as Chapter 6 and Chapter 4, respectively. Figure 8.10(d) shows a cut-view of interior node placement by sweeping where good mesh quality shown in Fig.8.11 has been achieved.

8.3 Summary

In this chapter, we describe the integration of some important sweeping pieces, which have been demonstrated in details from all the previous chapters: surface mesh mapping based on *harmonic mapping* between s/t surfaces, interior node placement by mapping from idealized model to the physical model, edge patch imprinting based on edge faceting and mapping method without volume decomposition, the optimal corner assignment for *Submapping* and some sweepability assessments based on homeomorphism. They have to be implemented in an united way so that the overall swept volume mesh quality can be improved. In summary, our multi-sweeping algorithm can generate the swept volume meshes with better mesh quality, less user interaction and less user effort of decomposition.

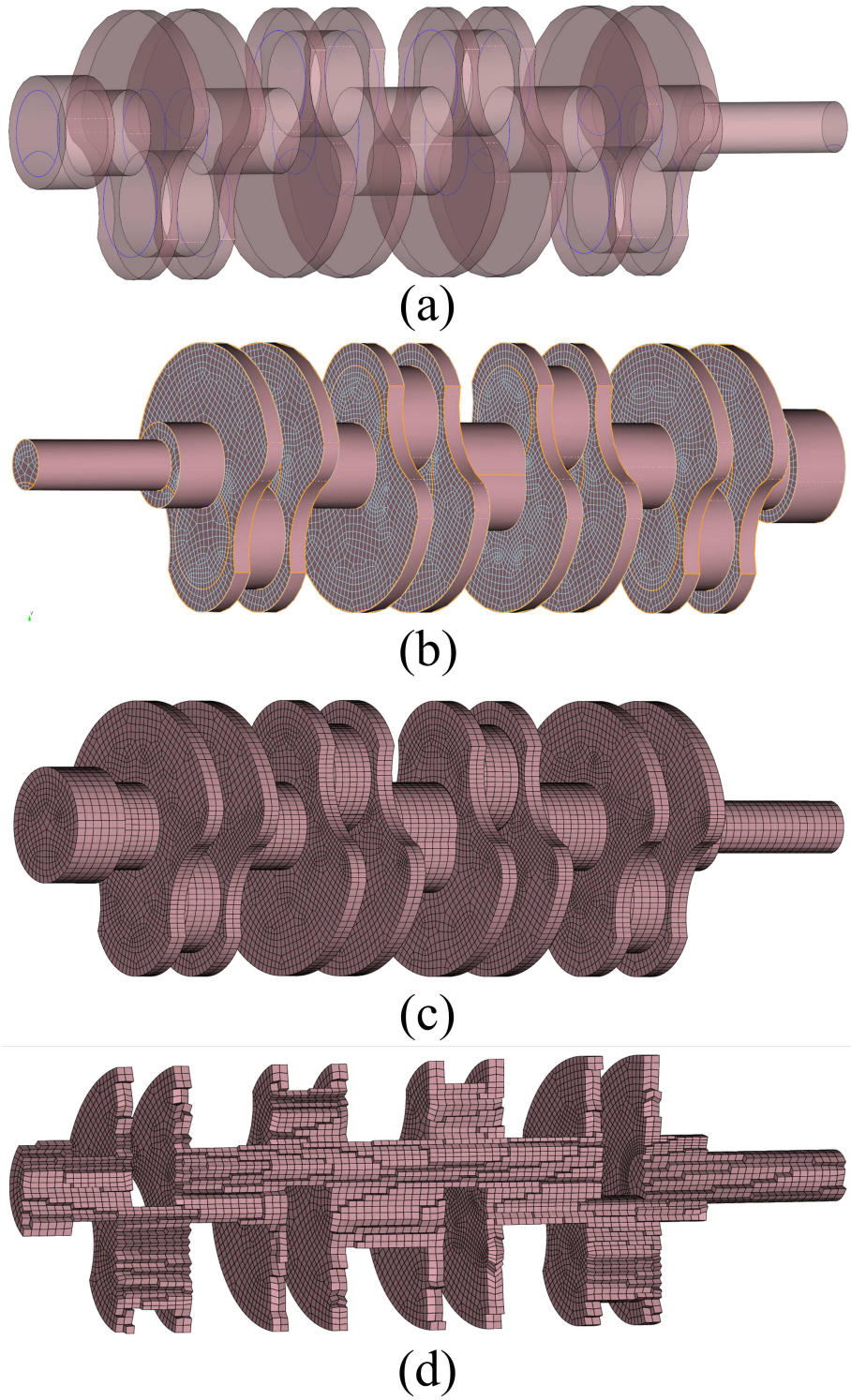

Figure 8.10: All-hexahedral mesh generation for the crankshaft model by sweeping: (a)the imprinting results; (b)*target* surface meshes by mapping; (c)all-hexahedral meshes by sweeping; (d)a cut-view of all-hexahedral meshes by our approach

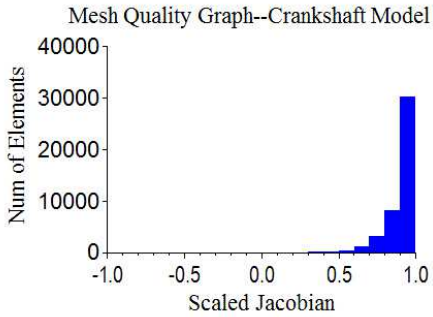

Figure 8.11: Mesh quality histogram for the crankshaft model in Fig.8.10

9 CONCLUSIONS

In this work, we have improved the sweeping algorithm by enhancing capabilities of several pieces of mesh sweeping: *target* surface mesh mapped from the *source* surface mesh based on *harmonic mapping* and surface faceting; a new interior node placement algorithm inside volumes constrained by the bounding surfaces based on cage-based method; an enhanced edge patch imprinting algorithm for resolving curves and surface patches during multi-sweeping based on cage-based morphing and edge faceting; an optimal corner assignment algorithm for structured grid generation on the *linking* surfaces during *Submapping*. Hence, the mesh quality of the swept volume meshes can be improved. The details are described in Section 9.1. However, there are some open problems for the sweeping algorithm to be solved in the near future, which is depicted in Section 9.2.

9.1 Contributions

In this dissertation, we have accomplished five goals addressed in Chapter 1 to develop a swept mesh generation technology. We have detailed these goals in the previous chapters of this dissertation. Therefore, we summarize the main contributions as follows.

1 **We have proposed a new surface mesh mapping method for *1-1* sweeping which effectively maps *source* surface mesh onto the *target* surface even though they are concave or multiply-connected surfaces.** In chapter 3, we have presented a new surface mesh mapping method based on *harmonic mapping* which maps both the *source* and *target* surface facetings over the common domain where the *source* surface meshes can be embedded and mapped back to the physical *target* surface. Our method works for convex, concave and multiply-connected surfaces and good mesh quality has been achieved on the *target* surface. This is

critical both for obtaining a high quality mesh on the volume boundary and for setting up the cage-based interior node placement in the interior of volumes.

2 **We have proposed a new interior node placement algorithm that can locate interior nodes inside volumes by sweeping with good mesh quality.** The surface meshes on the *source*, *target* and *linking* surfaces constrain the interior node placement inside volumes. Current existing methods fail to correctly locate interior nodes for models which have complicated internal structures since they use the linear *affine transformation* and fail to deal with the multiply-connected or concave domain. Linear *affine transformation* is a global method and lacks the local deformation. Therefore, in Chapter 4, we have developed an interior node placement method based on the cage-based method where interpolation functions(functions of interior nodes with bounding meshes as inputs) are computed in the idealized cage and interior nodes in the physical model are interpolated with the bounding surface meshes as inputs including *source*, *target* and *linking* surfaces. Good mesh quality has been achieved for locating interior nodes inside volumes, which is an important piece comprising a high quality swept mesh.

3 **We have proposed a new edge patch imprinting algorithm during multi-sweeping that can generate a list of *1-1* paired *source* and *target* surface patches.** The inherent characteristics of sweeping algorithm requires that any mesh face element on the *source* surfaces must match a specific mesh face element on the *target* surfaces. This requires the solution of two inherent problems during multi-sweeping: the first one is to resolve curves on the *source* and *target* surface so that they can be matched; The other one is to match *source* and *target* surface patches, which results in producing specific sweeping schemes during multi-sweeping: which *source* surface or part of *source* surface will be swept onto a specific *target*

surface. Therefore, in Chapter 5, we have developed an edge patch imprinting algorithm to solve multi-sweeping problem for the swept volume mesh generation: first sweeping levels are built based on the parametric space of *linking* surfaces and *breadth-first search* algorithm; then edge patches are imprinted between the *source* and *target* surfaces based on cage-based morphing which can correctly place interior edge patches on the concave or multiply-connected domains; *source* and *target* surfaces are partitioned so that each *source* surface patch matches only one specific *target* surface patch. For each paired s/t surface patch, the *target* surface mesh can be generated by using our method in Chapter 3. In addition, the edge patch interval assignment on the *source* and *target* surfaces can be solved as well through the edge patch imprinting.

