UCRL-TR-218661

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Adaptive Mesh Refinement Algorithms for Parallel Unstructured Finite Element Codes

I. D. Parsons and J. M. Solberg

February 7, 2006

Disclaimer

Auspices Statement

This work was performed under the auspices of the U. S. Department of Energy (DOE) by the University of California, Lawrence Livermore National Laboratory (LLNL) under Contract No. W-7405-Eng-48. The project (03-ERD-027) was funded by the Laboratory Directed Research and Development Program at LLNL.

FY05 LDRD Final Report

Adaptive Mesh Refinement Algorithms for Parallel Unstructured Finite Element Codes

LDRD Project Tracking Code: 03-ERD-027

I. D. Parsons, Principal Investigator
J. M. Solberg, co-Principal Investigator

Executive summary

This project produced algorithms for and software implementations of adaptive mesh refinement (AMR) methods for solving practical solid and thermal mechanics problems on multi-processor parallel computers using unstructured finite element meshes. The overall goal is to provide computational solutions that are accurate to some prescribed tolerance, and adaptivity is the correct path toward this goal. These new tools will enable analysts to conduct more reliable simulations at reduced cost, both in terms of analyst and computer time. Previous academic research in the field of adaptive mesh refinement has produced a voluminous literature focused on error estimators and demonstration problems; relatively little progress has been made on producing efficient implementations suitable for large-scale problem solving on state-of-the-art computer systems. Research issues that were considered include: effective error estimators for nonlinear structural mechanics; local meshing at irregular geometric boundaries; and constructing efficient software for parallel computing environments.

Introduction and background

The objective of this research is to implement adaptive mesh refinement (AMR) algorithms in unstructured finite element codes used for solving nonlinear, coupled solid and thermal engineering problems on ASCI-class, multi-processor parallel (MPP) computers.

AMR has long been the subject of finite element research (e.g., [1][2][3][4][5]). A solution on a given finite element mesh is improved by either subdividing elements with high estimated errors (*h*-refinement), increasing the order of polynomials used to interpolate the unknown variables in elements with high estimated errors (*p*-refinement), or relocating selected nodes while maintaining the topology of the original mesh (*r*-refinement); see Figure 1 (all figures are contained in Appendix B at the end of this report). Variations on these basic themes have been developed, such as *h*-refinement involving complete remeshing with smaller elements specified in high error regions and *hp*-refinement that both subdivides and increases the order of the basis functions in selected elements. The primary benefit of these approaches is that nearly optimum meshes can be produced automatically with little or no user intervention, thereby reducing the cost of an analysis and increasing its reliability.

Previous research has generally concentrated on developing effective error estimators for tractable problems, rather than on implementing AMR methodologies in a high-performance computing environment and refining them for solving practical engineering problems. Effective error estimators are the essential first step in producing effective AMR schemes; the rarity of advanced applications is partly a consequence of the unsuitability of the architectures of legacy finite element codes for both porting to MPP environments and the introduction of AMR algorithms.

Structural analysis codes at LLNL have successfully employed large numbers of low order hexahedral elements to solve nonlinear problems using both implicit and explicit time integration rules. Tetrahedral and high order hexahedral elements have inherent disadvantages when used to solve problems involving plastic flow, large deformations and contact. To date, no general-purpose meshing tool has been developed that can automatically generate high-quality hexahedral meshes; therefore, unstructured meshes are built by analysts using TrueGrid. The success of this general approach indicates that LLNL structural analysis codes will benefit from local *h*-refinement of existing low order hexahedral elements in unstructured meshes initially generated using TrueGrid. Complete remeshing based on error estimators is currently impractical, and *p*-refinement introduces element technology issues that may be difficult to solve. An opportunity currently exists for implementing AMR schemes in an MPP environment to solve complex solid mechanics problems at LLNL because research in error estimators has matured to provide several candidate schemes for practical analysis, and Diablo, an object-based analysis code for unstructured implicit finite element computations on MPP machines is currently under development. Diablo's modern software architecture is more readily adapted to the data structures required by an efficient parallel AMR strategy than legacy codes.

This report is structured in the following fashion. First, the various components of the serial AMR code infrastructure are described that enable adaptive mesh refinement to proceed. Second, the error estimators for nonlinear solid mechanics that drive the refinement are outlined. Parallelization of the AMR algorithms is then explained, followed by a series of applications that highlight the capabilities developed during the course of this project. Appendix A contains the references, and Appendix B contains all of the figures.

Serial AMR code infrastructure

The serial AMR capability that has been added to Diablo includes the data structures and algorithms required to refine a user-defined mesh and utilize error estimators based on patch recovery techniques. This section describes the various components of the software infrastructure that implement these features.

Element refinement and nodal constraints

The AMR implementation is able to perform both isotropic and anisotropic refinement as well as derefinement of a mesh. Anisotropic refinement (i.e., refinement based on a directional error estimator) and derefinement are crucial for solving highly transient problems and implicit analysis, where the cost of solving equations can increase substantially with problem size. Figure 2 shows both isotropic and anisotropic *h*-refinements of a single coarse element. Isotropic refinement is directionally invariant, causing a single hexahedron to be refined into eight new hexahedrons. Anisotropic refinement requires that the error estimator identify local element directions for refinement. As shown in Figure 2, this may cause a hexahedral element to be split into two or four new elements. Anisotropic refinement is useful for problems that contain strong directional dependencies, or for meshes that possess poor aspect ratios.

When an element undergoes isotropic or anisotropic refinement, hanging nodes will generally be created as shown in Figure 3. A hanging node appears on the boundary of an element adjacent to the element being refined, and must receive special attention so that the inter-element compatibility of fields is preserved. In this work, this compatibility is maintained by applying algebraic constraints to the degrees-of-freedom at the hanging nodes. For example, in Figure 3, the degrees-of-freedom at nodes *c* and *d* are constrained to be

$$u_c = \tfrac{1}{2} u_a + \tfrac{1}{2} u_b$$

and

$$u_d = \tfrac{1}{4} u_a + \tfrac{3}{4} u_b ,$$

respectively. The weights applied to the degrees-of-freedom at the master nodes *a* and *b* are determined by the relative locations of the various nodes.

Figure 3 shows that the refinement scheme supports meshes of arbitrary irregularity. In an *n*-irregular mesh, adjacent elements differ by no more than *n* levels of refinement; by inspection, the meshes in the figure are 1- and 2-irregular. Anisotropic refinement in particular will tend to produce meshes with an irregularity greater than one, and therefore the refinement scheme must be able to support the generation of such meshes.

The basic data object that is employed in the mesh refinement procedures is a single AMR block. The AMR block can be viewed as a single hexahedral element on the initial coarse mesh presented for adaptive refinement. Thus, isotropic or anisotropic refinement of a mesh is implemented as refinement of the individual AMR blocks, which in turn produces new elements and nodes in the AMR database that are eventually transferred to the global mesh database. During the refinement procedures, the elements and nodes in an AMR block are referenced using two sets of octrees. Manipulation of selected elements and nodes is then readily performed using suitable recursive procedures.

The element octree is straightforward: each parent leaf has up to eight child leaves pointing to the data produced after refinement of the parent element. The node octree uses an (i, j, k) index space to locate nodes within the space occupied by the AMR block. Figure 4 illustrates the basic idea for a two dimensional AMR block (in two dimensions, the octree is replaced by a quadtree). The vertices of the block are assigned integer coordinates (i, j) as shown in the figure, such that $(i, j) \in (0, 2^r)$ where *r* is the maximum number of permitted refinements. As elements are refined, nodes are produced with new index coordinates $(i, j) \in (0, 2^1, \ldots, 2^{r-1}, 2^r)$. These coordinates are generated using integer arithmetic as shown in Figure 5 for new nodes. The location of the nodes in the index space can be conveniently arranged in a tree as shown in Figure 6 and Figure 7 for a simple two dimensional example. The four vertices of the AMR block form the four child leaves of the tree root at level 1. Each of these leaves may produce up to four children at level two; in the refinement shown in Figure 6, nodes 1 and 3 produce nodes 5 and 6, respectively. During subsequent refinement, inactive leaves for nodes 7 and 8 must be introduced at level 2 to enable the introduction of these nodes on level 3. In three dimensions, the AMR block nodal data are organized in an octree that may be searched to prevent the creation of duplicate nodes and to identify hanging nodes that must be constrained.

After the elements within an AMR block have been refined, hanging nodes are identified by

searching the nodal tree for all nodes present within the index space defined by each element in the AMR block. This is illustrated in Figure 8. The (i, j) coordinates of the shaded element define the region to be searched; this produces the list of nodes consisting of the element's four vertices and nodes 7 and 12. These two nodes therefore are located on the boundary of the element and must be constrained to the element vertices. The appropriate constraint weights are easily computed using the (i, j) index coordinates of the constrained nodes and element vertices. Composite constraints, in which a node is constrained to another constrained node, are identified by a recursive search of all the constraint relations produced by the search of the nodal octree. For example, the octree search will constrain node 9 to nodes 7 and 8 (i.e., $u_9 = \frac{1}{2}u_7 + \frac{1}{2}u_8$); this constraint is modified to contain the constraint imposed on node 7 via the recursive constraint search (i.e., $u_9 = \frac{1}{4}u_5 + \frac{1}{4}u_6 + \frac{1}{2}u_8$). The constraints are then applied to the degrees-of-freedom at the hanging nodes during the assembly of element quantities into the global system of equations.

Identification and constraint of nodes across AMR block boundaries is achieved using similar procedures applied to nodal data structures constructed for interblock faces and edges. Nodal and element trees are generated on each block face and edge as the refinement proceeds. The AMR block data structure includes pointers to these block boundary data as shown in Figure 9, allowing each block to search the topology of the adjacent block faces and edges in the same way described for the interior of the block. This arrangement of data greatly simplifies the parallelization of the refinement algorithms, as is subsequently explained.

Special considerations for anisotropic refinement

Intrinsically, isotropic refinement produces hanging nodes that can always be constrained to preserve inter-element compatibility. However, this is not always the case with unfettered anisotropic refinement, and so some restrictions must be placed on this type of refinement. To demonstrate, consider the refinement pattern shown in Figure 10, where two elements undergo two levels of anisotropic refinement. Figure 11 shows views of the refinement on the shared face between the two original elements, viewed from the top and from the bottom. In both figures, hanging nodes are shown in red. After the first refinement, inter-element compatibility can be maintained across the shared face by applying constraints to the hanging nodes in the usual way. However, this is not possible after the second refinement. On the top part of the shared face, the value of the degree-of-freedom at point x in Figure 11 is

$$u_x = \tfrac{1}{4}u_b + \tfrac{1}{4}u_c + \tfrac{1}{2}u_e;$$

on the bottom half, it is

$$u_x = \tfrac{1}{4}u_a + \tfrac{1}{4}u_b + \tfrac{1}{4}u_c + \tfrac{1}{4}u_d.$$

Clearly, in general these two values cannot be equal, and so inter-element compatibility is lost.

The problem is caused by the pattern of the first refinement, in which element faces overlap one another. This pattern, dubbed the "cross of death," must be avoided. This is done by monitoring the refinement of adjacent elements when employing anisotropic refinement.

Treatment of non-planar geometries

Mesh refinement on non-planar surfaces must be done so that new nodes are placed on the surfaces. For solid mechanics problems involving only small deformations, this is relatively simple since the undeformed configuration can be captured using standard CAD geometric objects, and the

new nodes subsequently projected onto these geometric objects. This is standard practice with mesh generation software packages. However, if large deformations must be considered, non-planar surfaces are themselves undergoing deformation, and can no longer be described using the intitial geometry. To solve this problem, Gregory patches [6] are used to resolve non-planar surfaces using the initial mesh. Figure 12 shows the face of an element in the initial unrefined mesh that has been modeled using a Gregory patch. This patch produces a curved surface using the 12 control points shown, which depend on the nodal coordinates and the surface normals at the nodes. As the mesh deforms, this patch is recomputed and used when projecting new nodes onto the curved boundary of the mesh.

