2005-11-28

Interior Node Projection Techniques in Sweeping Algorithms

Michael Andrew Scott
Brigham Young University - Provo

Follow this and additional works at: https://scholarsarchive.byu.edu/etd
Part of the Civil and Environmental Engineering Commons

BYU ScholarsArchive Citation
https://scholarsarchive.byu.edu/etd/335

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
INTERIOR NODE PROJECTION TECHNIQUES IN
SWEEPING ALGORITHMS

by
Michael A. Scott

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Civil and Environmental Engineering
Brigham Young University
April 2006
BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Michael A. Scott

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date

Steven E. Benzley, Chair

Date

Richard J. Balling

Date

Steven J. Owen
As chair of the candidate’s graduate committee, I have read the thesis of Michael A. Scott in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Steven E. Benzley
Chair, Graduate Committee

Accepted for the Department

E. James Nelson
Graduate Coordinator

Accepted for the College

Alan R. Parkinson
Dean, Ira A. Fulton College of Engineering and Technology
ABSTRACT

INTERIOR NODE PROJECTION TECHNIQUES IN SWEEPING ALGORITHMS

Michael A. Scott

Department of Civil and Environmental Engineering

Master of Science

The enhancement of node projection techniques in sweeping is the subject of this thesis. Sweeping is a method used to produce all-hexahedral finite element meshes on certain classes of geometry. The placement of nodes in the interior of the geometry during the sweeping process remains a difficult problem. This thesis presents advancements in this area which improve the speed of the algorithm and the resulting element quality. A comparison of existing projection methods was performed. The existing Faceted projection sweeping method was extended to be applicable to more general classes of sweepable geometry. This comparison and extension of node projection algorithms led to the development of a new node projection technique: the SmartAffine method. This method builds on previous techniques and is characterized by its speed. Finally, a technique for coupling node projection techniques is presented. This technique characterizes the complexity of the sweepable geometry and applies the most appropriate node projection scheme. This is accomplished without user interaction and improves the speed of the sweeping algorithm and the quality of swept meshes.
Table of Contents

List of Figures viii

List of Tables ix

1 INTRODUCTION 1

2 POLYMORPHIC MANY-TO-ONE SWEEP TOOL 3

2.1 Introduction .................................................. 3

2.1.1 One-to-One Sweeping ................................. 3

2.1.2 Many-to-One Sweeping .............................. 3

2.1.3 Interior Node Placement ................................. 4

2.1.4 Comparison with Existing Methods .................. 6

2.2 Polymorphic Many-to-One Sweep Tool ................... 7

2.2.1 Ordering or Layering of Source Surfaces ........ 7

2.2.2 Creation of One-to-One Sweep Blocks ............... 8

2.2.3 Creation of the Node Projector Tool ............... 9

2.2.4 Creation of Hexahedral Layers ..................... 15

2.2.5 Locating and Updating the Next Sweep Block ...... 15

2.3 Comparisons .................................................. 17

2.3.1 Timing Results ....................................... 17

2.3.2 Element Quality Results ............................. 19

2.4 Examples ...................................................... 22

2.5 Conclusions .................................................. 24

3 ADAPTIVE SWEEPING TECHNIQUES 27

3.1 Introduction .................................................. 27
3.1.1 One-to-One Sweeping ............................................ 27
3.1.2 Many-to-One or Many-to-Many Sweeping .................. 29
3.1.3 Interior Node Placement ................................. 29
3.1.4 Adaptive Sweeping ........................................... 32
3.2 Analysis of Existing Methods ................................. 33
  3.2.1 LinearAffine .............................................. 33
  3.2.2 Faceted ..................................................... 34
  3.2.3 BoundaryError ........................................... 35
  3.2.4 Smoothing ............................................... 36
  3.2.5 Auto ....................................................... 36
  3.2.6 Timing Results ........................................... 36
  3.2.7 Element Quality Results ................................. 37
3.3 Development of Adaptive Node Projection Schemes ........ 39
  3.3.1 Selection Criteria for BoundaryError .................. 39
  3.3.2 The SmartAffine Method ................................ 40
3.4 Development of the AdaptiveSweeper ....................... 44
  3.4.1 Creation of the Target Mesh ......................... 44
  3.4.2 Characterization of Linking Curvature ............. 45
  3.4.3 Selection of Projection Scheme .................... 46
3.5 Results ......................................................... 46
  3.5.1 Test Case 1 ............................................. 47
  3.5.2 Test Case 2 ............................................. 48
  3.5.3 Test Case 3 ............................................. 49
  3.5.4 Test Case 4 ............................................. 50
  3.5.5 Test Case 5 ............................................. 53
3.6 Conclusion .................................................... 53

4 CONCLUSION .......................... 57

Bibliography ............................................. 61
List of Figures

2.1 Characterization of many-to-one geometry .......................... 5
2.2 Assignment of layer numbers ..................................... 8
2.3 Initial block configurations ...................................... 9
2.4 Source surface after triangulation ................................. 11
2.5 Projection of nodes to nearest facet ............................... 12
2.6 Updating the next sweep block .................................. 16
2.7 Overall timing results ............................................. 18
2.8 Overall average quality results .................................. 20
2.9 Overall minimum quality results ................................. 21
2.10 Geometry requiring automatic block decomposition ............. 23
2.11 Geometry with curved source and target surfaces ............... 24
2.12 Geometry with changes in linking curvature ..................... 25
3.1 Steps for sweeping a one-to-one geometry ....................... 28
3.2 Many-to-one geometry ............................................. 30
3.3 Example of ineffective capture of source and target curvature .. 31
3.4 Geometry with changes in linking curvature ..................... 32
3.5 Source-target curvature capture ability .......................... 41
3.6 Differences in mesh quality between LinearAffine and SmartAffine .. 43
3.7 SmartAffine captures linking, source and target curvature .......... 44
3.8 Test Case 1: Variable curvature on all surfaces .................. 47
3.9 Test Case 2: Variable curvature on a linking surface ............ 48
3.10 Test Case 3: Variable curvature on source and target surfaces .. 49
3.11 Test Case 4: Many-to-one sweep ................................. 51
3.12 Sweep blocks of Test Case 4 ..................................... 52
3.13 Test Case 5: Many-to-one sweep ................................ 53
3.14 Sweep blocks of Test Case 5   .......................... 54
List of Tables

2.1 Average number of elements per run ................................. 17
2.2 Average time per 1000 elements created (sec/1000 elem) .......... 18
2.3 Ratio of other 3 schemes speed to the Faceted method ............ 19
2.4 Ratio of other 3 schemes speed to the Boundary method ........... 19
2.5 Average element quality results ........................................ 20
2.6 Minimum Element Quality Results ..................................... 22
2.7 Ratio of other 3 schemes minimum quality to the Faceted method .. 22
2.8 Ratio of other 3 schemes minimum quality to the Boundary method . 22
2.9 Comparisons between BoundaryError and Smoothing ............... 24
3.1 Overall average timing results (seconds/1000 elements created) .... 37
3.2 Average element quality results ........................................ 38
3.3 Minimum element quality results ..................................... 39
3.4 Threshold values of $F$ ................................................... 46
3.5 Results of projection methods on Test Case 1 ......................... 48
3.6 Results of projection methods on Test Case 2 ......................... 49
3.7 Results of projection methods on Test Case 3 ......................... 50
3.8 Results for Test Case 4 (Time in seconds) ............................ 51
3.9 Results for Test Case 5 (Time in seconds) ............................ 54
1 INTRODUCTION

The computational problems faced by the engineering and scientific community are becoming more complex. This has fostered a demand for the development of robust design and analysis tools. A report by the United States National Committee on Theoretical and Applied Mechanics summarized this need in the following way:

It is the fundamentally important part of computational science and engineering concerned with the use of computational approaches to characterize, predict, and simulate physical events and engineering systems governed by the laws of mechanics...It has transformed much of classical Newtonian theory into practical tools for prediction and understanding of complex systems. These are used in the simulation and design of current and future advances in technology throughout the developed and developing world. These have had a pervasive impact on manufacturing, communication, transportation, medicine, defense and many other areas central to modern civilization. By incorporating new models of physical and biological systems based upon quantum, molecular and biological mechanics, computational mechanics has an enormous potential for future growth and applicability[1].

Part of this effort is the development of robust algorithms for finite element hexahedral mesh generation. Hexahedral meshes provide advantages over tetrahedral meshes but are currently more restrictive in the types of geometry they can fill [2][3]. The work presented in this thesis deals with improving a specific class of all-hexahedral mesh generation algorithm known as sweeping. Because a general all-hexahedral mesh generation algorithm has not been discovered, algorithms that handle specific classes of geometry have been developed. Sweeping algorithms are one of these geometry specific algorithms. The work presented in this thesis has improved the speed and accuracy of the sweeping algorithm and minimized user interaction. These advances were accomplished by improving the interior node projection techniques used in the sweeping process.
Chapters 2 and 3 contain papers that were published. Chapter 2 was published in the International Journal of Numerical Methods in Engineering. The lead author was Michael A. Scott. Chapter 3 was published in the Proceedings of the 14th International Meshing Roundtable. The lead author of this paper was Michael A. Scott. Because each paper was published independently, there is some overlap in the material covered in each. This overlap was allowed to remain to preserve the integrity of each paper.

Chapter 2 introduces a new many-to-one sweep tool, the Polymorphic Many-to-One Sweep Tool or PMOST. Through its polymorphic design it is able to project nodes through “modular” node projection schemes. Two node projection schemes were implemented with PMOST: Faceted and BoundaryError node projectors. PMOST was compared with the current many-to-one sweep tool implemented in the CUBIT code base. The Faceted node projection scheme was the most efficient algorithm and the BoundaryError method produced the highest minimum quality elements while also performing efficiently. When speed and quality are both taken into consideration it is apparent that PMOST outperformed the current algorithm in CUBIT.

