Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 00 (2015) 000–000

Procedia Engineering

www.elsevier.com/locate/procedia

24th International Meshing Roundtable (IMR24)

A Template-Based Approach for Parallel Hexahedral Two-Refinement

Steven J. Owen[a,*], Ryan M. Shih[b]

[a] *Sandia National Laboratory, Albuquerque, New Mexico, USA.*
[b] *University of California, Berkeley, California, USA.*

Abstract

In this work we provide a template-based approach for generating locally refined all-hex meshes. We focus specifically on refinement of initially structured grids utilizing a 2-refinement approach where uniformly refined hexes are subdivided into eight child elements. The refinement algorithm consists of identifying marked nodes that are used as the basis for a set of four simple refinement templates. The target application for 2-refinement is a parallel grid-based all-hex meshing tool for high performance computing in a distributed environment. The result is a parallel consistent locally refined mesh requiring minimal communication and where minimum mesh quality is greater than scaled Jacobian 0.4 prior to smoothing.

Peer-review under responsibility of organizing committee of the 24th International Meshing Roundtable (IMR24).

Keywords: hexahedron; mesh refinement; 2-refinement; adaptivity; parallel;

1. Introduction

Massively parallel platforms, such as those deployed at the U.S. Department of Energy Laboratories, have enabled computational simulation of enormous complexity. For applications requiring hexahedral elements, traditional methods of mesh generation can require significant user interaction which will not easily scale for these problems. Fully automatic, scalable and embedded meshing methods are an increasingly important requirement for these next-generation computing platforms. Mesh generation based on an overlay grid procedure is an ideal candidate for high performance computing, however to be effective it must provide for geometry-sensitive mesh size adaptation.

Overlay grid procedures for generating all-hex meshes [1][2][3] usually rely on some form of refinement strategy to capture small features. Most of these methods begin with a regular three-dimensional Cartesian grid that is adaptively refined based on various geometric criteria to form an octree subdivided mesh. A Boundary representation (B-rep) of the geometry of interest is then super-imposed on the octree mesh, where nodes are snapped to the geometry and elements falling outside of the B-Rep are discarded.

* Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000

E-mail address: sjowen@sandia.gov

Peer-review under responsibility of organizing committee of the 24th International Meshing Roundtable (IMR24).

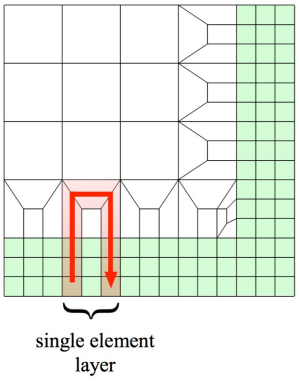


Fig. 1. Example 2D 3-refinement showing pillow loops accomplished within a single layer of elements.

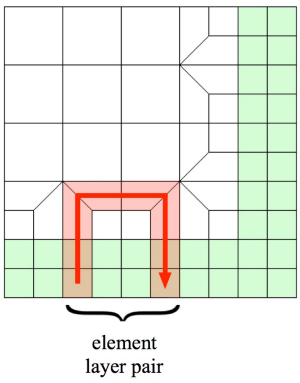


Fig. 2. Example 2D 2-refinement showing pillow loops requiring at least a pair of adjacent element layers.

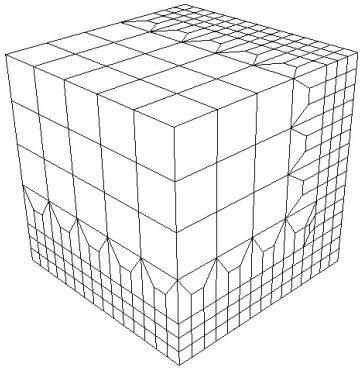


Fig. 3. Example 3D mesh using a 3-refinement procedure

In order to maintain continuity between refined and unrefined elements in the mesh, transition patterns are normally imposed. These patterns can utilize either a 3-refinement or a 2-refinement methodology. For 3-refinement, each edge of the uniformly refined grid is divided into three segments. On a three-dimensional hex element, this results in a 3x3x3 subdivision or 27 elements. 2-refinement, on the other hand, will split each edge in two resulting in a 2x2x2 subdivision with 8 elements.

Most refinement operations can be thought of as introducing a pillow layer of hexes surrounding a column of hexes. Figure 1 illustrates a 2D 3-refined mesh where the green elements were initially marked for uniform refinement. An example continuous pillow layer of quads (shown in red) is shown wrapping a single column of quads, noting that the same pattern is repeated throughout the mesh. Figure 2 illustrates a 2-refined mesh with a similar pillow layer surrounding two columns of quads. In 3-refinement, We note that the pillow layer can be accomplished within a single quad layer, whereas 2-refinement requires at least a pair of quad columns to accomplish the pillow. This problem extends to 3D (shown in figure 3) where sheets of hexes must be introduced to accomplish the refinement. For 3-refinement, since the pillowed sheet of hexes can be effected within a single column of hexes, each refinement transition can be performed independently and within a single element, making its implementation relatively straightforward. In contrast, 2-refinement must determine a consistent pairing of hex layers to effect the refinement transitions, making its implementation more challenging.

Although the 3-refinement strategy can produce far more elements resulting in high mesh size gradients in transition regions, it remains the most popular form of refinement because of its ease of implementation. The 3-refinement pattern, illustrated in figure 1, has well-defined templates for transition elements that can be introduced based on a set of marked nodes. First introduced by Schneiders [5] the number and pattern of marked nodes on a cell define the precise subdivision template to be used. This deterministic template-based approach to refinement, is relatively well-understood and easy to implement. Although beneficial for implementation, the change in element size and resulting mesh quality in the transition regions for 3-refinement can be problematic.

Although 2-refinement, shown in figure 2, by most measures is a more desirable approach, its implementation is more difficult, particularly for parallel distributed domains. Complications in 2-refinement can arise when the pairing patterns to form transition elements from nearby refinement zones do not match, or when the pairing of element layers must extend across processor boundaries. Several methods have been proposed which appear to present good results for serial applications, however implementation details for some of these methods are sparse and their application to distributed memory parallel is not addressed.