Code architecture

In order to carefully plan and document this work, the Unified Modeling Language (UML) [7] was used to model the software system. Figure 13 and Figure 14 show class and sequence diagrams, respectively, that contain portions of the model. This work has ensured that an organized and efficient software architecture is used to implement the features discussed in this section, and is expected to enhance the usability of the AMR implementation.

Error estimators for nonlinear solid mechanics

Patch-based recovery error estimators were developed for specific application to large-deformation solid mechanics. These procedures smooth computed solution gradients to produce an improved estimate of the true solution; the estimated error is then the difference between the original and smoothed solutions [8].

Figure 15 demonstrates the generation of a patch employed in patch-recovery error estimation. A patch of elements is identified with each interior node as the elements adjacent to that node. The computed Gauss point gradients are smoothed using a least-squares fit to a first order polynomial, thereby producing a smoothed value of the nodal gradients, $\overline{\nabla \varphi_h}$ in Figure 15. The nodal gradients are interpolated over all of the elements using the standard element basis functions, enabling the estimated error to be calculated at each element Gauss point as $\left\| \overline{\nabla \varphi_h} - \nabla \varphi_h \right\|$. Boundary node values are calculated using patches developed for internal nodes, whereas smoothed nodal gradients at constrained nodes are computed by applying the constraints to the nodal gradients at the relevant master nodes. All of these operations are performed using the data structures present at the global mesh level (i.e., the AMR block structures discussed above are not required) and can be computed separately, independent of the AMR computations.

This basic patch-recovery scheme was modified to drive the anisotropic refinement in the following way. Figure 16 shows a two-dimensional parent element in (ξ, η) parametric space. The smoothed stress is denoted as σ_{rec} and has the values σ_a, σ_b, σ_c and σ_d at the four Gauss points. In order to evaluate the error in the ξ-direction, the Gauss point stresses are first averaged in the η-direction to produce the element stress distribution denoted as $\sigma_{rec,\bar{\eta}}$ in Figure 16. The ξ-direction error is then computed as

$$\delta_{\bar{\eta}} = \left\| \sigma_{rec} - \sigma_{rec,\bar{\eta}} \right\|,$$

with the subscript $\bar{\eta}$ indicating that the η-dependence has been eliminated by averaging. Similarly, the η-direction error is computed as

$$\delta_{\bar{\xi}} = \left\| \sigma_{rec} - \sigma_{rec,\bar{\xi}} \right\|.$$

If the error is substantially greater in the ξ-direction, i.e., if $\delta_{\bar{\eta}} \le \varepsilon_{tol} \delta_{\bar{\xi}}$ with ε_{tol} being a user-defined tolerance, refinement occurs in the ξ-direction only, as shown in Figure 17. The same criteria is used to control refinement in the η-direction, again shown in the figure.

This algorithm provides an inexpensive, yet robust scheme for producing anisotropically refined meshes.

Parallelization of the AMR algorithms

The object-oriented design of the serial AMR code infrastructure described earlier in this report made parallelization of the various algorithms straightforward. Figure 18 shows an initial mesh partitioned for processing on eight processors. Each one of the elements on this mesh is treated as an independent AMR block, with refinement on each processor being conducted using the serial AMR procedure. Figure 19 shows the local refinement that may typically occur on one of the processors. During this refinement, inter-block constraints must be applied between blocks local to the processor as well as between blocks on remote processors. Figure 20 illustrates how the inter-processor constraints are handled. For an AMR block that shares a face with a block on a remote processor, the AMR block uses the same pointer data structure employed in the serial case. However, the adjacent face and edge topology data must now be communicated between processors when needed using standard MPI routines. A subsequent example will show that the cost of this communication is minimal.

Applications

A variety of applications are presented in this section to demonstrate the accuracy and performance of the AMR algorithms presented in the previous sections of this report.

Code and algorithm verification

It is extremely important to verify that the AMR algorithms are correctly implemented in the code. A useful set of verification problems are shown in Figure 21, in which a single element is subjected to both isotropic and anisotropic refinement based on random numbers. On any given mesh, each element is assigned a random number between 0 and 1; those elements given a number greater than 0.5 are refined, the remainder are left unchanged. This produces refined meshes generally similar to those shown in Figure 21. The problem domain is assigned boundary conditions consistent with a constant stress state (known as a patch test). If the code is correct, then all refined meshes will compute this constant stress state.

These series of randomly-refined patch tests proved to be a very useful debugging tool, and demonstrate that the code and algorithms are correct.

Non-planar surfaces

Figure 22 shows a problem used to demonstrate the use of Gregory patches to track non-planar deforming boundaries. A square plate with a circular hole is subjected to end tractions t_0 and a pressure p_0 applied to the interior of the hole. The figure also shows the initial mesh. Figure 23 shows the refined mesh after the tractions have been applied in step 1, and after the additional application of the pressure in step 2. The refinement controlled by the Gregory patches has

obviously been able to capture the deformations of the circular hole.

Anisotropic refinement

Figure 24 shows a comparison between isotropic and anisotropic refinement of an initial mesh that was subjected to a pressure load over the top surface of the innermost element. This problem highlights the efficiencies that can be gained using anisotropic refinement. For the isotropic case, 6,751 elements were used and 63 secs of CPU time required to solve the problem to an accuracy of 1%. Anisotropic refinement reduced the elements by 44% (4,677 elements) and the time by 40% (45 secs) for the same 1% accuracy.

Parallel performance

The cost of the parallel implementation of the AMR algorithms can be measured using the time required to communicate the adjacent face and edge topology data between processors. Figure 25 shows the initial and refined meshes used to test the parallel performance of the algorithms on 8 processors. Figure 26 shows a breakdown of the refinement and associated communication costs measured in wall clock time. Generally, the communication cost was about 10% of the total refinement cost; the reader should note that the total solution time was about 60 secs, which implies that the communication costs associated with the adaptive mesh refinement were about 1% of the total solution time.

Multi-mechanics simulations

To demonstrate the combination of adaptive mesh refinement and multimechanics simulations in Diablo, consider the problem geometry shown in Figure 27. A square plate with a central square hole is subjected to both mechanical (uniform horizontal tractions on the outside vertical faces) and thermal (point heat fluxes located at various locations on the outside vertical faces, zero heat flux elsewhere on the outside faces and a fixed temperature on the inside surfaces) loads. Figure 28 shows the sequence of adapted meshes generated when using Diablo to solve this problem; Figure 29 and Figure 30 show the computed distributions of mechanical stress and thermal gradients on the finest of these discretizations.

The AMR scheme described in this report is capable of capturing the localized gradients present in both the solid and thermal mechanics. Local refinement is evident near both the reentrant corner and the locations of the point heat fluxes.

Exit plan

The customers for the AMR capability created by this project are the Engineering analysts who will use the solid mechanics simulation code. As the AMR capabilities are applied to a wider range of problems, it is expected that funding to support the inevitable future code development will be forthcoming from the Engineering programs that benefit from the inherent efficiencies of AMR. These capabilities uniting AMR and unstructured finite element simulations for parallel computers will extend the state-of-the-art for production capabilities and may generate work-for-other funding opportunities from organizations such as DoD.

Publications are currently being prepared for submission to peer-reviewed academic journals (such as IJNME, CMAME and Computational Mechanics) that fully document the work described in this report. Tentative titles for these works are: “AMR strategies for unstructured multimechanics simulations,” “Parallel performance of an unstructured AMR framework” and “Effective error estimation for nonlinear structural mechanics.” Abstracts and presentations have already appeared at

the following conferences: Seventh USNCCM, July 2003; DOE Trilab Engineering Conference, October 2003; NECDC, October 2004; Eighth USNCCM, July 2005; DOE Trilab Engineering Conference, October 2005.

Appendix A - References

[1] I. Babuska and W. C. Rheinboldt, 1979. Adaptive approaches and reliability estimates in finite element analysis. Computer Methods in Applied Mechanics and Engineering, 17, 519-540.

[2] C. Johnson, 1987. Numerical solution of partial differential equations by the finite element method. Cambridge University Press.

[3] O. C. Zienkiewicz and J. Z. Zhu, 1987. A simple error estimator and adaptive procedure for practical engineering analysis. International Journal for Numerical Methods in Engineering, 24, 337-357.

[4] M. Ainsworth and J. T. Oden, 2000. A posteriori error estimation in finite element analysis. Wiley Interscience.

[5] P. Ladeveze and J. T. Oden (editors), 1998. Advances in adaptive computational methods in mechanics. Elsevier.

[6] M. A. Puso and T. A. Laursen, 2002. A 3D contact smoothing method using Gregory patches. International Journal for Numerical Methods in Engineering, 54, 1161-1194.

[7] G. Booch, J. Rumbaugh and I. Jacobson. The Unified Modeling Language User Guide. Addison-Wesley, 1999.

[8] J. Z. Zhu and O. C. Zienkiewicz, 1990. Superconvergence recovery technique and a posteriori error estimators. International Journal for Numerical Methods in Engineering, 30, 1321-1339.

Appendix B – Figures

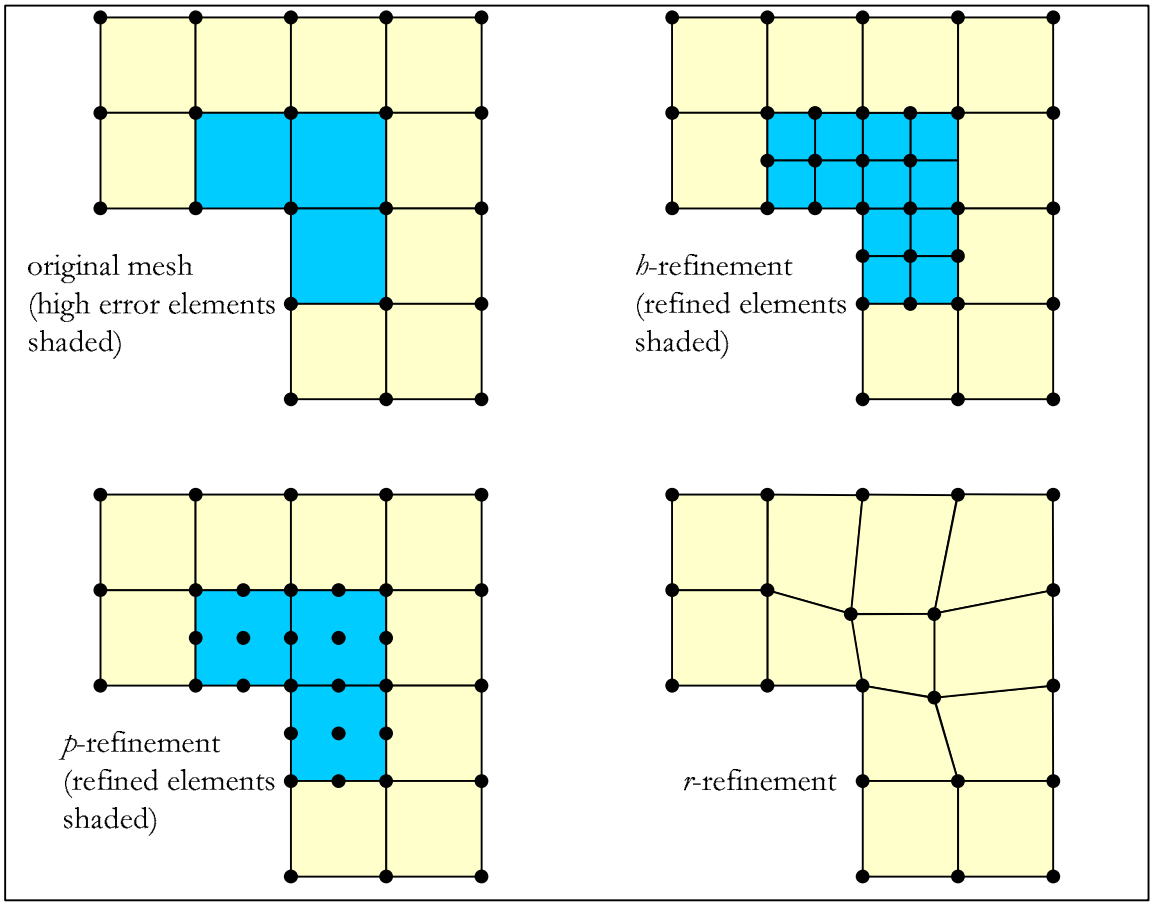

Figure 1. Adaptive mesh refinement schemes.