Chapter 3 introduces and describes adaptive sweeping techniques. This adaptivity is based on the geometric constraints of the model in question. The main contribution of this body of work is the effective automation of node projection schemes on a block-by-block basis. This allows for the optimization of both speed and quality in the generation of finite element meshes on many-to-one geometry. Through the careful characterization of existing projection techniques, two node projection schemes, each having unique strengths were implemented in the AdaptiveSweeper. Because both schemes capture source and target curvature, only linking curvature needs to be characterized. This is done through the use of empirically determined $F$ threshold values for the two node projection schemes. All these techniques enhance the automation, speed and quality of swept meshes.
2 POLYMORPHIC MANY-TO-ONE SWEEP TOOL

2.1 Introduction

With the use of Finite Element Analysis (FEA) growing across many disciplines, the need for accurate underlying discretizations or meshes has become increasingly important. Because of the inherent difficulty in generating an all-hexahedral mesh to general three-dimensional geometry, other methods of producing meshes on special classes of geometry have become popular [4, 5, 6, 7]. One of the most common methods is “sweeping” or “projecting.” Sweeping requires that the geometry in question be two-and-one-half dimensional (e.g. generalized cylinders). This requirement ensures that the geometry can be meshed with all-hexahedral finite elements [8].

2.1.1 One-to-One Sweeping

Sweeping of two-and-one-half dimensional geometry begins by identifying a “source” surface and any connected “linking” surfaces. The source surface is then usually meshed with quadrilaterals using an unstructured scheme such as paving [9]. Each linking surface must also be meshed with a mapped [10] or submapped mesh [11]. The source mesh is then swept or extruded one layer at a time along the mapped mesh on the linking surfaces toward the target mesh, which may or may not be meshed. This type of sweep is also called a “one-to-one” sweep because of the one-to-one correspondence between the source and target surfaces.

2.1.2 Many-to-One Sweeping

Because of sweeping’s strict requirements, few geometries satisfy the topological constraints required to generate a swept mesh. This problem has resulted in the
exploration of various methods that decompose more complex geometry into two-and-one-half dimensional sweep “blocks” or “barrels” which can then be swept \[4, 5, 6, 7\] To facilitate extension of the one-to-one sweep method to geometries with multiple source surfaces and a single target surface, an internal decomposition approach can be used. Although actual decomposition of the geometry does not occur, the method internally subdivides the geometry into individual one-to-one sweepable topologies that can be meshed with standard one-to-one procedures. The geometric entities that satisfy the requirements explained above are referred to as “many-to-one” geometry. Figure 2.1(a) shows a many-to-one geometry. Figure 2.1(b) shows the submapped linking (L) surfaces. Figure 2.1(c) shows the source (S) surfaces and the single target (T) surface. The Polymorphic Sweep Tool (PMOST) presented in this paper is capable of meshing many-to-one and one-to-one geometry. PMOST has been implemented in the CUBIT mesh generation toolkit code base \[12\]. The term polymorphic, from a computer science perspective, means to allow the same definitions to be used with different types of data (specifically, different classes of objects), resulting in more general and abstract implementations. This polymorphic characteristic allows the use of multiple node projection techniques with minimal user interaction. In the current implementation, a many-to-one sweepable geometry is automatically detected using the auto-sweep detection algorithm discussed in \[8\]. Necessary and sufficient conditions for sweepable geometries that are applicable for PMOST are outlined in this work.

### 2.1.3 Interior Node Placement

A vital component of the sweep process is the accurate projection or extrusion of interior points in the volume. Historically, this part of the process has been a weak link and it continues to contribute to occasional poor quality elements. In the Polymorphic Sweep Tool presented in this paper, two node projection schemes are implemented for the user to choose from: Faceted and BoundaryError. The Faceted node projection scheme is based on the two-and-one-half dimensional node projection technique, “BMSweep”, introduced by Staten et. al. \[13\]. The BoundaryError scheme
Figure 2.1: Characterization of many-to-one geometry
is based on the weighted residual method introduced by Blacker [4] and described in White [14]. These node projection schemes attempt to improve the placement of the interior nodes of the mesh to improve the overall quality of the resulting meshes. As previously published, the BMSweep method has been implemented only for one-to-one sweep cases. In addition, the weighted residual method, although demonstrated on many-to-many geometry, its implementation was not described. Using PMOST as a common framework, we propose extensions to both of these methods to make them applicable to many-to-one sweepable volumes and describe their implementation. Both node projection methods and their application to many-to-one volumes are presented in this paper.

2.1.4 Comparison with Existing Methods

To determine the overall effectiveness of PMOST and its node projection schemes, comparisons were made with the existing many-to-one sweeping algorithm currently functioning in CUBIT. This sweeping scheme has three node projection techniques available to the user. The first method, introduced by Knupp [7, 15, 16], uses a linear transformation and a traditional structured node smoothing scheme known as weighted Winslow smoothing to calculate final point placement. This smoothing method treats each layer of the sweep independently, optimizing the node locations of the quadrilateral vertices. It also attempts to maintain a reasonable copy or morph of the source surface mesh on each layer of the sweep by calculating a weight for each interior source node which is then applied to all matching interior nodes. The smoothing scheme is run on every layer of the sweep. This specific scheme will be referred to as the Smoothing scheme. The second method, also introduced by Knupp [7], determines on a layer-by-layer basis whether to use a linear transformation exclusively or a linear transformation followed by a weighted Winslow smoothing routine. This method will be referred to as the Auto routine. The third method uses only linear transformations to place nodes on each layer of the sweep. Because this method has proven to be inferior to the previous two in almost every case it will not be tested with the other four methods. Four of the five node projection schemes, with the exception
of purely linear transformations, with their accompanying many-to-one sweep tools were tested on a wide array of geometry. Overall speed and mesh quality results were recorded and analyzed.

2.2 Polymorphic Many-to-One Sweep Tool

The Polymorphic Sweep Tool’s engine performs 6 steps to complete the sweeping of a volume. They are:

1. Order the source surfaces in the order that they will be swept.

2. Create one-to-one sweep blocks for each source surface.

3. For each block, create the user-specified node projector tool. This tool will create all the layers of interior nodes in a sweep block.

4. With the layers of interior nodes created, form new layers of hexahedral element layers.

5. If the target surface has not been reached, update the next sweep block with the last layer of quadrilateral elements in the now completed current sweep block.

6. Iteratively repeat this process for all the sweep blocks until the target surface has been reached.

Each of these steps will be treated in more detail.

2.2.1 Ordering or Layering of Source Surfaces

The sweep layering tool recursively searches through the list of source surfaces that pertain to the current volume and assigns a layer number to each surface based on the mesh applied to the linking surfaces. When this process is completed, the source surface that lies farthest from the single target surface will be assigned the number zero. The distance from the target surface is determined using the mapped or unmapped linking surfaces. As can be seen in Figure 2.2, the spherical source surface is assigned the layer number zero. Layer seven is assigned this number because it is seven “layers” from layer zero along the linking surface’s unmapped mesh.
2.2.2 Creation of One-to-One Sweep Blocks

With the assignment of layer values to each of the source surfaces complete, an initial decomposition of the volume occurs by creating “sweep blocks.” A sweep block is a region between two consecutive source surfaces that will admit a one-to-one sweep. Sweep blocks are initialized before any sweeping begins with the list of the quadrilateral faces of the lowest numbered layer in the block. All other pertinent information that is required during the sweep can be calculated from this initial list of quadrilateral faces and the linking surface mesh. Specifically, the one-to-one node projection tools require the following information from each sweep block:

1. The volume that is currently being swept.

2. Boundary node loops as determined from the list of quadrilateral faces. There will always be one list of exterior boundary loops and possibly one or more lists of interior boundary node loops for each layer.
3. A list of all the quadrilaterals on the layer that constitute from where the sweep
will begin.

The initial block configuration for the geometry in Figure 2.2 is shown in
Figure 2.3.

2.2.3 Creation of the Node Projector Tool

Now that the full set of sweep blocks are created, an iterative process is com-
menced that will create nodes in the interior of each sweep block. For each sweep
block, starting at layer zero, a node projector tool is created via polymorphism.
For the implementation of the node projector tools in PMOST, polymorphic design
implemented in C++ was used. The following justification is given for the use of
polymorphic design. First, the portion of PMOST which is responsible for the de-
composition of the geometry into one-to-one sweepable blocks and the creation of
hexahedral elements, is distinct and separate from the actual projection of interior
nodes in the volume. The sweep engine performs the same functions regardless of
the type of projection scheme desired by the user. Second, it has been determined
through experience and experimentation, (see Section 2.3), that different node projection schemes perform well on certain types of geometry and not as well on others. This necessitates a type of “tool box” of projection schemes that give the user the option of selecting the node projection scheme most suited for the problem at hand. The development of these node projection “modules” should not require changes to the sweep engine. Third, the uncoupling of the node projection scheme from the sweep engine allows for the rapid development and implementation of new node projection schemes as technology advances. The similarities between the two implemented node projection schemes in PMOST will be discussed, followed by the characteristics unique to each one.

**Similarities in the Node Projectors**

Each individual node projector tool inherits from a generic node projector class which is responsible for all functionality common to both child classes. This parent class is responsible for creating a boundary node “ribbing” and transferring the source surface mesh to the target. A “rib” in this case, is essentially a linked list of nodes that originates at a source surface boundary node and continues to the matching target boundary node. In addition to this correspondence, each source boundary node points to its matching target node and vice-versa. With this ribbing in place the source surface boundary mesh can be transferred to the target surface. To transfer the nodes from the source surface mesh to the target surface, an initial triangulation of the source surface boundary nodes is computed. This is accomplished by projecting the source surface to a best-fit plane in parametric space and then performing a Delaunay triangulation, similar to the method described in [13], on the source surface boundary nodes.

Figure 2.4 shows the source surface associated with block 0 of the geometry in Figure 2.2. The triangulation has been completed on the boundary loop nodes and it has been projected back to real space. Next, the interior nodes on the source surface mesh need to be projected to the nearest facet. Once a node’s nearest facet is found,
that facet’s information is stored with the node for future use. Figure 2.5 shows how the source surface mesh nodes are projected to the nearest facet.