4 **We have proposed a new vertex classification algorithm based on *LP* that improves the robustness and quality of *Submapping* scheme assignment for *linking* surfaces.** The structured quadrilateral meshes by *Submapping* on the *linking* surfaces connecting the *source* and *target* surfaces guide the sweeping process for the swept meshes. However, the heuristic method fails to classify vertices on surfaces during *submapping* in that it is purely based on surface internal angles at vertices. In order to correct the invalid vertex classification, in Chapter 6, we describe the use of ***LP*** to adjust vertex types on surfaces so that *linking* surfaces are more often submappable: first templates are used to relax the angle-based constraints around the round, fillet or chamfer features; then ***LP*** is used to correct vertex types with some conditions satisfied so that surfaces are submappable. Therefore, the optimal corner assignment for *Submapping* can be successfully generated. So do the structured grids on the *linking* surfaces.

5 **We have discussed the sweepability assessment that can determine whether**

a geometry is sweepable or not instead of running sweeping algorithm itself. Whether a model is sweepable through multisweeping or not is very important and this can avoid extra computational costs if a model is not meshable by sweeping. In Chapter 7, we discuss the sweepability assessment based on the geometrical constraints, topological constraints and prescribed constraints. From the point of topological constraint, the sweepability is determined based on whether the *source* and *target* collection are homeomorphic or not. For the geometric constraint, the sweepability is determined based on angles between s/t surfaces and *linking* surfaces in order to avoid poor swept meshes. The prescribed constraint is used to determine sweepability problems by identifying user's matching errors.

6 **Multi-sweeping requires robust solutions for all these pieces, assembled in an implementation.** in Chapter 8, the integration of sweeping includes an algorithm for mapping surface meshes between any two surfaces, a robust *submapping* algorithm for structured quadrilateral mesh generation on the *linking* surfaces, a robust interior node placement algorithm for interior nodes inside volumes, a robust edge patch imprinting algorithm for multi-sweeping in order to map the *source* surface meshes onto the *target* surfaces and a sweepability assessment to determine whether a volume is sweepable or not before generating a swept volume mesh.

9.2 Future Work

After this work, the state of art for sweeping is: an improved *target* surface mesh mapping method during ***1-1*** sweeping based on *harmonic mapping*, a good interior node placement method for sweeping based on cage-based method, a robust edge patch imprinting algorithm for multi-sweeping problems and an enhanced optimal corner assignment algorithm for

Submapping on the *linking* surfaces. The work carried out in this dissertation leaves some open research lines that should be investigated in the near future.

(1). **Idealized Cage creation**

Cage-based method is used to locate interior nodes inside volumes during sweeping. The idealized cage has to be created by deriving from the physical model. However, the issue how to automatically generate a good idealized cage for any sweepable geometry when placing interior nodes by sweeping should be solved in the near future.

(2) ***Submapping***

In this dissertation, our focus for *submapping* is to generate an optimal corner assignment for surfaces. However, there are some other issues to be solved for *submapping* in the near future. The first one is the edge parameterization for the multiply-connected surfaces during *submapping*: since holes are disconnected with the outmost boundary on a multiply-connected surface, how to generate the consistent edge parameterization for the outmost boundaries and hole boundaries is very important for improving the mesh quality of structured grids on the *linking* surfaces. The second one is the interior node interpolation by *Transfinite Interpolation* (***TFI***) during *submapping*: this becomes an apparent issue when there is a curved surface since ***TFI*** is purely based on the edge parameterization and the surface features such as curvatures are not taken into account. Hence, generally, it is required to smooth the structured grids on the *linking* surfaces after interpolating interior nodes on surfaces.

(3). **Parallel Meshing**

Current implementation of sweeping methods in this dissertation is serial computing only. There are several pieces during sweeping which can be parallelized in order to generate the swept meshes faster:

(a). Chains of *linking* surfaces can be meshed by *Submapping* for structured quadrilateral meshes. They can be parallelized by setting different chains of *linking* surfaces to the different processors.

(b). During multi-sweeping, a list of *source* and *target* surface patches has been generated after imprinting and the *target* surface patches can be meshed by mapping the surface meshes on the *source* surface patches. Since there is ***1-1*** correspondence between the *source* and *target* surface patch, the *target* surface patches can be meshed by parallel meshing which assigns different pairs of *source* and *target* surface patches to the different processors.

(c). During the interior node placement inside volumes, cage-based method is used by using the bounding surfaces as cages. The binding process during cage-based method is done by computing influences of each cage vertex on all the interior nodes. This can be parallelized by assigning different cage vertices to the different processors and computing influences of each cage vertex on all the interior nodes separately. The process of interpolating interior nodes inside volumes can be parallelized as well by assigning different processors with different interior nodes but with the same bounding surface meshes.

(4). **Geometric Constraints in Sweepability**

The sweepability problem is very important in that it can save a lot of computational costs by telling users whether a solid model is sweepable or not. However, the geometrical constraints such as the shape of *linking* surfaces can make a geometry

unsweepable in some case. In the near future, the issue how to define, quantify and integrate the geometry constraints into the sweepablity assessment should be solved.

REFERENCES

[1] A., Mitchell S. 2000. High Fidelity Interval Assignment. *International Journal of Computational Geometry & Applications* 10(04):399–415.

[2] Alexa, M. 2001. Local Control for Mesh Morphing. *SMI International Conference on IEEE* 209–215.

[3] Alexa, M. 2003. Differential Coordinates for Local Mesh Morphing and Deformation. *The Visual Computer* 19(2):105–114.

[4] Austin, D. 2015. Moving Remy in Harmony: Pixar's Use of Harmonic Functions.

[5] Baker, T. J. 2001. The Generation of Hexahedral Meshes for Assembly Geometry: Survey and Progress. *International Journal for Numerical Methods in Engineering* 50(12):2617–2642.

[6] Beck, R. E. 1995. *Elementary linear programming with applications*. Gulf Professional Publishing.

[7] Benzley, S. E., E. Perry, K. Clark, and G. Sjaardama. 1995. A Comparison of All Hexagonal and All Tetrahedral Finite Element Meshes for Elastic and Elasto-Plastic Analysis. In *Proceedings of 4th International Meshing Roundtable*, 179–191.

[8] Benzley, S. E., E. Perry, K. Merkley, B. Clark, and G. Sjaardama. 1995. A Comparison of All Hexagonal and All Tetrahedral Finite Element Meshes for Elastic and Elasto-Plastic Analysis. *4th International Meshing Roundtable* 179–191.

[9] Biswas, R., and R. C. Strawn. 1988. Tetrahedral and hexahedral mesh adaptation for cfd problems. *Applied Numerical Mathematics* 26(1–2):135–151.

[10] Blacker, T. D. 1996. The Cooper Tool. In *Proceedings of 5th International Meshing Roundtable*, 13–29.

[11] Blacker, T. D. 2001. Automated Conformal Hexahedral Meshing Constraints, Challenges and Opportunities. *Engineering with Computers* 17(3):201–210.

[12] Blacker, T. D., and R. J. Myers. 1993. Seams and Wedgers in Plastering: A 3D Hexahedral Mesh Generation Algorithm. *Engineering With Computers* 2(1):83–93.

[13] Blacker, T. D., and M. B. Stephenson. 1991. Paving: a New Approach to Automated Quadrilateral Mesh Generation. *International Journal for Numerical Methods in Engineering* 32(4):811–847.

[14] Blanding, R. L., G. M. Turkiyyah, D. W. Storti, and M. A. Ganter. 2000. Skeleton-based Three Dimensional Geometric Morphing. *Computational Geometry* 15(1): 129–148.

[15] Blender developer community. 2015. *Meshdeform modifier*. http://wiki.blender.org/index.php/Doc:2.4/Manual/Modifiers/Deform/MeshDeform.

[16] Cai, S., and T. Tautges. 2013. Robust one-to-one sweeping with harmonic st mappings and cages. In *Proceedings of 22th International Meshing Roundtable*, 1–18.

[17] Cai, S., and T. Tautges. 2014. One-to-one sweeping based on harmonic st mappings of facet meshes and their cages. *Engineering with Computers* 1–14.

[18] Cai, S., and T. Tautges. 2014. Surface mesh generation based on imprinting of s-t edge patches. In *Proceedings of 23th International Meshing Roundtable*.

[19] Cifuentes, A. O., and A. Kalbag. 1992. A Performance Study of Tetrahedral and Hexahedral Elements in 3D Finite Element Structural Analysis. *Finite Elements in Analysis and Design* 12(3-4):74–78.