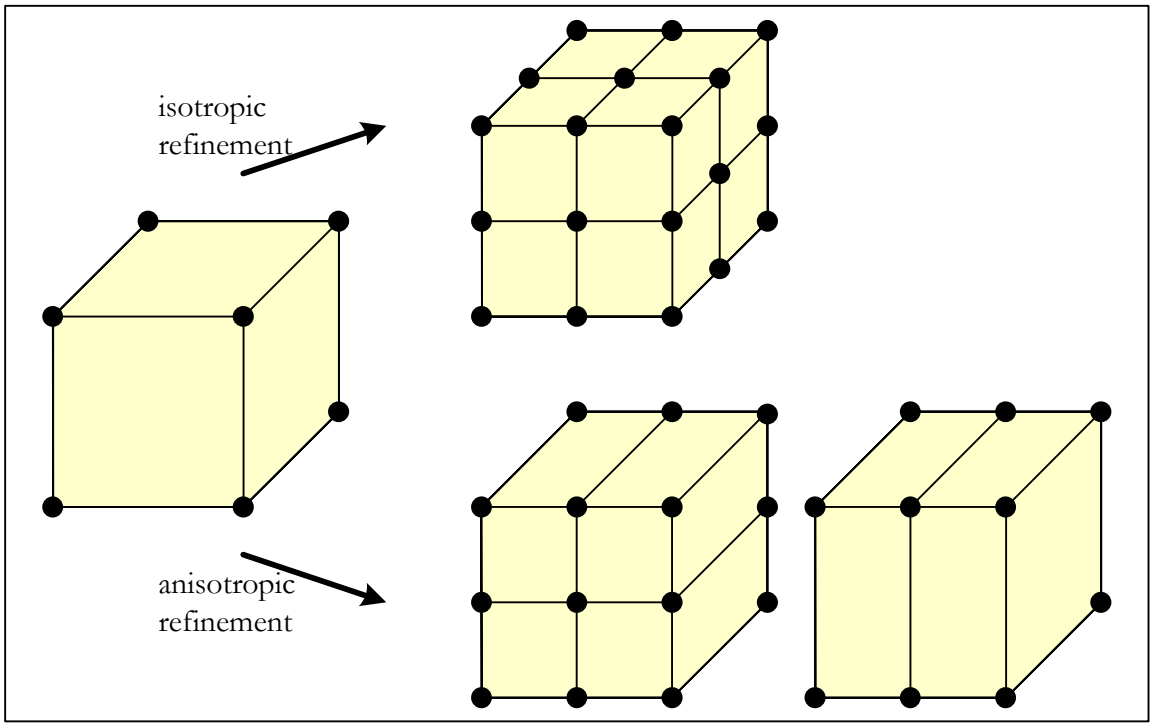

Figure 2. Isotropic and anisotropic refinement of hexahedral elements.

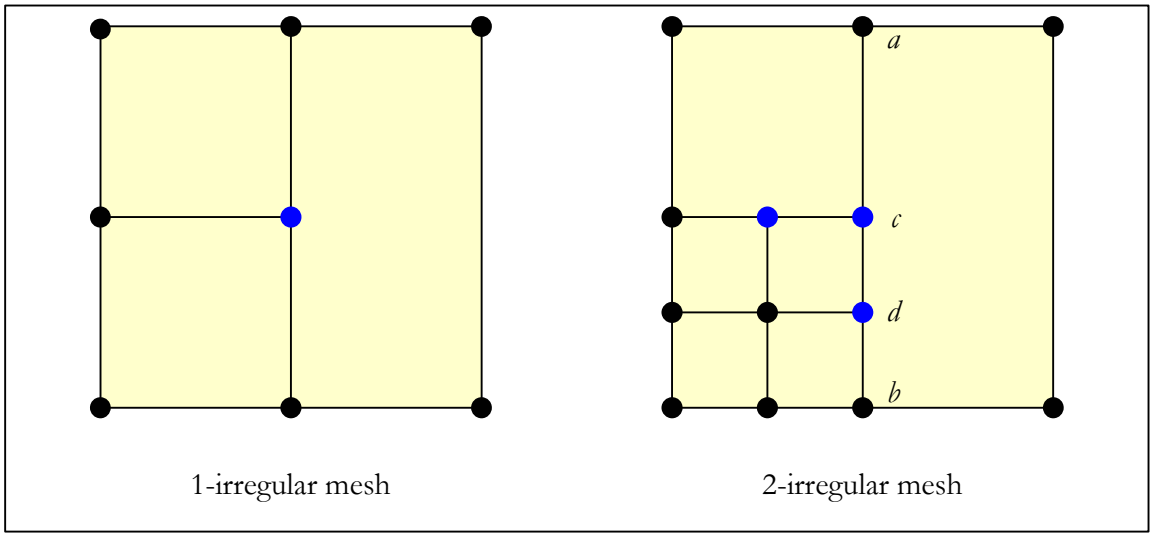

Figure 3. Generation of hanging nodes in irregular meshes.

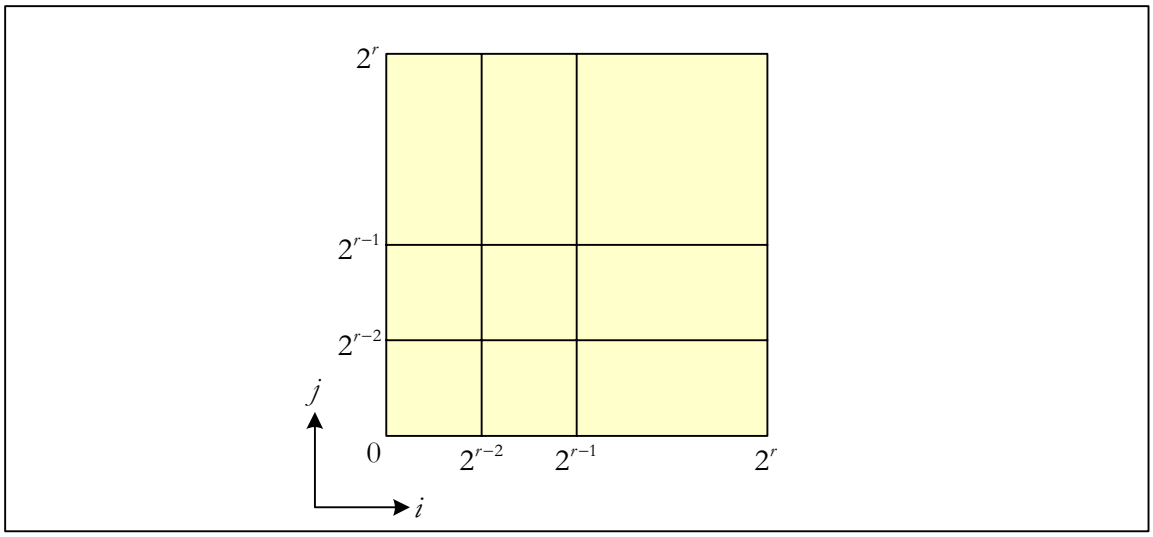

Figure 4. Index space used to track location of nodes.

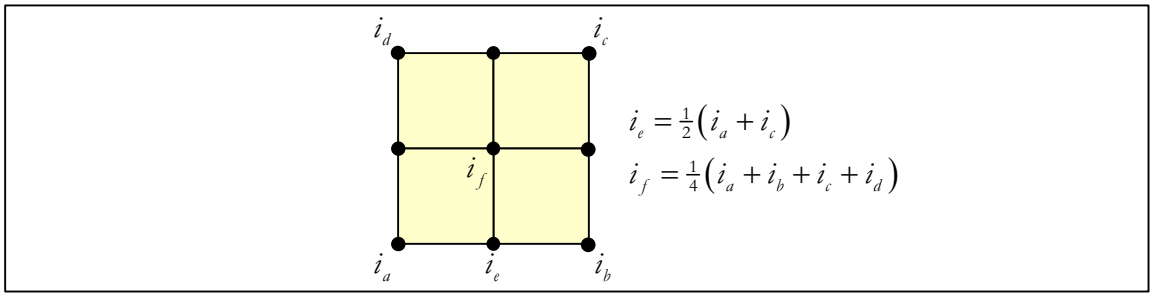

Figure 5. Computation of node indexes for refined elements.

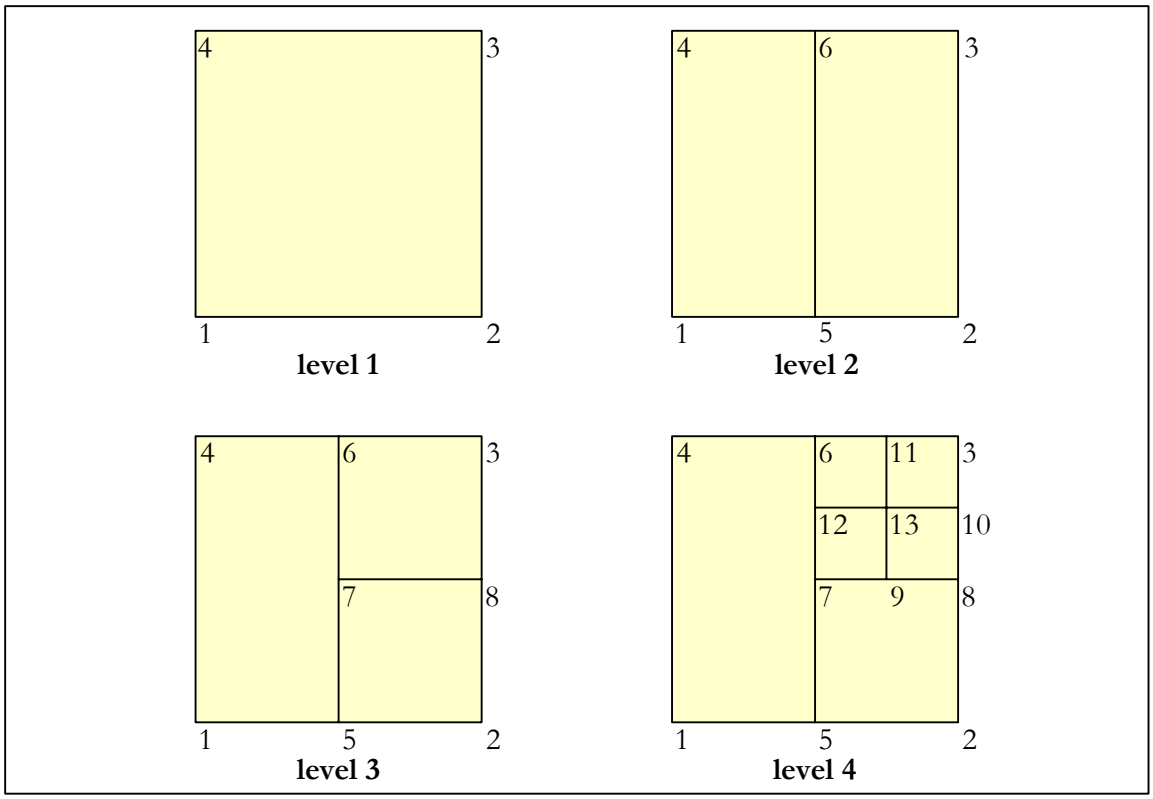

Figure 6. Two dimensional refined meshes to demonstrate node index tree.