The location of the source surface node on its nearest facet is used to compute a barycentric location for that node on the facet. The offset distance from the nodes location on the source surface to its location on the nearest facet is also computed. Next, the source surface mesh topology must be transferred to the target surface. This is accomplished by creating a matching faceting for the target surface boundary nodes and then using the barycentric coordinates calculated for the source surface nodes to find Cartesian coordinates on the target facets. The nodes are then projected to the target surface. Note that these barycentric coordinates and offset distances are also saved with the corresponding target node for possible use later. Because there can be one or more sweep blocks, only if the target surface has been reached should a target mesh be created. If this is the case, a mesh is created on the target surface with the same connectivity as the source surface mesh. The final step is to then run a Winslow smoothing routine on the target surface mesh to assure optimal overall quality. At this point, the child node projector classes assume the responsibility of placing the interior nodes in the volume.
Faceted Node Projections

With the barycentric coordinates and offsets computed for both the source and target surface interior nodes, a linear interpolation of this information is performed to place interior nodes. Staten describes this procedure in [13], but it is also presented here for clarity. The placement of interior nodes occurs by creating a rib that originates at each source surface interior node and places nodes on each successive layer until it reaches the matching target node. The first step is to calculate the distance $x_i$ between each successive layer and to calculate $L_n$, the total distance from the source to the target.

$$L_n = \sum_{i=0}^{n} x_i$$ (2.1)

The next step is to create a linear interpolation $d_i$ of the target offset $d_t$, and the source offset $d_s$, for the current layer.

$$d_i = d_s \left(1 - \frac{x_i}{L_n}\right) + d_t \left(\frac{x_i}{L_n}\right)$$ (2.2)

Now, a linear interpolation of the source and target barycentric coordinates $A_s$ and $A_t$ is needed to create a barycentric coordinate $A_i$ for the current layer.
\[ A_i = A_s \left(1 - \frac{x_i}{L_n}\right) + A_i \left(\frac{x_i}{L_n}\right) \]  \hspace{1cm} (2.3)

Now that the interpolations are calculated, the general sweep direction \( \vec{V}_i \) is determined. This is calculated by determining vectors \( \vec{V}_1, \vec{V}_2, \) and \( \vec{V}_3 \) between the three points on the previous facet, the three points on the current facet, and the barycentric coordinates \( a_i, b_i, \) and \( c_i \) of the point on the current facet.

\[ \vec{V}_i = a_i \cdot \vec{V}_1 + b_i \cdot \vec{V}_2 + c_i \cdot \vec{V}_3 \]  \hspace{1cm} (2.4)

We are now ready to place an interior node on the current layer. Using the three points of the current facet \( P_{1i}, \ P_{2i}, \) and \( P_{3i}, \) we can calculate the new interior nodal location \( P_i. \)

\[ \vec{P}_i = (a_i \cdot P_{1i} + b_i \cdot P_{2i} + c_i \cdot P_{3i}) + d_i \cdot \vec{V}_i \]  \hspace{1cm} (2.5)

This process is repeated for each node on the current rib and for each rib in the block until all interior nodes have been calculated.

**BoundaryError Projections**

The BoundaryError method, introduced in [4] and described in [14], places nodes using an affine transformation algorithm and a subsequent least-squares residual error correction.

The affine transformation algorithm, developed by Knupp [7], uses a transformation matrix and the centroidal locations of the current and next boundary loops to place interior nodes. This type of transformation robustly handles the translation, rotation, and scaling of interior nodes in each sweep layer until the target is reached. A brief explanation of how the transformation is developed and used to place interior nodes follows. For a complete treatment of the subject see [7].

A \( 3 \times 3 \) non-singular linear transformation \( T \) is computed between current and next boundary loop nodes, \( x_k \) and \( \tilde{x}_k \) where \( k = 1, 2, \cdots, K \) with \( K \geq 3. \) This transformation \( T \) is used with the loop center points, \( c \) and \( \tilde{c}, \) to project interior nodes from the current layer to the next. Equation (2.6) shows how the transform is used
with the loop center points \( c \) and \( \tilde{c} \) computed in (2.7) and (2.8). Notice that \( x_k \) and \( \tilde{x}_k \) may be replaced with current and next layer interior points.

\[
\tilde{x}_k - \tilde{c} = T(x_k - c) \quad (2.6)
\]

\[
c = \frac{1}{K} \sum_{k=1}^{K} x_k \quad (2.7)
\]

\[
\tilde{c} = \frac{1}{K} \sum_{k=1}^{K} \tilde{x}_k \quad (2.8)
\]

The BoundaryError method, in addition to using the affine transformation \( T \) for initial node placement, calculates two residual errors, once sweeping from the source surface and terminating at the target surface, and then sweeping from the target and terminating at the source. These two error distances are then interpolated for final interior node placement. The critical step in this method is the calculation of the residual error, \( E \), defined by (2.9) which is applied to the current interior node being projected. Once the affine transformation is computed between the current and next layer it is used to project each of the nodes in the current boundary layer to where it believes it should be placed on the next layer. Because the actual location of the boundary node on the next layer is known, the difference, \( e \), between the actual location and the computed location can be computed. This process is repeated for each node in the current boundary list. The next step is to compute the corrected location of each interior node on the next layer. This is done by first projecting the interior node using the affine transformation calculated above. The error, \( e \), associated with each boundary node, and the distance, \( d \), from the current interior node to each boundary node is then used to calculate a least-squares weighted error, \( E \), which is added to the location of the current interior node. Note that \( n \) is the number of nodes in the boundary loop.

\[
\tilde{E} = \sum_{i=1}^{n} \frac{\tilde{c}_i}{\sum_{j=1}^{n} d_i} \quad (2.9)
\]
The final error, $E_{\text{final}}$, is the linear interpolation of the error, $\vec{E}_s$, from the source and the error, $\vec{E}_i$, from the target and is defined by (2.10). The value $n_{\text{layers}}$ is the total number of layers in the sweep and $i$ is the layer we are currently creating.

$$E_{\text{final}} = \vec{E}_s (1 - \frac{i}{n_{\text{layers}}}) + \vec{E}_i (\frac{i}{n_{\text{layers}}})$$  (2.10)

2.2.4 Creation of Hexahedral Layers

The next step in the sweep process is the creation of hexahedral mesh element layers. This is easily done using consecutive layers of boundary and interior nodes created in the current node projector tool.

2.2.5 Locating and Updating the Next Sweep Block

If the target surface has not been reached after the successful sweep of the current sweep block, the next sweep block must be isolated and updated so that it may be successfully swept. Because each sweep block knows at which layer of the sweep it should begin, finding the next sweep block is simply a matter of adding the current sweep block beginning layer number to the number of iterations or the number of layers created in order to completely sweep the current block. This number will represent the next sweep block to be swept and the block that needs to be updated. The next sweep block then needs to update itself using the last layer of quadrilateral faces created in the current sweep block. Figure 2.6 illustrates the updating of the next sweep block. As block one is swept, the last layer of quadrilateral faces needs to be merged with the top layer of quadrilateral faces on block 7. The boundary loops also need to be updated. Block 7 now has only one sweep boundary loop as opposed to the two that it started with (see Figure 2.3).

Updating Boundary Node Loops

Because the automatic and accurate updating of boundary node loops is vital to the success of PMOST, the algorithm for combining boundary loops to form the new boundary loops for the next sweep block will be described. This algorithm
extracts the required loop information from the topology of the quadrilaterals in the final layer of the previous block combined with the first layer of the next block. The first half of this algorithm consists of the creation, intersection and union of various sets of mesh elements (i.e. nodes, edges, faces). The first half of the algorithm is described in the following steps:

1. Get all the boundary loop edges $E_{bl}$, from the list of quadrilateral faces $Q_{all}$.

2. Get all the boundary nodes $N_b$, from the edges in $E_{bl}$.

3. Get all the quadrilateral faces on the boundary $Q_b$. This is determined by finding the quadrilaterals in $Q_{all}$ that have one of their four edges in $E_{bl}$.

4. Find all edges $E_{adj}$ shared between adjacent faces in $Q_b$.

With these sets of mesh elements now formed, the boundary node loops can be created. Each edge in $E_{adj}$ is used to “walk” around the boundary of the quadrilateral faces or surface mesh. Once the current edge is equal to the starting edge a loop has been completed and all those edges are deleted from $E_{adj}$. This process is repeated until $E_{adj}$ is empty and all boundary node loops have been formed.
Table 2.1: Average number of elements per run

<table>
<thead>
<tr>
<th>Run</th>
<th>Element Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run 1</td>
<td>6205</td>
</tr>
<tr>
<td>Run 2</td>
<td>18521</td>
</tr>
<tr>
<td>Run 3</td>
<td>77212</td>
</tr>
</tbody>
</table>

2.3 Comparisons

PMOST was compared with the many-to-one sweeper that already exists in CUBIT. Thirty models were chosen to test the capabilities of the two sweeping algorithms and the four node projection schemes. Special attention was taken to select models with curved source, target and linking surfaces. This was done in order to test the ability of each node projection scheme to capture geometric variation during the sweeping of each layer. General trends with respect to algorithm speed and hexahedral element quality were noted.

2.3.1 Timing Results

The two many-to-one sweeping algorithms and the four node projection schemes were tested on models of differing complexity and variation. Each of the four projection methods and accompanying many-to-one sweepers was run on the thirty models three different times. Each time the average number of hexahedral elements needed to mesh the volume was increased (see Table 2.1). The timing results were then recorded for all the tests and trends were identified and graphed.