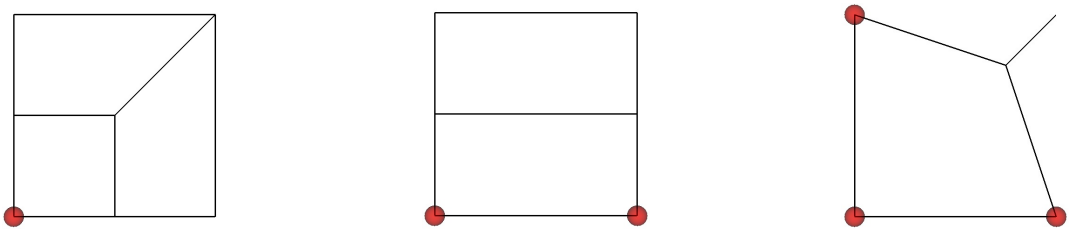
We address the need for hexahedral mesh refinement for distributed memory parallel environments. A refinement strategy that uses a deterministic algorithm that yields the same results regardless of the domain decomposition strategy is desirable. Because templates can provide local criteria for subdivision, they provide an attractive solution for parallel applications where very little inter-processor communication is necessary.

In this work we introduce a new deterministic 2-refinement strategy based on templates and marked nodes. We limit our scope to structured grids since they are used as the basis for our target overlay grid application [4]. We propose a method that provides the following characteristics:

1. One-to-eight uniform refinement
2. Controlled mesh size gradients
3. Minimum mesh quality within transition elements of 0.4 scaled Jacobian
4. Parallel-consistent (Same result regardless of decomposition)
5. Scalable to massively parallel environments
6. Multiple levels of refinement

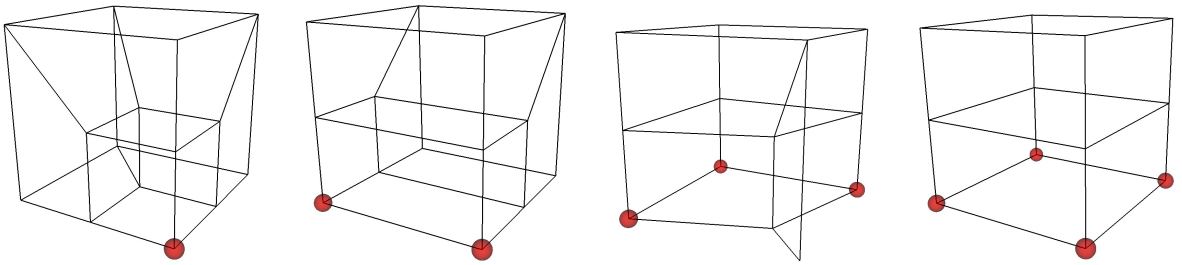
2. Previous Work

Most localized hex refinement strategies in the literature have largely been based on Schneiders' initial work on Octree meshing [5]. A detailed set of 3-refinement templates were developed and used to locally capture feature sizes. Schneiders later expands these procedures to include 2-refinement [6], introducing templates similar to those shown in figures 4 and 5. In this work, the concepts of directionally refined parallel layers is introduced, as well as a node marking strategy based on alternating nodes surrounding the refinement region to enforce pairing. We note that because of the selected overlapping strategy of the intersecting pillowing [7] operations, that transitions can quickly become complex, especially for concave regions reducing resulting mesh quality.



(a) *1-template* (b) *2-template* (c) *3-template*

Fig. 4. Complete set of 2D 2-refinement templates. Templates are identified based on number of marked nodes on a quad or hex face. Note that for 2D, only the 1-template and 2-template are required. The 2D *3-template* is used only in the context of 3-refinement for refining a hex face of the 3D *3-template* (Fig. 5c).



(a) *1-template* (b) *2-template* (c) *3-template* (d) *4-template*

Fig. 5. Complete set of 3D 2-refinement templates. Templates are identified based on the number of marked nodes on a hex

Harris [8] et. al., and later expanded by Edgel [9] and Malone [10] also provide background and implementation of a directional parallel pillowing strategy for 2-refinement. These works do not address the use of templates, instead use pillow operations on arbitrarily selected adjacent hex layers based on expansion from an initial uniform refinement zone. Mesh quality in transition zones as well as generality and extension for concave regions also was problematic in these works. In contrast, in this work, we extend Harris' parallel pillowing strategy to use a global deterministic

template-based approach that is applicable for any shaped region while maintaining mesh quality in excess of 0.4 scaled Jacobian in transition regions.

Other 2-refinement strategies include Ebedia et. al. [11], Marechal [3] and Zhang et. al. [12]. While each of these methods offers unique benefits, they each begin with a balanced octree decomposition of an initial uniform grid. They later define column groupings of four elements extended from the octree surrounding a common edge. Transitions are generated to cap these column groupings using a set of four *1-templates* as shown in figure 5a. To handle concave regions, Ebedia [11] and Zhang [12] utilize a buffer pillow layer surrounding the uniformly refined hexes through which the transition elements are defined. Although specific metrics are not reported for elements in the transition region, it appears the elements formed in the pillowed transition layers would likely be poor. Ebeida [11] and Zhang [12] also demonstrate several cases for reconciling non-manifold conditions in order to insert the pillow surrounding the uniformly refined elements, a condition which is avoided in the proposed work. Shared memory parallelism is also addressed in Ebeida's work showing examples with notable efficiency, though distributed parallelism is not tackled.

Although Marechal's [3] initial approach is similar, he uses the octree subdivision and subsequent four-column groupings as a dual representation over which a primal set of hexes is later generated. Several impressive examples are shown using the Hexotic software tool using his approach, though initial refined element quality is not reported.

We recognize the important contributions made by these previous works, however to our knowledge, the current literature does not address the parallel consistency issues for distributed domains. We note that a successful 2-refinement strategy must require pairing of adjacent layers or columns of hexes which should be applied in a globally consistent manner. In this work we propose a method that consistently applies pairing in a global manner for distributed domains, and delineates a simple set of local templates that can be consistently applied with minimal inter-processor communication.

2-refinement strategies are most often used as a precursor to an overlay grid method. These methods tend to impose severe constraints on the elements at geometric boundaries. As such, an initial refined base grid that maximizes mesh quality is advantageous. Although most current methods provide adequate topology for 2-refinement, the initial mesh quality can be lacking. As a result we attempt to provide a minimum quality mesh exceeding scaled Jacobian 0.4 prior to the boundary capture and smoothing procedures, providing a more robust platform on which to generate the mesh.

3. 2-refinement

Algorithm 1 outlines the procedure for applying 2-refinement to a set of selected elements using the templates shown in figures 4 and 5. A more detailed discussion of each of the steps of the algorithm follows using the 2D example illustrated in figures 6 to 13. See the section in parenthesis for additional details.