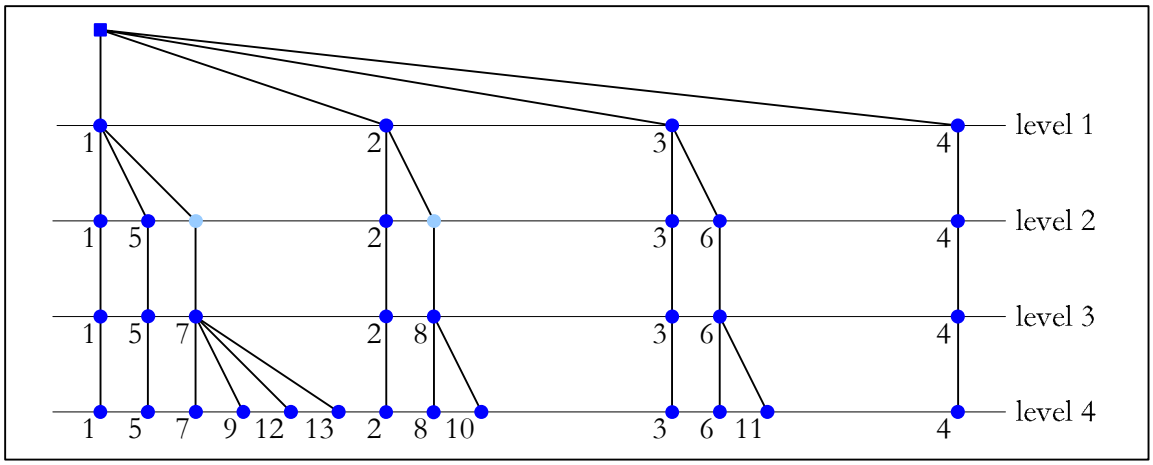

Figure 7. Node index tree for the sequence of refined meshes.

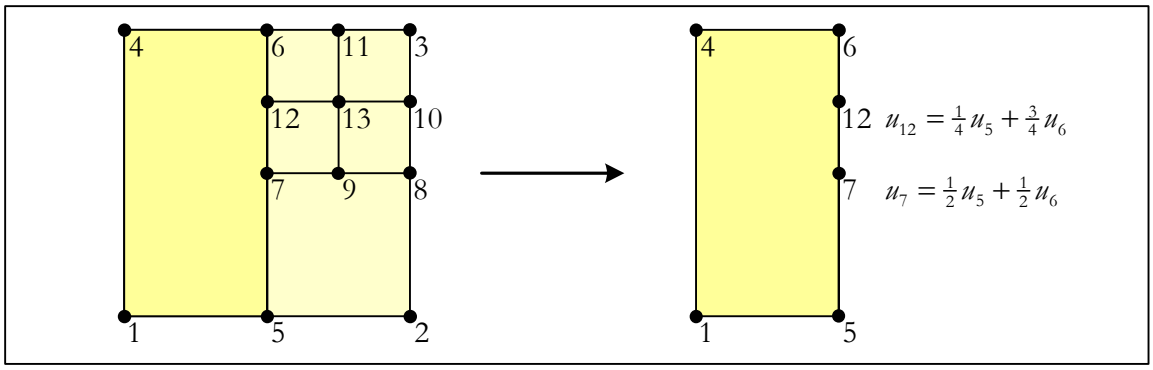

Figure 8. Application of constraints to the hanging nodes.

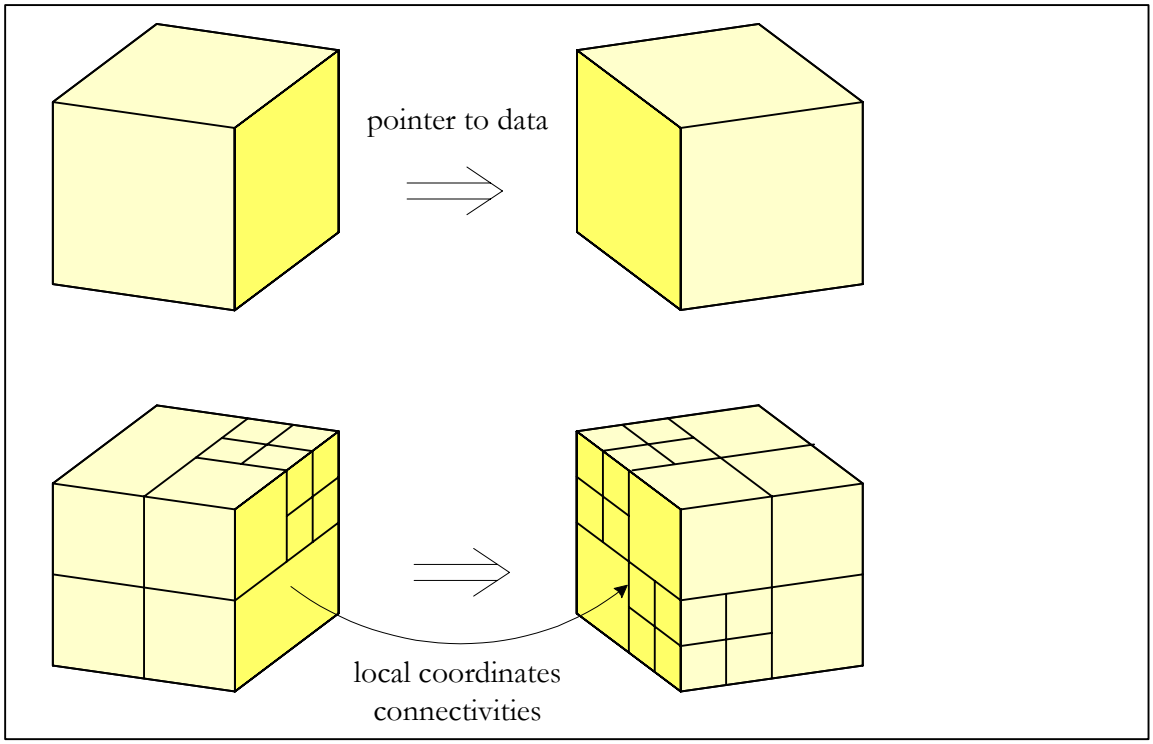

Figure 9. Serial data structures for application of adjacent block constraints.

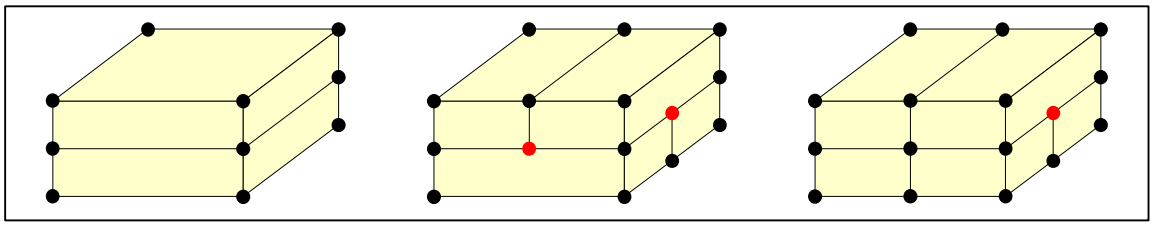

Figure 10. Anisotropic refinement producing incompatible fields.

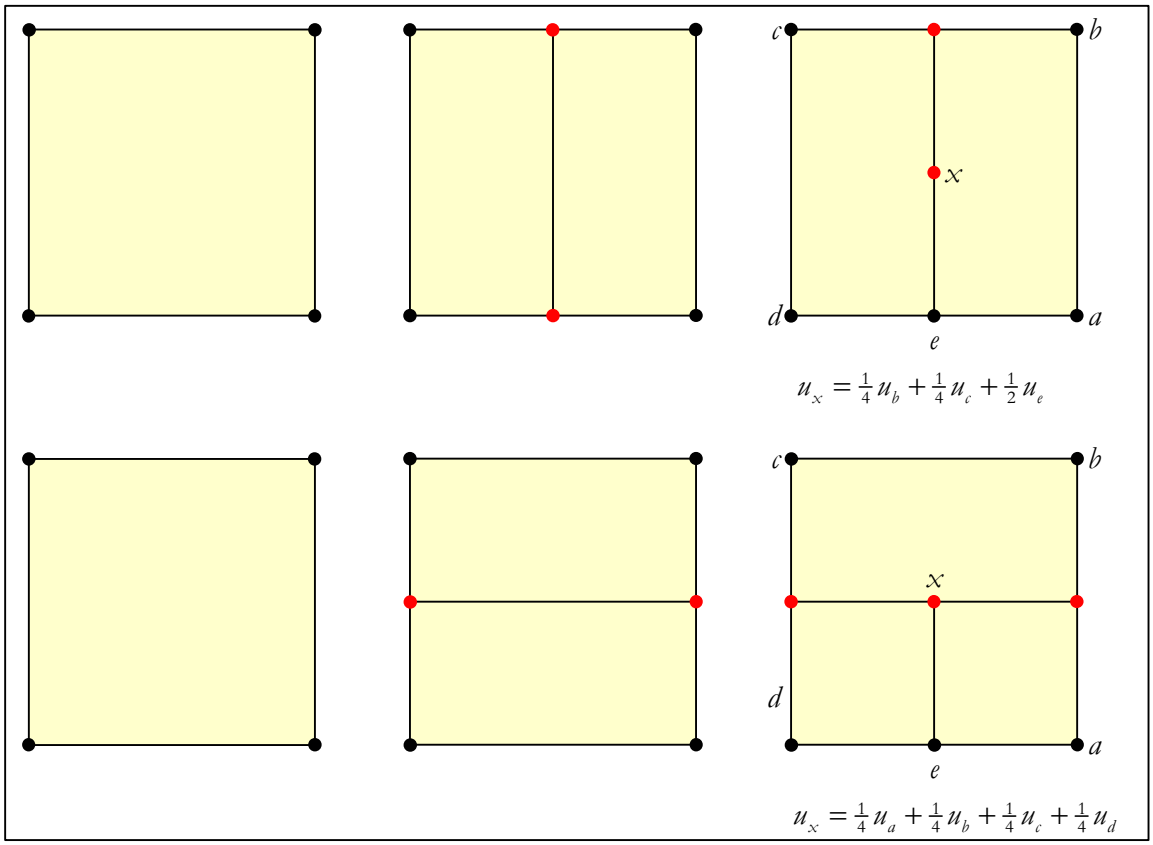

Figure 11. Incompatible fields across block faces.