Figure 2.7 and Table 2.2 present the results of the timing tests for the four different node projection methods. It can be seen that both the Faceted and Boundary-Error node projection schemes had the fastest times. These results can be attributed to the difference in computational intensity of the four algorithms. For example, the Faceted scheme only needs to be aware of the current interior node’s location with respect to its matching node on the source and target surface and the locations of the vertices of the current and next facet. The BoundaryError scheme, while less computationally expensive then smoothing, still needs to be aware of the location
Figure 2.7: Overall timing results

Table 2.2: Average time per 1000 elements created (sec/1000 elem)

<table>
<thead>
<tr>
<th>Run</th>
<th>Auto</th>
<th>Smooth</th>
<th>Boundary</th>
<th>Faceted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run 1</td>
<td>0.210</td>
<td>0.515</td>
<td>0.254</td>
<td>0.194</td>
</tr>
<tr>
<td>Run 2</td>
<td>0.259</td>
<td>0.568</td>
<td>0.229</td>
<td>0.123</td>
</tr>
<tr>
<td>Run 3</td>
<td>0.368</td>
<td>0.767</td>
<td>0.279</td>
<td>0.104</td>
</tr>
</tbody>
</table>

of all the boundary nodes on the current and next layer and use this information in interior node placement. It is interesting to note that the auto scheme, while fast when only linear transformations are needed, slowed down considerably because of the need at times to perform smoothing on the current layer of interior nodes on complex or highly curved geometries.

Tables 2.3 and 2.4 summarize the results with respect to the Faceted scheme (Table 2.3) and the BoundaryError schemes (Table 2.4) in particular. A simple ratio was calculated with the average speed of one of the other three methods in the numerator and the average speed of either the BoundaryError or Faceted schemes in the denominator. Any values less than one indicates an instance when Faceted or BoundaryError outperformed its counterpart. As the number of elements in each run increases the Faceted and BoundaryError schemes increase the speed gap between.
Table 2.3: Ratio of other 3 schemes speed to the Faceted method

<table>
<thead>
<tr>
<th>Run</th>
<th>then Auto</th>
<th>then Smooth</th>
<th>then Boundary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run 1</td>
<td>1.083</td>
<td>2.653</td>
<td>1.310</td>
</tr>
<tr>
<td>Run 2</td>
<td>2.095</td>
<td>4.606</td>
<td>1.852</td>
</tr>
<tr>
<td>Run 3</td>
<td>3.544</td>
<td>7.388</td>
<td>2.683</td>
</tr>
</tbody>
</table>

Table 2.4: Ratio of other 3 schemes speed to the Boundary method

<table>
<thead>
<tr>
<th>Run</th>
<th>then Auto</th>
<th>then Smooth</th>
<th>then Faceted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run 1</td>
<td>0.826</td>
<td>2.024</td>
<td>0.763</td>
</tr>
<tr>
<td>Run 2</td>
<td>1.131</td>
<td>2.487</td>
<td>0.540</td>
</tr>
<tr>
<td>Run 3</td>
<td>1.321</td>
<td>2.754</td>
<td>0.373</td>
</tr>
</tbody>
</table>

themselves and the auto and smoothing schemes. The Faceted scheme was nearly $7\frac{1}{2}$ times faster than smoothing on run 3.

2.3.2 Element Quality Results

Perfect element quality in a hexahedral mesh can only be obtained if all the elements in the mesh are perfect cubes. Because this is impossible to obtain in all cases, the element quality is instead maximized as much as possible so as to produce a mesh still suitable for finite element analysis. As was done in the timing tests, the minimum and average element quality [17] was recorded for all the tests during the three different runs. The number of elements produced was increased in each run.

Average Element Quality Results

The average element quality provides a good estimate of the overall quality of the mesh. It also provides a glimpse into the ability of the four node projection algorithms to produce high quality elements. Figure 2.8 and Table 2.5 show the results of the tests, and the average element quality each scheme produced.

The four methods produced similar results across the board. It is interesting to note that smoothing was the only method that continued to produce a higher quality mesh each time the number of elements created was increased. The element quality
Figure 2.8: Overall average quality results

Table 2.5: Average element quality results

<table>
<thead>
<tr>
<th>Run</th>
<th>Auto</th>
<th>Smooth</th>
<th>Boundary</th>
<th>Faceted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run 1</td>
<td>0.750</td>
<td>0.774</td>
<td>0.765</td>
<td>0.755</td>
</tr>
<tr>
<td>Run 2</td>
<td>0.763</td>
<td>0.780</td>
<td>0.781</td>
<td>0.770</td>
</tr>
<tr>
<td>Run 3</td>
<td>0.759</td>
<td>0.787</td>
<td>0.776</td>
<td>0.763</td>
</tr>
</tbody>
</table>
of the other three methods deteriorated slightly after the twenty thousand element mark. A portion of this deterioration in the Faceted and BoundaryError schemes can be attributed to the breakdown in the accuracy of the interpolation between the source and the target as the distance between interior nodes on neighboring layers converges. A small error in a fine mesh will produce distorted low quality elements. These errors affect the Faceted node projector more than the BoundaryError scheme because of the fewer boundary nodes used in the final interpolation and placement of interior nodes. The improvement of this interpolation is a subject for future research.

**Minimum Element Quality Results**

The minimum element quality results help identify where the “weak link” element is in the mesh. If the result is zero, then a negative Jacobian element was produced. The results for all the tests were analyzed and will be presented in this section. Figure 2.9 and Table 2.6 show the average results of the tests. The BoundaryError node projection scheme generated the highest minimum element quality. The Smoothing and Faceted schemes were within a small percentage of one another.

Tables 2.7 and 2.8 compare the BoundaryError and Faceted schemes to the other three methods. A simple ratio was calculated with the minimum quality of one
Table 2.6: Minimum Element Quality Results

<table>
<thead>
<tr>
<th>Run</th>
<th>Auto</th>
<th>Smooth</th>
<th>Boundary</th>
<th>Faceted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run 1</td>
<td>0.331</td>
<td>0.423</td>
<td>0.436</td>
<td>0.394</td>
</tr>
<tr>
<td>Run 2</td>
<td>0.299</td>
<td>0.397</td>
<td>0.425</td>
<td>0.387</td>
</tr>
<tr>
<td>Run 3</td>
<td>0.287</td>
<td>0.385</td>
<td>0.412</td>
<td>0.377</td>
</tr>
</tbody>
</table>

Table 2.7: Ratio of other 3 schemes minimum quality to the Faceted method

<table>
<thead>
<tr>
<th>Run</th>
<th>then Auto</th>
<th>then Smooth</th>
<th>then Boundary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run 1</td>
<td>0.841</td>
<td>1.075</td>
<td>1.108</td>
</tr>
<tr>
<td>Run 2</td>
<td>0.773</td>
<td>1.026</td>
<td>1.100</td>
</tr>
<tr>
<td>Run 3</td>
<td>0.761</td>
<td>1.023</td>
<td>1.092</td>
</tr>
</tbody>
</table>

of the other three methods in the numerator and the minimum quality of either the BoundaryError or Faceted schemes in the denominator. Any values less than one indicates an instance when Faceted or BoundaryError outperformed its counterpart.

The BoundaryError scheme outperformed the other three methods in general. The results for smoothing and Faceted did remain within ten percent of the BoundaryError method in every case.

2.4 Examples

Figure 2.10 shows a geometry that required a considerable amount of automatic block decomposition before it could be successfully swept. PMOST was able to identify each sweep block and set up the one-to-one correspondences. Each block was then swept using the Faceted projection scheme. The projections could also have

Table 2.8: Ratio of other 3 schemes minimum quality to the Boundary method

<table>
<thead>
<tr>
<th>Run</th>
<th>then Auto</th>
<th>then Smooth</th>
<th>then Faceted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run 1</td>
<td>0.760</td>
<td>0.970</td>
<td>0.903</td>
</tr>
<tr>
<td>Run 2</td>
<td>0.703</td>
<td>0.933</td>
<td>0.909</td>
</tr>
<tr>
<td>Run 3</td>
<td>0.697</td>
<td>0.936</td>
<td>0.915</td>
</tr>
</tbody>
</table>
been performed using the BoundaryError method. It should be noted that the automatic selection of the most appropriate projection scheme is an avenue of future research.

The geometry in Figure 2.11 appears simple. Unfortunately, the highly curved source and target surfaces can create difficulties when the interior points are placed. This is because the curvature on both the source and target surfaces needs to be interpolated successfully at each interior node. The existing sweep tool in CUBIT was unable to produce a high quality mesh. The auto scheme produced a negative Jacobian mesh and the smoothing scheme produced a minimum quality of 0.028. Both the BoundaryError and Faceted projection methods in PMOST produced a mesh with a minimum quality of 0.305 which is considerably higher.

As was demonstrated in Sections 2.3.1 and 2.3.2 the BoundaryError method outperformed the other node projection schemes in terms of minimum quality while approximating the speed of the Faceted algorithm. This is an important result because BoundaryError can now be used to replace the inefficient Smoothing method while still maintaining the quality of the resulting mesh. As meshes get larger, the difference in speed between Smoothing and BoundaryError will diverge dramatically.
Figure 2.11: Geometry with curved source and target surfaces

Table 2.9: Comparisons between BoundaryError and Smoothing

<table>
<thead>
<tr>
<th>Method</th>
<th>Time(sec)</th>
<th>Min. Element Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boundary</td>
<td>0.560</td>
<td>0.2924</td>
</tr>
<tr>
<td>Smoothing</td>
<td>12.64</td>
<td>0.3091</td>
</tr>
</tbody>
</table>

Figure 2.12 was meshed with BoundaryError and then with Smoothing. It should be noted that on a difficult mesh like that shown in Figure 2.12, the Auto method will perform a smoothing routine on each layer. Therefore, the results of Smoothing and Auto approximate one another as a mesh gets more difficult to produce. The results of the two methods are presented in Table 2.9. The minimum element quality resulting from Smoothing was slightly higher than that produced by BoundaryError. However, the dramatic difference in time required to produce the same mesh offsets the small quality benefits Smoothing might provide in this case. This trend is consistent on all meshes where a majority of the layers of the sweep are smoothed. Figure 2.12 was meshed with 8780 hexahedral elements.

