Open access • Journal Article • DOI:10.1002/NME.1008

Object-oriented hierarchical mesh refinement with CHARMS — Source link

Petr Krysl, Abhishek Trivedi, Baozhi Zhu

Institutions: University of California, San Diego

Published on: 28 Jun 2004 - International Journal for Numerical Methods in Engineering (John Wiley & Sons, Ltd.)

Topics: Object-oriented design, Subdivision surface, Polygon mesh, Software framework and Finite element method

Object-Oriented Hierarchical Mesh Refinement with CHARMS

Petr Krysl,*, Abhishek Trivedi, Baozhi Zhu

University of California, San Diego,
9500 Gilman Dr, La Jolla, CA 92093-0085,
pkrysl@ucsd.edu

KEY WORDS: Finite element, mesh refinement, hierarchical, adaptive approximation, object-oriented design

SUMMARY

A new approach to the construction of adaptive approximations on finite element meshes and also in more general settings, including approximations on subdivision surfaces, had been recently proposed by Krysl, Grinspun, and Schröder. This paper outlines how the general refinement algorithms may be specialized to some common finite element discretizations in two and three dimensions, discusses the design of nested meshes, and delves into the algorithmic, design, and implementation issues in an object oriented software framework (http://hogwarts.ucsd.edu/~pkrysl/CHARMS).

INTRODUCTION

Finite element approximations construct basis functions from piecewise polynomial functions defined on a mesh. The resolution of the basis function set may be controlled by increasing or decreasing the characteristic dimensions of the mesh elements, in other words by adjusting the mesh density.

In this paper we describe an implementation of the so-called mesh refinement which varies the density of the mesh by performing local *refinement (or coarsening – unrefinement)* of the existing mesh so that in some regions existing finite elements are split to decrease their "size", in other regions they are merged to reduce the resolution. This is in contrast to remeshing, which builds a new mesh in the whole domain or locally. Here we assume that mesh *refinement* had been adopted as the method of choice, but both methods have their advantages and disadvantages.

The element-centric viewpoint that evolved over the years in the finite element community resulted in the adoption of the geometric division of the individual finite elements as the starting point of all current mesh refinement algorithms. However, refinement based on the element-by-element splitting does not in general ensure global *compatibility* of the refined mesh, and a number of approaches are in use trying to resolve this problem: (i) the unknowns of the incompatibly placed nodes are constrained with respect to other nodes so that

compatibility of the resulting approximation is ensured although the mesh itself remains incompatible; (ii) compatibility is weakly enforced with Lagrangian multipliers or penalty methods; (iii) compatibility of the mesh is achieved by splitting additional elements until the mesh becomes globally compatible. A number of specialized refinement schemes have been proposed for a variety of practically important meshes: triangular and quadrilateral meshes in two dimensions [1, 2, 3, 4], tetrahedral [5, 6, 7, 8, 9, 10], or hexahedral meshes in three dimensions [11]. In general, existing mesh refinement technology tends to be quite complex (constraint methods, splitting of neighboring elements), or leads to unappealing algorithmic properties (Lagrange multipliers, penalty methods).

A general, flexible, and robust mesh refinement algorithm had been recently proposed by Krysl, Grinspun, and Schröder [12] and Grinspun, Krysl, and Schröder [13]. (These algorithms will be refered to here by the acronym CHARMS, which stands for Conforming, Hiearchical, Adaptive Refinement MethodS.) It constructs a compatible refined approximation by drawing on the properties of the *refinement equation* (well-known in wavelet analysis and elsewhere), and has a number of very attractive properties:

- The same approach applies in any number of dimensions, and in particular algorithms on one-dimensional meshes are of the same complexity as algorithms for three- or four-dimensional meshes;
- The approximation order of the basis functions is immaterial, and the same algorithm may be applied to piecewise linear or piecewise cubic, or indeed on non-polynomial, approximations;
- The support of the individual basis functions may be the neighborhood consisting of all finite elements sharing a node, or it might be a wider neighborhood, for instance a two-ring of triangles on surfaces supporting the so-called subdivision elements [13], to which current mesh refinement techniques cannot be applied.

This paper has the following goals: We wish to describe how the CHARMS may be implemented for some classical finite element approximations, such as those on triangular or quadrilateral meshes in two dimensions, or on hexahedral meshes in three dimensions. For simplicity, we apply our adaptive system to a simple partial differential equation of steady diffusion. We discuss an object-oriented implementation in the C++ language, including the details of some of the most important (and novel or unusual) algorithms manipulating the CHARMS datastructures. Mesh refinement should ideally guarantee efficiency, good mesh quality, and robustness. An important benefit follows if the refined mesh is nested in the coarser levels, since that property directly supports efficient multilevel solvers [14, 15]. We show how our algorithms meets all of the above requirements.

1. CHARMS

We start by considering a set of linearly independent scalar basis functions $\mathcal{B} = \{\phi_i(x)\}$, supported on the finite elements in the mesh M, which span the finite element space $\mathcal{X}$,

$$\mathcal{X} = \left\{ v(x) : v(x) = \sum_i \phi_i(x) v_i \right\} , \tag{1}$$

with $x \in \mathbb{R}^n$, $v(x) \in \mathbb{R}^m$, $v_i \in \mathbb{R}^m$ ($n \geq 1$, $m \geq 1$), and $\phi_i(x) \in \mathbb{R}$. We propose to create a new, "refined" mesh, M', such that the set of functions $\phi'_i(x)$ constructed on mesh M' constitutes a basis for the *finer* space $\mathcal{X}'$, such that $\mathcal{X}'$ is related to the *coarser* space $\mathcal{X}$ by inclusion:

$$\mathcal{X} \subset \mathcal{X}' \ . \tag{2}$$

We shall show in Section 2 how to construct appropriate meshes M', but here we shall assume that given a coarser mesh M with the associated basis functions $\phi_i(x)$ it is possible to construct a finer mesh M', and the associated basis functions $\phi'_i(x)$, such that the nesting relation (2) holds. By recursively extending the above argument, a *hierarchy* of approximation spaces $\mathcal{X}^{(j)}$ is obtained. The coarsest space in the hierarchy is $\mathcal{X}^{(0)}$, progressively finer spaces have increasing indices $\mathcal{X}^{(1)}$, $\mathcal{X}^{(2)}\ldots$, and this sequence may be either infinite or finite, resulting in the hierarchical nesting relation

$$\mathcal{X}^{(0)} \subset \mathcal{X}^{(1)} \subset \mathcal{X}^{(2)} \subset \ldots \subset \mathcal{X}^{(m)} \ . \tag{3}$$

1.1. REFINEMENT EQUATION

Relation (3) makes it possible to apply the *refinement* (also known as the *multiresolution* or *dilation*) equation [16] to finite element mesh refinement. Since $\mathcal{X}^{(j)} \subset \mathcal{X}^{(j+1)}$, any basis function $\phi_i^{(j)}(x)$ on level j may be exactly resolved (reproduced) in the basis with finer resolution $\{\phi_i^{(j+1)}(x)\}$:

$$\phi_i^{(j)}(x) = \sum_k \beta_{ik}^{(j+1)} \phi_k^{(j+1)}(x) \ , \tag{4}$$

where $\beta_{ik}^{(j+1)} \neq 0$ are the coefficients of the linear combination. We will usually attempt to construct the spaces $\mathcal{X}^{(j)}$ in such a way that the sum in (4) has a finite, and fairly small, number of terms.

The refinement equation (4) is the key to our adaptivity approach: Given an approximation space $\mathcal{X}^{(j)}$ with basis $\mathcal{B}^{(j)}$, one can derive a custom space $\mathcal{X}^{'}$, with basis $\mathcal{B}^{'}$ whose resolution is locally finer, by replacing one or more parent basis functions from $\mathcal{B}^{(j)}$ by children functions from $\mathcal{B}^{(j')}$ on levels $j' > j$. In this sense, the refinement equation may be viewed as two statements: Firstly, the *approximation properties* of the basis function set $\mathcal{B}^{'}$ preserve or enhance those of $\mathcal{B}^{(j)}$ if the children basis functions are substituted for the parent basis function. Secondly, the *continuity* of the children functions is greater than or equal to the continuity of the parent function (since any finite linear combination of C^k functions is C^k). Current mesh refinement techniques have to wrestle with continuity because the refinement operations are applied to isolated *pieces* of basis functions; CHARMS, on the other hand, achieves compatibility by construction.