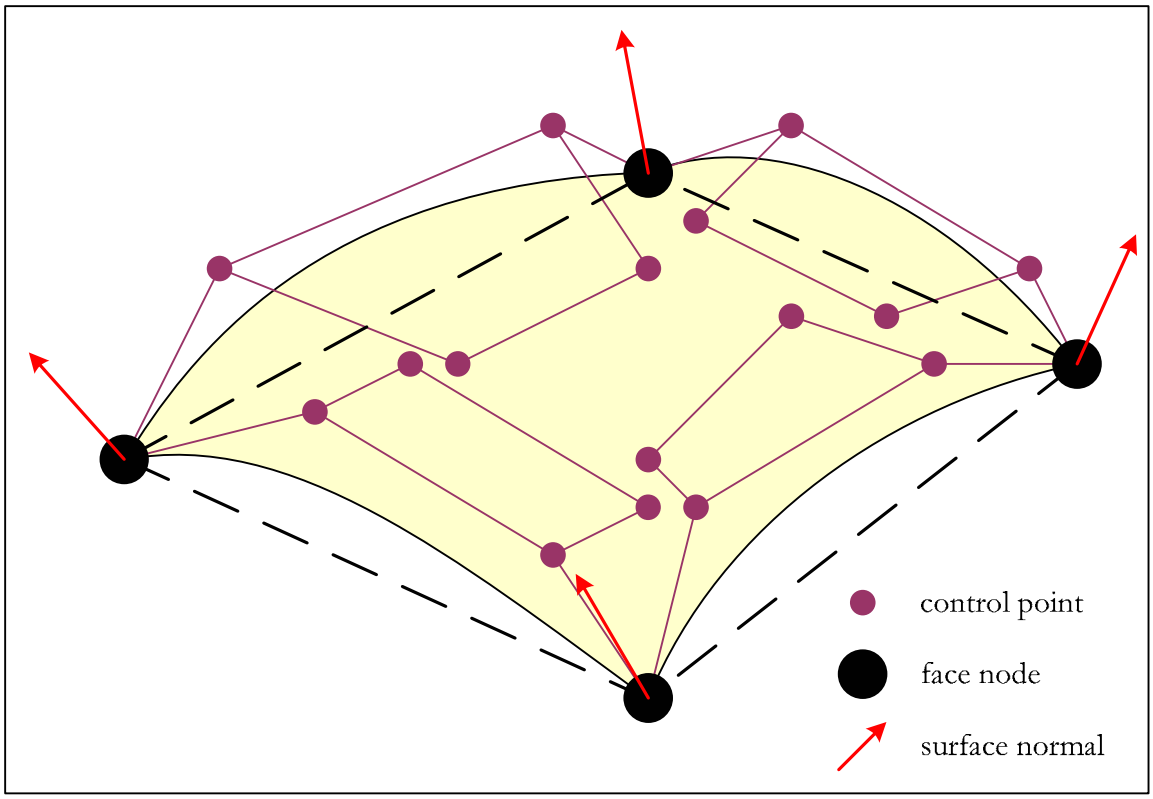

Figure 12. Gregory patch for capturing non-planar block faces.

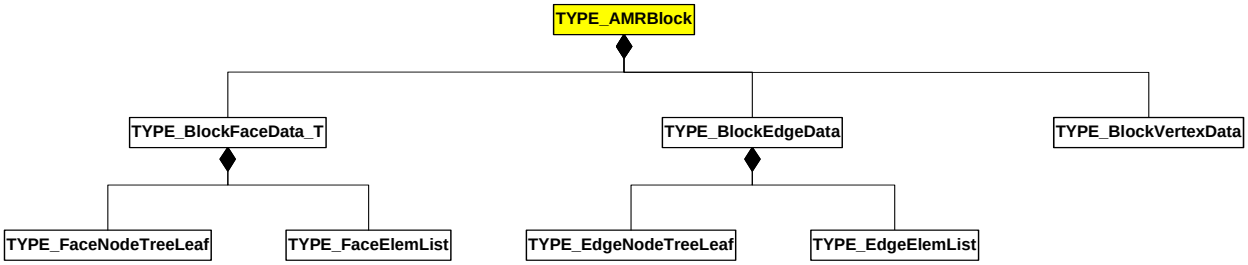

Figure 13. UML class diagram for AMR computations.

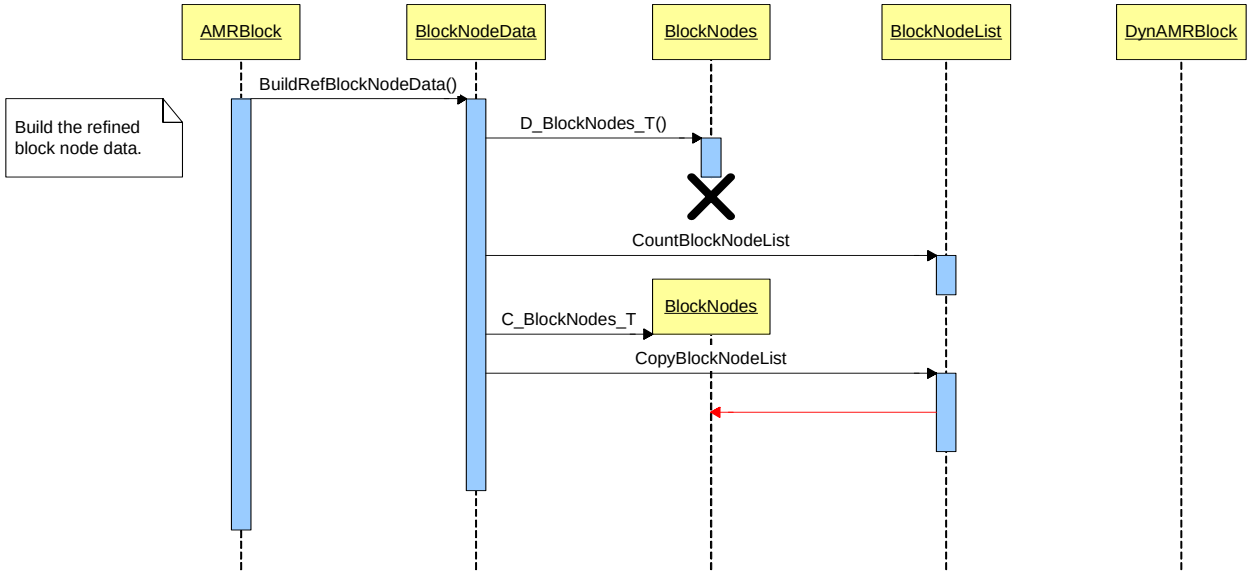

Figure 14. UML sequence diagram for AMR computations.

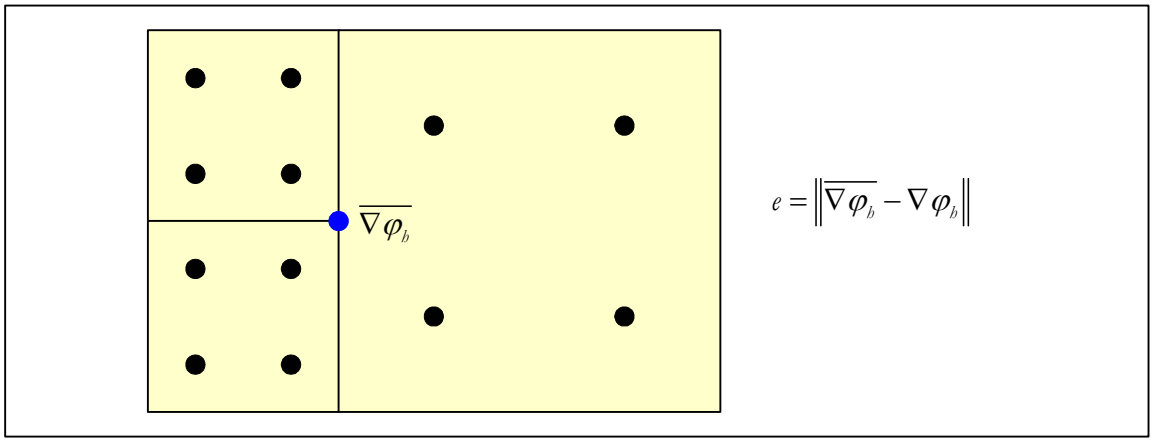

Figure 15. Element patch used to compute smoothed gradients.

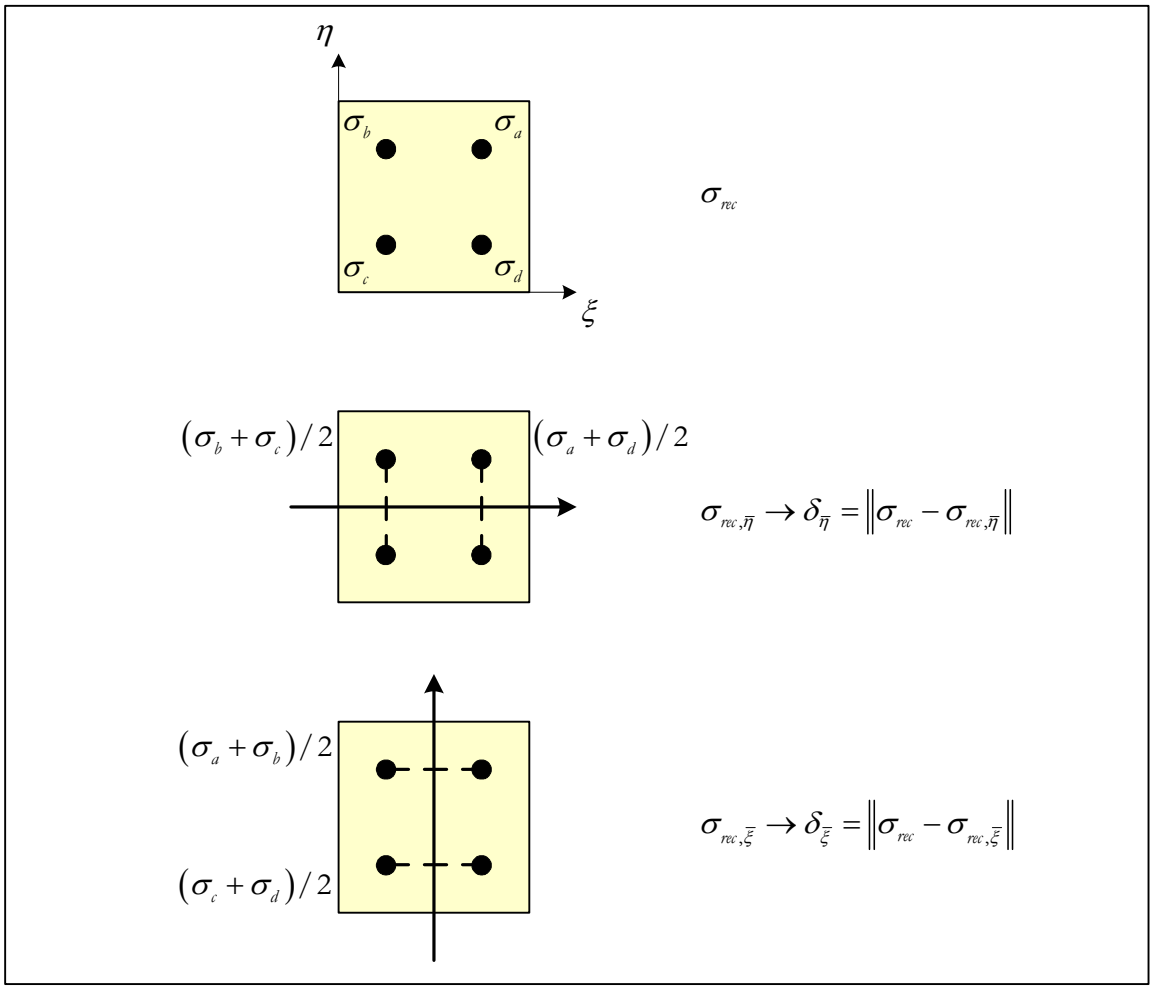

Figure 16. Directional averaging for anisotropic error estimators.