2.5 Conclusions

This paper presents a new many-to-one sweep tool, the Polymorphic Many-to-One Sweep Tool or PMOST. Through its polymorphic design it is able to project
nodes through “modular” node projection schemes. Two node projection schemes were implemented with PMOST: Faceted and BoundaryError node projectors. The Faceted method is an extension of Staten’s “BMSweep” method [13] to many-to-one geometry and the BoundaryError method is a modification of Blacker’s weighted residual method as described in [14] to many-to-one geometry. PMOST was compared with the current many-to-one sweep tool implemented in the CUBIT code base. The Faceted node projection scheme was the most efficient algorithm and the BoundaryError method produced the highest minimum quality elements while also performing efficiently. When speed and quality are both taken into consideration it is apparent that PMOST outperformed the current algorithm in CUBIT.
3 ADAPTIVE SWEEPING TECHNIQUES

3.1 Introduction

With the use of Finite Element Analysis (FEA) growing across many disciplines, the need for accurate underlying discretizations or meshes has become increasingly important. Because of the inherent difficulty in generating an all-hexahedral mesh on general three-dimensional geometry, methods of producing meshes on special classes of geometry have become popular. One of the most common methods is “sweeping” or “projecting.” Sweeping requires that the geometry in question be two-and-one-half dimensional or decomposable into two-and-one-half dimensional sub-geometries (e.g. generalized cylinders). This requirement ensures that the geometry can be meshed with all-hexahedral finite elements [8].

3.1.1 One-to-One Sweeping

Sweeping of “one-to-one” geometry is visually represented in Figure 3.1. Sweeping of “one-to-one” geometry begins by identifying “source”, “target”, and connected “linking” surfaces. The source surface is then usually meshed (see Figure 3.1(a)) with quadrilaterals using an unstructured scheme such as paving [9]. Each linking surface must also be meshed with a mapped [10] or submapped [11] mesh (see Figure 3.1(b)). The surface mesh on the source is then swept or extruded one layer at a time along the mapped mesh on the linking surfaces toward the target mesh, which may or may not be meshed. This type of sweep is termed “one-to-one” because of the one-to-one correspondence between the source and target surface.
Figure 3.1: Steps for sweeping a one-to-one geometry
3.1.2 Many-to-One or Many-to-Many Sweeping

Because of sweeping’s strict requirements, few geometries satisfy the topological constraints required to generate a swept mesh. This problem has resulted in the exploration of various methods that decompose more complex geometry into two-and-one-half dimensional sweep “blocks” or “barrels” which can then be swept [4, 5, 6, 7, 18, 14].

Two such methods, many-to-many and many-to-one, use an internal decomposition approach to generate a series of individual one-to-one sweepable topologies which are then processed by a one-to-one sweep engine. A many-to-many geometry is characterized by multiple source and target surfaces. A many-to-one geometry is characterized by multiple source surfaces and a single target surface. It is important to note that actual decomposition of the geometry does not occur, only an internal characterization of sweep blocks. Sweepable geometries or geometries that may be decomposed into sweepable parts can be detected automatically with a fair amount of success [8].

The sweeper presented in this work operates on many-to-one and one-to-one geometry. Figure 3.2 is an example of a many-to-one geometry. The sweep axis originates at the left side of the geometry and arches to the top of the geometry. Notice the three source surfaces converging to a single target surface.

3.1.3 Interior Node Placement

A vital component of the sweep process is the accurate placement of interior points in the volume. Although a number of projection schemes have been proposed for sweeping [5, 7, 13, 15], the problem of efficiently and accurately placing nodes in all types of two-and-one-half dimensional volumes remains elusive. Each scheme has strengths and weaknesses that need to be understood by the user before a sweeping operation can be performed. Requiring the user to understand the details associated with each scheme is a major weakness in current sweeping algorithms. The problems associated with interior point placement schemes can be grouped into three general categories.
Ineffective Capture of Source and Target Curvature  If the curvature associated with the source and target surfaces is different, it is important that each interior point reflect this difference in curvature throughout the sweep. As the sweep is initiated at the source, the layers closer to the source should reflect the source surface curvature more than the curvature on the target. The opposite needs to occur as the sweep approaches the target. If a sweeping algorithm begins at the source and blindly moves toward the target, unconscious of target curvature, a mesh that poorly represents the curved geometry may be produced. Figure 3.3 shows an example where a node projection scheme did not account for the differences in source and target curvature. Resultant hexahedral elements on the last layer of the sweep (Figure 3.3(b)) are shown in this example. Notice that the elements reflect the curvature of the source but not the target.

Ineffective Capture of Changes in Linking Curvature  Perhaps an even more serious shortcoming in a node projection scheme is the inability to effectively
Figure 3.3: Example of ineffective capture of source and target curvature

capture changes in linking curvature. Figure 3.4 shows a geometry that has a drastic change in linking curvature along the sweep axis. The sweep axis originates at the right of the figure and moves toward the left. A node projection scheme that is unable to resolve these changes in linking curvature as the sweep progresses may project nodes outside the volume and produce degenerate elements.

**Efficiency Problems** The more advanced node projection schemes may suffer from performance problems due to the large number of calculations required to accurately place interior nodes. For example, a basic node projection scheme may be used in connection with a layer smoothing routine [16, 15]. As each layer of nodes is placed, smoothing is used to reorient the points into a more optimal configuration. These smoothing routines are characteristically slow and on large difficult meshes may render the sweeper useless due to the tremendous slow-down imposed by the smoother. Efficiency also becomes a concern when a geometry is composed of many one-to-one sweep blocks. As the geometry is decomposed and each block is swept, one
block may be successfully swept with an efficient basic node projection algorithm while another block may require a computationally intense algorithm because of different, more complex geometric constraints. If a node projection scheme must be selected by the user up front, then the worst case scenario will control and a less efficient, more advanced scheme will be selected. This scheme will mesh both the complicated block as well as the blocks that could be meshed with a faster projection algorithm, resulting in an overall time loss.

3.1.4 Adaptive Sweeping

The adaptive sweeping approach described in this paper addresses the three general shortcomings described above through the development and implementation of two node projection schemes, SmartAffine and BoundaryError. Both these schemes are used in a many-to-one context, or, in other words, a many-to-one sweep tool was built around them that decomposes a given geometry into one-to-one sweep blocks which are then passed to the most appropriate projection scheme. Each scheme is only used on one-to-one sweep blocks that have been carefully characterized and found well-suited to the relative strengths of that algorithm. In this way, the sweeping algorithm is able to adapt and apply the most appropriate projection scheme on a block-by-block basis. This process is formalized in this paper and called the AdaptiveSweeper.
3.2 Analysis of Existing Methods

A careful analysis of existing node projection schemes was conducted to understand the relative strengths and weaknesses of each and to develop appropriate schemes for the AdaptiveSweeper. The analysis centered on two main aspects of each algorithm. First, the speed of each algorithm was compared. Second, the quality of the resulting hexahedral elements was compared. Five projection schemes were implemented and analyzed. They are:

- LinearAffine
- Faceted
- BoundaryError
- Smoothing
- Auto

3.2.1 LinearAffine

The LinearAffine method, developed by Knupp [7], uses an affine transformation matrix and the centroidal locations of the current and next boundary loops to place interior nodes. This type of transformation robustly handles the translation, rotation, and scaling of interior nodes in each sweep layer until the target is reached. A brief explanation of how the transformation is developed and used to place interior nodes follows. For a complete treatment of the subject see [7].

A $3 \times 3$ non-singular linear transformation $T$ is computed between current and next boundary loop nodes, $x_k$ and $\tilde{x}_k$ where $k = 1, 2, \ldots, K$ with $K \geq 3$. This transformation $T$ is used with the loop center points, $c$ and $\tilde{c}$, to project interior nodes from the current layer to the next. Equation (3.1) shows how the transform is used with the loop center points $c$ and $\tilde{c}$ computed in (3.2) and (3.3). Notice that $x_k$ and $\tilde{x}_k$ may be replaced with current and next layer interior points.

$$\tilde{x}_k - \tilde{c} = T(x_k - c)$$ (3.1)
\[ c = \frac{1}{K} \sum_{k=1}^{K} x_k \]  

(3.2)

\[ \tilde{c} = \frac{1}{K} \sum_{k=1}^{K} \tilde{x}_k \]  

(3.3)

Because, in general, a single \( T \) may not exist between arbitrary loops, a least-squares fit to the bounding loop data is performed by minimizing the non-negative function (3.4) where \( u_k = x_k - c \) and \( \tilde{u}_k = \tilde{x}_k - \tilde{c} \)

\[ F(T) = \frac{1}{2} \sum_{k=1}^{K} |\tilde{u}_k - Tu_k|^2 \]  

(3.4)

Notice that if \( T \) is the identity matrix, then loop translation is achieved. Also, if \( F \) is not zero at the minimum, then \( T \) does not necessarily send all \( x_k \) to \( \tilde{x}_k \).

### 3.2.2 Faceted

The Faceted method is based on the one-to-one BMSweep method introduced by Staten et. al [13], but extended to be usable in a many-to-one setting [18]. The Faceted method assumes that a topologically similar mesh is on both the source and target surfaces. It then calculates a faceted mesh using the source surface boundary loops. That faceted representation is used in two ways. First, it is used to calculate the barycentric coordinates of each source interior node on its closest facet. Second, the distance or offset between the newly calculated barycentric coordinate and the source interior node’s actual location is determined. These two calculations are also performed for the topologically similar target mesh. The sweep begins at the source surface and the facets are transferred to the next layer of boundary nodes on the linking surfaces. The sweep direction is calculated using the vertices of the facet on the current layer and the vertices of the facet on the next layer. Using the sweep direction, the interpolated barycentric coordinate of the point being projected on the next facet and the interpolated offset information for the next layer, the node can be projected to the next layer. This process is repeated for each node on each layer until the sweep reaches the target surface, which is already meshed.
3.2.3 BoundaryError

The BoundaryError method, introduced in [4] and described in [14], places nodes using the LinearAffine algorithm described above and a subsequent least-squares residual error correction. The BoundaryError method, to successfully capture source and target curvature, calculates the residual error twice, once sweeping from the source surface and terminating at the target surface, and then sweeping from the target and terminating at the source. These two error distances are then interpolated for final interior node placement. The critical step in this method is the calculation of the residual error, $E$, defined by (3.5) which is applied to the current interior node being projected. Once the affine transformation is computed between the current and next layer it is used to project each of the nodes in the current boundary layer to where it believes it should be placed on the next layer. Because the actual location of the boundary node on the next layer is known, the difference, $e$, between the actual location and the computed location can be computed. This process is repeated for each node in the current boundary list. The next step is to compute the corrected location of each interior node on the next layer. This is done by first projecting the interior node using the affine transformation calculated above. The error, $e$, associated with each boundary node, and the distance, $d$, from the current interior node to each boundary node is then used to calculate a least-squares weighted error, $E$, which is added to the location of the current interior node. Note that $n$ is the number of nodes in the boundary loop.