[20] Cook, W. A., and W. R. Oakes. 1982. Mapping Methods for Generating Three-dimensional Meshes. *Computers in Mechanical Engineering* 8:67–72.

[21] Davis, T. A. 2006. *Direct Methods for Sparse Linear Systems*, vol. 2. Siam.

[22] Derose, T., and M. Meyer. 2006. Harmonic coordinates. In *Pixar Technical Memo 06-02, Pixar Animation Studios.*

[23] Dudley, C. R., and S. J. Owen. 2014. Degenerate Hex Elements. *23rd International Meshing Roundtable* 301–312.

[24] Duren, P., and W. Hengartner. 1997. Harmonic mappings of multiply connected domains. *Pacific Journal of Mathematics* 180(2):201–220.

[25] Fan, Z. W., X. G. Jin, J. Q. Feng, and H. Q. Sun. 2005. Mesh Morphing Using Polycube-based Cross-parameterization. *Computer Animation and Virtual Worlds* 16(3–4):499–508.

[26] Floater, M. S. 2003. Mean Value Coordinates. *Computer Aided Geometric Design* 20(1):19–27.

[27] Floater, M. S. 2005. Mean Value Coordinates in 3D. *Computer Aided Geometric Design* 22(7):623–631.

[28] Gamelin, T. W. and Greene, R. E. 1983. *Introduction to topology.* Courier Corporation.

[29] Gass, Saul I. 2010. *Linear Programming: methods and applications*. Dover Publications.

[30] Gomes, H. 1999. *Warping and morphing of graphical objects*, vol. 1. Morgan Kaufmann.

[31] Goodrich, D. 1997. Generation of All-Quadrilateral Surface Meshers by Mesh Morphing. Master thesis, Briham Young University.

[32] Halpern, M. 1997. Industrial Requirements and Practices in Finite Element Meshing: a Survey of Trends. *Proceedings of 6th International Meshing Roundtable* 399–411.

[33] Joshi, P., M. Meyer, T. DeRose, B. Green, and T. Sanocki. 2007. Harmonic Coordinates for Character Articulation. In *ACM Transactions on Graphics (TOG)*, vol. 26, 71.

[34] Kanai, T., H. Suzuki, and F. Kimura. 1998. Three-dimensional geometric metamorphosis based on harmonic maps. *The Visual Computer* 14(4):166–176.

[35] Kanai, T., H. Suzuki, and F. Kimura. 1998. Three-Dimensional Geometric Metamorphosis based on Harmonic Maps. *The Visual Computer* 14(4):166–176.

[36] Kaneko, K., and Y. Okada. 2008. Skeleton based 3D Model Morphing Using Barycentric Map. *Computer Graphics, Imaging and Visualization, Fifth International Conference on IEEE* 132–137.

[37] Kjell Eikland. 2010. *lpsolve*. sourceforge.net/projects/lpsolve.

[38] Knepley, M. G., and D. A. Karpeev. 2009. Mesh Algorithms for PDE with Sieve i: Mesh Distribution. *Scientific Programming* 17(3):215–230.

[39] Knupp, P. M. 1998. Next-Generation Sweep Tool: a Method for Generating All-Hex Meshes on Two-and-One-Half Dimensional Geometries. In *Proceedings of 7th International Meshing Roundtable*, 505–513.

[40] Knupp, P. M. 1999. Applications of Mesh Smoothing: Copy, Morph, and Sweep on Unstructured Quadrilateral Meshes. *International Journal for Numerical Methods in Engineering* 45(1):37–45.

[41] Lai, M., S. Benzley, and D. White. 2000. Automated Hexahedral Mesh Generation by Generalized Multiple Source to Multiple Target Sweeping. *International Journal for Numerical Methods in Engineering* 49:261–275.

[42] Ledoux, F., and J. C. Weill. 2008. An Extension of the Reliable Whisker Weaving Algorithm. *Proceedings of 16th International Meshing Roundtable* 215–232.

[43] Lee, A., D. Dobkin, W. Sweldens, and P. Schroder. 1999. Multiresolution Mesh Morphing. In *Proceedings of ACM SIGGRAPH*, 343–350.

[44] Lee, A. W. F., W. Sweldens, P. Schroder, L. Cowsar, and D. Dobkin. 1998. MAPS: Multiresolution Adaptive Parameterization of Surfaces. In *Proceedings of ACM SIGGRAPH*, 95–104.

[45] Lerios, A., C. D. Garfinkle, and M. Levoy. 2010. Feature-based Volume Metamorphosis. *22nd Annual Conference on Computer Graphics and Interactive Techniques, ACM* 899–914.

[46] Li, T. S., R. M. McKeag, and C. G. Armstrong. 1995. Hexahedral Meshing Using Midpoint Subdivision and Integer Programming. *Computer Methods in Applied Mechanics and Engineering* 124(1):171–193.

[47] Lin, C. H., T. Y. Lee, H. K. Chu, and C. Y. Yao. 2005. Progressive Mesh Metamorphosis. *Computer Animation and Virtual Worlds* 16(3–4):487–498.

[48] Lipman, Y., D. Levin, and D. Cohen-Or. 2008. Green Coordinates. In *Proceedings of ACM Transactions on Graphics (TOG)*, vol. 20, 19–27.

[49] Martin D. Buhmann. 2003. *Radial Basis Functions: Theory and Implementations*. Cambridge University Press.

[50] Michikawa, T., T. Kanai, M. Fujita, and H. Chiyokura. 2001. Multiresolution Interpolation Meshes. *Computer Graphics and Applications* 60–69.

[51] Mingwu, L., S. E. Benzley, G. Sjaardema, and T. Tautges. 1996. A multiple source and target sweeping method for generating all-hexahedral finite element meshes. *Proceedings of the 5th International Meshing Roundtable* 217–225.

[52] Mitchell, S. A. 2014. Simple and Fast Interval Assignment Using Nonlinear and Piecewise Linear Objectives. *Proceedings of 22th International Meshing Roundtable* 203–221.

[53] Miyoshi, K., and T. D. Blacker. 2000. Hexahedral Mesh Generation Using Multi-Axis Cooper Algorithm. In *Proceedings of 9th International Meshing Roundtable*, 89–97.

[54] Mueller-Hannemann, M. 1999. Hexahedral Mesh Generation by Successive Dual Cycle Elimination. *Engineering with Computers* 15(3):269–279.

[55] Nealen, A., O. Sorkine, M. Alexa, and D. Cohen-Or. 2007. A Skeleton-based Interface for Detail-Preserving Mesh Editing. *ACM SIGGRAPH 2007, ACM* 42.

[56] Nieto, J. R., and A. Susin. 2013. Cage Based Deformations: a Survey. *Deformation Models, Springer Netherlands* 75–99.

[57] Nocedal, J., and S. J. Wright. 2006. *Conjugate Gradient Methods*. Springer, New York.

[58] o'Rourke, J. 1998. *Computational geometry in c*. Cambridge University Press, Cambridge, UK.

[59] Owen, S. J. 1998. A Survey of Unstructured Mesh Generation. *Proceedings of 7th International Meshing Roundtable* 239–267.

[60] Owen, S. J., and Saigal S. 2000. H-Morph: an Indirect Appraoch to Advancing Front Hex Meshing. *International Journal for Numerical Methods in Engineering* 49(1–2):289–312.

[61] Owen, S. J., and T. R. Shelton. 2013. Validation of Grid-based Hex Meshes with Computational Solid Mechanics. *22nd International Meshing Roundtable* 39–56.

[62] Price, M. A., and C. G. Armstrong. 1995. Hexahedral Mesh Generation by Medial Surface Subdivision: Part I. *International Journal for Numerical Methods in Engineering* 38(19):3335–3359.

[63] Price, M. A., and C. G. Armstrong. 1997. Hexahedral Mesh Generation by Medial Surface Subdivision: Part II. Solids with flat and concave edges. *International Journal for Numerical Methods in Engineering* 40(1):111–136.

[64] Remacle, J. F., C. Geuzaine, and G. Compere. 2009. High quality surface remeshing using harmonic maps. *International Journal of Numerical Methods in Engineering* 83(4):403–425.

[65] Remacle, J. F., C. Geuzaine, and G. Compere. 2012. Least-squares Approximation of Affine Mappings for Sweep Mesh Generation: Functional Analysis and Applications. *International Journal of Numerical Methods in Engineering* 1–15.

[66] Roca, X., J. Sarrate, and A. Huerta. 2005. A new least-squares approximation of affine mappings for sweep algorithms. In *Proceedings of 14th International Meshing Roundtable*, 433–448.

[67] Roca, X., J. Sarrate, and A. Huerta. 2006. Mesh Projection between Parametric Surfaces. *Communications in Numerical Methods in Engineering* 22(6):591–603.