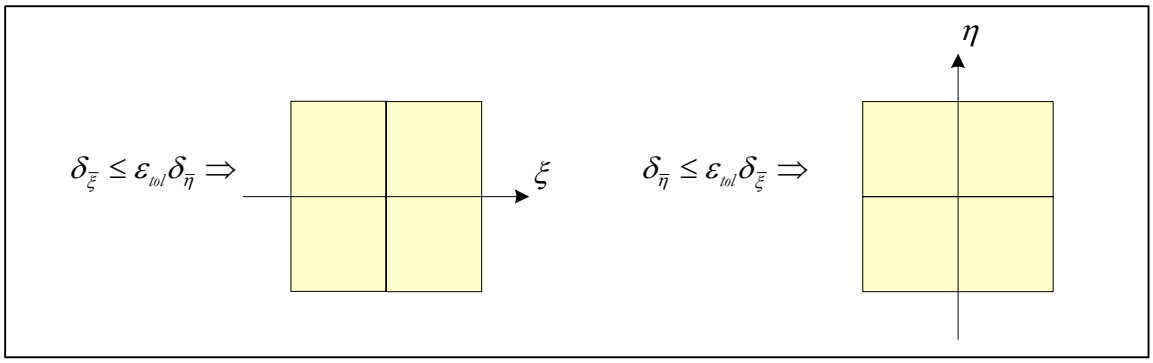

Figure 17. Anisotropic refinement criteria.

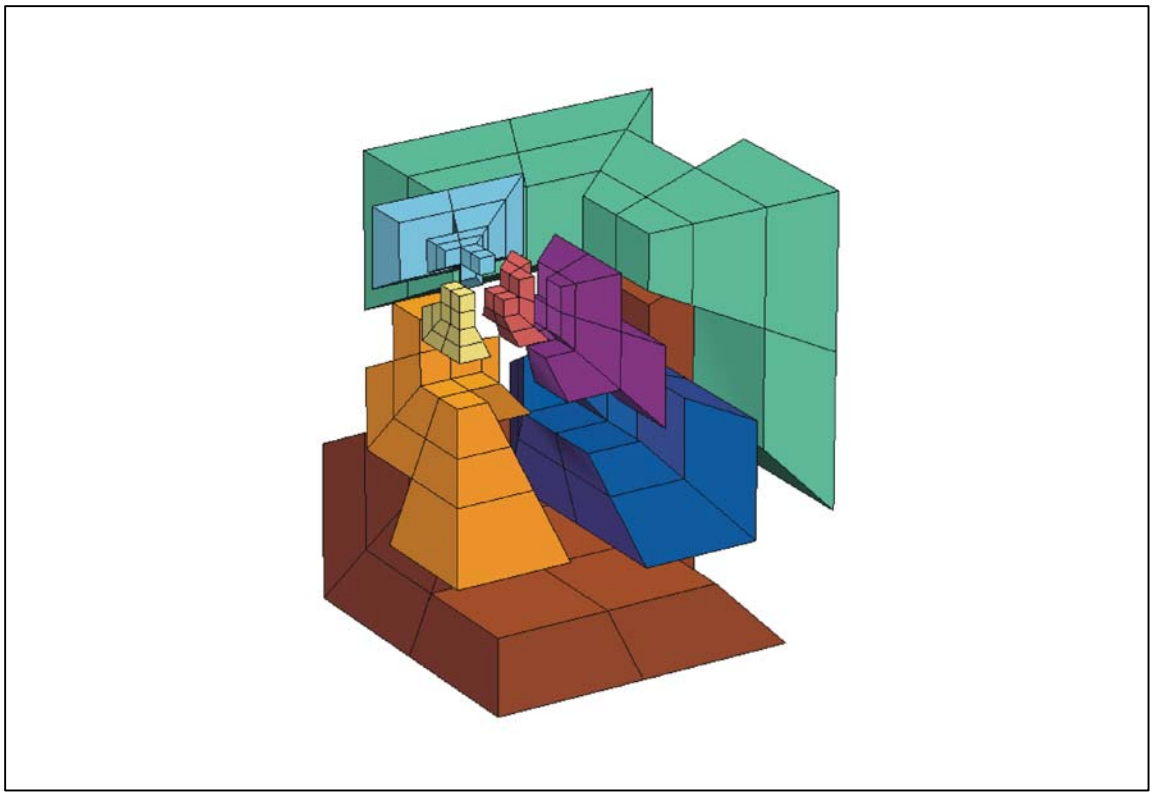

Figure 18. Parallel decomposition of an unrefined mesh.

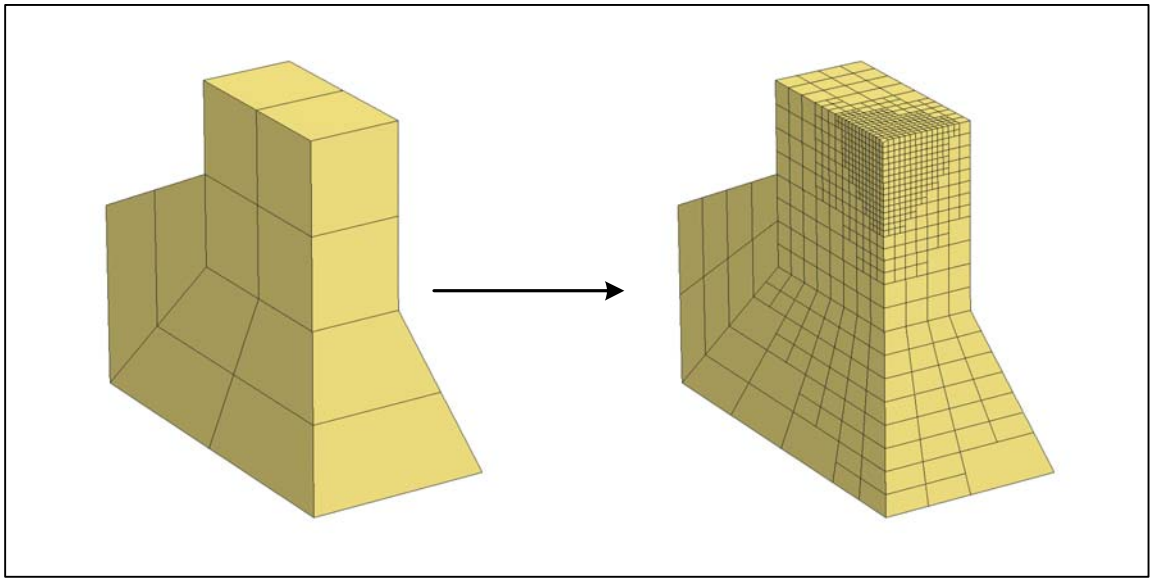

Figure 19. Local refinement of blocks assigned to a single processor.

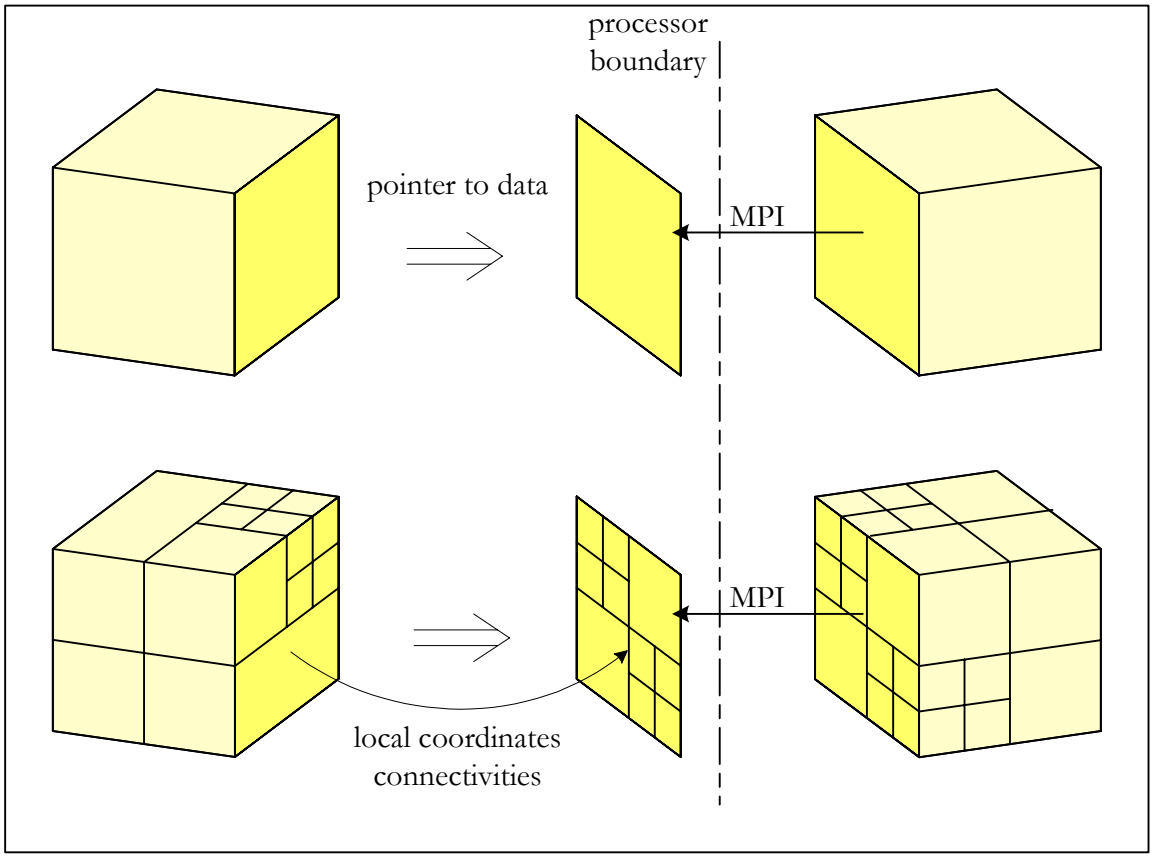

Figure 20. Parallel data structures for application of adjacent block constraints.

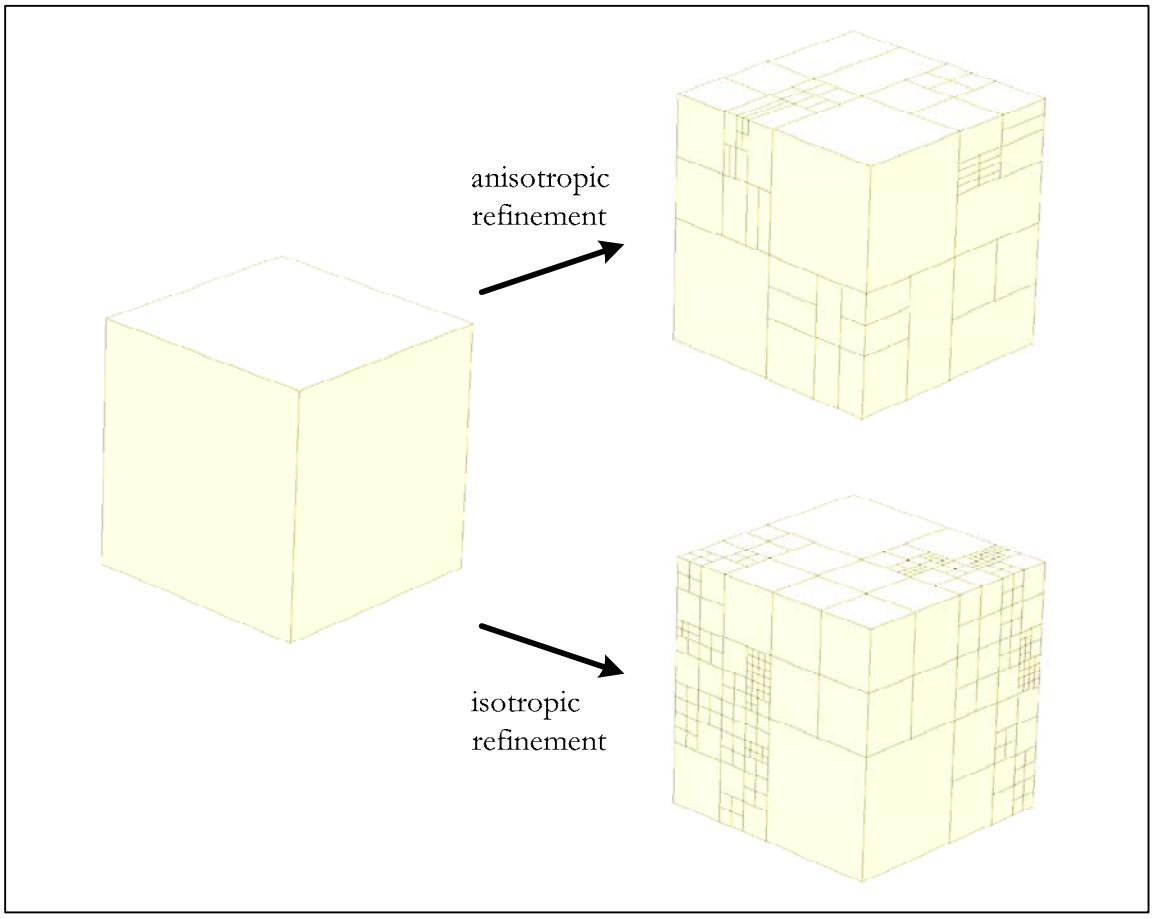

Figure 21. Patch test and random refinement for code verification.