Algorithm 1 Two Refinement Algorithm

1: Enumerate Cartesian grid (3.1)
2: Identify and mark elements to be uniformly refined (3.2)
3: **for** each direction I, J, K **do**
4: Mark nodes between odd-even layers at marked elements (3.3, 3.6)
5: Apply refinement templates to elements with marked nodes (3.4, 3.7)
6: Expand marked elements to include recently refined elements (3.5, 3.8)
7: **end for**

3.1. Enumerate Cartesian grid

We limit our application to a Cartesian grid that has an arbitrary number of intervals in directions I, J and K. For our purpose [4] the Cartesian grid may be distributed amongst many processors. Figure 14 shows an example distributed Cartesian grid where the domain is shared amongst six processors. Each processor is responsible for independently refining selected elements within its domain. To ensure consistency between processors, we first identify the starting I, J, K indices for each processor based on the global decomposition. Since the 2-refinement procedure relies on grouping pairs of adjacent sheets of hexes, we determine whether the starting indices on each processor are either

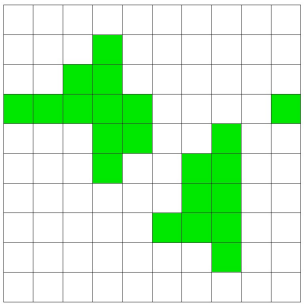


Fig. 6. Example 2D Cartesian grid showing elements identified for uniform refinement.

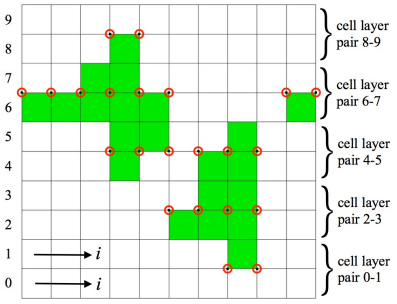


Fig. 7. Each odd-even row pair of quads are traversed and nodes between the rows adjacent to at least one marked quad is also marked.

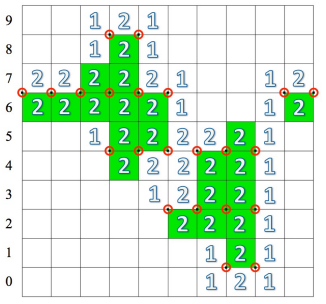


Fig. 8. Refinement templates identified based on marked nodes. In 2D only 1 and 2 templates are permitted.

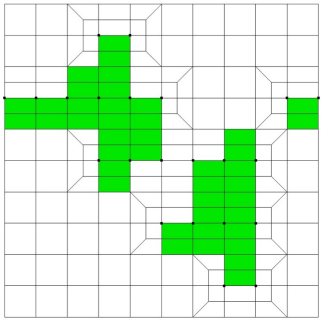


Fig. 9. Templates shown in figure 4 applied to marked nodes in I direction.

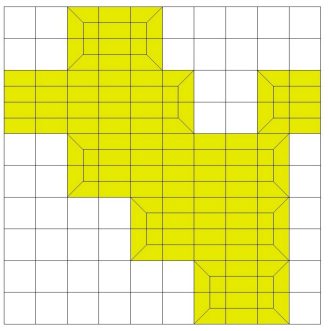


Fig. 10. Marked elements now expanded to include refined elements from I direction.

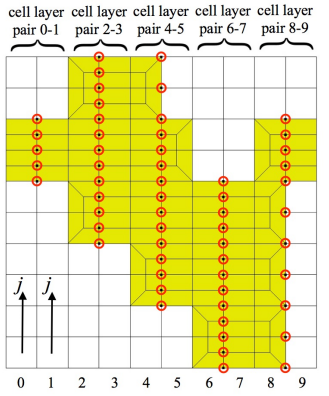


Fig. 11. Nodes marked in J direction at every other column adjacent marked elements.

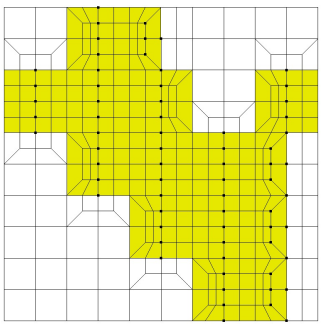


Fig. 12. Templates applied on marked nodes in J direction.

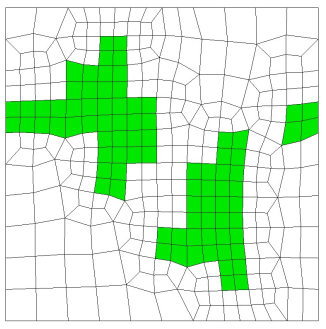


Fig. 13. 2D example from figure 12 following smoothing. Shaded quads illustrate the initial cells identified for uniform refinement.

odd or even. With this information we can consistently provide layer pairs that combine an odd-even layer across the global domain.

3.2. Identify elements to be uniformly refined

A set of elements with indices i, j, k are identified for refinement. Any criteria may be used for this identification. The criteria we use for our application is based on local geometric feature sizes, but can also be defined based on error

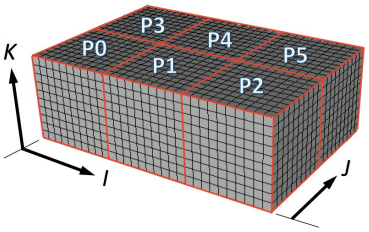


Fig. 14. Example distributed Cartesian grid on six processors

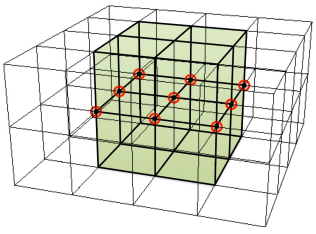


Fig. 15. 3D example showing marked nodes between an odd-even pair of layers

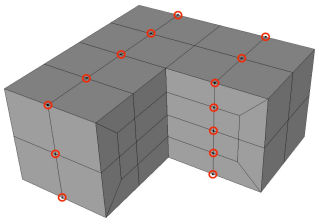


Fig. 16. Cut-away of example 3D mesh with nodes marked in *J* direction

measures, local physics or material properties, or any other criteria. We use a 2D example to illustrate in figure 6 where shaded cells are those identified for refinement. These cells are guaranteed to contain uniform refinement, or in this case, each cell will be split into four. Although by necessity, additional surrounding quads may also be uniformly refined, transition templates will be applied to other cells to achieve appropriate transitions to maintain a conformal mesh.