2. CONSTRUCTION OF MESHES OF HIGHER RESOLUTION

In this section we will outline a strategy for constructing the hierarchy of meshes M^j that will lead to the nesting relation (2). We assume that the *initial* mesh consists of an arbitrary number of element types (for instance, triangles mixed together with quadrilaterals and tetrahedra in a non-manifold geometry), but is *compatible*. Furthermore, we shall consider only *nodal basis functions*, i.e. the basis functions are associated with nodes, and are composed of pieces defined

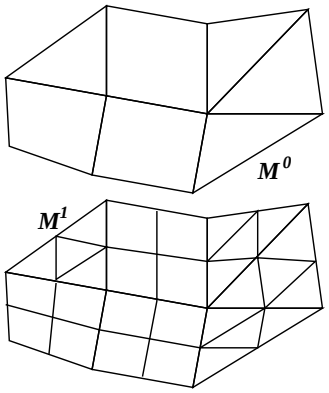

Figure 1. Uniform division of mesh M^0 that results in a compatible mesh M^1.

over the incident finite elements. On each level j of the nesting hierarchy, it is assumed that the basis functions $\phi_i^{(j)}$ verify the Kronecker delta property, i.e.,

$$\phi_i^{(j)}(x_k) = \delta_{ik} \ , \tag{5}$$

where x_k is the location of the node associated with function $\phi_k^{(j)}$. In other words, we assume that an approximation built of basis functions from any *single* level *interpolates* the nodal values.

Refining a basis function using Equation (4) will mean that basis functions with higher resolution need to be constructed from pieces over "finer" finite elements. Therefore, as one of the steps in our algorithm, we shall need to geometrically divide coarse finite elements into finer ones. Our mesh hierarchy will be built up in such a way that each element is divided into elements of the same type (triangles into triangles, quadrilaterals into quadrilaterals, etc.), and elements at each level will be divided using the same pattern (for instance, one triangle into four smaller triangles, no matter at which level in the hierarchy the parent triangle is).

Furthermore, we shall assume that refining uniformly one particular mesh in the mesh hierarchy results in a compatible mesh one level higher; compare with Figure 1. Therefore, if for instance two finite elements in the mesh M^j share an edge, we shall use a division pattern that divides the shared edge in the same way in both elements. The same principle applies to faces shared by three-dimensional elements, or in general to d-dimensional faces shared by $(d+1)$-dimensional elements. To specialize the above to some familiar element types: two-node line elements will be bisected, three-node triangles and four-node quadrilaterals will be quadrisected (which bisects their edges, making them compatible upon refinement with line elements attached along the edges of the quadrilateral or triangle), eight-node hexahedra will be octasected (again, their edges are bisected, and the divided faces are compatible with the refinement of a four-node quadrilateral).

The same principle may be applied unchanged in higher-dimensional elements of the Lagrange or Hermite type. For instance, if the basis function pieces over finite elements are constructed through the master element in the parametric space, which is then mapped to the physical space, the natural choice for refinement is uniform division in the parametric space. For instance, consider the quadratic 8-node quadrilateral element in Figure 2, where

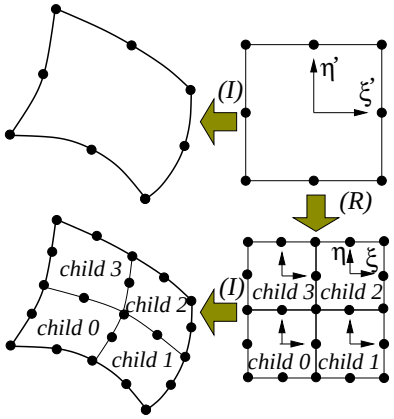

Figure 2. Refinement of 8-node quadrilateral finite elements: (I) stands for isoparametric mapping, (R) stands for refinement.

the isoparametric mappings are indicated by (I), and the refinement process is indicated by (R).

2.1. TRUE HIERARCHICAL BASIS

Equation (4) may be viewed also as a statement of equivalence: the left hand side (the coarse function) is equivalent to the right hand side (the set of finer functions) in the sense that if we combine both sides of (4) we get a linearly-dependent set. Furthermore, we may note that there is just one function too many in the combined set of the left and right hand side functions. Therefore, we could move all but one of the right hand side (finer) functions to the left hand side, obtaining

$$\phi_i^{(j)}(x) - \sum_{k\neq m} \beta_{ik}^{(j+1)} \phi_k^{(j+1)}(x) = \beta_{im}^{(j+1)} \phi_m^{(j+1)}(x) \ . \tag{6}$$

That is a statement of the same kind as before, an equivalence, but now the function that is being "reproduced" is on the finer level. Symmetry considerations dictate that the function to leave on the right hand side is the one whose node "stems" from the node of the coarse function. For instance, we show in Figure 3 the patch (one triangle and three quadrilaterals) that supports the function indicated by the filled circle (left). In the right hand part we show the refined patch, where the filled circles indicate functions on level $j+1$ on the left hand side of (6), and the empty circle stands for the single function on the right hand side, $\phi_m^{(j+1)}(x)$; in other words, the filled circles indicate the linearly independent functions, the empty circle the one finer function that is dependent on the rest. We shall call the functions on level $j+1$ that were moved to the left hand side of (6) the ***detail functions*** of $\phi_i^{(j)}(x)$.

The above discussion may suggest to the reader a way of constructing the refined approximation spaces: To refine a function from the coarse space $\mathcal{B}$, we shall add its detail functions to $\mathcal{B}$ to obtain $\mathcal{B}^{'}$. The resulting approximation will be dubbed the ***true hierarchical*** basis, and, indeed it is not a new concept in itself, since we get the well-known hierarchical

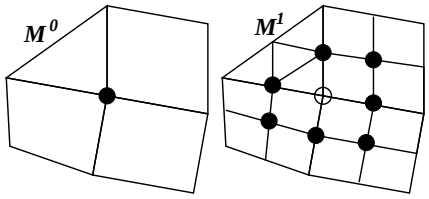

Figure 3. Illustration of Equation (6).

basis discussed, for example, by Yserentant [17]. However, our use of this concept for selective refinement is novel.

2.2. QUASI-HIERARCHICAL BASIS

To formalize somewhat, we shall use the term ***active function*** for functions selected from a given basis set, and the symbol $\widehat{\mathcal{B}}^{(k)}$ will be used for the set of active functions from basis set $\mathcal{B}^{(k)}$. Now, the approximation basis function set $\mathcal{B}^{'}$ may be written as

$$\mathcal{B}^{'} = \bigcup_{j=0}^{\infty} \widehat{\mathcal{B}}^{(j)} \ , \tag{7}$$

that is as a union of active sets from all mesh levels. For the true hierarchical basis described above, the active set on level $j = 0$ includes all the basis functions defined on the initial mesh, and each active set on level $j > 0$ includes only the detail functions. Now we shall describe an alternative refinement strategy, the quasi-hierarchical basis.

As before we proceed from the refinement equation (4). In difference to the true hierarchical approximation, we shall interpret the refinement equation as a recipe for *replacing* coarse functions (left hand side) by "finer" (higher-resolution) functions (right hand side). Thus, the refinement will delete (deactivate) coarse functions, replacing them in a lossless manner by finer functions, and unrefinement will delete (deactivate) fine functions and activate coarse functions. Clearly, if this type of refinement is applied globally, all coarse functions are replaced by fine functions and the interpolating partition of unity form of finite element approximation is recovered; on the other hand, if the refinement is graded, transition needs to be made from coarse to fine functions, and in the transition regions the resulting approximation resembles the true hierarchical basis in that the basis function do not necessarily add up to unity. This is illustrated in Figure 4, where on the left we show the true hierarchical basis, compared to the quasi-hierarchical basis on the right. Even though we replace coarse functions instead of augmenting them with finer functions, the resulting approximation basis is still written in the form of Equation (7), the only change being that the active sets $\widehat{\mathcal{B}}^{(j)}$ may now include not only the detail functions, but all the functions from the right hand side of the refinement equation (4).

