Adaptive Sweeping Techniques

Michael A. Scott[1], Matthew N. Earp[2], Steven E. Benzley[3] and Michael B. Stephenson[4]

[1] Brigham Young University Provo, UT U.S.A. mas88@et.byu.edu
[2] Brigham Young University Provo, UT U.S.A. mne2@et.byu.edu
[3] Brigham Young University Provo, UT U.S.A. seb@byu.edu
[4] M. B. Stephenson & Associates, Provo, UT U.S.A. mbsteph@sandia.gov

Summary. This paper presents an adaptive approach to sweeping one-to-one and many-to-one geometry. The automatic decomposition of many-to-one geometry into one-to-one "blocks" and the selection of an appropriate node projection scheme are vital steps in the efficient generation of high-quality swept meshes. This paper identifies two node projection schemes which are used in tandem to robustly sweep each block of a one-to-one geometry. Methods are also presented for the characterization of one-to-one geometry and the automatic assignment of the most appropriate node projection scheme. These capabilities allow the sweeper to adapt to the requirements of the sweep block being processed. The identification of the two node projection schemes was made after an extensive analysis of existing schemes was completed. One of the node projection schemes implemented in this work, BoundaryError, was selected from traditional node placement algorithms. The second node projection scheme, SmartAffine, is an extension of simple affine transformations and is capable of efficiently sweeping geometry with source and/or target curvature while approximating the speed of a simple transform. These two schemes, when used in this adaptive setting, optimize mesh quality and the speed that swept meshes can be generated while minimizing required user interaction.

Key words: decomposition, hexahedra, mesh generation, node projection, smoothing, sweeping, automatic

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 [DTJ00].

1.1 One-to-One Sweeping

Sweeping of "one-to-one" geometry begins by identifying "source", "target", and connected "linking" surfaces. The source surface is then usually meshed with quadrilaterals using an unstructured scheme such as paving [T.91]. Each linking surface must also be meshed with a mapped [WW82] or submapped [DLSG95] mesh. 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.

1.2 Many-to-One or Many-to-Many Sweeping

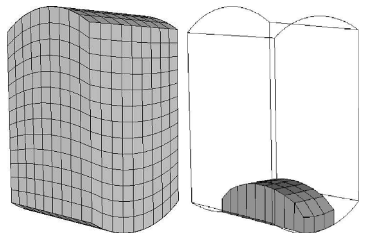
Because of sweeping's strict requirements, few geometries satisfy the topological constraints required to generate a swept mesh. This problem has resulted in the exploration of various methods that decompose more complex geometry into two-and-one-half dimensional sweep "blocks" or "barrels" which can then be swept [T.96, LSGT96, JSPD00, P.98a, MSS, DSS04].

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 [DTJ00].

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 [LSGT96, P.98a, MSS98, P.99], 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. 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 1 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 1(b)) are shown in this example. Notice that the elements reflect the curvature of the source but not the target.



(a) Geometry with source and target curvature

(b) Elements near target

Fig. 1. Example of ineffective capture of source and target curvature

Ineffective Capture of Changes in Linking Curvature Perhaps an even more serious shortcoming in a node projection scheme is the inability to effectively capture changes in linking curvature. Figure 2 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.

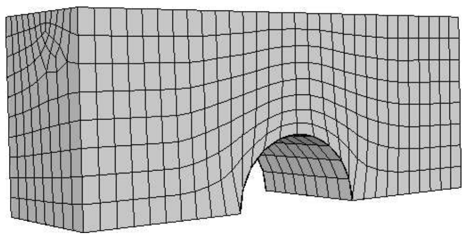


Fig. 2. Geometry with changes in linking curvature

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 [P.98b, P.99]. 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.

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.

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

2.1 LinearAffine

The LinearAffine method, developed by Knupp [P.98a], 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 [P.98a].

A 3×3 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 (1) shows how the transform is used with the loop center points c and $\tilde{c}$ computed in (2) and (3). Notice that x_k and $\tilde{x_k}$ may be replaced with current and next layer interior points.

$$\tilde{x_k} - \tilde{c} = T(x_k - c) \tag{1}$$

$$c = \frac{1}{K} \sum_{k=1}^{K} x_k \tag{2}$$

$$\tilde{c} = \frac{1}{K} \sum_{k=1}^{K} \tilde{x_k} \tag{3}$$

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 (4) where $u_k = x_k - c$ and $\tilde{u_k} = \tilde{x_k} - \tilde{c}$

$$F(T) = \frac{1}{2} \sum_{k=1}^{K} |\tilde{u_k} - T u_k|^2 \tag{4}$$

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 $\tilde{x_k}$.