3.3. Mark nodes in I direction

To classify elements based on a limited set of templates, we mark some of the nodes in the grid. To enforce the paired layer objective needed for 2-refinement we mark nodes only on every other row of the grid, beginning with the *I* direction. Each odd-even row pair of cells are traversed and nodes between the rows that are adjacent to at least one uniform refinement cell are marked. This is illustrated on our 2D example in figure 7.

Similarly in 3D, we traverse parallel odd-even layers of cells. A simple 3D example is shown in figure 15 where the 9 nodes touching the uniform refinement cells are marked.

3.4. Apply refinement templates to marked nodes in I direction

Having marked nodes in the *I* direction, we identify all elements with at least one node marked and apply one of the 2-refinement templates. Figure 4 shows the templates that would be applied for the 2D problem. Figure 8 illustrates the 2D refinement templates to be used for each of the affected cells while figure 9 shows the resulting 2D refinement. In 3D, the templates shown in figure 5 are used. It should be noted that the faces of each 3D element shown in figure 5 correspond to one of the 2D templates of figure 4. We also note that this small set of templates shown in figures 4 and 5 is complete, since we limit the marking to every other odd-even set of layers.

Included in the 2-refinement templates is the *3-template*, based on three marked nodes. This template is shown in figure 4(c) for 2D and in 3D in figure 5(c). A similar template is also described by Schneiders in [6]. We note the *3-template* is only used in 3D and is useful for capturing concave regions. Additionally, the 2D *3-template* is used only to describe one of the faces of the 3D *3-template* and would not occur in a planar quad refinement problem. To facilitate its use we merge nodes on two of its adjacent elements. Figure 17 illustrates the *3-template* as it might be used adjacent to two *2-templates*.

3.5. Expand refinement elements from I direction refinement

To prepare for the next stage of refinement we first include all elements that have been previously refined within our set of marked elements. This expands the refinement region, ensuring that further transition templates will only be defined within unrefined elements helping to control element quality and size gradient. A 2D example of this expanded set of marked elements is shown in figure 10.

3.6. Mark nodes in J direction

We now repeat a similar procedure to mark nodes in the *J* direction. We consider all elements grouped within the initial odd-even cell layers of the initial Cartesian grid. All nodes within the refinement region that form the interface

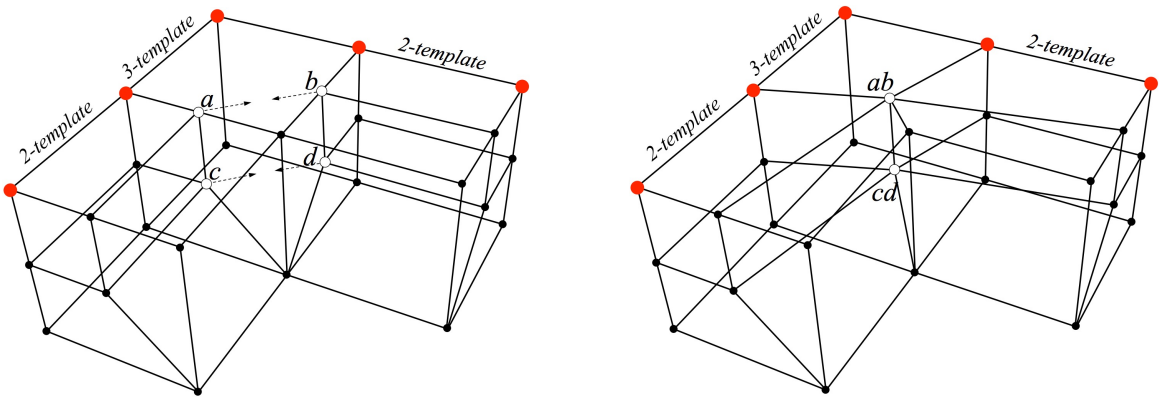


Fig. 17. (a) Two completed *2-templates* adjacent a partially completed *3-template*. Nodes *a* and *b* are merged as well as nodes *c* and *d* (b) View of completed *3-template* with modified adjacent *2-templates* after merging nodes.

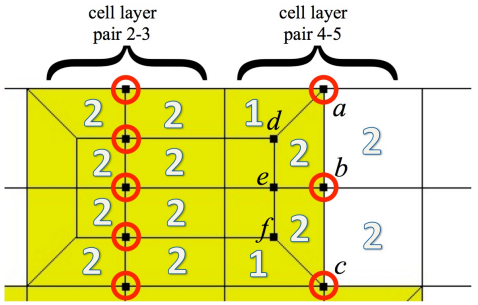


Fig. 18. Close-up of node marking and resulting templates based on procedure described above.

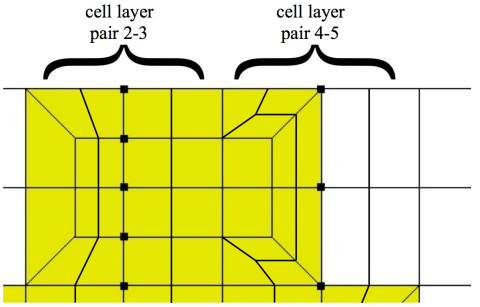


Fig. 19. Resulting mesh from basic marking procedure. Note poor angles produced from *1-templates* adjacent the marked nodes *a* and *c* in fig. 18.

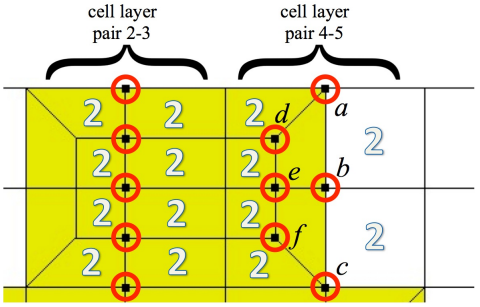


Fig. 20. Example of propagation of node marking and resulting templates. The marking at nodes *a*, *b* and *c* are propagated to nodes *d*, *e* and *f* respectively.