$$
E = \sum_{i=1}^{n} \frac{e_i}{d_j}
$$

(3.5)

The final error, $E_{final}$, is the linear interpolation of the error, $E_s$, from the source and the error, $E_t$, from the target and is defined by (3.6). The value $n_{layers}$ is the total number of layers in the sweep and $i$ is the layer we are currently creating.

$$
E_{final} = E_s(1 - \frac{i}{n_{layers}}) + E_t(\frac{i}{n_{layers}})
$$

(3.6)
3.2.4 Smoothing

The Smoothing method introduced by Knupp [15], also uses the LinearAffine method for initial interior node placement. Once the interior points are placed using a simple transformation a traditional structured node smoothing scheme known as weighted Winslow smoothing is used to calculate final point placement. The weight functions associated with this smoother are calculated from the initial source mesh and strive to ensure a faithful copy of the source mesh on all subsequent layers until the target is reached. Specifically, without an attempt at weighting the initial source mesh, any initial biasing of the source mesh would be destroyed by the smoother. In this method the smoothing algorithm is run on every layer of the sweep.

3.2.5 Auto

The Auto sweeping method is a natural enhancement of the Smoothing method [15]. The Smoothing method performs a layer smoothing operation on each layer and is therefore inefficient in terms of algorithmic speed. Because the quality of the affine transformation is determined by how successfully (3.4) is minimized, a scaled $F$ factor can be computed on a layer by layer basis. This factor will specify the quality of the impending transformation, with zero being a perfect translation that will not require any layer smoothing. This factor may be used to, in effect, turn layer smoothing on and off, depending on the anticipated quality of the affine transformation.

3.2.6 Timing Results

Because the LinearAffine method is, by far, the least computationally complex of the five methods it was always the fastest algorithm on all the tests conducted. For this reason, it will not be included in the results that follow. It may be assumed that it always outperformed its counterparts in terms of speed. The other four methods were run on thirty test models three different times. Each time the average number of elements required to mesh the model was increased. The timing results were then recorded for all the tests and trends were identified. The results of the testing are show in Table 3.1.
Table 3.1: Overall average timing results (seconds/1000 elements created)

<table>
<thead>
<tr>
<th>Run</th>
<th>Auto</th>
<th>Smooth</th>
<th>Boundary</th>
<th>Faceted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run 1</td>
<td>0.210</td>
<td>0.515</td>
<td>0.254</td>
<td>0.194</td>
</tr>
<tr>
<td>Run 2</td>
<td>0.259</td>
<td>0.568</td>
<td>0.229</td>
<td>0.123</td>
</tr>
<tr>
<td>Run 3</td>
<td>0.368</td>
<td>0.767</td>
<td>0.279</td>
<td>0.104</td>
</tr>
</tbody>
</table>

These results can be attributed to the difference in computational intensity of the four algorithms. For example, the Faceted scheme only needs to be aware of the current interior node’s location with respect to its matching node on the source and target surface and the locations of the vertices of the current and next facet. The BoundaryError scheme, while less computationally expensive than Smoothing still needs to be aware of the location of all the boundary nodes on the current and next layer and use this information in interior node placement. It is interesting to note that the Auto scheme, while fast when only linear transformations are needed, slowed down considerably because of the need, at times, to perform smoothing on the current layer of interior nodes on complex or highly curved geometry.

3.2.7 Element Quality Results

Perfect element quality in a hexahedral mesh can only be obtained if all the elements are perfect cubes [17]. Because this is impossible to obtain in all cases, the element quality is instead maximized as much as possible to produce a mesh still suitable for finite element analysis. It should be noted that for all types of shape quality metrics, a perfect element will return a quality metric value of one and a degenerate element will return zero. As was done in the timing tests, the results from LinearAffine are not included because it will never outperform the other methods due to the fact it is used as a basic building block in the other four schemes. The minimum and average element quality was recorded for all the tests during three different runs. The number of elements was increased in each run.
Table 3.2: Average element quality results

<table>
<thead>
<tr>
<th>Run</th>
<th>Auto</th>
<th>Smooth</th>
<th>Boundary</th>
<th>Faceted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run 1</td>
<td>0.750</td>
<td>0.774</td>
<td>0.765</td>
<td>0.755</td>
</tr>
<tr>
<td>Run 2</td>
<td>0.763</td>
<td>0.780</td>
<td>0.781</td>
<td>0.770</td>
</tr>
<tr>
<td>Run 3</td>
<td>0.759</td>
<td>0.787</td>
<td>0.776</td>
<td>0.763</td>
</tr>
</tbody>
</table>

**Average Element Quality Results**

The average element quality provides a good estimate of the overall quality of a mesh. It also provides a glimpse into the ability of the four node projection algorithms to produce high quality elements. Table 3.2 shows the results of the tests.

The four methods produced similar results across the board. It is interesting to note that Smoothing was the only method that continued to produce a higher quality mesh each time the number of elements created was increased. The element quality of the other three methods deteriorated slightly. A portion of this deterioration in the Faceted and BoundaryError schemes can be attributed to the breakdown in the accuracy of the interpolation between the source and target as the distance between interior nodes on neighboring layers converges. A small error in a fine mesh will produce distorted low quality elements. These errors affect the Faceted scheme more than the BoundaryError method because fewer boundary nodes are used in the final interpolation and placement of interior nodes.

**Minimum Element Quality Results**

The minimum element quality results help identify where the “weak link” element is in the mesh. If the result is zero, then a negative Jacobian element was produced. The results for all the tests were analyzed and the results are presented in this section.

Table 3.3 shows the averaged minimum results for the tests. The Boundary-Error method generated the highest minimum element quality. The Smoothing and Faceted schemes were within a small percentage of one another.
Table 3.3: Minimum element quality results

<table>
<thead>
<tr>
<th>Run</th>
<th>Auto</th>
<th>Smooth</th>
<th>Boundary</th>
<th>Faceted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run 1</td>
<td>0.331</td>
<td>0.423</td>
<td>0.436</td>
<td>0.394</td>
</tr>
<tr>
<td>Run 2</td>
<td>0.299</td>
<td>0.397</td>
<td>0.425</td>
<td>0.387</td>
</tr>
<tr>
<td>Run 3</td>
<td>0.287</td>
<td>0.385</td>
<td>0.412</td>
<td>0.377</td>
</tr>
</tbody>
</table>

The BoundaryError method outperformed the other three methods in general. The results for Smoothing and Faceted were within ten percent of the BoundaryError method in every case.

3.3 Development of Adaptive Node Projection Schemes

The results obtained in Section 3.2 were used to select and develop the node projection schemes to be implemented in the AdaptiveSweeper. Two schemes were implemented in the sweeper. BoundaryError was selected from the list of existing methods. The second scheme, SmartAffine, is a modification of the LinearAffine method and is presented in Section 3.3.2. The ideas governing the selection and development process are presented in this section.

3.3.1 Selection Criteria for BoundaryError

A surprising result of the analysis performed on the node projection methods was the ability of the BoundaryError method to equal or outperform the Smoothing method in terms of the minimum quality of elements generated. This is an important result because it allows for the use of BoundaryError on difficult sweeps instead of much slower smoothing routines. Table 3.1 shows that, as far as speed is concerned, BoundaryError is, on average, twice as fast as the Smoothing method describe in 3.2.4. BoundaryError is also able to capture changes in source and target curvature as opposed to Smoothing which is not. Figure 3.5 shows two cross-sections of a mesh swept with Smoothing and BoundaryError. Notice the difference in the shape of the hexahedral elements on the layers closest to the source and target. Smoothing produces long rectangular elements of low quality while BoundaryError produces
high quality elements throughout the mesh. The Faceted method was not chosen over BoundaryError because it did not produce as high quality elements as BoundaryError even though it does have an advantage in terms of speed. Resulting minimum element quality was the controlling factor in the selection of BoundaryError. In the Sweeping algorithm presented in this paper, BoundaryError is responsible for producing high quality meshes on difficult geometry. The SmartAffine method, which is presented in the next section (3.3.2, is responsible for producing large meshes quickly on less difficult geometry.

3.3.2 The SmartAffine Method

In this section, the SmartAffine method is introduced. The SmartAffine method closely approximates the speed of LinearAffine and is able to robustly sweep much more general problems than the simple translations, rotations and scalings of the source surface that the LinearAffine method is designed to handle. SmartAffine uses simple affine transformations and their inverses to capture source and target curvature. It is also able to, in a limited sense, handle problems that have linking curvature through the interpolation and effective “spreading out” of sweeping errors. Because both SmartAffine and BoundaryError robustly handle changes in source and target curvature it eliminates the need to explicitly make this check during the adaptive sweeping process.

Implementation Details for SmartAffine

The SmartAffine method assumes that the source and target are both meshed with topologically similar meshes. It then performs LinearAffine transformations beginning at the source and target simultaneously. These sweeps will each produce a layer of nodes at the central layer of the volume. The positions of matching nodes on the two layers can then be analyzed to determine the relative error between the source and target sweeps. Let \( \hat{p}_s \) and \( \hat{p}_t \), be the positions of matching nodes at the central layer of the sweep. The point \( \hat{p}_s \) is the point from the sweep that originated at the source and \( \hat{p}_t \) is the point from the sweep that originated at the target. We
(a) Geometry with source-target curvature

(b) Cross-section of Smoothing swept mesh

(c) Cross-section of BoundaryError swept mesh

Figure 3.5: Source-target curvature capture ability
can now calculate two interpolating factors, \( n_s = \frac{n_{\text{total}}}{2} \) and \( n_t = n_{\text{total}} - n_s \), where \( n_{\text{total}} \) is the total number of projections needed to sweep the geometry. It will always be one less than the number of layers in the sweep. This step needs to be performed carefully because \( n_{\text{total}} \) may be odd.