2.3. APPROXIMATION ON CHARMS-REFINED MESHES

Both the quasi-hierarchical and the true hierarchical approximation may be expressed in exactly the same way, and in fact often it is possible to devise refinement sequences that yield precisely the same spans for both basis types. This is not surprising, since both basis

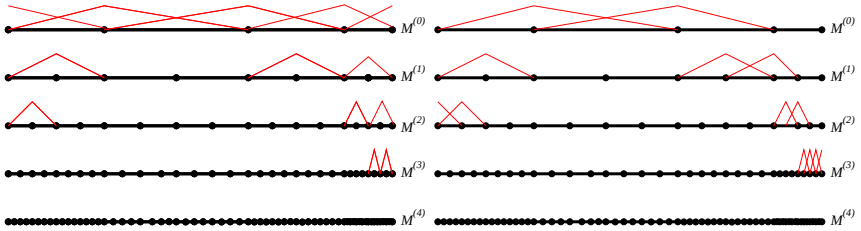

Figure 4. Comparison of the true hierarchical basis (left) with the quasi-hierarchical basis (right).

types follow from the refinement equation, and as shown in the next section, the construction of both types of basis may be expressed as the activation or deactivation of basis functions from the conceptual hierarchy of nested meshes. The equivalence of the quasi-hierarchical and the hierarchical basis allows us to write the finite element approximation in the unified way

$$u_h(x) = \sum_{j:\widehat{\mathcal{B}}^{(j)}\neq\emptyset} \left(\sum_{i:\phi_i^{(j)}(x)\in\widehat{\mathcal{B}}^{(j)}} \phi_i^{(j)} u_i^{(j)} \right) ,$$

where $\widehat{\mathcal{B}}^{(j)}$ is the set of active functions on level j, and $u_i^{(j)}$ are the nodal parameters. Evidently, it would be possible to express the finite element approximation with a single sum, as usual, but the above form makes it clear that the basis functions "live" on different refinement levels.

The appromation properties of the CHARMS-adapted basis set are identical for the discretization error to classical finite element basis using the same type of element. On the other hand, the hierarchical character of the basis may lead to a deterioration in the conditioning of the system matrices [17]. This will be more pronounced for the true hierarchical basis than for the quasi-hierarchical basis. For a small number of overlaps (up to about five levels of refined basis functions interacting in the transition regions) the effect seems minor. However, so far this aspect has not been studied in sufficient detail.

It remains to formulate algorithms for the construction of the active sets, and that is the goal of the next section.

3. FORMAL STATEMENT OF THE REFINEMENT AND UNREFINEMENT ALGORITHMS

It is of advantage in applications when the active functions constitute a *basis*, i.e. when they are linearly independent. In order to make the present work self-contained, we re-phrase here the algorithms enunciated in Reference [12], but without proofs. These algorithms activate and deactivate functions from the nesting hierarchy, and guarantee the linear independence of the active function set.

For the finite element meshes we consider in this paper, no basis function support is entirely enclosed by the support of another basis function, i.e.,

$$\text{supp}\left[\phi_k^{(j)}\right] \not\subseteq \text{supp}\left[\phi_i^{(j)}\right] \quad \text{for} \quad k \neq i \text{ and } \forall j \geq 0 \, . \tag{8}$$

This is an important tool in our proofs [12]: Consider two functions on the same level; because of the above property, the refinement set of either must contain at least one function not present in the refinement set of the other.

There are many possible algorithms for building adapted bases, and here we present algorithms based upon three **rules**:

1. The refining/unrefining of a function on level j may affect that function or any of its children on level $j+1$; no other function may be involved.
2. A function on level $j+1$ ($j+1 \geq 1$) may be refined only when all its parents on level j have been refined.
3. A function on level j may be unrefined only if (a) it is currently refined and (b) none of its children on level $j+1$ are refined.

If the basis functions are completely supported by one ring of elements around a node, then rules 2 and 3 enforce the common rule of one-level-difference refinement of neighbors; this rule has been applied to finite element meshes, as well as to spatial datastructures such as quadtrees and octrees in various contexts (graphics, mesh generation, spatial searches, etc.) [18, 19].

We now show that if the (un)refinement is applied atomically, i.e. it is either entirely executed or not at all, then linear independence of the adapted basis is guaranteed. Furthermore, our algorithms ensure that the refinement step is *lossless*; the span of the resulting set includes the span of the original set. (Unrefinement is not lossless, in general: some information is always going to be lost, since the goal of unrefinement is to decrease the span of the approximation space.)

3.1. QUASI-HIERARCHICAL BASIS

We begin by discussing the quasi-hierarchical refinement strategy. We first describe the refinement operation, and show that it preserves the linear independence requirement and is lossless; we then describe the unrefinement operation, and show that it too preserves the linear independence requirement.

3.1.1. Refinement The ***refinement set*** of the coarser function $\phi_i^{(j)}$ supported by the mesh on level $j+1$ is denoted by $\mathcal{R}^{(j+1)}[\phi_i^{(j)}]$, and contains exactly those basis functions that contribute to the right-hand side of the refinement equation with a non-zero coefficient:

$$\mathcal{R}^{(j+1)}[\phi_i^{(j)}] = \left\{ \phi_k^{(j+1)} \mid \beta_{ik}^{(j+1)} \neq 0 \right\} . \tag{9}$$

If $\phi_k^{(j+1)}$ belongs to $\mathcal{R}^{(j+1)}[\phi_i^{(j)}]$, we say that "$\phi_i^{(j)}$ *is a* ***parent*** *of* $\phi_k^{(j+1)}$," and "$\phi_k^{(j+1)}$ *is a* ***child*** *of* $\phi_i^{(j)}$."

Given an initial basis function set, $\mathcal{B}$, which contains $\phi_i^{(j)}$, and satisfies the linear independence requirement, we choose to produce another function set $\mathcal{B}'$ by deactivating $\phi_i^{(j)}$ and activating all of its children $\mathcal{R}^{(j+1)}[\phi_i^{(j)}]$. We refer to this algorithm, which maps $\mathcal{B}$ to $\mathcal{B}'$, as *quasi-hierarchical refinement.*

We claim that quasi-hierarchical refinement (a) preserves the linear independence requirement and (b) is lossless. See Reference [12] for proofs.

3.1.2. Unrefinement Given an initial basis function set, $\mathcal{B}$, that satisfies the linear independence requirement, and given a previously refined $\phi_i^{(j)}$, not in $\mathcal{B}$ and eligible for unrefinement under rule 3, we choose to produce another function set $\mathcal{B}^{'}$ by activating $\phi_i^{(j)}$ and deactivating those children of $\phi_i^{(j)}$ which have no other currently refined (i.e. inactive) parent. More concisely, the members of the following set are deactivated:

$$\left\{ \phi_m^{(j+1)} \in \mathcal{R}^{(j+1)}[\phi_i^{(j)}] \quad \wedge \quad \forall\, r \neq i \;\; \phi_m^{(j+1)} \in \mathcal{R}^{(j+1)}[\phi_r^{(j)}] \rightarrow \phi_r^{(j)} \in \widehat{\mathcal{B}}^{(j)} \right\} \; . \tag{10}$$

We refer to this algorithm, which maps $\mathcal{B}$ to $\mathcal{B}^{'}$, as *quasi-hierarchical unrefinement.* We claim that quasi-hierarchical unrefinement preserves the linear independence requirement. (See Reference [12] for proofs.)

3.2. HIERARCHICAL BASIS

We have completed the algorithm of quasi-hierarchical refinement, which treats refinement as the *replacement* of coarse-level functions by finer-level functions. Let us turn to the alternative strategy for constructing adapted bases: hierarchical refinement treats refinement as the *addition* of finer-level "detail functions" to an unchanged set of coarse-level functions. Before we continue, let us formalize the concepts of a detail function and a detail (function) set that had been introduced above.