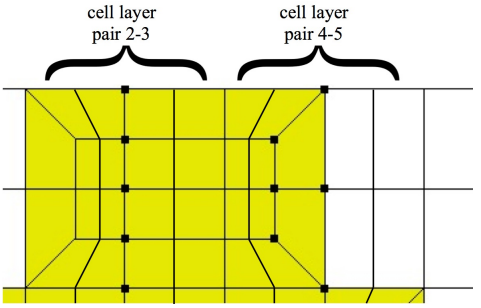


Fig. 21. Example of resulting mesh where node propagation has been performed.

between odd and even cell layers are marked, including newly added nodes as a result of refinement in the *I* direction. Figure 11 shows a 2D example of nodes marked in the *J* direction. A simple 3D example is shown in figure 16

An exception to the basic procedure of marking nodes between odd-even layers of elements is illustrated in figures 18 to 21. Figure 18 shows a close-up of some of the odd-even layers of hexes from the example in figure 11. It also illustrates the resulting templates that would result from the basic marking procedure. Figure 19 shows the resulting refinement as it would occur with the basic marking procedure. The *1-templates*, that may result in poorer quality elements, can however be eliminated by propagating the marking to the nodes on the interior of the cell layer as shown in figure 20. For this example, the marking at nodes *a*, *b* and *c* is propagated to nodes *d*, *e* and *f* respectively. Note that

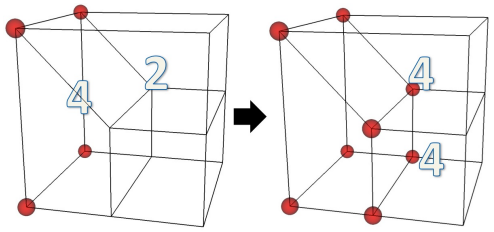


Fig. 22. Example of 3D node marking propagation on an element previously refined using a *2-template*.

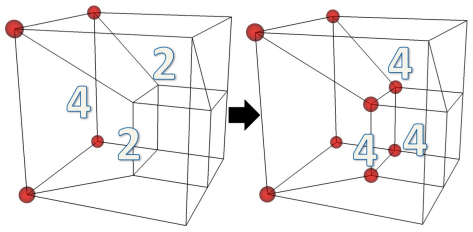


Fig. 23. Example of 3D node marking propagation on an element previously refined using a *1-template*.

nodes *a*, *b* and *c* remain marked in figure 20 resulting in quads with all four nodes marked. Quads or hexes with all nodes marked are ignored by the refinement procedure. The resulting 2D quad mesh using the propagation procedure in this example is shown in figure 21.

A 3D example of the node marking propagation procedure is also shown in figures 22 and 23. In figure 22 the initial marked nodes result in a *4-template* and a *2-template*. By propagating the marked nodes we avoid the *2-template* in favor of a *4-template*. Similarly in figure 23, two *2-templates* are replaced with two *4-templates*. Note that conditions for propagation will only occur when a cell has previously been refined using a *1-template* or *2-template*.

3.7. Apply refinement templates to marked nodes in J direction

Once the nodes have been marked and appropriately propagated as needed, each element with at least one node marked is refined using the templates illustrated in 4 and 5. At this stage, our 2D refinement example is complete. To illustrate its effectiveness, figure 13 shows our 2D example after smoothing has been performed. The shaded elements show the initial region that was marked for uniform refinement.

3.8. K direction refinement

For the 3D problem we continue the procedure by expanding the set of marked elements to all elements that have been previously refined by both *I* and *J* direction refinement procedures. Odd-even pairs of cells in the *K* direction are then used in a similar manner to mark nodes between the layers, including nodes from elements previously refined. The procedure for propagating node marking to the interior of the cells is also used for the *K* direction. Finally, all elements with at least one node marked are then refined using the templates illustrated in figure 5.

3.9. Additional node marking requirements in 3D

We also note two other conditions in 3D where it is necessary to mark additional nodes that may not be included in the set of marked elements. These cases are illustrated in figures 24 and 25. If we consider the plane between the odd-even layers of cells, it is possible to mark nodes such that invalid templates could be identified. The case illustrated in figure 24 shows face *A* at the odd-even layer interface where only nodes *a* and *c* would be marked. Since diagonal marking on a single face is not handled by our templates, we simply add the other diagonal nodes *b* and *d* into the set of marked nodes.

Figure 25 illustrates a case where faces *A* and *B* lie between the odd-even layers of hexes with surrounding marked elements as shown. Without additional node marking, both faces *A* and *B* would require *3-templates* for refinement. Because a *3-template* will merge nodes from neighboring elements, we restrict the use of the *3-template* so that two adjacent *3-templates* cannot be defined. When a case such as this is identified, the fourth node, in this example node *f*, is marked eliminating this condition.

4. Multiple levels of refinement

In the previous section we outlined a strategy for two refinement involving transitioning from 1 to 8 elements. It is also possible to incorporate multiple levels of refinement where a second level would split 8 elements into 64, a third

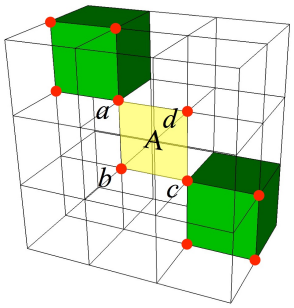


Fig. 24. Nodes b and d at face A would also be marked to avoid an invalid template for adjacent cells at A

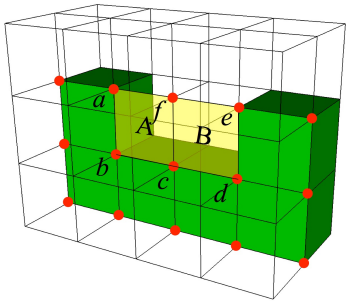


Fig. 25. Node f would also be marked to avoid two adjacent *3-templates*

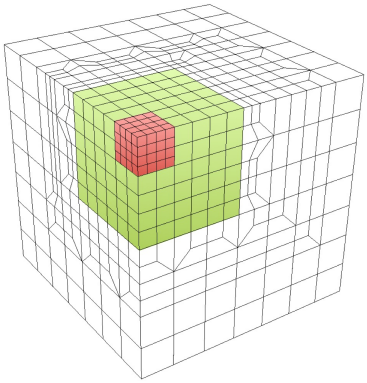


Fig. 26. Example 3D refinement showing first level of refinement from green cells. Green cells will serve as transition zone for second level of refinement from uniformly refined red cells

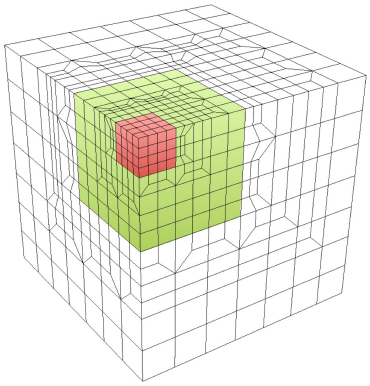


Fig. 27. Transition elements for second level of refinement added.