2.2 Faceted

The Faceted method is based on the one-to-one BMSweep method introduced by Staten et. al [MSS98], but extended to be usable in a many-to-one setting [MSS]. 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.

2.3 BoundaryError

The BoundaryError method, introduced in [T.96] and described in [DSS04], 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 sweeeping 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 (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.

$$\mathbf{E} = \sum_{i=1}^{n} \frac{\mathbf{e}_i}{\sum_{j=1}^{n} \frac{d_i}{d_j}} \tag{5}$$

The final error, $\mathbf{E}_{final}$, is the linear interpolation of the error, $\mathbf{E}_s$, from the source and the error, $\mathbf{E}_t$, from the target and is defined by (6). The value n_{layers} is the total number of layers in the sweep and i is the layer we are currently creating.

$$\mathbf{E}_{final} = \mathbf{E}_s(1 - \frac{i}{n_{layers}}) + \mathbf{E}_t(\frac{i}{n_{layers}}) \tag{6}$$

2.4 Smoothing

The Smoothing method introduced by Knupp [P.99], 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.

2.5 Auto

The Auto sweeping method is a natural enhancement of the Smoothing method [P.99]. 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 (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.

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 1.

Table 1. Overall average timing results (seconds/1000 elements created)

Run	Auto	Smooth	Boundary	Faceted
Run 1	0.210	0.515	0.254	0.194
Run 2	0.259	0.568	0.229	0.123
Run 3	0.368	0.767	0.279	0.104

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.

2.7 Element Quality Results

Perfect element quality in a hexahedral mesh can only be obtained if all the elements are perfect cubes [P.03]. 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 element quality was recorded for all the tests during three different runs. The number of elements was increased in each run.

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 2 shows the averaged minimum results for the tests. The BoundaryError method generated the highest minimum element quality. The Smoothing and Faceted schemes were within a small percentage of one another.

Table 2. Minimum element quality results

Run	Auto	Smooth	Boundary	Faceted
Run 1	0.331	0.423	0.436	0.394
Run 2	0.299	0.397	0.425	0.387
Run 3	0.287	0.385	0.412	0.377

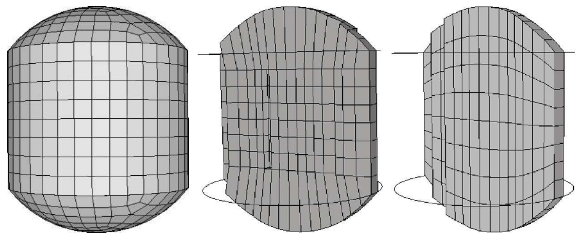
The BoundaryError method outperformed the other three methods in general. The results for Smoothing and Faceted were within ten percent of the BoundaryError method in every case.

3 Development of Adaptive Node Projection Schemes

The results obtained in Section 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 3.2. The ideas governing the selection and development process are presented in this section.

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 1 shows that, as far as speed is concerned, BoundaryError is, on average, twice as fast as the Smoothing method describe in 2.4. BoundaryError is also able to capture changes in source and target curvature as opposed to Smoothing which is not. Figure 3 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.



(a) Geometry with source-target curvature (b) Cross-section of Smoothing swept mesh (c) Cross-section of BoundaryError swept mesh

Fig. 3. Differences in quality of source-target curvature capture between Smoothing and BoundaryError

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, $\mathbf{p}_s$ and $\mathbf{p}_t$, be the positions of matching nodes at the central layer of the sweep. The point $\mathbf{p}_s$ is the point from the sweep that originated at the source and $\mathbf{p}_t$ is the point from the sweep that originated at the target. We can now calculate two interpolating factors, $n_s = \frac{n_{total}}{2}$ and $n_t = n_{total} - n_s$, where n_{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_{total} may be odd.