[68] Roca, Xevi. 2009. Paving the path towards automatic hexahedral mesh generation. PhD dissertation, Universitat PolitÃÍcnica de Catalunya.

[69] Roca, Xevi, Josep Sarrate, and Antonio Huerta. 2004. Surface mesh projection for hexahedral mesh generation by sweeping. In *Proceedings of 13th International Meshing Roundtable*, 169–180.

[70] Ruiz-Girones, E., X. Roca, and J. Sarrate. 2009. A new procedure to compute imprints in multi-sweeping algorithms. In *Proceedings of 18th International Meshing Roundtable*.

[71] Ruiz-Girones, E., X. Roca, and J. Sarrate. 2011. Using a Computational Domain and a Three-Stage Node Location Procedure for Multi-Sweeping Algorithms. *Advances in Engineering Software* 42(9):700–713.

[72] Ruiz-Girones, E., and J. Sarrate. 2010. Generation of Structured Hexahedral Meshes in Volumes with Holes. *Finite Elements in Analysis and Design* 46(10):792–804.

[73] Ruiz-Girones, E., and J. Sarrate. 2010. Generation of Structured Meshes in Multiply Connected Surfaces Using SubMapping. *Advances in Engineering Software* 41(2): 379–387.

[74] Ruiz-Girones, Eloi. 2011. Automatic Hexahedral Meshing Algorithms: From Structured to Unstructured Meshes. PhD dissertation, Universitat Politecnica de Catalunya, Barcelona, Spain.

[75] Samareh, J. A. 2005. Geometry and grid/mesh generation issues for CFD and CSM shape optimization. *Optimization and Engineering* 6(1):21–32.

[76] Schneiders, R. 1996. A Grid-based Algorithm for the Generation of Hexahedral Element Meshes. *Engineering with Computers* 12(3–4):168–177.

[77] Schneiders, R. 1996. A Grid-based Algorithm for the Generation of Hexahedral Element Meshes. *Engineering with Computers* 12(3–4):168–177.

[78] Scott, M. A., S. E. Benzley, and S. J. Owen. 2006. Improved Many-to-One Sweeping. *International Journal for Numerical Methods in Engineering* 65(3):332–348.

[79] Scott, M. A., M. N. Earp, S. E. Benzley, and M. B. Stephenson. 2005. Adaptive Sweeping Techniques. 417–432.

[80] Sheffer, A., and V. Kraevoy. 2004. Pyramid Coordinates for Morphing and Deformation. *3D Data Processing, Visualization and Transmission, 2nd International Symposium on IEEE* 68–75.

[81] Shepherd, J. F. 2007. Topologic and geometric constraint-based hexahedral mesh generation. PhD dissertation, University of Utah.

[82] Shepherd, J. F., S. Benzley, and S. Mitchell. 2000. Interval Assignment for Volumes with Holes. *International Journal for Numerical Methods in Engineering* 49(1–2): 277–288.

[83] Sieger, D., S. Menzel, and M. Botsch. 2013. High Quality Mesh Morphing Using Triharmonic Radial Basis Functions. In *Proceedings of 21th International Meshing Roundtable*, 1–15.

[84] Sigma team, Argonne National Lab. 2015. *The lasso relations tool*. sigma.mcs.anl.gov/lasso-library.

[85] Sloan, S. W. 1993. A fast algorithm for generating constrained Delaunay triangulations. *Computers and Structures* 47(3):441–450.

[86] Staten, M. L., S. A. Canann, and S. J. Owen. 1998. BMSweep: Locating Interior Nodes During Sweeping. In *Proceedings of 7th International Meshing Roundtable*, 7–18.

[87] Staten, M. L., R. A. Kerr, S. J. Owen, and T. D. Blacker. 2006. Unconstrained Paving and Plastering: Progress Update. *Proceedings of 15th International Meshing Roundtable* 469–486.

[88] Staten, M. L., S. J. Owen, and T. D. Blacker. 2005. Unconstrained Paving and Plastering: a New Idea for All Hexahedral Mesh Generation. *Proceedings of 14th International Meshing Roundtable* 469–486.

[89] Staten, M. L., S. J. Owen, and T. D. Blacker. 2005. Unconstrained paving and plastering: a new idea for all hexahedral mesh generation. *14th International Meshing Roundtable* 399–416.

[90] T., Ju, S. Schaefer, and J. Warren. 2005. Mean Value Coordinates for Closest Triangular Meshes. *Pacific Journal of Mathematics* 180(2):201–220.

[91] Taniguchi, T., T. Goda, and H. Kasper. 1996. Hexahedral mesh generation of complex composite domain. In *Proceedings of 5th International Conference on Grid Generation in Computational Field Simmulations*, 699–707.

[92] Tarini, M., K. Hormann, P. Cignoni, and C. Montani. 2004. Polycube-maps. *ACM Transactions on Graphics (TOG)* 23(3):853–860.

[93] Tautges, T. 2004. MOAB-SD: Integrated Structured and Unstructured Mesh Representation. *Engineering with Computers* 20(3):286–293.

[94] Tautges, T. J. 2001. The Generation of Hexahedral Meshes for Assembly Geometry: Survey and Progress. *International Journal for Numerical Methods in Engineering* 50(12):2617–2642.

[95] Tautges, T. J., T. D. Blacker, and S. A. Mitchell. 1996. The Whisker-Weaving Algorithm: A Connectivity Based Method for Constructing All-Hexahedral Finite Element Meshes. *International Journal for Numerical Methods in Engineering* 39(19): 3327–3349.

[96] Tautges, T. J., and R. Jain. 2012. Creating geometry and mesh models for nuclear reactor core geometries using a lattice hierarchy-based approach. *Engineering with Computers* 28(4):319–329.

[97] Tautges, T. J., R. Meyers, K. Merkley, C. Stimpson, and C. Ernst. 2004. MOAB: a mesh-oriented database. SAND2004-1592, Sandia National Laboratories.

[98] Tautges, Timothy J. 2001. CGM: A Geometry Interface for Mesh Generation, Analysis and Other Applications. *Engineering with Computers* 17:299–314.

[99] Tautges, Timothy J., and Rajeev Jain. 2012. Harmonic mappings of multiply connected domains. *Engineering with Computers* 28(4):319–329.

[100] Toth, S., and A. Aszodi. 2010. CFD Analysis of Flow Field in a Triangular Rod Bundle. *Nuclear Engineering and Design* 240(2):352–363.

[101] Van, Z., A. Boer, and H. Bijl. 2007. Higher-order time integration through smooth mesh deformation for 3D fluid-structure interaction simulations. *Engineering with Computers* 224(1):414–430.

[102] Verma, C., and T. Tautges. 2012. Jaal Engineering a high quality all-quadrilateral mesh generator. *Proceedings of 20th International Meshing Roundtable* 511–530.

[103] Verma, C. S., P. F. Fischer, S. E. Lee, and F. Loth. 2005. An All-Hex Meshing Strategy for Bifurcation Geometries in Vascular Flow Simulation. *Proceedings of 14th International Meshing Roundtable* 363–375.

[104] Wang, F., and M. Luca. 2012. Automated Hex Meshing for Turbomachinery Secondary Air System. *Proceedings of 21th International Meshing Roundtable* 549–566.

[105] Wang, Y., M. Gupta, and S. Zhang. 2005. High Resolution Tracking of Non-rigid 3D Motion of Densely Sampled Data Using Harmonic Maps. *Computer Vision, ICCV, 10th IEEE International Conference on IEEE* 388–395.

[106] Wang, Y., M. Gupta, S. Zhang, S. Wang, X. Gu, D. Samaras, and P. Huang. 2008. MAPS: Multiresolution Adaptive Parameterization of Surfaces. In *Proceedings of IEEE International Conference on Computer Vision*, 283–300.

[107] Weingarten, V. I. 1994. The Controversy over Hex or Tet Meshing. *Machine Design* 74–78.

[108] Wendland, H. 2005. *Scattered data approximation*. Cambridge University Press, Cambridage, UK.

[109] White, D. R., M. Lai, S. E. Benzley, and G. D. Sjaardema. 1995. Automated Hexahedral Mesh Generation by Virtual Decomposition. *Proceedings of 4th International Meshing Roundtable* 165–176.

[110] White, D. R., R. W. Leland, S. Saigal, and S. J. Owen. 2001. The Meshing Complexity of a Solid: an Introduction. *10th International Meshing Roundtable* 373–384.

[111] White, D. R., S. Saigal, and S. J. Owen. 2003. Meshing Complexity of Single Part CAD Models. *12th International Meshing Roundtable* 121–134.

[112] White, D. R., S. Saigal, and S. J. Owen. 2004. CCSweep: automatic decomposition of multi-sweep volumes. *Engineering with Computers* 20(3):222–236.