Definition: Given a function $\phi_i^{(j)}$, construct the set of all functions $\phi_k^{(j+1)} \in \mathcal{R}^{(j+1)}[\phi_i^{(j)}]$ such that they vanish at the location x_i of node i

$$\mathcal{D}^{(j+1)}[\phi_i^{(j)}] = \left\{ \phi_k^{(j+1)} \mid \phi_k^{(j+1)} \in \mathcal{R}^{(j+1)}[\phi_i^{(j)}] \text{ and } \phi_k^{(j+1)}(x_i) = 0 \right\} \; . \tag{11}$$

The set $\mathcal{D}^{(j+1)}[\phi_i^{(j)}]$ is the ***detail set*** of $\phi_i^{(j)}$. Functions that belong to at least one detail set are called ***detail functions***.

Note that our definition of the detail set guarantees that there is *precisely one* fine function $\phi_i^{(j+1)}$ such that $\mathcal{R}^{(j+1)}[\phi_i^{(j)}] = \mathcal{D}^{(j+1)}[\phi_i^{(j)}] \bigcup \phi_i^{(j+1)}$.

3.2.1. Refinement Given an initial basis function set, $\mathcal{B}$, which contains $\phi_i^{(j)}$, and satisfies the linear independence requirement, we choose to produce a refined set $\mathcal{B}^{'}$ by activating $\mathcal{D}^{(j+1)}[\phi_i^{(j)}]$. We refer to this algorithm, which maps $\mathcal{B}$ to $\mathcal{B}^{'}$, as *hierarchical refinement.* As proved in Reference [12], the hierarchical refinement (a) preserves the linear independence requirement and (b) is lossless.

3.2.2. Unrefinement Given an initial basis function set, $\mathcal{B}$, that satisfies the linear independence requirement, and given a previously refined $\phi_i^{(j)}$, also in $\mathcal{B}$ and eligible for unrefinement (rule 3), we choose to produce another function set $\mathcal{B}^{'}$ by deactivating functions $\phi_m^{(j+1)} \in \mathcal{D}^{(j+1)}[\phi_i^{(j)}]$ that are absent from all the refinement sets of currently refined functions on level j. We refer to this algorithm, which maps $\mathcal{B}$ to $\mathcal{B}^{'}$, as *hierarchical unrefinement.* It is easy to show that hierarchical unrefinement preserves the linear independence requirement [12].

4. DESIGN OF AN ADAPTIVE SOLVER

In this section we will pursue the discussion of the CHARMS technology into the design phase, which will be illustrated with UML diagrams [20]. Figure legends use slanted type to indicate abstract classes; the inheritance and delegation relationships are included with multiplicities.

4.1. GEOMETRIC CELL (GCELL)

The computational domain is assumed to be a non-manifold object embedded in a d-dimensional Euclidean space. The constituent manifolds are assumed to be locally coordinatized by m Cartesian coordinates, $m \leq d$, depending on the manifold dimension.

The domain is discretized into geometric cells. The manifold dimension of these cells corresponds to the manifold being discretized. Geometric cell (`GCELL`) is one of the basic classes in our implementation. However, `GCELL` is an abstract class, and only its specializations are being instantiated. The specialization branches in the direction of manifold dimension: thus there are geometric cells of manifold dimension zero (0) (`GCELL_POINT_P1`), of one (1) (e.g. `GCELL_LINE_L2`), two (2) (for instance `GCELL_SURF_Q4`, or `GCELL_SURF_T3`), and three (3) (`GCELL_SOLID_H8`, or `GCELL_SOLID_T4`). Figure 5 shows the class diagram for the quadratic six-node triangle (`GCELL_SURF_T6`).

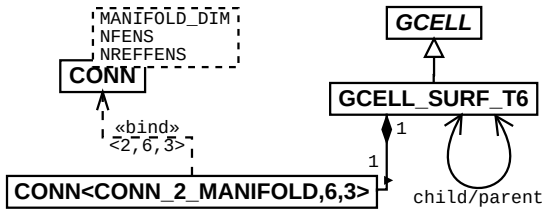

Figure 5. Class diagram for a gcell representing a six-node (quadratic) triangular finite element. Note the use of the templated `CONN` class for the specification of the connectivity.

It is assumed that the cells in the physical space are images of "master" cells in the parametric domain. The parametric coordinates constitute the charts for the pieces of the manifolds embedded in the d-dimensional Euclidean space to which the master cell maps. Each cell may be mapped into a different Cartesian coordinate system, but for simplicity we shall assume in this paper that all cells map to a single, global Cartesian coordinate system.

Geometric cells (`GCELL`'s) provide a number of services to the adaptive code. Thus, they are the means of defining pieces of basis functions (in the master parametric domain), but they are also carriers of the topological/geometrical refinement hierarchy. In other words, the h-refinement involves division of a `GCELL` into finer, nested `GCELL`'s. It is assumed here that this division results in `GCELL`'s of the same type. The topological/geometrical refinement is applied recursively, resulting in a tree of `GCELL`'s, the root being represented by the coarsest `GCELL`, and the leaves being cells without children. To give an example, Figure 6 shows the hierarchy of `GCELL`'s stemming from the root at the bottom, with the finest cells as the leaves at the top, for a four-node quadrilateral.

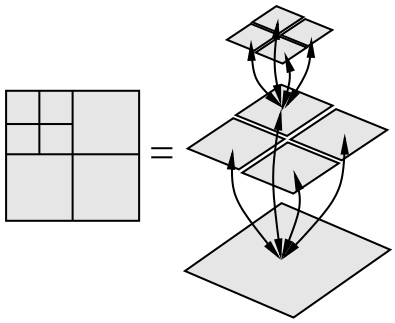

Figure 6. Schematic refinement tree for a quadrilateral `GCELL`. The upward pointing arrows symbolize the "child" relationship, the downward arrow points at the parent.

4.2. CONNECTIVITY AND REFINEMENT NODES

Connectivity `CONN` is a template class, parameterized with the manifold dimension, number of connected nodes, and number of refinement nodes (Figure 5). This allows us to distinguish between a line with 3 connected nodes, and a 3-node triangle, etc. From a refinement point of view, a very important connectivity type is the vertex (`CONN<CONN_0_MANIFOLD,1,1>`). All refinement nodes topologically located at the nodes of the coarser levels are classified on the connectivity `CONN<CONN_0_MANIFOLD,1,1>`.

encodes a topological division rule. In order to maximize the number of element types that may be mixed in a single finite element mesh we adopt the strategy that views elements of manifold dimension d as boundaries of elements of manifold dimension $d + 1$. This information is encoded in the instantiations of the template class `CONN`. Thus, vertices of cells are topological entities `CONN<CONN_0_MANIFOLD,1,1>`, that is their manifold dimension is zero, connect a single node, and are refined with one node. The edges of cells are cells in their own right, and vertices are their boundaries. For instance, `CONN<CONN_1_MANIFOLD,2,1>` is a two-node line segment, which is refined with a single node at the mid-point (origin of the parametric coordinates). Three-node triangles (`CONN<CONN_2_MANIFOLD,3,0>`) and four-node quadrilaterals (`CONN<CONN_2_MANIFOLD,4,1>`) are bounded by connectivities `CONN<CONN_1_MANIFOLD,2,1>`, which makes it possible to mix those two in a single mesh. Three-node triangles have no refinement nodes of their own (all refinement nodes are located at the edges, none in the interior), whereas the four-node quadrilaterals have one interior refinement node. As the last illustration, consider the quadratic isoparametric triangle. The class diagram for the quadratic six-node triangle in Figure 5 refers to the connectivity type `CONN<CONN_2_MANIFOLD,6,3>`, that is a 2-manifold that connects 6 nodes, and has 3 internal refinement nodes. The boundaries of this cell are of type `CONN<CONN_1_MANIFOLD,3,2>`, and the connectivity of the quadratic triangle is compatible with the refinement of the 10-node tetrahedron.