The error terms to be applied to each half of the total sweep, $\mathbf{e}_s$ and $\mathbf{e}_t$, are calculated in (7) and (8). There will be one such error term associated with each node in both the central source and target layers.

$$\mathbf{e}_s = (\mathbf{p}_t - \mathbf{p}_s)\frac{n_s}{n_{total}} \tag{7}$$

$$\mathbf{e}_t = (\mathbf{p}_s - \mathbf{p}_t)\frac{n_t}{n_{total}} \tag{8}$$

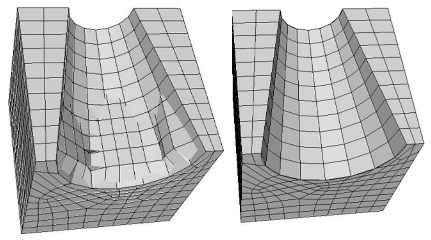
These errors are now applied to each layer of their respective sweeps starting at the mid-layer and working backward to either the source or target surface. Because the error terms were calculated at the central layer they need to be scaled appropriately on each layer of the sweep. This can be accomplished 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 (9) and (10), where i represents the current layer being adjusted.

$$\mathbf{e}_{s,adj} = T^{-1}(\mathbf{e}_s \frac{i}{n_s}) \tag{9}$$

$$\mathbf{e}_{t,adj} = T^{-1}(\mathbf{e}_t \frac{i}{n_t}) \tag{10}$$

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 3×3 affine transformation matrices. Figure 4 shows an example of a geometry with changes in linking curvature. In Figure 4(a) LinearAffine has projected nodes outside of the geometry and produced a degenerate mesh. In Figure 4(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.



(a) LinearAffine projects points outside of geometry

(b) SmartAffine produces a high quality mesh

Fig. 4. Differences in mesh quality between LinearAffine and SmartAffine

Figure 5 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.

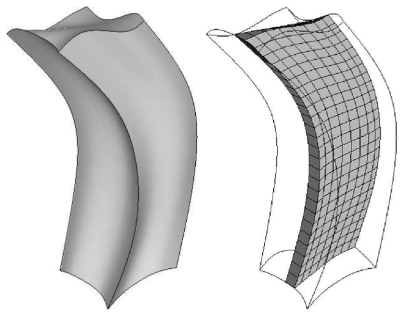

(a) Geometry with linking, source and target curvature (b) Curvature captured by SmartAffine

Fig. 5. SmartAffine captures linking, source and target curvature

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.

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.

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 [P.99]. 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 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.

4.2 Characterization of Linking Curvature

A more difficult operation is the characterization of changes in linking curvature. To do this (4) is used to measure the quality of the transformation from the current layer to the next layer [P.98a]. 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 is not used in the AdaptiveSweeper because of the efficiency of all the projection schemes. Instead, all the layers are evaluated according to the criteria just described and the worst case transformation is 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 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.

4.3 Selection of Projection Scheme

Using the results from 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.

Table 3. Threshold values of F

F Value	Projection Scheme
$0 \leqslant F < 0.03$ and $F_{st} \leqslant 10$	SmartAffine
$0.03 \leqslant F < \infty$ or $F_{st} \geqslant 10$	BoundaryError

The values in Table 3, 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.

5 Results

The following three 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 1.3). Two one-to-one and one 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). 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.

5.1 Test Case 1

Test Case 1 (Figure 6) 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.

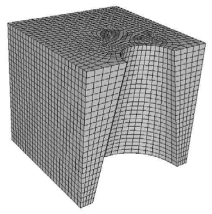


Fig. 6. Test Case 1: Variable curvature on a linking surface

Table 4 shows the results for the sweep of Test Case 1. 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 seconds. The methods in the AdaptiveSweeper for automatic scheme selection chose BoundaryError for Test Case 1 because of a worst-case F value of 0.121 and a F_{st} value of 17.72. $17,525$ hexahedral elements were produced for this geometry.

5.2 Test Case 2

Test Case 2 (Figure 7) has variable curvature on the source and target surfaces. Notice that the curvature is slight and the linking surfaces do not have variable curvature.

Table 4. Results of projection methods on Test Case 1

Scheme	Min	Ave	Speed(sec)
Linear	0.0000	0.9071	0.28
Smart	0.3262	0.9089	0.31
Boundary	0.3261	0.9087	1.09
Smooth	0.3814	0.9168	22.36

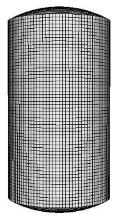


Fig. 7. Test Case 2: Variable curvature on source and target surfaces

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 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.