[113] White, D. R., and T. J. Tautges. 2000. Automatic Scheme Selection for Toolkit Hex Meshing. *International Journal for Numerical Methods in Engineering* 49(1): 127–144.

[114] Whiteley, M., D. White, S. Benzley, and T. Blacker. 2006. Two and Three-Quarter Dimensional Meshing Facilitators. *Engineering with Computers* 12(3):144–154.

[115] Yan, H. B., S. M. Hu, and R. Martin. 2007. 3D Morphing Using Strain Field Interpolation. *Journal of Computer Science and Technology* 22(1):147–155.

[116] Yang, H. P., and B. Juttler. 2007. 3D Shape Metamorphosis based on T-Spline Level Sets. *The Visual Computer* 23(12):1015–1025.

[117] Yoo, D. J. 2007. Three-Dimensional Morphing of Similar Shapes Using a Template Mesh. *International Journal of Precision Engineering and Manufacturing* 10(1): 147–155.

[118] Zeng, W., X. T. Yin, M. Zhang, F. Luo, and X. F. Gu. 2009. Generalized koebe's method for conformal mapping multiply connected domains. In *SIAM/ACM Joint Conference on Geometric and Physical Modeling, ACM*, 89–100.

[119] Zhang, Y., and C. Bajaj. 2006. Adaptive and Quality Quadrilateral/Hexahedral Meshing from Volumetric Data. *Computer Methods in Applied Mechanics and Engineering* 195(9–12):942–960.

[120] Zhang, Y., C. Bajaj, and B. S. Sohn. 2005. 3D Finite Element Meshing from Imaging Data. *Computer Methods in Applied Mechanics and Engineering* 194(48–49):5083–5160.

A APPENDIX

A.1 Affine Transformation

Suppose there is a layer mesh of points $p_j \in R^3$ with the bounding loop L consisting of points $\{\mathbf{x}_k\}, k = 1, 2, \cdots, K, K >= 3$. A second loop L' consisting of K points $\{\mathbf{x}'_k\}$ in R^3 is given as well. A vector $\mathbf{b} \in R^3$ and a 3×3 non-singular linear transformation $\boldsymbol{T}$ between two loops are computed such that

$$\mathbf{x}'_k = \mathbf{T}\mathbf{x}_k + \mathbf{b} \tag{A.1}$$

The loop center for L and L' is defined as follows

$$\mathbf{c} = \frac{1}{K}\sum_k \mathbf{x}_k \tag{A.2}$$

$$\mathbf{c}' = \frac{1}{K}\sum_k \mathbf{x}'_k \tag{A.3}$$

Let

$$\mathbf{u}_k = \mathbf{x}_k - \mathbf{c} \tag{A.4}$$

$$\mathbf{u}'_k = \mathbf{x}'_k - \mathbf{c}' \tag{A.5}$$

Except for certain ideal cases such as pure translations or rotations, a transformation between two arbitrary loops does not exist. A *Least-Square* fit to the bounding loop data is computed by minimizing the non-negative function

$$F(\mathbf{T}) = \frac{1}{K}\sum_{k=1}^{K} |\mathbf{u}'_k - \mathbf{T}\mathbf{u}_k| \tag{A.6}$$

In order to find $\boldsymbol{T}$, set $\partial F/\partial \boldsymbol{T}_{ij} = 0$ to get three uncoupled linear systems that can be written as

$$\mathbf{TM} = \mathbf{F} \tag{A.7}$$

$$\mathbf{M} = \sum_k (\mathbf{u}_k \otimes \mathbf{u}_k) \tag{A.8}$$

$$\mathbf{F} = \sum_k (\mathbf{u}_k \otimes \mathbf{u}'_k) \tag{A.9}$$

Where $\otimes$ is the vector outer product.

In order to avoid singular problem if there is planar loops, sets of vectors $\mathbf{u}_k$ and $\mathbf{u}'_k$ are redefined as follows

$$\mathbf{u}_k = \mathbf{x}_k - (2\mathbf{c} - \mathbf{c}') \tag{A.10}$$

$$\mathbf{u}'_k = \mathbf{x}'_k - \mathbf{c} \tag{A.11}$$

A.2 Least-Square Approximation

Let $X = \{\mathbf{x}^i\}_{i=1,\cdots,m} \subset R^n$ be a set of *source* points, and $X' = \{\mathbf{x}'^i\}_{i=1,\cdots,m} \subset R^n$ be a set of *target* points with $m >= n$. The goal is to find a mapping $\phi : R^n \to R^n$ such that

$$\mathbf{x}'^i = \phi(\mathbf{x}^i), i = 1, \cdots, m \tag{A.12}$$

The affine mapping is determined by a *Least-Square* fitting of the given data. Thus, ϕ is computed by minimizing the functional

$$E(\phi) = \sum_{i=1}^{m} \parallel \mathbf{x}'^i - \phi(\mathbf{x}^i) \parallel \tag{A.13}$$

The affine mapping is optimized when the sum of the square of distances between the *target* points and the image of the *source* points is minimized. Let

$$\mathbf{c} = \frac{1}{m} \sum_{i=1}^{m} \mathbf{x}^i \tag{A.14}$$

$$\mathbf{c}^{X'} = \frac{1}{m} \sum_{i=1}^{m} \mathbf{x}'^i \tag{A.15}$$

be the geometric centers of X and X'. A new formulation with the *Least-Square* form is formulated as follows

$$H(\mathbf{T}; \mathbf{u}^X, \mathbf{u}^{X'}) = \sum_{i=1}^{m} \parallel \mathbf{x}'^i - \mathbf{c}^{X'} - \mathbf{T}(\mathbf{x}^i - \mathbf{c}^X) \parallel + \parallel \mathbf{u}^{X'} - \mathbf{T}\mathbf{u}^X \parallel \tag{A.16}$$

where $\mathbf{u}^X \in R^n$ and $\mathbf{u}^{X'} \in R^n$. Parametric vectors $\mathbf{u}^X$ and $\mathbf{u}^{X'}$ can be properly selected in order to obtain several desired properties of functional H. They can be selected as follows.

1 X hyperplanar and X' hyperplanar: $\mathbf{u}^X = \mathbf{n}^X$ and $\mathbf{u}^{X'} = \mathbf{n}^{X'}$

2 X hyperplanar and X' non-hyperplanar: $\mathbf{u}^X = \mathbf{n}^X$ and $\mathbf{u}^{X'} = \tilde{\mathbf{n}}^{X'}$

3 X non-hyperplanar and X' hyperplanar: $\mathbf{u}^X = 0$ and $\mathbf{u}^{X'} = 0$

4 X non-hyperplanar and X' non-hyperplanar: $\mathbf{u}^X = 0$ and $\mathbf{u}^{X'} = 0$

where $\tilde{\mathbf{n}}^{X'}$ is a measure of the normal vector to X'. $\mathbf{n}^X$ and $\mathbf{n}^{X'}$ are unitary normal vector of X and X'.

A.3 Mean Value Coordinates

The Mean Value Coordinates(***MVC***) was proposed by Floater[26, 27] and extended by Ju[90]. Give points $\boldsymbol{x}$ to be deformed and a closed mesh, for each triangle T in the mesh with vertices $\{\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3\}$ and associated values $\{f_1, f_2, f_3\}$.

(1) Compute the mean vector $\boldsymbol{m}$

$$\boldsymbol{m} = \sum_i \frac{1}{2}\theta_i \boldsymbol{n}_i \tag{A.17}$$

where θ_i is the arc length and $\boldsymbol{n}_i$ is the normal vector such as Fig.A.1.

(2) Compute the weight w_i

$$w_i = \frac{\boldsymbol{n}_i \cdot \boldsymbol{m}}{\boldsymbol{n}_i \cdot (\boldsymbol{p}_i - \boldsymbol{x})} \tag{A.18}$$

(3) Update the denominator and numerator of $\hat{f}[\boldsymbol{x}]$ respectively by adding $\sum_i w_i$ and $\sum_i w_i f_i$.

$$\hat{f}[\boldsymbol{x}] = \frac{\sum_k \sum_i w_i^k \cdot f_i^k}{\sum_k \sum_i w_i^k} \tag{A.19}$$

After the above three steps, the ***MVC*** coordinates can be computed. The new objects under the deformed cage are interpolated as follows.

$$\boldsymbol{x} = \sum_{i=1}^{N_c} f_i^x \boldsymbol{p}_i' \tag{A.20}$$

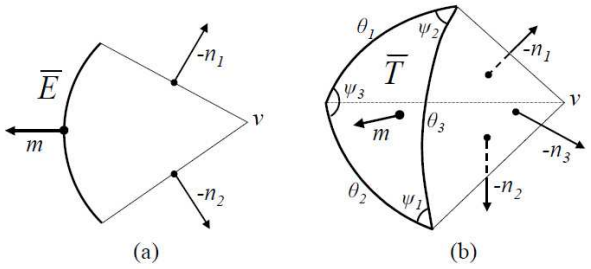

Figure A.1: Mean vector m on a circular arc $\overline{E}$ with edge normals $\boldsymbol{n}_i$ (a) and on a spherical triangle $\overline{T}$ with arc lengths θ_i and face normals $\boldsymbol{n}_i$[90]

Algorithm 5 Pseudo Code of Mean Value Coordinates on a Closed Triangular Mesh

INPUT: Cage vertex $\boldsymbol{p}$, arc length θ_i on the cage faces, object points $\boldsymbol{x}$ to be deformed.