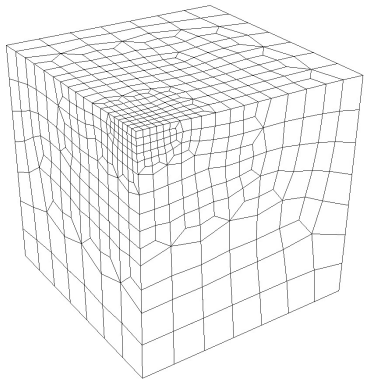


Fig. 28. Smoothing applied to the model in figure 27

into 256 and so on. Because the proposed 2-refinement strategy will work only within a structured set of cells, we limit the introduction of additional levels of refinement to only uniformly refined parent cells. We also note that since our strategy can take up to three additional hex layers to effect the transition, we first expand the uniform refinement zone of the parent cells to allow for at least three layers surrounding the child uniform refinement. This is illustrated in figures 26 and 27. Once the lowest level of refinement is determined based on local feature sizes, a balancing procedure is performed to ensure sufficient transition layers are provided for each level of refinement. In figure 26 we use two additional layers of the initial hexes (shown in green) surrounding the second level of refinement (shown in red). After the first level of refinement is complete, four layers of uniformly refined hexes are available to be used for the transition elements of the second level of refinement shown in figure 27. In figure 28 we show the same mesh following a smoothing operation.

To perform the refinement at subsequent levels, the same procedure outlined in Algorithm 1 is performed. This can be accomplished recursively for as many levels as desired. Since the parent cells will have already been uniformly refined, the parent cell layers will each contain an odd-even set of child layers that can be used for the marking procedure, thus maintaining the parallel consistent nature of the algorithm through multiple levels of refinement.

5. Parallel implementation

One of the principal objectives of this work is to provide a scalable implementation that can be used in a massively parallel distributed environment. To accomplish this it was necessary to provide a parallel consistent algorithm that required minimal inter-processor communication. For our purposes we limit the global domain to be a Cartesian grid that has been distributed across multiple processors, where each domain is itself a Cartesian grid. Section 3.1

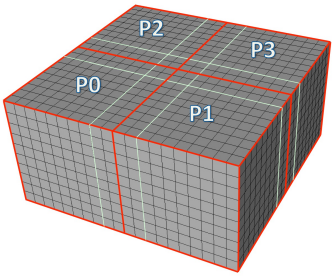


Fig. 29. Example parallel distribution of Cartesian grid on four processors

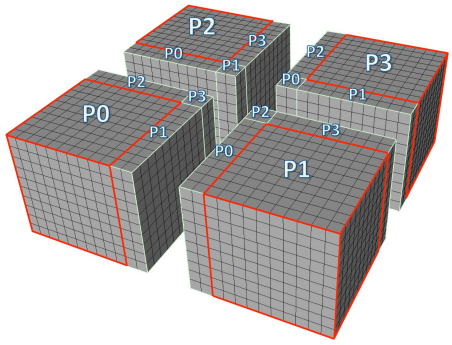


Fig. 30. Exploded view of 4 processors where ghosted cells of neighboring processors are included for each processor

describes the initial global enumeration of the grid where the starting i, j, k indices for each processor are identified as either odd or even. This provides the basis for the odd-even layer pairs used in the node marking procedure and ensures that all templates will be conformal across processors with minimal additional communication.

In order to maintain a conformal mesh across processors, two required instances of inter-processor communication are noted. Communication is effected by using ghosted cells at the boundaries of each domain. A four processor example is illustrated in figures 29 and 30. Initially, two layers of cells are included in the ghosted regions of each processor. For each hex and its refined children, we keep track of their owning processor, where any refined element is owned by its parent's processor. Redundant and identical refinement operations are performed on the owning processor and its ghosts on neighboring processors. With consistent node marking provided by the global odd-even layer pairing, we are guaranteed a conformal mesh across processor domains provided the following two communication steps are effected.

5.1. *Communication of node marks*

Although the basic node marking procedure described in section 3 can be done independently on each processor, the additional cases for node marking described in section 3.9 may expand the marking into a neighboring processor. In addition, when multiple levels of refinement are used, as described in section 4, the expansion of the uniformly refined region necessary to include multiple levels may extend across processors. To allow for these cases, nodes marked within the ghosted region of a neighboring processor are communicated prior to performing template-based refinement. This is done by identifying nodes on the current processor that have been marked that have a corresponding ghosted node on a neighboring processor. Messages are passed between neighboring processors to update the marked state to reflect the status of each node's owning processor. Communication of marked nodes is done once for each direction I, J and K. In algorithm 1 above, this communication would occur following line 4, prior to performing refinement in each direction.

5.2. *Communication of* 3-template *node movement*

For most templates, Node locations can be simply determined as the mid-point of edges, faces and hexes. This provides a parallel consistent implementation across processor boundaries. The one exception to this case is when a *3-template* is applied near a processor boundary. Figure 31 shows a 2D representation where a *3-template* is identified on processor 0 and node n_0 is relocated to form the template. Since processor 1 has only a single node from the *3-template*, it will define a different location for the same node (n_1) at the mid-point of an edge. To account for this case, we include an additional communication step following template refinement to update node locations. This is done by keeping track of node locations updated as a result of *3-template* operations on the owning processor. These new node locations are communicated to neighbor processors and locations updated if the node lies at a domain boundary. In algorithm 1 above, this communication would occur following line 5, following refinement in each direction.

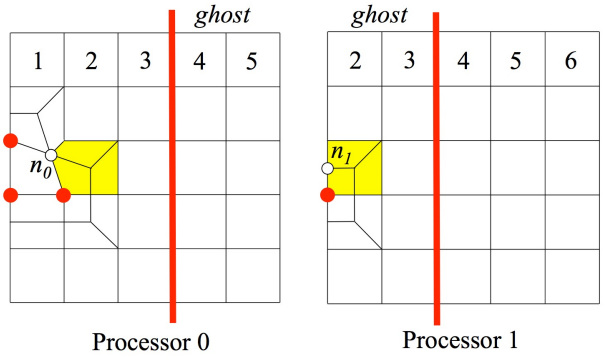


Fig. 31. Illustration of discrepancy between node location n_0 and n_1 on neighboring processors resulting from a *3-template* near a processor boundary.