Table 5. Results of projection methods on Test Case 2

Scheme	Min	Ave	Speed(sec)
Linear	0.05178	0.9129	0.55
Smart	0.6404	0.9368	0.68
Boundary	0.6404	0.9368	3.52
Smooth	0.1017	0.9456	54.41

5.3 Test Case 3

Test Case 3 shows the results on a many-to-one geometry. As can be seen in Figure 8, Test Case 3 is a many-to-one geometry with six sweep blocks. Notice there are five blocks protruding from the top of the geometry. In the results that follow, the larger of the five protruding blocks will be called block one, the four smaller identical protrusions will be called blocks two through five and the large base will be block six.

Table 6 shows the results for the tests run on Test Case 3. The time required to complete a sweep on each block as well as the overall time were recorded. The overall

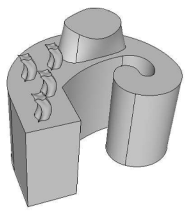


Fig. 8. Test Case 3: Many-to-one sweep

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 6, 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 3 was meshed with 202, 688 elements.

Table 6. Results for Test Case 3 (Time in seconds)

Scheme	Block 1 Time	Block 2–5 Time	Block 6 Time	Total Time	Min Qual	Ave Qual
Linear	0.11	0.01	1.22	1.40	0.0000	0.9178
Smart	0.14	0.01	1.67	1.85	0.4613	0.9529
Boundary	0.63	0.015	33.68	34.37	0.4618	0.9529
Smooth	12.99	0.22	40.3	54.17	0.01381	0.9348
Adaptive	0.63	0.01	1.61	2.28	0.4618	0.9529

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 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.

[DLSG95] White D., Mingwu L., Benzley S., and Sjaardema G. Automated Hexahedral Mesh Generation by Virtual Decomposition. In *Proceedings 4th International Meshing Roundtable*, pages 165–176. Sandia National Laboratories, 1995.

[DSS04] White D., Saigal S., and Owen S. CCSweep: Automatic Decomposition of Multi–Sweep Volumes. *Engineering with Computers*, 20:222–236, 2004.

[DTJ00] White D., Tautges T., and Timothy J. Automatic Scheme Selection for Toolkit Hex Meshing. *International Journal for Numerical Methods in Engineering*, 49:127–144, 2000.

[JSPD00] Shepherd J., Mitchell S., Knupp P., and White D. Methods for MultiSweep Automation. In *Proceedings 9th International Meshing Roundtable*, pages 77–87. Sandia National Laboratories, 2000.

[LSGT96] Mingwu L., Benzley S., Sjaardema G., and Tautges T. A Multiple Source and Target Sweeping Method for Generating All Hexahedral Finite Element Meshes. In *Proceedings 5th International Meshing Roundtable*, pages 217–225. Sandia National Laboratories, 1996.

[MSS] Scott M., Benzley S., and Owen S. Improved Many-to-One Sweeping. Submitted for publication.

[MSS98] Staten M., Canaan S., and Owen S. BMSweep: Locating Interior Nodes During Sweeping. In *Proceedings 7th International Meshing Roundtable*, pages 7–18. Sandia National Laboratories, 1998.

[P.98a] Knupp P. Next–Generation Sweep Tool: A Method for Generating All–Hex Meshes On Two–And–One–Half Dimensional Geometries. In *Proceedings 7th International Meshing Roundtable*, pages 505–513. Sandia National Laboratories, 1998.

[P.98b] Knupp P. Winslow Smoothing on Two–Dimensional Unstructured Meshes. In *Proceedings 7th International Meshing Roundtable*, pages 449–457. Sandia National Laboratories, 1998.

[P.99] Knupp P. Applications of Mesh Smoothing: Copy, Morph, and Sweep on Unstructured Quadrilateral Meshes. *International Journal for Numerical Methods in Engineering*, 45:37–45, 1999.

[P.03] Knupp P. Algebraic Mesh Quality Metrics for Unstructured Initial Meshes. *Finite Elements in Analysis and Design*, 39:217–241, 2003.

[T.91] Blacker T. A New Approach to Automated Quadrilateral Mesh Generation. *International Journal for Numerical Methods in Engineering*, 32:811–847, 1991.

[T.96] Blacker T. The Cooper Tool. In *Proceedings 5th International Meshing Roundtable*, pages 13–29. Sandia National Laboratories, 1996.

[WW82] Cook W. and Oakes W. Mapped Methods for Generating Three Dimensional Meshes. *Computers in Mechanical Engineering*, pages 67–72, 1982. CIME Research Supplement.