OUTPUT: weight f_x for object point $\boldsymbol{x}$ with respect to cage vertices.

1. **FOR EACH** vertex $\boldsymbol{p}_j$ with values f_j
2. $\quad d_j \leftarrow \| \boldsymbol{p}_j - \boldsymbol{x} \|$
3. $\quad$ **IF** $d_j < \eta$
4. $\quad\quad$ **RETURN** f_j
5. $\quad \boldsymbol{u}_j \leftarrow (\boldsymbol{p}_j - \boldsymbol{x})/d_j$
6. $totalF \leftarrow 0$
7. $totalW \leftarrow 0$
8. **FOR EACH** triangle with vertices $\boldsymbol{p}_1$, $\boldsymbol{p}_2$, $\boldsymbol{p}_3$ and values f_1,f_2,f_3
9. $\quad l_i \leftarrow \| \boldsymbol{u}_{i+1} - \boldsymbol{u}_{i-1} \| \; //for \quad i = 1, 2, 3$
10. $\quad h \leftarrow (\sum \theta_i)/2$
11. $\quad$ **IF** $\pi - h < \eta$
    $\quad\quad //\boldsymbol{x}$ lies on t, use 2D barycentric coordinates
12. $\quad\quad w_i \leftarrow sin[\theta_i] l_{i-1} l_{i+1}$
13. $\quad\quad$ **RETURN** $(\sum w_i f_i)/(\sum w_i)$
14. $\quad c_i \leftarrow (2sin[h]sin[h - \theta_i])/(sin[\theta_{i+1}]sin[\theta_{i-1}]) - 1$
15. $\quad s_i \leftarrow sign[det[u_1, u_2, u_3]]\sqrt{1 - c_i^2}$
16. $\quad$ **IF** $\exists i, |s_i| \leqslant \eta$
    $\quad\quad //\boldsymbol{x}$ lies outside t on the same plane, ignore t
17. $\quad\quad$ **CONTINUE**
18. $\quad w_i \leftarrow (\theta_i - c_{i+1}\theta_{i-1} - c_{i-1}\theta_{i+1})/(d_i sin[\theta_{i+1}] s_{i-1})$
19. $\quad totalF+ = \sum w_i f_i$
20. $\quad totalW+ = \sum w_i$
21. $f_x \leftarrow totalF/totalW$

A.4 Green Coordinates

To achieve a natural deformation with shape preserving, Green coordinates[48] have been proposed where cage face data is added to the deformation operator besides cage vertex data.

$$\boldsymbol{v} = F(\boldsymbol{v}; C) = \sum_{i \in I_{\boldsymbol{V}}} \varphi_i(\boldsymbol{v}) \boldsymbol{v}_i + \sum_{j \in I_{\boldsymbol{T}}} \phi_j(\boldsymbol{v}) \boldsymbol{n}(t_j) \tag{A.21}$$

where $C = (\boldsymbol{V}, \boldsymbol{T})$ is the cage, $\boldsymbol{V}$ are the cage vertices, $\boldsymbol{T}$ are the cage simplexes (edges or faces), φ_i are the coordinates w.r.t. cage's vertices, ϕ_j are coordinates with respect to the normals $\boldsymbol{n}(t_j)$ of cage face t_j, $I(\boldsymbol{V})$ is the set of cage vertices and $I(\boldsymbol{T})$ is the set of cage faces. With new cage $C' = (\boldsymbol{V}', \boldsymbol{T}')$ as inputs, the deformed object can be interpolated as follows

$$\boldsymbol{v}' \to F(\boldsymbol{v}'; C') = \sum_{i \in I_{\boldsymbol{V}}} \varphi_i(\boldsymbol{v}) \boldsymbol{v}'_i + \sum_{j \in I_{\boldsymbol{T}}} \phi_j(\boldsymbol{v}) s_j \boldsymbol{n}(t'_j) \tag{A.22}$$

where $\boldsymbol{v}'_i$ and t'_j are the modified vertices and faces of C', respectively. In Equation (A.22), a new term $\{s_j\}_{j \in I_{\boldsymbol{T}}}$ is added to ensure some properties such as scale invariance.

Algorithm 6 Pseudo Code of 3D Green Coordinates Algorithm

INPUT: cage $P = (V, T)$, object point set $\Lambda = \eta$ to be deformed , normal $\mathbf{n}$ of cage faces

OUTPUT: 3D GC $\phi_i(\boldsymbol{\eta})$,$\varphi_j(\boldsymbol{\eta})$,$i \in I_V$ (set of cage vertices),$j \in I_T$ (set of cage triangle faces),$\boldsymbol{\eta} \in \Lambda$

SET all $\phi_i = 0$ and $\varphi_j = 0$
FOR EACH *point* $\boldsymbol{\eta} \in \Lambda$
  FOR EACH face $j \in \mathbf{I}_T$ with vertices $\boldsymbol{v}_{j_1}$,$\boldsymbol{v}_{j_2}$,$\boldsymbol{v}_{j_3}$
    FOR $l = 1, 2, 3$
      $\boldsymbol{v}_{jl} = \boldsymbol{v}_{jl} - \boldsymbol{\eta}$
    $\boldsymbol{p} = (\boldsymbol{v}_{j_1} \cdot \boldsymbol{n}(t_j))\boldsymbol{n}(t_j)$
    FOR $l = 1, 2, 3$
      $s_l = sign(((\boldsymbol{v}_{jl} - \boldsymbol{p}) \times (\boldsymbol{v}_{j,l+1} - \boldsymbol{p})) \cdot \boldsymbol{n}(t_j))$
      $I_l = \mathbf{GCTriInt}(\boldsymbol{p}, \boldsymbol{v}_{jl}, \boldsymbol{v}_{j,l+1}, 0)$
      $II_l = \mathbf{GCTriInt}(0, \boldsymbol{v}_{j,l+1}, \boldsymbol{v}_{jl}, 0)$
      $\boldsymbol{q}_l = \boldsymbol{v}_{j,l+1} \times \boldsymbol{v}_{jl}$
      $\boldsymbol{N}_l = \boldsymbol{q}_l / \parallel \boldsymbol{q}_l \parallel$
    $I = -|\sum_{k=1}^{3} s_k I_k|$
    $\varphi_j(\boldsymbol{\eta}) = -I$
    $\boldsymbol{\omega} = \boldsymbol{n}(t_j) I + \sum_{k=1}^{3} \mathbf{N}_k II_k$
    IF $\parallel \boldsymbol{\omega} \parallel > \epsilon$
      FOR $l = 1, 2, 3$
        $\phi_{jl}(\boldsymbol{\eta}) = \phi_{jl}(\boldsymbol{\eta}) + \frac{\mathbf{N}_{l+1} \cdot \boldsymbol{\omega}}{\mathbf{N}_{l+1} \cdot \boldsymbol{v}_{jl}}$

---

function $\mathbf{GCTriInt}(\boldsymbol{p}, \boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{\eta})$
$\alpha = cos^{-1}(\frac{(\boldsymbol{v}_2 - \boldsymbol{v}_1) \cdot (\boldsymbol{p} - \boldsymbol{v}_1)}{\|\boldsymbol{v}_2 - \boldsymbol{v}_1\| \|\boldsymbol{p} - \boldsymbol{v}_1\|})$
$\beta = cos^{-1}(\frac{(\boldsymbol{v}_1 - \boldsymbol{p}) \cdot (\boldsymbol{v}_2 - \boldsymbol{p})}{\|\boldsymbol{v}_1 - \boldsymbol{p}\| \|\boldsymbol{v}_2 - \boldsymbol{p}\|})$
$\lambda = \parallel \boldsymbol{p} - \boldsymbol{v}_1 \parallel^2 sin(\alpha)$
$c = \parallel \boldsymbol{p} - \boldsymbol{\eta} \parallel^2$
FOR $\theta = \pi - \alpha, \pi - \alpha - \beta$
  $S = sin(\theta)$
  $C = cos(\theta)$
  $I_\theta = \frac{-sign(S)}{2}[2\sqrt{c}tan^{-1}(\frac{\sqrt{c}C}{\sqrt{\lambda + S^2 c}}) + \sqrt{\lambda}log(\frac{2\sqrt{\lambda}S^2}{(1-C)^2}(1 - \frac{2cC}{c(1+C) + \lambda + \sqrt{\lambda^2 + \lambda c S^2}}))]$
RETURN $\frac{-1}{4\pi}|I_{\pi-\alpha} - I_{\pi-\alpha-\beta} - \sqrt{c}\beta|$