The topological division starts with the connectivities of the lowest manifold dimension, that is zero (vertices). Then edges are divided, followed by faces, and finally volumes. Thus, an eight-node hexahedron would first divide its vertices, then its edges, faces, and finally it would conclude the process by generating one refinement node at its barycenter.

4.3. FIELD

The class `FIELD` is a fundamental concept in our software framework. The term "field" implies spatial variation [21], and in the present context, field u is represented by the sum

$$u_h(x) = \sum_j \phi_j(x) u_j \ , \tag{12}$$

where $u_h(x)$ is the usual finite element expansion of function $u(x)$, $\phi_j(x)$ are the finite element basis functions, and u_j are the nodal parameters. The approximated function may be a scalar, vector, or tensor function of a vector argument, for instance the displacement field over the domain (vector function of a vector argument), or temperature field (scalar function of a vector argument). Each term in the sum (12) is expressed through an object, the `FIELD_PAIR`. It relates two objects: the basis function, `BFUN`, and the degree-of-freedom parameter (DOFPARAM for short). The value of the DOFPARAM object may change freely, or some or all of its components might be prescribed as an expression of an essential boundary condition. `FIELD_PAIR` is a class parameterized with the number of components in the DOFPARAM object, and in that way the current implementation is able to represent scalar degrees of freedom (one component), or vectors in 3D space (three components), or an array of an arbitrary number of components.

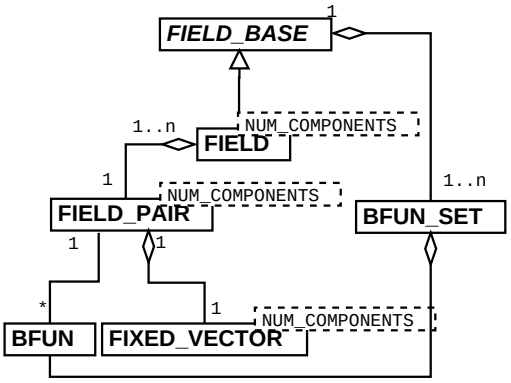

Figure 7. Class diagram for the field pair and field.

4.4. BASIS FUNCTION (BFUN)

Figure 8 shows the class diagram for the basis function and basis function set types. `BFUN` serves as the abstract base class. `BFUN_FE` builds up the first concrete class upon `BFUN` and provides access to the standard finite element basis functions. `BFUN_FE` maintains a list of all the `GCELL`'s over which the individual pieces of the given basis function are defined. Thus, for instance the basis function associated with the node indicated by the black dot in the coarse mesh in Figure 3 would include in the list one triangle (`GCELL_SURF_T3`) and three quadrilaterals (`GCELL_SURF_Q4`).

4.5. BASIS FUNCTION SET (BFUN_SET)

The basis function set (`BFUN_SET`) collects basis functions defined on a particular geometric mesh. Any particular `BFUN_SET` maybe associated with any number of fields. The `BFUN_SET` provides the field with a very important service: The `BFUN_SET` generates an opaque identifier (`BFUN_DOFPARAM_PAIR_ID`) which may be used to access a field pair in constant time (it is in fact implemented as an array access). The `BFUN_SET` generates this identifier based on the pointer to an opaque, system-wide unique identifier which is carried by a finite element node. That in turn implies that there is a one-to-one link between the unique identifier (finite element node) and a basis function. The `BFUN_DOFPARAM_PAIR_ID` identifier is used for instance by solvers so they can distribute computed solutions to the field pairs.

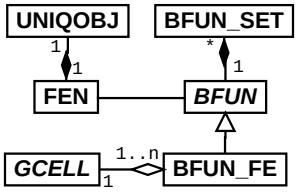

Figure 8. Class diagram for the basis function and basis function set.

5. IMPLEMENTATION OF REFINEMENT/UNREFINEMENT

The implementation of the refinement and unrefinement algorithms as enunciated earlier is straightforward. However, it seems worthwhile to point out that the refinement of a single basis function involves the generation of child `GCELL`'s, which in its turn requires the addition of refinement nodes. Some of these nodes are shared by two or more `GCELL`'s, and care must be taken to introduce only a single copy of each shared refinement node so that compatibility of the refinement functions is ensured.

To enable the sharing of refinement nodes during the creation of the child geometric cells, we use the concept of a refinement context. It acts as the dispenser of refinement nodes.

The present code labels refinement nodes with an instance of the connectivity object on which they are classified. For instance, for an eight-node hexahedron, the refinement nodes are either located at the vertex nodes, at the mid-points of the edges, at the barycenters of the faces, or at the center of the volume: compare with Table I. The request to supply a refinement node on a given connectivity object (for instance, on an edge connecting nodes I and J) is processed by the refinement context: If the refinement node had not been created for the connectivity object before, it is created, and the reference to the connectivity is remembered for future lookups. Each `GCELL` may thus proceed with the refinement independently of its neighbors; the sharing of refinement nodes is managed transparently.

Location of refinement node	Classified at (CONN<MANIFOLD_DIM,NFENS,NREFFENS>)
Vertex	CONN<CONN_0_MANIFOLD,1,1>
Edge	CONN<CONN_1_MANIFOLD,2,1>
Face	CONN<CONN_2_MANIFOLD,4,1>
Volume	CONN<CONN_3_MANIFOLD,8,1>

Table I. Refinement nodes of an 8-node hexahedron.

6. EXAMPLE: EQUATION OF STEADY DIFFUSION

To illustrate the adaptive code, we shall consider a simplified version of the linear partial differential equation of steady diffusion. In strong form: Given $f : \Omega \leftarrow R$, $g : \Gamma_g \leftarrow R$, and $h : \Gamma_h \leftarrow R$, find $u : \bar{\Omega} \leftarrow R$

$$\begin{aligned} -\kappa \Delta u &= f \quad \text{in } \Omega \\ u &= g \quad \text{on } \Gamma_g \\ u_{,i} n_i &= h \quad \text{on } \Gamma_h \ , \end{aligned}$$

where Ω is the domain (two- or three-dimensional), Γ_h and Γ_g are disjoint parts of the boundary, $\Gamma = \Gamma_h \cup \Gamma_g$, Δ is the Laplace operator, "$,i$" in the subscript means differentiation with respect to the i-th Cartesian coordinate, κ is the conductivity (assumed to be constant in Ω), and the function g and h are the prescribed boundary values. A weak form of Eq. (13) is derived in a standard way [22] as

$$a(w,u) = (w,f) + (w,h)_{\Gamma_h} \ , \tag{13}$$

where

$$a(w,u) = \int_\Omega \kappa w_{,i} u_{,i} \, d\Omega \ , \tag{14}$$

$$(w,f) = \int_\Omega w f \, d\Omega \ , \tag{15}$$

$$(w,h)_{\Gamma_h} = \int_{\Gamma_h} w h \, d\Gamma_h \ . \tag{16}$$

The functions u and w are the trial and test function respectively, $u \in \mathcal{S}$ and $w \in \mathcal{V}$, where $\mathcal{S}$ is the trial space and $\mathcal{V}$ is the test space.

Next, we choose finite dimensional approximations of the trial and test spaces as the span of some suitable finite element basis functions. Adopting a Galerkin formulation, the test functions $v^h \in \mathcal{V}^h$ will all vanish on the boundary Γ_g, and the trial functions u^h will be decomposed as

$$u^h = v^h + g^h \ , \tag{17}$$

where g^h will satisfy (approximately) the boundary condition $u = g$ on Γ_g, and $v^h \in \mathcal{V}^h$.

The homogeneous-part of the trial functions is written as

$$v^h = \sum_B \phi_B d_B \ ,$$

which leads to the discrete system of linear equations

$$\sum_{B \notin \Gamma_h} a(\phi_A, \phi_B) d_B = (\phi_A, f) + (\phi_A, h)_\Gamma - \sum_{B \in \Gamma_h} a(\phi_A, \phi_B) g_B \ , \quad A \notin \Gamma_h \ . \tag{18}$$

On the left-hand side of (18) is the product of the conductivity matrix with the vector of unknowns, and the source terms are all on the right-hand side.