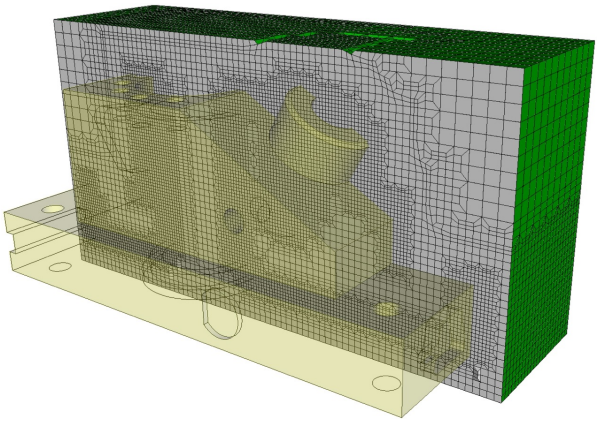


Fig. 32. Cut-away of refined base mesh for anc101 model

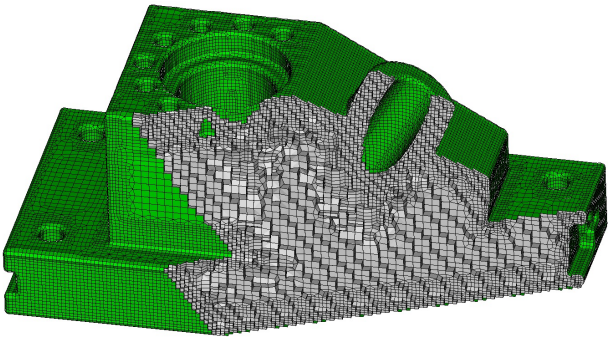


Fig. 33. Sculpt mesh of anc101 model

6. Examples

We illustrate the proposed algorithm with a limited set of examples. The intended objective of the proposed refinement algorithm is its use within Sandia's Sculpt [4] tool. In this tool, a regular, user-defined Cartesian grid is specified and the refinement procedure applied based on geometric indicators. For the examples shown, we use criteria similar to those described in [1], however integration of our 2-refinement procedure is on-going and note that although refinement criteria is vital, it is beyond the scope of the current work. Once a refined grid is complete, the Sculpt procedure, as described in [4] is employed to generate an all-hex finite element mesh.

We show examples of the refined Cartesian grids in figures 32, 34 and 36 where we have limited the number of refinement levels to 2. We note that following refinement, the minimum scaled Jacobian for each refined grid is 0.408. Initial results using the Sculpt tool are also shown in figures 33, 35 and 37, although at this writing, additional optimization of Sculpt to use the proposed 2-refinement operations are still underway.

Also illustrated is figure 39 are initial performance times versus numbers of processors for the model anc101 shown in figures 32 and 38. Strong scaling tests were performed on up to 256 processors on Sandia's Redsky and TLCC2 Chama platforms. Tests measured the time to refine an initial Cartesian grid bounding the anc101 model with two different resolutions. The first series of tests started with approximately 32K hexes with refined mesh at 730K hexes and the second started with 499K hexes and finished with 4.75M. We note in all cases the resulting meshes were identical for their respective tests with minimum scaled Jacobian of 0.408, demonstrating the algorithm's parallel consistency.

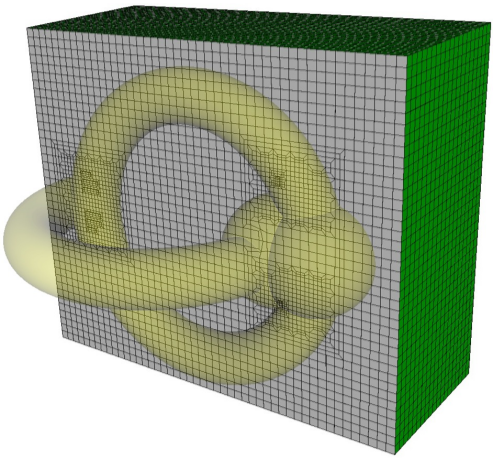


Fig. 34. Cut-away of refined base mesh for a027 model

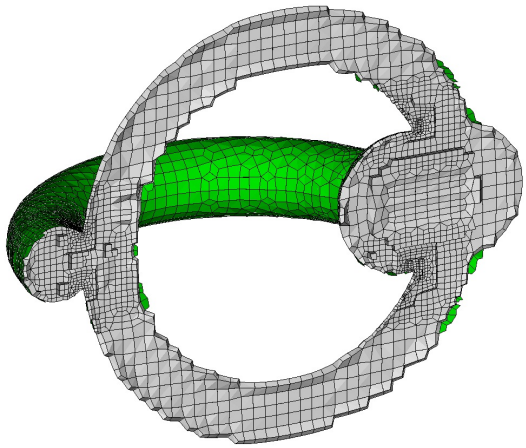


Fig. 35. Sculpt mesh of a027 model

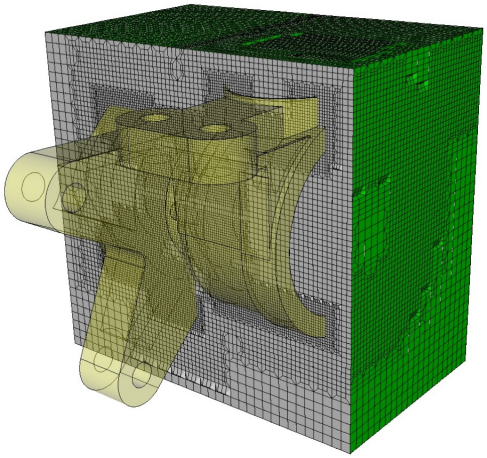


Fig. 36. Cut-away of refined base mesh for driver model

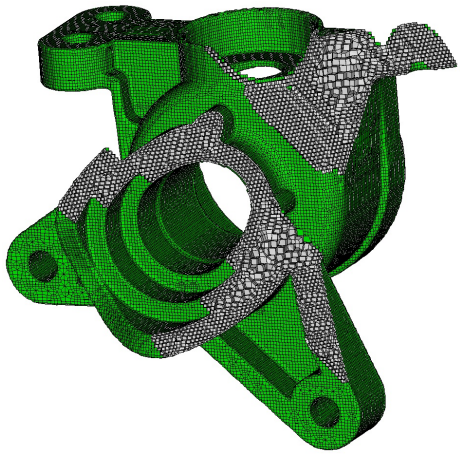


Fig. 37. Sculpt mesh of driver model

7. Conclusion

2-refinement is generally preferred over 3-refinement methods for capturing features in overlay grid methods. We have proposed and demonstrated a 2-refinement procedure that is applicable for a distributed memory parallel environment that maintains consistent hex layer pairing across processors. A modest set of refinement templates combined with a node marking procedure result a relatively simple implementation that avoids some of the layer pairing issues arising in other published methods. We note that this method avoids the non-manifold resolution and additional pillowing operations sometimes required by other 2-refinement procedures [11][12]. It also extends previous layer pairing procedures [8][10] to effectively incorporate concave regions while using a simple set of refinement templates. Examples show that mesh quality exceeds 0.4 scaled Jacobian in all cases prior to any smoothing, which appears to be a significant improvement over alternative methods. The method also provides for multiple levels of refinement although current implementation and examples have been limited to two levels.