A.5 Triharmonic Radial Basis Method

Employing a space deformation approach, the interior node placement inside volumes can be treated as an abstract scattered data interpolation problem. We are looking for a deformation function $\mathbf{d} : \boldsymbol{IR}^3 \rightarrow \boldsymbol{IR}^3$ that (i) exactly interpolates the prescribed boundary displacements $d(\boldsymbol{v}_i) = (\boldsymbol{v}'_i - \boldsymbol{v}_i)$ and (ii) smoothly interpolates these displacements into the mesh interior. ***RBF***s are well known to be suitable for solving this type of problem[108]. An ***RBF*** deformation is represented as a linear combination of radially symmetric kernels $\varphi_j(\mathbf{x}) = \varphi(\parallel \mathbf{x}_j - \mathbf{x} \parallel)$, located at centers $\mathbf{x}_j \in \boldsymbol{IR}^3$ and weighted by $\mathbf{w}_j \in \boldsymbol{IR}^3$, plus a linear polynomial that guarantees linear precision

$$d(\mathbf{x}) = \sum_{j=1}^{m} \mathbf{w}_j \varphi_j(\mathbf{x}) + \sum_{k=1}^{4} \mathbf{q}_k \pi_k(\mathbf{x}) \tag{A.23}$$

where $\{\pi_1, \pi_2, \pi_3, \pi_4\} = \{x, y, z, 1\}$ is a basis of the space of linear trivariate polynomials, weighted by coefficients $\mathbf{q}_k \in \boldsymbol{IR}^3$. Note that the polynomial term is important, since it guarantees to find the optimal affine motion (translation, rotation, scaling or combinations of them) contained in the prescribed displacements $\boldsymbol{v}_i \rightarrow \boldsymbol{v}'_i$.

The choice of the kernel function $\boldsymbol{\varphi} : \boldsymbol{IR}^3 \rightarrow \boldsymbol{IR}^3$ has a significant influence on the quality of interior node placement and corresponding hexahedral meshes. It is proposed to choose $\varphi(r) = r^3$[83], which by construction results in a triharmonic deformation function($\bigtriangledown^3 \mathbf{d} = 0$). This is due to the fact that the ***RBF*** kernels φ_j are fundamental solutions of the *tri-Laplacian* equation. Consequently, the fairness energy is minimized by the ***RBF*** deformation.

$$\iiint_{IR^3} \parallel \frac{\partial^3 \mathbf{d}}{\partial x^3} \parallel^2 + \parallel \frac{\partial^3 \mathbf{d}}{\partial x^2 \partial y} \parallel^2 + ... + \parallel \frac{\partial^3 \mathbf{d}}{\partial z^3} \parallel^2 \, dxdydz \tag{A.24}$$

Satisfying the interpolation constraints $\mathbf{d}(\boldsymbol{v}_i) = (\boldsymbol{v}'_i - \boldsymbol{v}_i)$ amounts to placing ***RBF***

kernels at the constraint positions (i.e., $\boldsymbol{x}_j = \boldsymbol{v}_j$) and finding the coefficients $\mathbf{w}_j$ and $\mathbf{q}_k$ by solving the $(m+4) \times (m+4)$ linear system.

$$\begin{pmatrix} \varphi_1(\mathbf{v}_1) & \cdots & \varphi_m(\mathbf{v}_1) & \pi_1(\mathbf{v}_1) & \cdots & \pi_4(\mathbf{v}_1) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \varphi_1(\mathbf{v}_m) & \cdots & \varphi_m(\mathbf{v}_m) & \pi_1(\mathbf{v}_m) & \cdots & \pi_4(\mathbf{v}_m) \\ \pi_1(\mathbf{v}_1) & \cdots & \pi_1(\mathbf{v}_m) & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \pi_4(\mathbf{v}_1) & \cdots & \pi_4(\mathbf{v}_m) & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} \mathbf{w}_1^T \\ \vdots \\ \mathbf{w}_m^T \\ \mathbf{q}_1^T \\ \vdots \\ \mathbf{q}_4^T \end{pmatrix} = \begin{pmatrix} (\mathbf{v}'_1 - \mathbf{v}_1)^T \\ \vdots \\ (\mathbf{v}'_m - \mathbf{v}_m)^T \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{pmatrix} \tag{A.25}$$

After the linear system is solved, interior nodes can be placed in M' by simply evaluating ***RBF*** deformation at each interior node in the idealized cage in the physical space:

$$\boldsymbol{\eta}'_i = \boldsymbol{\eta}_i + \mathbf{d}(\boldsymbol{\eta}_i) \tag{A.26}$$

A.6 Ruiz-Girones's *LP* Method for *Submapping*

In order to deal with shortcomings from heuristic vertex classification during *Submapping*, E. Ruiz-Girones et al.[73] used ***LP*** to slightly correct the vertex classification based on angles once it fails to satisfy the *Submapping* constraint(6.4).

The basic idea of Ruiz-Girones's method is to minimize differences between a new vertex classification and heuristic vertex classification based on angles while it is subject to the *Submapping* constraint(6.4). The above ***LP*** model can be summarized as follows.

$$objective \qquad min \sum_{i=1}^{N} |\alpha_i - \overline{\alpha_i}| \tag{A.27}$$

s.t.

$$\sum_{i=1}^{N} \alpha_i = 4 \qquad \alpha_i \in \{-2, -1, 0, 1\} \tag{A.28}$$

where α_i is a new vertex classification at a vertex i and N is the total number of vertices which define a geometry. In order to avoid big variation difference of vertex types between α_i and $\overline{\alpha_i}$, one more constraint is added

$$|\alpha_i - \overline{\alpha_i}| \leq 1 \qquad i = 1, \cdots, N \tag{A.29}$$

The constraint(A.29) ensures that the variation of vertex classification at each vertex is bounded. For example, an ***END*** vertex could be converted to ***SIDE***; a ***SIDE*** vertex could be converted to ***END*** or ***CORNER***; a ***CORNER*** vertex could be converted to ***SIDE*** or ***REVERSAL*** and a ***REVERSAL*** vertex could be converted to ***CORNER***. The constraint(6.4) ensures that the optimal solution of ***LP*** model is always a valid vertex classification. One example is shown in Fig.A.2 where the heuristic method fails to classify all the vertices successfully due to the *Submapping* constraint(6.4) while Ruiz-Girones's