6.1. ADAPTIVE ALGORITHM

The overall adaptive algorithm is described as

algorithm ALGO_STEADY_DIFFUSION
do
1. Assemble the conductivity matrix;
2. Assemble the right-hand side (the source terms);
3. Solve the system of linear equations;
4. Evaluate the required mesh size $h(x)$;
5. Adapt the FE basis function set $\mathcal{V}^h$ to the function $h(x)$;

while not converged

Refinement or unrefinement follows an estimate of the required mesh size in the form of a mesh-size function, $h(x)$. Our code uses residual-based estimates to compute per-cell errors, which are then processed to yield per-basis-function errors. As an example of such a procedure we have used the error density, which is obtained as the ratio of the summed error from all `GCELL`'s supporting the given basis function and their volume (area, in two dimensions). Basis functions with high error density are candidates for refinement, basis functions with low error density are considered for unrefinement. The principle of equidistribution of error then leads to a mesh-size function $h(x)$.

CHARMS make it very easy to re-use the solution from the coarse mesh in the process of solving for the fine-mesh solution. The solution field from the coarse mesh is transferred to the fine mesh (see next section), and is then passed along to the linear equation solver as the initial guess of the fine-mesh solution. Iterative solution methods then can make use of this initial guess to converge to the fine-mesh solution more efficiently.

6.2. FIELD TRANSFER

To satisfy the needs of an adaptive procedure, we need a ***field transfer*** from one field (the source) to another (the destination). In our case, the two fields are defined on two distinct refined meshes resulting from the refinement of the same initial mesh. The field transfer operation is in general formulated as the computation of the nodal parameters of the destination field from the condition that the destination field be somehow "close" to the source field. Since our refinement algorithm is lossless, we can transfer a field from a coarse mesh to a destination field on a fine mesh exactly. However, if the destination field is defined fully or partially on a mesh coarser than the source field, the transfer is going to involve some loss of information.

Approximations generated by CHARMS are in general non-interpolating. This makes it seemingly hard to come up with an efficient computation of the nodal parameters. A function

given on the source field

$$\widetilde{u}(x) = \sum_k \widetilde{\phi}_k(x)\widetilde{u}_k \ ,$$

where $\widetilde{\phi}_k \in \widetilde{\mathcal{B}}$ and $\widetilde{u}_k$ are the basis functions and nodal parameters of the source field, is to be approximated by (transfered to) the destination field

$$u(x) = \sum_j \phi_j(x)u_j$$

where $\phi_j \in \mathcal{B}$ and u_j are the basis functions and nodal parameters of the destination field. The parameters $\widetilde{u}_k$ are known, but parameters u_j are all unknown. Thus we have a classical function approximation problem, and a number of schemes may be used to solve for the unknown parameters; interpolation, and least square fitting being perhaps the most often used ones. Instead of striving for the most general solution, we take advantage of the special nature of the nested finite element spaces, and use an interpolating prolongation when transferring from coarse to fine, and the dropping of details during restriction in the opposite direction. The refinement equation (4) in combination with the refinement and unrefinement algorithms of Section 3 yields an efficient algorithm for the computation of the nodal parameters in the destination field. Due to space constraints, the algorithm is not described here and the reader is referred to Reference [23].

6.3. EVALUATION CELL (ECELL)

In our Galerkin example, the mesh is needed for the evaluation of the integrals (14), which in turn involves the basis function derivatives. Note that there are clearly two separate and orthogonal operations here:

- Evaluate the basis functions (and their derivatives); and
- Use those computed basis function values or their derivatives to evaluate the integrals.

The second task will change with each new problem. The first task (the evaluation of the basis functions), on the other hand, will be the same independently of the problem to be solved. Our design therefore separates the evaluation of the basis functions into a task performed by the geometric cells, and all the other operations are factored into responsibilities for the so-called ***evaluation cells*** (`ECELL`'s).

6.4. PROTOCOLS

The protocol `PROTO_STEADY_DIFFUSION` for the problem at hand consists predictably of three functions, `assemble_conductivity_matrix`, `assemble_source_terms`, and `solve`. The role of the protocol is simply to bundle the functionality (creation of the evaluation cells, looping over those cells to assemble the contributions to the left- and right-hand side of the linear system, and finally the solution of the system) into a reusable component. The protocol enlists the services of a linear equation solver component, which we do not have the space to describe in any detail. The protocol is employed by the algorithm in steps 1–3.

6.5. CALCULATION OF BASIS FUNCTION TABLES

When the basis function tables (the function value, and values of the derivatives with respect to the spatial coordinates) are needed at a particular quadrature point, they need to be

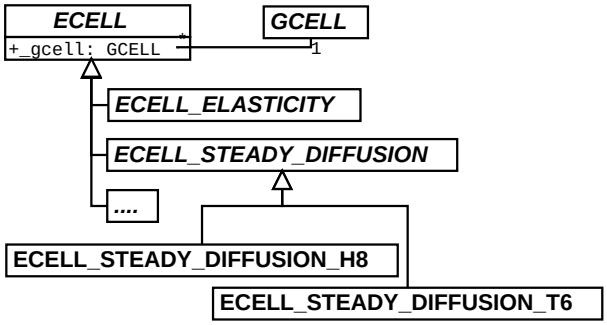

Figure 9. Class diagram for the steady-diffusion evaluation cell of for an 8-node hexahedron.

computed for the hierarchy of overlapping basis functions. The geometric cells on which the basis functions are defined are related through the parent/child relationship.

To evaluate the spatial integrals we perform numerical quadrature on the geometric cells that are the leaves of the refinement hierarchy. In other words, the interaction of basis functions from different levels is evaluated over geometric cells that support the finest basis function. (An alternative approach that evaluates interactions of pairs of basis functions over the support of the finer function had been proposed in Reference [13].)

Because of our chosen quadrature scheme (all integrals are evaluated over the leaf cells), we need to traverse the hierarchy from the leaf towards the root. Furthermore, since all the parent/child relationships are transitive, it is sufficient to consider in this discussion just two levels from the mesh hierarchy: the parent level and the child level.

Without loss of generality we may consider a particular element type, for instance the isoparametric eight-node (serendipity) quadrilateral (Figure 2) [24]. The basis functions of a single element are defined in the parametric coordinates in the biunit square ($-1 \leq \xi \leq +1$, $-1 \leq \eta \leq +1$) as

$$\mathbf{\Phi}^T(\xi,\eta) = \begin{bmatrix} -1/4(1-\xi)(1-\eta)(1+\xi+\eta) \\ -1/4(1+\xi)(1-\eta)(1-\xi+\eta) \\ -1/4(1+\xi)(1+\eta)(1-\xi-\eta) \\ -1/4(1-\xi)(1+\eta)(1+\xi-\eta) \\ 1/2(1-\xi)(1+\xi)(1-\eta) \\ 1/2(1-\eta)(1+\eta)(1+\xi) \\ 1/2(1-\xi)(1+\xi)(1+\eta) \\ 1/2(1-\eta)(1+\eta)(1-\xi) \end{bmatrix}, \tag{19}$$

where the superscript T indicates transpose. The derivatives of the basis functions with respect to the spatial coordinates are evaluated as usual by the chain rule, which can be put in matrix form as

$$\begin{bmatrix} \dfrac{\partial \mathbf{\Phi}(\xi,\eta)}{\partial x} \\ \dfrac{\partial \mathbf{\Phi}(\xi,\eta)}{\partial y} \end{bmatrix} = \mathbf{J}^{-1} \begin{bmatrix} \dfrac{\partial \mathbf{\Phi}(\xi,\eta)}{\partial \xi} \\ \dfrac{\partial \mathbf{\Phi}(\xi,\eta)}{\partial \eta} \end{bmatrix}, \tag{20}$$

with the jacobian matrix defined as

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} . \tag{21}$$

The elements of the jacobian matrix are computed from the expansions of x and y in terms of the basis functions

$$x = \sum_i \Phi_i(\xi, \eta) x_i \ , \quad y = \sum_i \Phi_i(\xi, \eta) y_i \ , \tag{22}$$

where x_i and y_i are the nodal geometry parameters.