The error terms to be applied to each half of the total sweep, \( \tilde{e}_s \) and \( \tilde{e}_t \), are calculated in (3.7) and (3.8). There will be one such error term associated with each node in both the central source and target layers.

\[
\tilde{e}_s = (\tilde{p}_t - \tilde{p}_s) \frac{n_s}{n_{\text{total}}} \quad (3.7)
\]

\[
\tilde{e}_t = (\tilde{p}_s - \tilde{p}_t) \frac{n_t}{n_{\text{total}}} \quad (3.8)
\]

These errors are now applied to each layer of their respective sweeps starting at the mid-layer and working backward to either the source or target surface. Because the error terms were calculated at the central layer they need to be scaled appropriately on each layer of the sweep. This can be accomplished by inverting the affine transformation matrix used to initially place the nodes and interpolating the error term as each layer is processed. The adjusted error terms are presented in (3.9) and (3.10), where \( i \) represents the current layer being adjusted.

\[
e_{s, \text{adj}} = T^{-1}(\tilde{e}_s \frac{i}{n_s}) \quad (3.9)
\]

\[
e_{t, \text{adj}} = T^{-1}(\tilde{e}_t \frac{i}{n_t}) \quad (3.10)
\]

**SmartAffine Results**

The SmartAffine method allows for the sweeping of one-to-one geometry that does not meet the simple translation, rotation or scaling requirements of LinearAffine. It also creates meshes nearly as fast as LinearAffine due to the fact that it only proceeds half-way through the geometry before reversing and heading back to where it started. Notice that SmartAffine and LinearAffine are of the same order in terms of algorithmic complexity. Most of the additional computational complexity in
SmartAffine can be attributed to inverting the $3 \times 3$ affine transformation matrices. Figure 3.6 shows an example of a geometry with changes in linking curvature. In Figure 3.6(a) LinearAffine has projected nodes outside of the geometry and produced a degenerate mesh. In Figure 3.6(b) SmartAffine has produced a mesh with a minimum shape metric of 0.3813, which is very good considering the highly variable curvature in the linking surfaces.

Figure 3.7 shows a mesh with changes in curvature in the linking, source and target surfaces. SmartAffine produced a good quality mesh with a minimum shape metric of 0.4144 and LinearAffine produced a degenerate mesh.

SmartAffine and LinearAffine are comparable in terms of speed, with SmartAffine being, on average, within fifteen to twenty five percent of LinearAffine. For example, a test was run on a model with one million elements and the time spent projecting nodes in LinearAffine was 3.21 seconds while it took SmartAffine 5.51 seconds to perform projections on the same model. This demonstrates that, even on very large models, there is not a sharp divergence in speed between the two.

Figure 3.6: Differences in mesh quality between LinearAffine and SmartAffine
Figure 3.7: SmartAffine captures linking, source and target curvature

3.4 Development of the AdaptiveSweeper

Now that both projection methods are developed and implemented in a many-to-one setting, the next step is the development of routines that automate the task of selecting them so that the strengths of each are captured. To accomplish this, three steps must be followed. First, a close copy of the source mesh must be placed on the target surface. Second, the linking surfaces must be characterized to determine how drastic the changes in linking curvature are from one layer of the sweep to the next. Third, using the information generated in steps one and two, the projection scheme best suited to the geometry is selected. This automated selection of projection schemes allows the sweeper to “adapt” to each sweep block as a many-to-one or one-to-one geometry is processed.

3.4.1 Creation of the Target Mesh

A high quality copy of the source mesh to the target surface is vital to the overall effectiveness of the projection schemes. If large discrepancies exist between the source and target surface meshes then these errors will be propagated throughout
the mesh as the sweeper places interior points. The method used to create the target mesh is based on the mesh morph/copy work of Knupp [15]. An affine transformation is computed between the source and target boundary loops. The source interior nodes are then projected as near to the target surface as possible. Because the nodes most likely will not be on the target surface or even near it if the target surface is highly curved, a simple vector calculation is used to iteratively move the nodes in the sweep direction until the target surface is reached. This will ensure that the nodes are in a near optimal position before smoothing is invoked. With the nodes on the surface, they are then smoothed using a weighted Winslow smoothing routine (see 3.2.4) which preserves biasing and attempts to produce a near copy of the source surface. It is important to note that the iterative vector moving procedure will greatly reduce the number of iterations required to smooth the target mesh and ensure a more faithful copy of the source mesh. On a highly curved target surface, without the iterative vector moving procedure, the smoother would begin with a highly distorted representation of the source mesh and may be unable to recover and produce a close match to the source surface mesh.

3.4.2 Characterization of Linking Curvature

A more difficult operation is the characterization of changes in linking curvature. To do this (3.4) is used to measure the quality of the transformation from the current layer to the next layer [7]. As was stated, if $F$ is zero simple loop translation exists, and the larger the value of $F$ the less accurate the affine transformation will be. A layer-by-layer approach was not used in the AdaptiveSweeper because of the efficiency of all the projection schemes. Instead, all the layers were evaluated according to the criteria just described and the worst case transformation was used in projection scheme selection. An additional check that is made is how large the $F_{st}$ (source-to-target) value is when calculated from the source boundary layer to the target boundary layer. It is possible, especially on a very fine mesh, that the layer-by-layer $F$ values remains small while the overall change in shape from the source to
Table 3.4: Threshold values of $F$

<table>
<thead>
<tr>
<th>$F$ Value</th>
<th>Projection Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 \leq F &lt; 0.03$ and $F_{st} \leq 10$</td>
<td>SmartAffine</td>
</tr>
<tr>
<td>$0.03 \leq F &lt; \infty$ or $F_{st} \geq 10$</td>
<td>BoundaryError</td>
</tr>
</tbody>
</table>

the target surface is large. Both checks need to be made in order to fully capture what is occurring on the linking surfaces throughout the sweep.

3.4.3 Selection of Projection Scheme

Using the results from 3.4.2, an appropriate projection scheme can be selected for the current sweep block. Conservative threshold values of $F$ were determined experimentally and the results are presented in Table 3.4.

The values in Table 3.4, coupled with the source and target curvature capture capabilities of each projection method fully define a way to select both node projection schemes in the AdaptiveSweeper on a block-by-block basis. BoundaryError will be used if the $F$ or $F_{st}$ threshold values of the sweep block are exceeded. SmartAffine will be used if neither the $F$ or $F_{st}$ values are exceeded. Notice that, through this method, a complicated many-to-one geometry with multiple blocks can be meshed efficiently without user intervention on a block-by-block basis.

3.5 Results

The following five examples demonstrate the capabilities of the AdaptiveSweeper. One-to-one and many-to-one geometries were selected that demonstrate each algorithms capability to robustly handle the three types of general problems most common in sweeping (see Section 3.1.3). Three one-to-one and two many-to-one geometries were selected. Notice that the Faceted method was not included in the tests that follow. It has been demonstrated that, in almost all cases, Faceted produces lower quality elements than BoundaryError (see Table 2.6). Smoothing was included because traditionally, smoothing schemes coupled with linear projections have been the
default method for sweeping difficult geometry. The comparisons that follow between Smoothing and BoundaryError demonstrate why BoundaryError may safely replace Smoothing as a projection method. LinearAffine is included in the tests to demonstrate the similarities in speed between LinearAffine and SmartAffine and SmartAffines superior ability to mesh general one-to-one geometry.

3.5.1 Test Case 1

Test Case 1 (Figure 3.8) is a geometry that exhibits changing curvature on source, target and linking surfaces. Because of the curvature constraints this is a difficult geometry to sweep while maintaining high quality.

Table 3.5 shows the quality and speed results for LinearAffine, SmartAffine, BoundaryError and Smoothing on Test Case 1. The results in Table 3.5 show that only BoundaryError was able to produce a valid mesh on Test Case 1. Smoothing attempted to produce a mesh but failed after 27.30 seconds. The methods in the AdaptiveSweeper for automatic scheme selection chose BoundaryError for Test Case 1 because of a worst-case $F$ value of 0.113. The results in Table 3.5 verify the correctness of that choice. 8,624 hexahedral elements were generated in this example.
Table 3.5: Results of projection methods on Test Case 1

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Min</th>
<th>Ave</th>
<th>Speed (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>0.0000</td>
<td>0.8034</td>
<td>2.16</td>
</tr>
<tr>
<td>Smart</td>
<td>0.0000</td>
<td>0.8497</td>
<td>2.16</td>
</tr>
<tr>
<td>Boundary</td>
<td>0.6074</td>
<td>0.8702</td>
<td>2.38</td>
</tr>
<tr>
<td>Smooth</td>
<td>0.0000</td>
<td>0.8538</td>
<td>27.30</td>
</tr>
</tbody>
</table>

Figure 3.9: Test Case 2: Variable curvature on a linking surface

3.5.2 Test Case 2

Test Case 2 (Figure 3.9) exhibits changing curvature on a linking surface (Note: The sweep axis is from the top to the bottom of the page). There are no changes in curvature on the source and target, but the target surface is not a scaled version of the source due to the linking curvature change.

Table 3.6 shows the results for the sweep of Test Case 2. The results demonstrate the ability of SmartAffine to mesh more general geometry than LinearAffine and to approximate the results of the more computationally intense BoundaryError and Smoothing routines. It should also be noted that SmartAffine approximates the results of LinearAffine in terms of speed, and produces a valid mesh while LinearAffine does not. Smoothing produced the highest quality mesh but required 22.36
Table 3.6: Results of projection methods on Test Case 2

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Min</th>
<th>Ave</th>
<th>Speed (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>0.0000</td>
<td>0.9071</td>
<td>0.28</td>
</tr>
<tr>
<td>Smart</td>
<td>0.3262</td>
<td>0.9089</td>
<td>0.31</td>
</tr>
<tr>
<td>Boundary</td>
<td>0.3261</td>
<td>0.9087</td>
<td>1.09</td>
</tr>
<tr>
<td>Smooth</td>
<td>0.3814</td>
<td>0.9168</td>
<td>22.36</td>
</tr>
</tbody>
</table>

Figure 3.10: Test Case 3: Variable curvature on source and target surfaces

seconds. The methods in the AdaptiveSweeper for automatic scheme selection chose BoundaryError for Test Case 2 because of a worst-case $F$ value of 0.121 and a $F_{st}$ value of 17.72. 17,525 hexahedral elements were produced for this geometry.