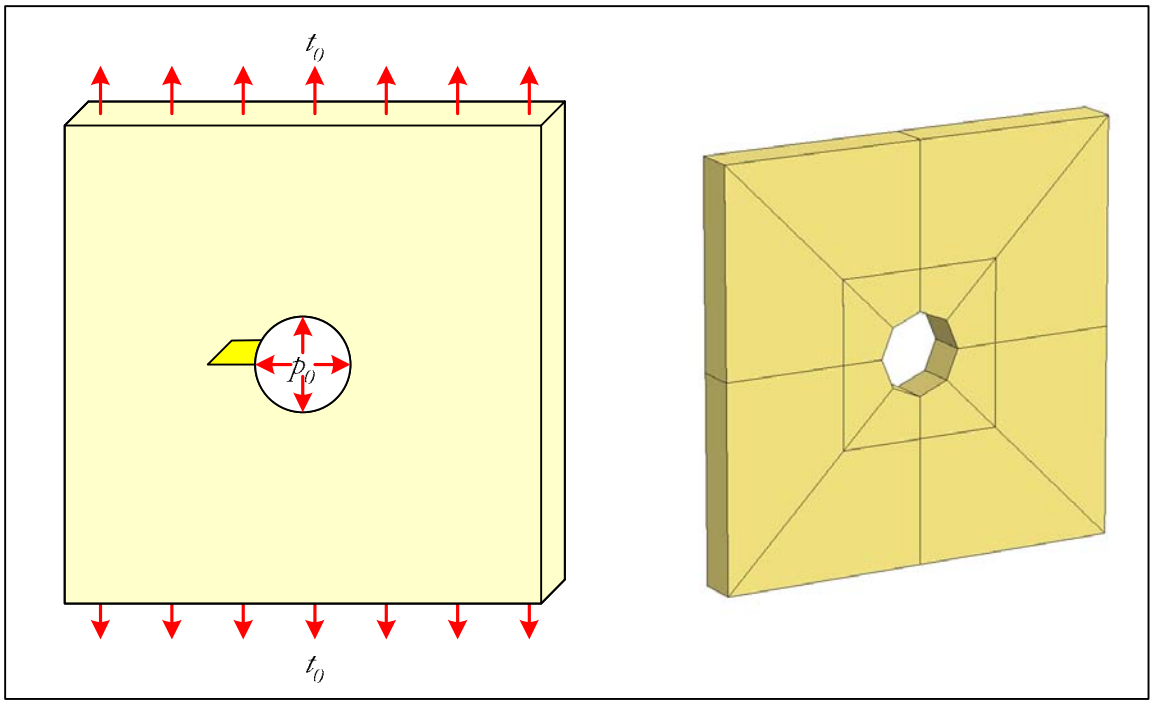

Figure 22. Cracked plate with hole.

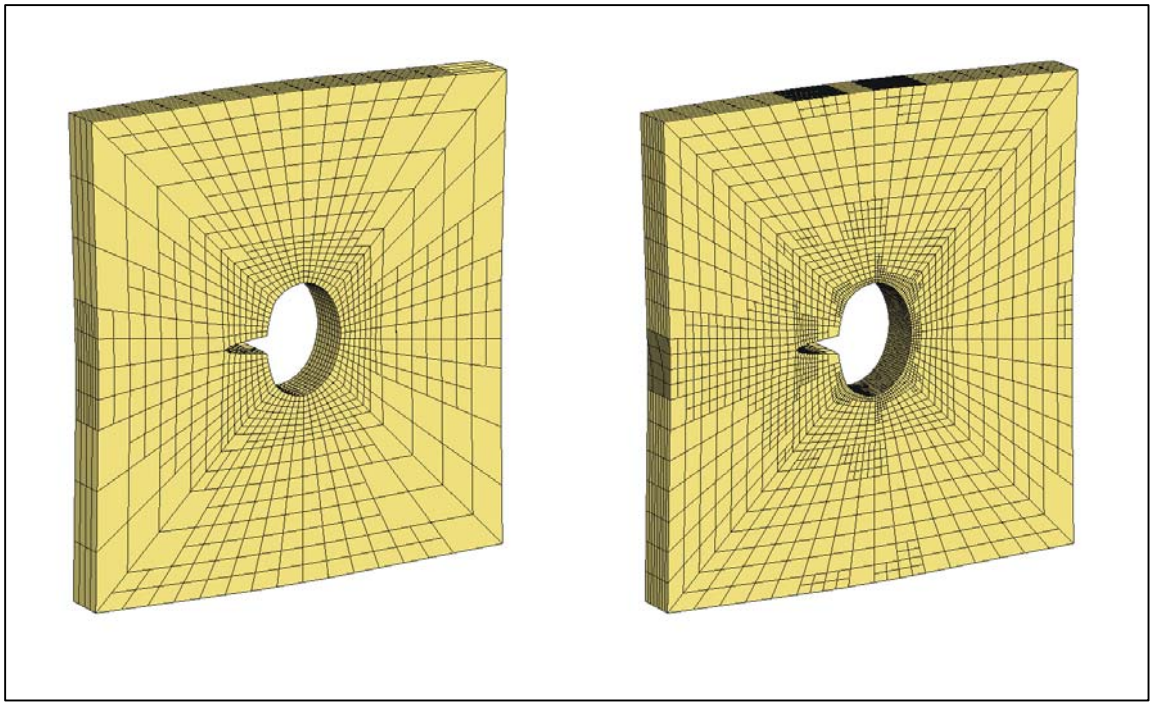

Figure 23. Refined meshes with non-planar deforming boundaries.

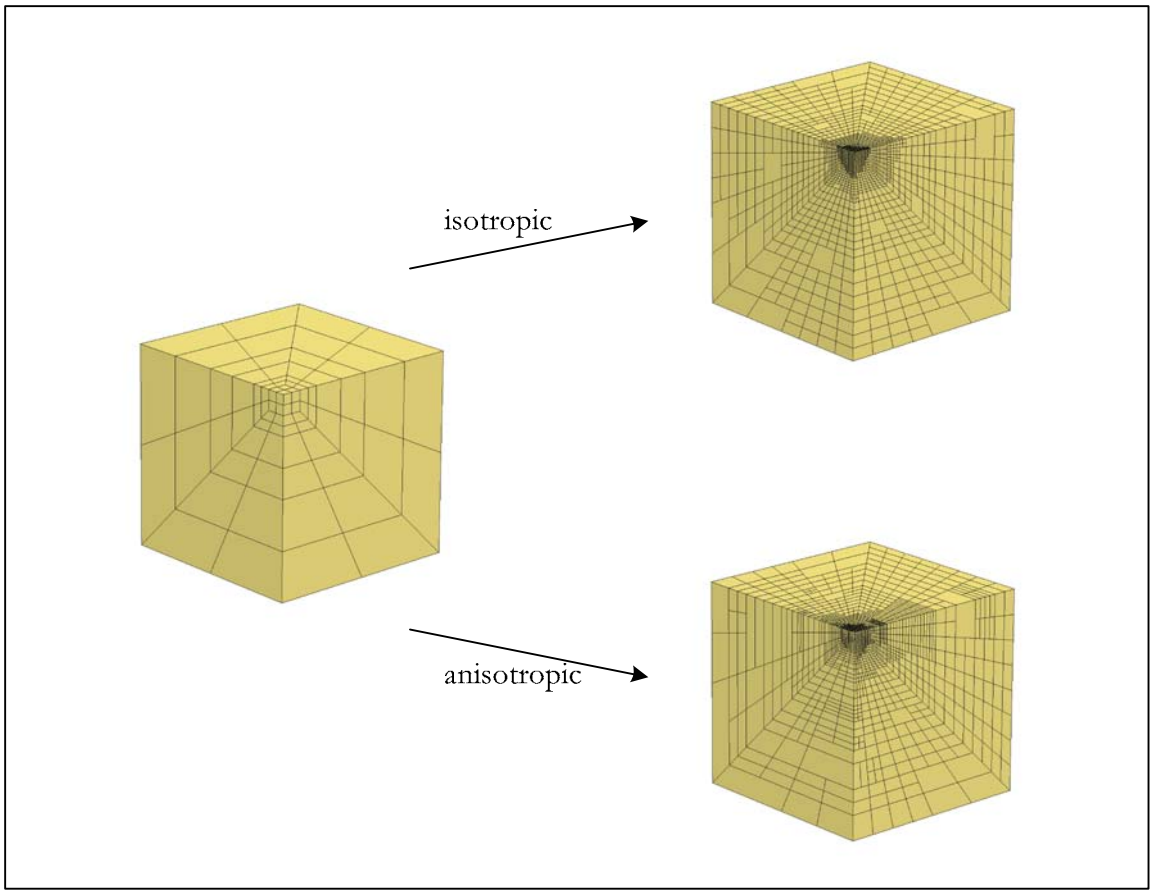

Figure 24. Isotropic and anisotropic refined meshes.

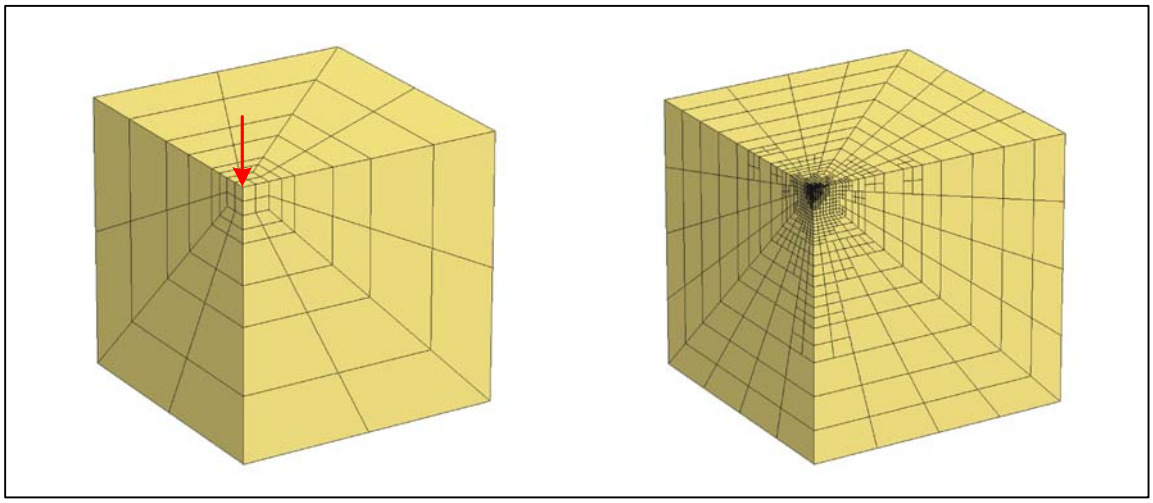

Figure 25. Parallel refined meshes.