If we now consider approximation on a mesh with basis functions at several levels, we may still keep Eq. (19), but the elements of the $\mathbf{\Phi}$ matrix are now functions $\phi_i^{(j)}$ at various levels j. Similarly, the derivatives with respect to spatial derivatives may be computed from Eq. (20), but it must be realized that the geometric cells in the mesh hierarchy have their own parametric coordinates (i.e. they are all mapped from the master element). To compute the jacobian matrix (21) all the derivatives $\partial \Phi_i / \partial \xi$ etc. need to be computed in a *single* coordinate system. In our implementation, we choose to compute all the derivatives in the parametric coordinates of the leaf geometric cell (i.e. the finest cell in the hierarchy). We use the notation ξ' and η' for the parametric coordinates of the parent, and ξ and η for the parametric coordinates of the child, and $\phi_i^{(j)}$ for the basis function defined on the parent, and $\phi_i^{(j+1)}$ for the basis functions defined on the child. The parent (top row in Figure 2) is divided into four children (bottom row) in its parametric space, and all elements, parent and the children, are mapped to the physical space using the standard isoparametric mapping in their own coordinates. We express the derivatives of the basis functions on the parent cell with respect to the parametric coordinates of the child needed in (21) through the chain rule

$$\frac{\partial \phi_i^{(j)}}{\xi} = \frac{\partial \phi_i^{(j)}}{\xi'} \frac{\partial \xi'}{\xi} \ , \tag{23}$$

and similarly for the relationship between η and η'. For our particular example, the mapping from the parent to the child is simple. For instance, for child 0 (compare with Figure 2):

$$\begin{aligned} \xi' &= 1/2(\xi - 1) \\ \eta' &= 1/2(\eta - 1) \ , \end{aligned} \tag{24}$$

and we get $\partial \xi' / \partial \xi = 1/2$ and $\partial \eta' / \partial \eta = 1/2$; it is easy to show that this relationship holds for all four children.

We are ready to state the algorithm `eval_bfun_set` for the computation of the values and derivatives with respect to the spatial coordinates of the basis functions active at a particular quadrature point: The inputs are the leaf cell and the parametric coordinates of a given quadrature point in that cell. Note that the tree needs to be descended from the leaf cell towards the root, and the algorithm may proceed by recursion or by iteration.

algorithm eval_bfun_set
1. Descend the refinement tree
   and compute $\phi_i^{(j)}$, $\partial\phi_i^{(j)}/\partial\xi$, ... for all active functions, $j = 0, 1, \ldots$ from (23)
2. Compute the jacobian matrix $\mathbf{J}$ of Equation (21),
   and then invert to obtain $\mathbf{J}^{-1}$
3. Using $\partial\phi_i^{(j)}/\partial\xi$, ... from step 1, and $\mathbf{J}$ from step 2,
   compute $\partial\phi_i^{(j)}/\partial x$, ..., $j = 0, 1, \ldots$, from Equation (20)

Note that the task of evaluating the individual basis functions and their derivatives in the parametric coordinates is performed by the `GCELL`'s, as usual in standard finite element codes. The only difference is that the child's coordinates need to be related to the parent's coordinates through the chain rule (the factor 1/2 in the example discussed above).

6.6. GEOMETRY

A remark is in order concerning the definition of the geometry of the domain. An isoparametric description of the finite element fields is used, and Equation (22) makes it clear that to evaluate the geometry, i.e. the location of a point given by its natural, parametric coordinates, the nodal parameters for the geometry are needed. Our adaptive framework represents the geometry as an ordinary field. In other words, the code makes no distinction between the unknown fields and the geometry. However, one difference needs to be noted: the geometry needs to be initialized at the beginning of the computation. Fortunately, that is an easy task, since the basis functions on the initial (input) mesh interpolate. Therefore, the nodal parameters of the initial geometry field are simply the coordinates of the nodes. The geometry fields for the refined approximations need to be computed by field transfer, however, since the approximation becomes non-interpolating upon refinement. If the initial mesh describes the domain exactly, the geometry is also described exactly on any refined mesh, because of the lossless character of field transfers from coarse to fine meshes. On the other hand, if the geometry of the domain is only approximated by the mesh, the nodal geometry parameters may be computed to improve the shape approximation by fitting curved boundaries [23].

6.7. HEXAHEDRAL DISCRETIZATION IN 3D

Hexahedral finite elements are often preferred to tetrahedra because of their slightly better performance in a number of applications. However, refining hexahedra by introducing compatible edges, or by using a technique similar to the mesh refinement of tetrahedra by bisection is a tough problem, since the element quality tends to deteriorate very quickly and without bounds. Due to the regular division of the elements by octasection, CHARMS have no problem with shape deterioration. The implementation of mesh refinement with CHARMS also proves extremely easy, and in fact it took the first author just a couple of hours to add the hexahedral refinement to the CHARMS framework once the code had been debugged in one dimension. Most of the implementation effort goes into the coding of the connectivity of the parent and its children: the information which nodes are connected by a given child needs to be recorded.

Figure 10 shows the mesh and the results of a sample simulation for a steady diffusion problem solved on a polyhedral domain. The initial grid and a refined grid are compared

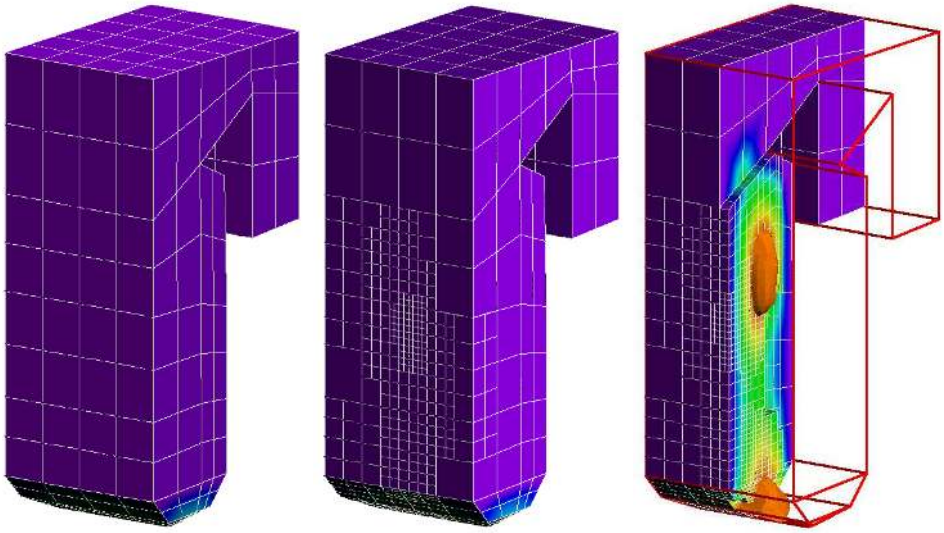

Figure 10. Steady diffusion equation solved in a three-dimensional domain. Left: initial grid; middle: step after two adaptations; right: cut-off grid with a solution contour. The integration cells are shown, with color coding corresponding to the temperature distribution.

side-by-side. The integration cells are shown in Figure 10 and the reader should take care to realize the apparently hanging nodes are a visualization artifact: no active basis functions are associated with the incompatibly located nodes.

6.8. REFINEMENT OF TRIANGULATIONS WITH QUADRATIC TRIANGLES

As an example of mesh refinement for higher-order approximations we discuss here an implementation for the quadratic (6-node) triangles. Quadrisection is used to divide each parent triangle into four children of the same type in the parametric space: viz Figure 11. The refinement nodes are numbered from 0 to 14. Refinement nodes $0 \ldots 5$ are classified at the nodes of the parent, nodes $6 \ldots 11$ are classified on the 3-node edges of the parent, and the last three nodes (12,13,14) are classified in the interior of the parent element. An important piece of information that the refinement algorithm needs is which functions at the nodes of the child refine a particular function of the parent. For instance, function at node 0 of the parent is refined by functions at refinement nodes 0, 6, 7, 13, 14, 10, 11 of the children (all but the function at node 0 are detail functions); function at node 4 of the parent is refined by functions at nodes 4, 8, 9, 12, 13, 14 of the children (all but the first are detail functions), etc.