3.5.3 Test Case 3

Test Case 3 (Figure 3.10) has variable curvature on the source and target surfaces. Notice that the curvature is slight and the linking surfaces do not have variable curvature.

The weaknesses in LinearAffine and Smoothing are highlighted in this example. Even with only slight curvature on the source and target, the resultant quality of the
Table 3.7: Results of projection methods on Test Case 3

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Min</th>
<th>Ave</th>
<th>Speed (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>0.05178</td>
<td>0.9129</td>
<td>0.55</td>
</tr>
<tr>
<td>Smart</td>
<td>0.6404</td>
<td>0.9368</td>
<td>0.68</td>
</tr>
<tr>
<td>Boundary</td>
<td>0.6404</td>
<td>0.9368</td>
<td>3.52</td>
</tr>
<tr>
<td>Smooth</td>
<td>0.1017</td>
<td>0.9456</td>
<td>54.41</td>
</tr>
</tbody>
</table>

mesh in both cases is very poor. Also, Smoothing required 54.41 seconds to sweep the volume. This is much more than the other methods required. SmartAffine and BoundaryError produced the same mesh with SmartAffine completing in 0.68 seconds as opposed to 3.52 seconds for BoundaryError. As expected, the AdaptiveSweeper selected SmartAffine as the projection scheme of choice to take advantage of the time-savings it provides in this example. The worst-case $F$ value is 0.0 and the $F_{st}$ value is 0.0. 61,422 hexahedral elements were produced in this example.

### 3.5.4 Test Case 4

Beginning with Test Case 4, the results on many-to-one geometry are given. As can be seen in Figure 3.11, Test Case 4 is a many-to-one geometry with six sweep blocks. Figure 3.12 shows how the blocks will be ordered in the results that follow.

Table 3.8 shows the results for the tests run on Test Case 4. The time required to complete a sweep on each block as well as the overall time were recorded. The overall minimum and average quality are also reported. The automation schemes in the AdaptiveSweeper assigned BoundaryError to the first five blocks and SmartAffine to block six. In this way the more complicated sweeps one through five took advantage of BoundaryError’s residual error correction and block six, which is a simple sweep, benefited from the speed of SmartAffine. As can be seen in Table 3.8, if BoundaryError had been used on block six, it would have required 34.37 seconds to complete just that block. The AdaptiveSweeper successfully balanced speed and quality in this example. Test Case 4 was meshed with 202,688 elements.
Table 3.8: Results for Test Case 4 (Time in seconds)

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Block 1 Time</th>
<th>Block 2–5 Time</th>
<th>Block 6 Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>0.11</td>
<td>0.01</td>
<td>1.22</td>
</tr>
<tr>
<td>Smart</td>
<td>0.14</td>
<td>0.01</td>
<td>1.67</td>
</tr>
<tr>
<td>Boundary</td>
<td>0.63</td>
<td>0.015</td>
<td>33.68</td>
</tr>
<tr>
<td>Smooth</td>
<td>12.99</td>
<td>0.22</td>
<td>40.3</td>
</tr>
<tr>
<td>Adaptive</td>
<td>0.63</td>
<td>0.01</td>
<td>1.61</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Total Time</th>
<th>Min Qual</th>
<th>Ave Qual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>1.40</td>
<td>0.0000</td>
<td>0.9178</td>
</tr>
<tr>
<td>Smart</td>
<td>1.85</td>
<td>0.4613</td>
<td>0.9529</td>
</tr>
<tr>
<td>Boundary</td>
<td>34.37</td>
<td>0.4618</td>
<td>0.9529</td>
</tr>
<tr>
<td>Smooth</td>
<td>54.17</td>
<td>0.01381</td>
<td>0.9348</td>
</tr>
<tr>
<td>Adaptive</td>
<td>2.28</td>
<td>0.4618</td>
<td>0.9529</td>
</tr>
</tbody>
</table>
Figure 3.12: Sweep blocks of Test Case 4
3.5.5 Test Case 5

Test Case 5 (Figure 3.13) is the most difficult of the five geometries to sweep. It has highly curved surfaces that require careful placement of interior nodes. The sweep blocks for Test Case 5 are shown in Figure 3.14.

Table 3.9 shows the results for the tests run on Test Case 5. The time required to complete a sweep on each block as well as the overall time were recorded. The overall minimum and average quality were also recorded. The automation schemes in the AdaptiveSweeper assigned SmartAffine to the first block and BoundaryError to blocks two and three. Table 3.9 shows that the AdaptiveSweeper optimized the speed and quality of the mesh through the intelligent use of SmartAffine and BoundaryError. Test Case 5 was meshed with 80,282 hexahedral elements.

3.6 Conclusion

The main contribution of this work is the effective automation of node projection schemes on a block-by-block basis. This allows for the optimization of both
Figure 3.14: Sweep blocks of Test Case 5

Table 3.9: Results for Test Case 5 (Time in seconds)

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Block 1 Time</th>
<th>Block 2 Time</th>
<th>Block 3 Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>0.06</td>
<td>0.13</td>
<td>0.39</td>
</tr>
<tr>
<td>Smart</td>
<td>0.07</td>
<td>0.15</td>
<td>0.54</td>
</tr>
<tr>
<td>Boundary</td>
<td>0.16</td>
<td>0.40</td>
<td>2.96</td>
</tr>
<tr>
<td>Smooth</td>
<td>0.83</td>
<td>3.23</td>
<td>65.27</td>
</tr>
<tr>
<td>Adaptive</td>
<td>0.08</td>
<td>0.38</td>
<td>2.88</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Total Time</th>
<th>Min Qual</th>
<th>Ave Qual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>0.58</td>
<td>0.0000</td>
<td>0.7137</td>
</tr>
<tr>
<td>Smart</td>
<td>0.76</td>
<td>0.0000</td>
<td>0.7207</td>
</tr>
<tr>
<td>Boundary</td>
<td>3.52</td>
<td>0.1824</td>
<td>0.7352</td>
</tr>
<tr>
<td>Smooth</td>
<td>69.97</td>
<td>0.0000</td>
<td>0.7289</td>
</tr>
<tr>
<td>Adaptive</td>
<td>3.34</td>
<td>0.1814</td>
<td>0.7346</td>
</tr>
</tbody>
</table>
speed and quality in the generation of finite element meshes on many-to-one geometry. Through the careful characterization of existing projection techniques, two node projection schemes, each having unique strengths were implemented in the AdaptiveSweeper. BoundaryError effectively replaces more computationally intense smoothing routines, while maintaining the quality of the mesh and providing a speed advantage. It is also capable of capturing source and target curvature. SmartAffine, introduced in this work, approximates the speed of LinearAffine while robustly capturing source, target and, to a limited extent, linking curvature. Because both schemes capture source and target curvature, only linking curvature needs to be characterized. This is done through the use of empirically determined $F$ threshold values for the two node projection schemes. All these techniques enhance the automation, speed and quality of swept meshes.

The AdaptiveSweeper can be enhanced through future work on the characterization of linking surface curvature. The $F$ values presented in this work are empirical in nature and can be enhanced if a more theoretical framework can be discovered and implemented. Also, the SmartAffine method can be enhanced by extending its capabilities to geometries with more extreme variations in linking surface curvature. This might be done through internal boundary-error detection similar to those present in BoundaryError. It would have to be done with care so as to not destroy the efficiency of the algorithm, which is one of its strengths.

The problem of meshing general three-dimensional geometry with hexahedral elements remains an elusive goal. The sweeping method has proven to be a robust method for placing these meshes on two-and-one-half dimensional geometries. It is however, unable to handle all cases, and future work needs to remain focused on improving the robustness of existing sweeping methods. The AdaptiveSweeper attempts to build on this existing technology and expand the generality of the sweeping method.
4 CONCLUSION

This thesis presents recent work in the area of hexahedral finite element mesh generation. Specifically, work was done in the area of sweeping algorithms.

First, a new many-to-one sweep tool, the Polymorphic Many-to-One Sweep Tool or PMOST is presented. Through its polymorphic design it is able to project nodes through “modular” node projection schemes. Two node projection schemes were implemented with PMOST: Faceted and BoundaryError node projectors. The Faceted method is an extension of Staten’s “BMSweep” method [13] to many-to-one geometry and the BoundaryError method is a modification of Blacker’s weighted residual method as described in [14] to many-to-one geometry. PMOST was compared with the current many-to-one sweep tool implemented in the CUBIT code base. The Faceted node projection scheme was the most efficient algorithm and the BoundaryError method produced the highest minimum quality elements while also performing efficiently. When speed and quality are both taken into consideration it is apparent that PMOST outperformed the current algorithm in CUBIT.

Second, a new approach to adaptive sweeping techniques is presented. This adaptivity is based on the geometric constraints of the model in question. The main contribution of this body of work is the effective automation of node projection schemes on a block-by-block basis. This allows for the optimization of both speed and quality in the generation of finite element meshes on many-to-one geometry. Through the careful characterization of existing projection techniques, two node projection schemes, each having unique strengths were implemented in the AdaptiveSweeper. BoundaryError effectively replaces more computationally intense smoothing routines, while maintaining the quality of the mesh and providing a speed advantage. It is also
capable of capturing source and target curvature. SmartAffine, introduced in this work, approximates the speed of LinearAffine while robustly capturing source, target and, to a limited extent, linking curvature. Because both schemes capture source and target curvature, only linking curvature needs to be characterized. This is done through the use of empirically determined $F$ threshold values for the two node projection schemes. All these techniques enhance the automation, speed and quality of swept meshes.
Bibliography