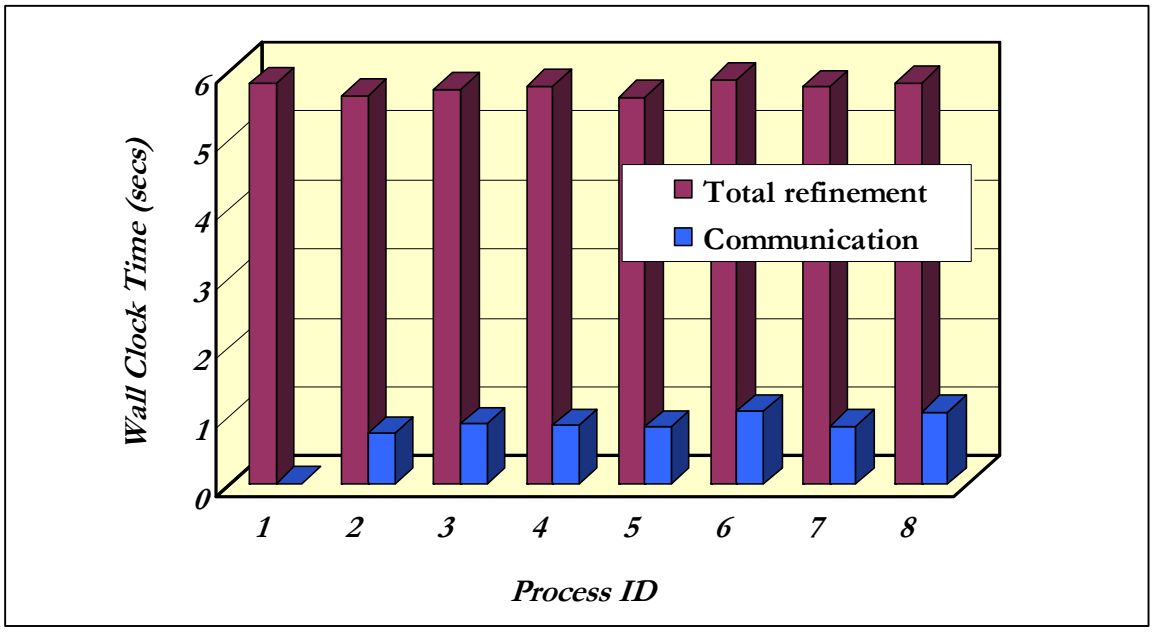

Figure 26. Parallel cost of the AMR algorithm.

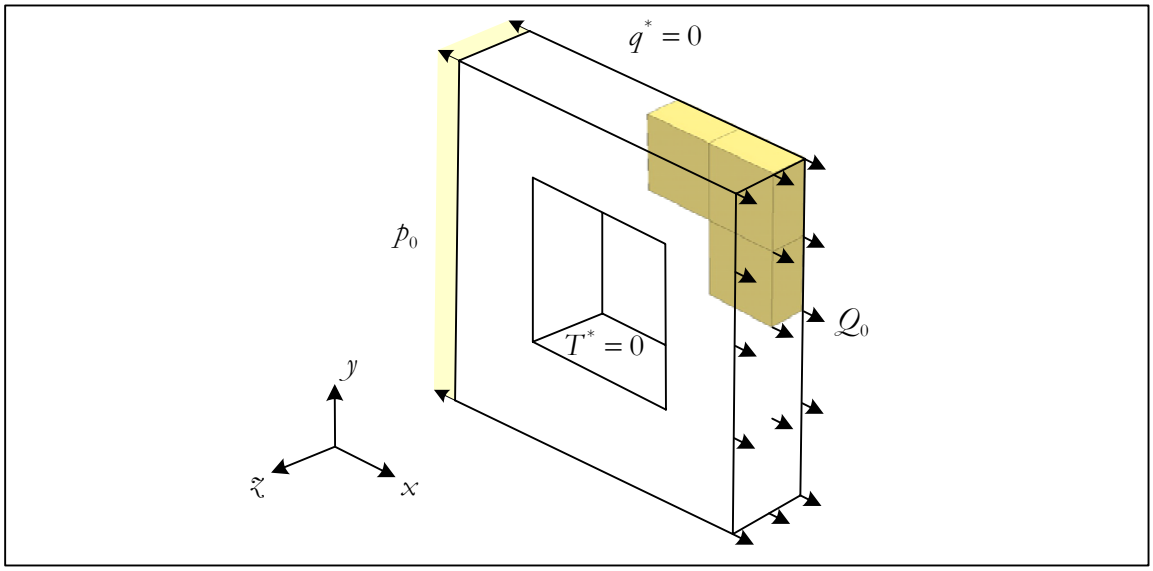

Figure 27. Multi-mechanics perforated plate problem.

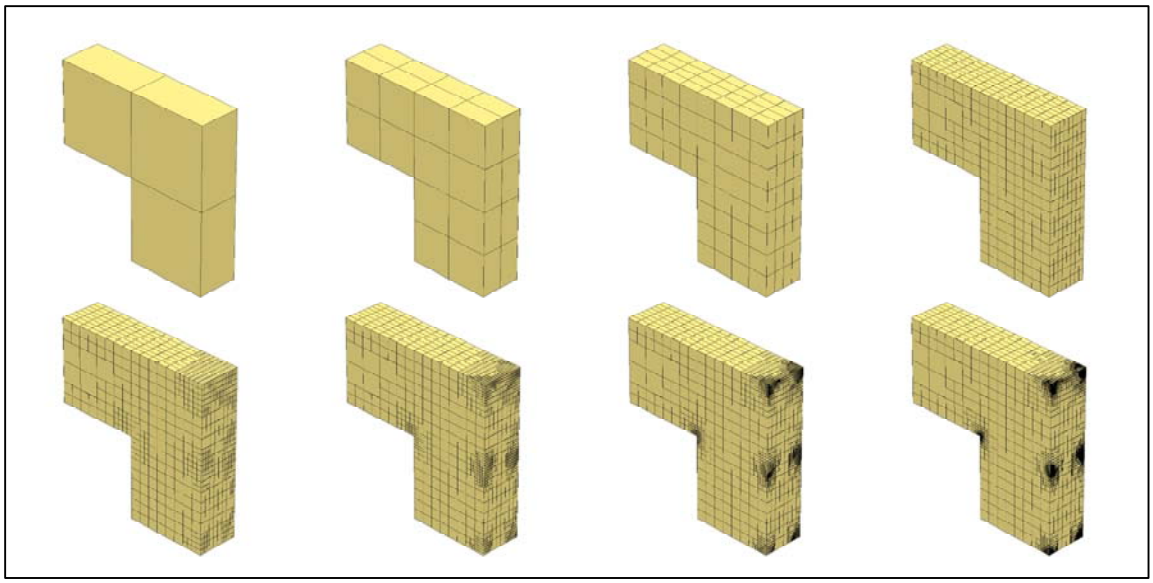

Figure 28. Refined meshes for the perforated plate problem.

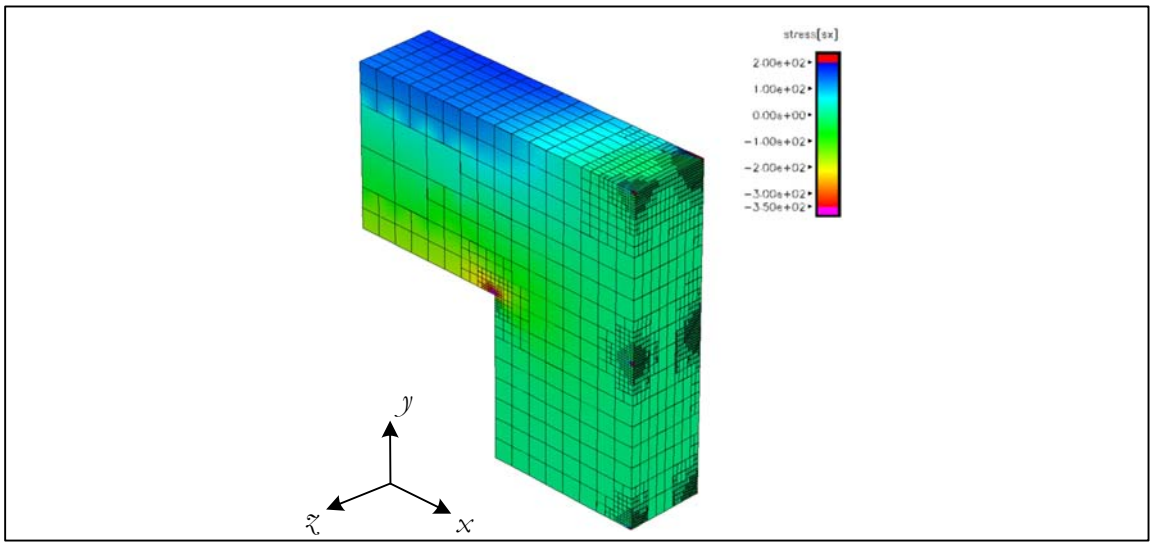

Figure 29. Distribution of σ_{xx} stress in the perforated plate.

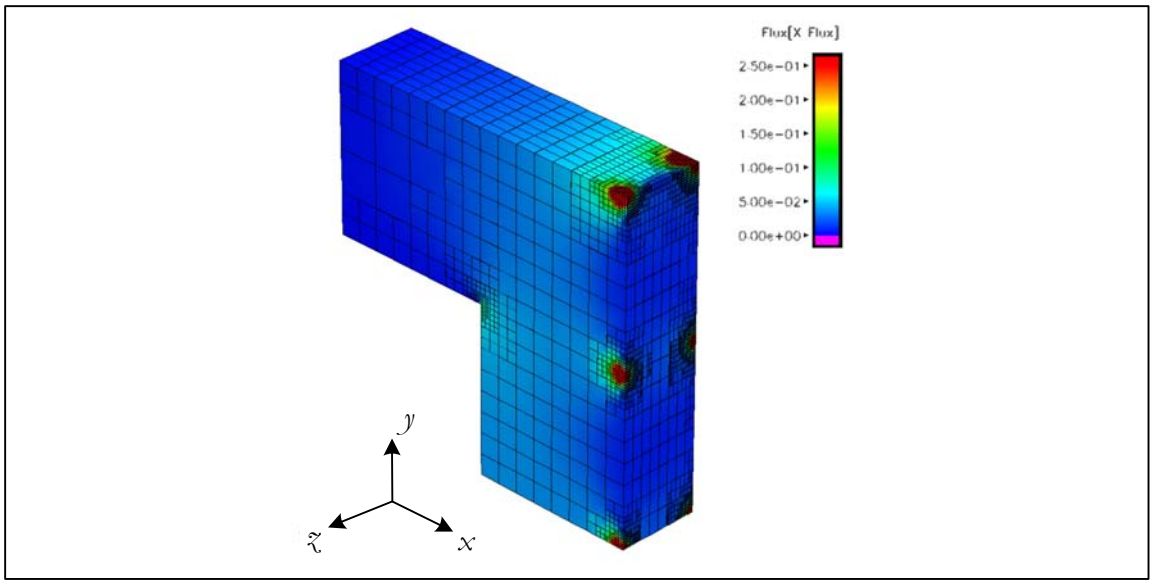

Figure 30. Distribution of heat flux in the perforated plate.