In contrast to 3-refinement and other published 2-refinement methods, we recognize that in order to achieve the observed mesh quality, the transition zone may extend up to three additional layers from the uniform refinement zone. The additional space provided by the proposed method provides for smooth size gradients in transition regions which can dramatically improve mesh quality in the final overlay grid mesh.

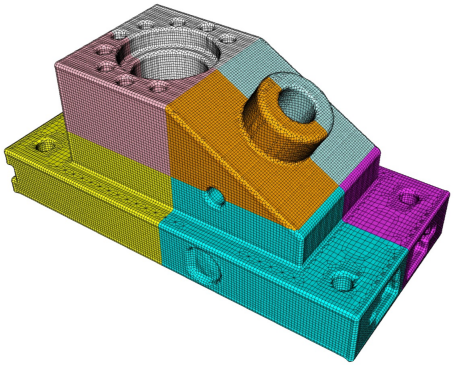


Fig. 38. anc101 model used in performance study. Illustrates final mesh generated with sculpt [4] on 8 processors. Color represents the different processor domains. Performance numbers in fig. 39 are for refinement algorithm only and do not include time to generate the final mesh.

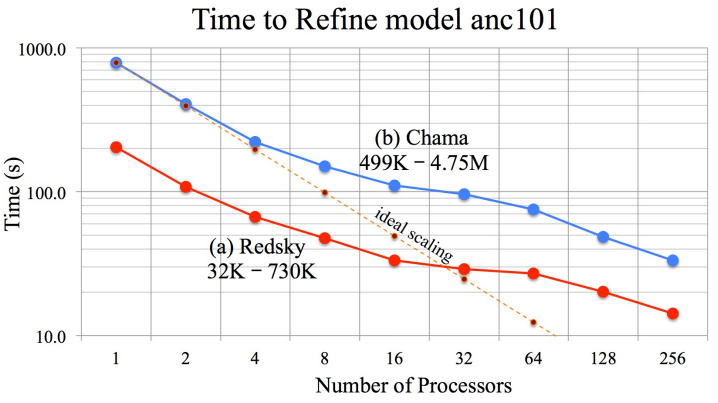


Fig. 39. Performance of refinement vs number of processors for model anc101. (a) Initial hexes in Cartesian grid: 32,400, Final number: 730,512. Tests performed on Sandia's Redsky platform (519/4,152 nodes/cores 2.93GHz Intel, 12GB/node) (b) Initial hexes in Cartesian grid: 498,960, Final number: 4,750,848. Tests performed on Sandia's TLCC2 Chama platform (1,232/19,712 nodes/cores 2.6GHz Intel, 64GB/node). Note all runs on respective machines produced identical meshes regardless of processor count.

In this work we have presented an algorithm for 2-refinement along with results to demonstrate its effectiveness. Future work will further optimize this method with the sculpt software [4] running in a massively parallel environment. While initial results have shown effective results on up to 256 processors with MPI, future work will require performance enhancements as we examine its usefulness and further scalability on massively parallel HPC machines.

8. Acknowledgements

Funding for this work was provided by the US Department of Energy Advanced Scientific Computing program. Special thanks to Brett Clark and Randy Morris for providing review and feedback. CAD Models used in examples provided by Ansys, Inc.

References

[1] Y. Zhang, C. Bajaj, Adaptive and Quality Quadrilateral/Hexahedral Meshing from Volumetric Data, Comput Methods Appl Mech Eng, (2006) 195(9): 942-960
[2] Y. Ito, A. M. Shih, B. K. Soni, Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates, Int. J. Numer. Meth. Engng, (2008) 77:1809–1833
[3] L. Marechal, Advances in Octree-Based All-Hexahedral Mesh Generation: Handling Sharp Features, In: Proceedings of the 18th International Meshing Roundtable, (2009), pp 65–84
[4] S. J. Owen, M. L. Staten, M. C. Sorensen, Parallel Hex Meshing from Volume Fractions, In: Proceedings 20th International Meshing Roundtable, (2011) pp. 161–178
[5] R. Schneiders, Quadrilateral and Hexahedral Meshes, Handbook of Grid Generation, J. F. Thompson et. al. (Eds.) CRC Press LLC. (1999) Chapter 21.
[6] R. Schneiders, Algorithms for quadrilateral and hexahedral mesh generation. In: Proceedings of the VKI lecture series on computational fluid dynamics (2000)
[7] S. A. Mitchell, T. J. Tautges, Pillowing doublets: Refining a mesh to ensure that faces share at most one edge, In: Proceedings 4th International Meshing Roundtable, (1995) pp. 231-240
[8] N. Harris N, S. Benzley, S. Owen, Conformal refinement of all-hexahedral element meshes based on multiple twist plane insertion, In: Proceedings 13th International Meshing Roundtable (2004), pp 157-168.
[9] J. Edgel, S. E. Benzley, S. J. Owen, An Adaptive Grid-Based All Hexahedral Meshing Algorithm Based on 2-Refinement, In: Proceedings 19th International Meshing Roundtable, Research Notes (2010)
[10] J. B. Malone., Two-Refinement by Pillowing for Structured Hexahedral Meshes, Thesis, Brigham Young University (2012)
[11] M. S. Ebeida, A. Patney, J. D. Owens, E. Mestreau, Isotropic Conforming Refinement of Quadrilateral and Hexahedral Meshes Using Two-Refinement Templates, Int. J. Numer. Meth. Engng 2011; 88:974-985
[12] Y. Zhang, X. Liang, G. Xu, A Robust 2-Refinement Algorithm in Octree and Rhombic Dodecahedral Tree Based All-Hexahedral Mesh Generation. In: Proceedings of the 21st International Meshing Roundtable (2013), pp. 155–172