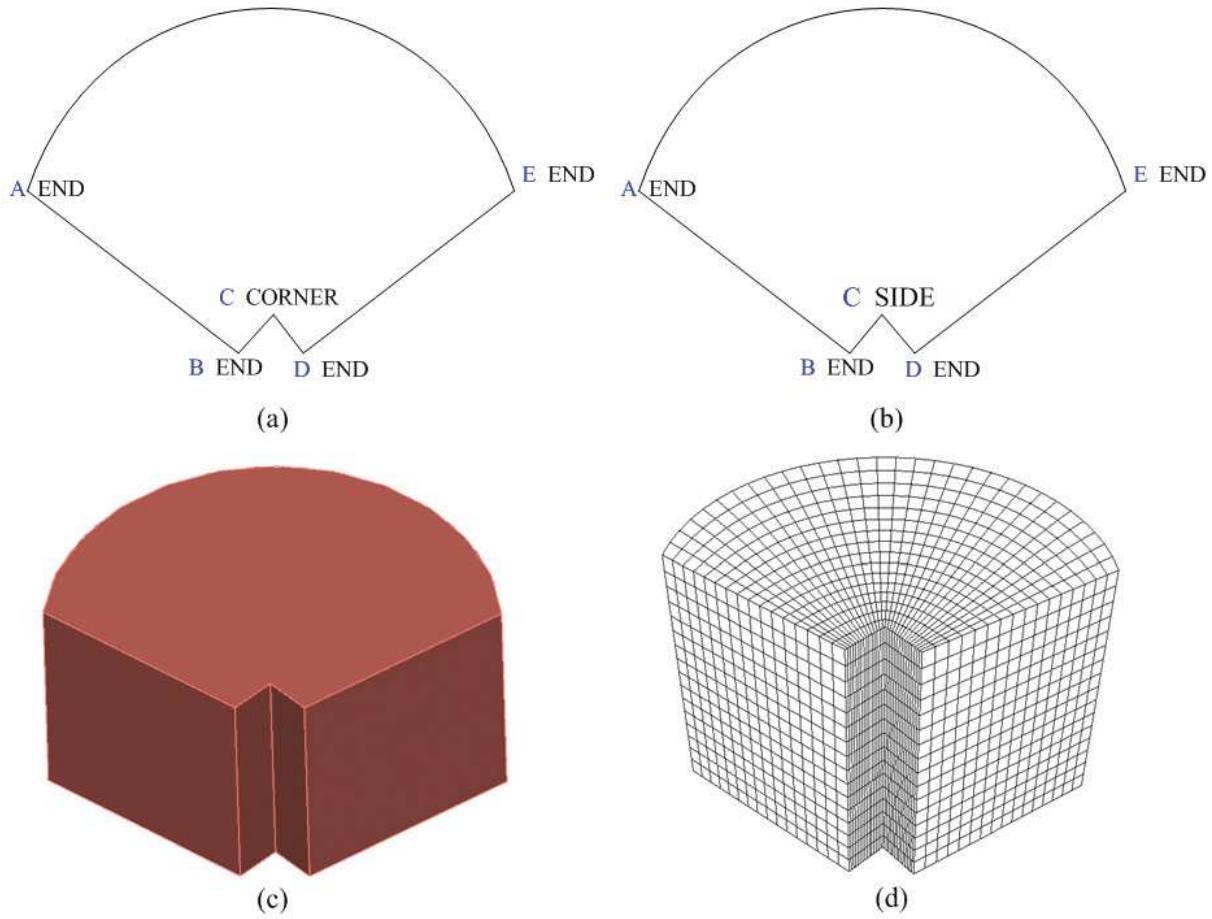

Figure A.2: An example of invalid vertex classification: (a)an invalid vertex classification purely based on angles; (b)new valid vertex classification based on ***LP*** model: a ***CORNER*** vertex is converted to be ***SIDE***; (c)geometry model; (d)structured quadrilateral mesh generated by *submapping*

method can correctly modify the initial vertex classification such that the top surface in Fig.A.2 is submappable.

However, the variation constraint (A.29) can sometimes cause problems and make ***LP*** fail to obtain a valid vertex classification. One example is shown in Fig.6.6 where Ruiz-Girones's method can not generate a valid vertex classification due to the variation constraint(A.29).

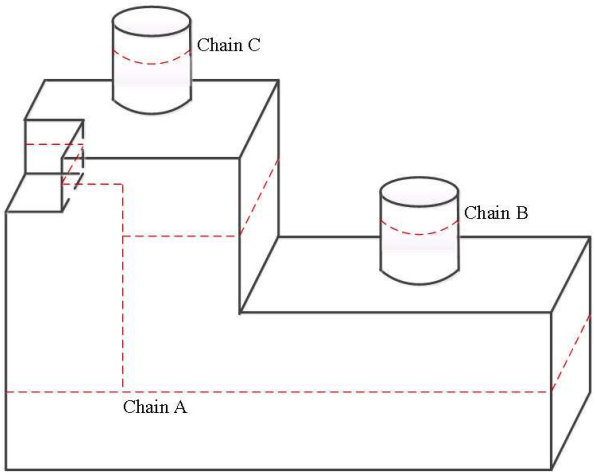

Figure A.3: An example with three chains, A, B and C

A.7 White's Method of Sweep Detection

In order to address the sweepability problem, chain definitions[113] are described as follows.

Chain: Consider a traversal across a mappable surface, from one side to the opposite side. If the surface sharing the side traversed to is mappable or submappable, it can be traversed in the same way. Traversal stops either when a non-mappable surface is encountered or when a side is encountered which has already been traversed. If all paths of the traversal end on a side that has already been traversed, the group of surfaces forms a loop of surfaces sharing edges of a given parameter; this loop is defined as a chain. In the simplest case, a chain can be a single periodic surface or a sequence of 4-sided mappable surfaces. Figure A.3 shows an example of chains where there are three chains.

In the reference[113], constraints which must be met for a volume to be sweepable are described, which are necessary but not sufficient conditions for sweepability.

Lemma A.1. A swept mesh is bounded by one or more non-intersecting chains.

Lemma A.2. The side (*linking*) surfaces generated by sweeping one or more contiguous sets of mesh faces into a third dimension are mapped if the boundary of the regions being

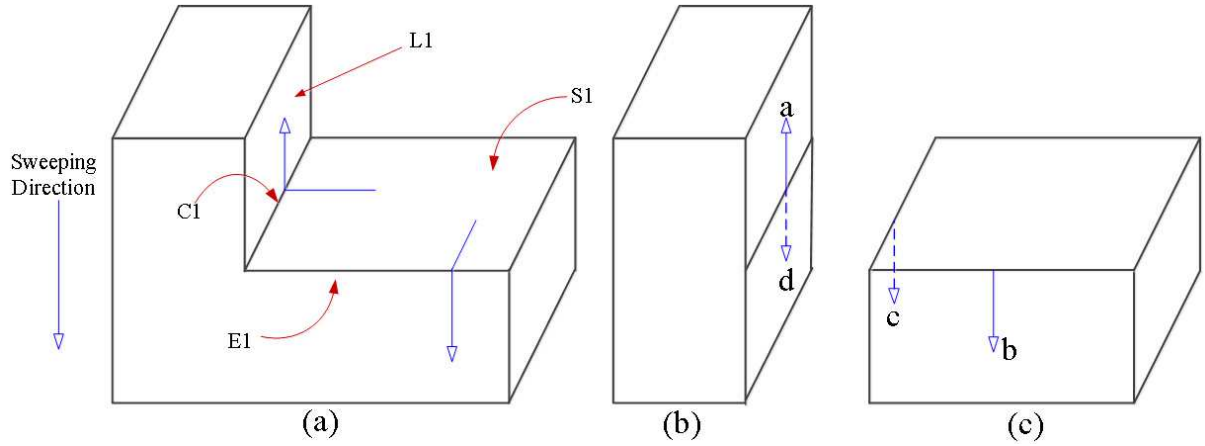

Figure A.4: Traversing from a *source* surface over a Corner-type edge (C1) and End-type edge (E1) results in traversing the *linking* surface in opposite directions along the same ij parameters

added or subtracted from the sweep do not intersect the original boundary of the sweep, or submapped if the boundaries do intersect.

Lemma A.3. In a swept volume:

(a) edges between *source*/*target* surfaces and *linking* surfaces are always of type ***END*** or ***CORNER***;

(b) edges between *source*/*target* surfaces are always of type ***SIDE*** or ***REVERSAL***;

(c) edges between *linking* surfaces that are perpendicular to the sweeping direction (i.e. edges that bound a single layer) are always of type ***SIDE*** or ***REVERSAL***; and

(d) edges between *linking* surfaces that are parallel to the sweep direction (i.e. edges that bound multiple layers) can be any type.

Lemma A.4. Each set of contiguous *source*/*target* surfaces is bounded by edges having a common value of the traversed parameter with respect to the bounding *linking* surface(s).

Lemma A.5. Traversing from a set of contiguous *source*/*target* surfaces over any boundary edge of type ***End*** results in traversing a *linking* surface in the non-traversed parameter, no matter which ***END***-type edge is traversed; similarly for traversing any ***CORNER***-type edge.

Lemma A.6. All *linking* surfaces in a swept mesh can be assigned a global, consistent ij parameterization.

Lemma A.7. Traversing from the same set of contiguous *source*/*target* surface(s) onto a *linking* surface over an ***END***-type versus a ***CORNER***-type edge will result in traversing the same ij parameter in the opposite directions (e.g. $+i$ and $-i$) on the *linking* surface.

Theorem A.1. A volume is sweepable only if

(a) The volume has one or more non-intersecting chains;

(b) For each set of contiguous *source*/*target* surfaces, traversing over all E-type or C-type edges bounding the set results in traversing in the same global ij parameter and direction on the *linking* surface(s).

(c) For each set of contiguous *source*/*target* surfaces bounded by both E-type and C-type edges, traversing over any E-type edge bounding the set results in traversing in the same global ij parameter and opposite direction on the *linking* surface as resulting from traversing over any C-type edge bounding the set.

Theorem A.2. To ensure element angles less than 180 degree and non-overlapping elements, ***End***-type edges must be less than 180 degrees, and ***Side***, ***Corner*** and ***Reversal*** edges must be less than 360 degree. All angles must be greater than 0 degree.

Theorem A.3. Volumes in which *source*/*target* surfaces share an edge of type ***Reversal*** require multiple *source* and *target* sweeping; volumes in which *source*/*target* and *linking* surfaces share edges of type ***Corner*** require multiple *source*/single *target* or multiple

source/*target* sweeping; volumes in which *linking* surfaces share edges perpendicular to the sweep direction of type ***Reversal*** require multiple *source*/single *target* sweeping; and volumes containing multiple *source*/*target* surfaces require multiple *source* and possibly multiple *target* sweeping.

Theorem A.4. Sweepable volumes contain constraints on the topology of *linking* surfaces; these constraints depend on the sweeping implementation being used.