Figure 12 shows the contours of the solution of the Poisson equation for a "dipole" source function. The results are displayed on the integration cells. (The visualization approximates the smooth quadratic variation by piecewise linear polygons.) Figure 13 illustrates the active basis function set for the refined mesh in the upper-left corner of the domain from Figure 12 by showing the active functions as red balls, and the geometric cells that support the active functions are filled triangles. Note that the active functions on level $j+1$ vanish along the boundaries of the geometric cells of level j. That is a visual confirmation of the preserved

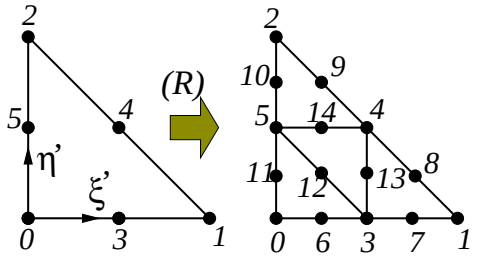

Figure 11. Refinement of the 6-node triangle.

compatibility.

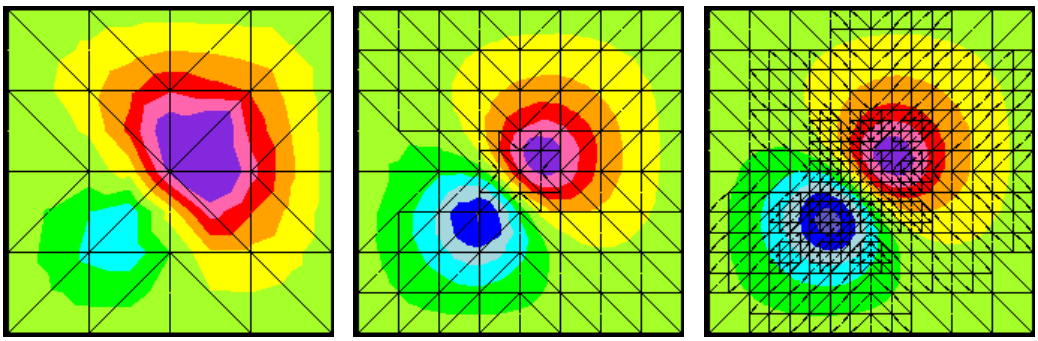

Figure 12. Dipole equation with homogeneous boundary conditions solved on a triangulation of square domain with 6-node quadratic triangles. Color coding of the field on three refined grids.

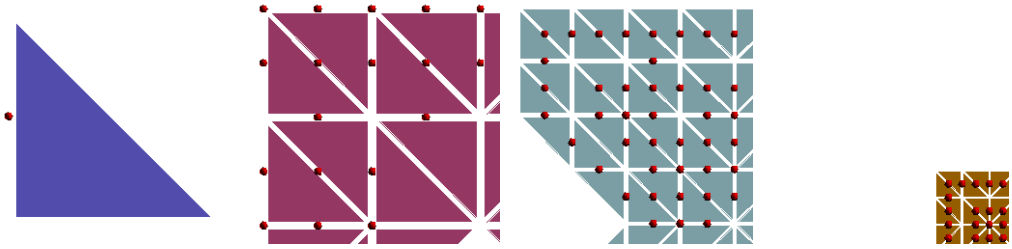

Figure 13. Dipole equation, 6-node quadratic triangles. The geometric cells from the upper-left corner of the computational domain that support basis functions on different levels: left to right, level 1, 2, 3, 4. The red balls indicate the active basis functions.

CONCLUSIONS

The conforming hierarchical adaptive refinement methods (CHARMS for short) have been presented from the conceptual view of the basic ideas up to the detailed view of the object oriented design and implementation.

The CHARMS framework allows for mesh refinement applications to be built quickly and robustly. The CHARMS element library currently includes two-node line elements, 3-node and 6-node triangles, 4-node and 8-node quadrilaterals, 4-node and 10-node tetrahedra (CHARMS proved essential in the solution of the tough problem of guaranteeing the shape quality of a refined tetrahedra [25]), and 8-node hexahedra. After the initial debugging phase for the generic framework, which could be done entirely with the one-dimensional finite element as a stub, the implementation never required serious debugging effort when adding a new element. To support the construction of approximation bases either in a different number of dimensions (perhaps the time-space), or for a different approximation order (cubics), a new element type may need to be added. The associated programming effort then needs to be invested into the coding of the connectivity of the new element, but the generic part of the library is not affected, and no special tricks are required.

This property is very attractive since implementations of current mesh refinement techniques often prove too expensive from the point of view of the initial investment effort, and later on they tend to be much too fragile to be maintained as the software complexity increases. Even though the software described in this work builds on some non-traditional design decisions, CHARMS techniques require little in terms of additional data structures, and may be readily introduced into existing finite element programs.

The object oriented software framework implementing the Conforming Hierarchical Adaptive Refinement MethodS (CHARMS) is available as open source at `http://hogwarts.ucsd.edu/~pkrysl/CHARMS`.

ACKNOWLEDGMENTS

This research was partially supported by the Hellman Fellowship 2001 (PK). The many fruitful discussions the first author has had with Eitan Grinspun and Peter Schröder at the California Institute of Technology are gratefully acknowledged.

REFERENCES

1. M.C. Rivara and G. Iribarren. The 4-triangles longest-side partition of triangles and linear refinement algorithms. *Mathematics of Computation*, 65(216):1485–1502, 1996.
2. R.E. Bank and J.C. Xu. An algorithm for coarsening unstructured meshes. *Numerische Mathematik*, 73(1):1–36, 1996.
3. M.C. Rivara. New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations. *International Journal for Numerical Methods in Engineering*, 40(18):3313–3324, 1997.
4. M.C. Rivara and P. Inostroza. Using longest-side bisection techniques for the automatic refinement of delaunay triangulations. *International Journal for Numerical Methods in Engineering*, 40(4):581–597, 1997.
5. S.O. Wille. A structured tri-tree search method for generation of optimal unstructured finite-element grids in 2 and 3 dimensions. *International Journal for Numerical Methods in Fluids*, 14(7):861–881, 1992.

6. A.W. Liu and B. Joe. Quality local refinement of tetrahedral meshes based on bisection. *SIAM Journal on Scientific Computing*, 16(6):1269–1291, 1995.
7. A.W. Liu and B. Joe. Quality local refinement of tetrahedral meshes based on 8-subtetrahedron subdivision. *Mathematics of Computation*, 65(215):1183–1200, 1996.
8. A. Plaza, M.A. Padron, and G.F. Carey. A 3d refinement/derefinement algorithm for solving evolution problems. *Applied Numerical Mathematics*, 32(4):401–418, 2000.
9. A. Plaza and G.F. Carey. Local refinement of simplicial grids based on the skeleton. *Applied Numerical Mathematics*, 32(2):195–218, 2000.
10. D. N. Arnold, A. Mukherjee, and L. Pouly. Locally adapted tetrahedra meshes using bisection. *SIAM Journal on Scientific Computing*, 22(2):431–448, 2001.
11. H. P. Langtangen. *Computational Partial Differential Equations*. Springer Verlag, 1999.
12. P. Krysl, E. Grinspun, and P. Schröder. Natural hierarchical refinement for finite element methods. *International Journal for Numerical Methods in Engineering*, 56(8):1109–1124, 2003.
13. E. Grinspun, P. Krysl, and P. Schröder. CHARMS: A simple framework for adaptive simulation. *ACM Transactions on Graphics, 21 (3): 281-290 JUL 2002*, 2002.
14. R.E. Bank, T.F. Dupont, and H. Yserentant. The hierarchical basis multigrid method. *Numerische Mathematik*, 52(4):427–458, 1988.
15. S.O. Wille. A tri-tree multigrid recoarsement algorithm for the finite element formulation of the navier-stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 135(1-2):129–142, 1996.
16. G. Strang. Wavelets and dilation equations: A brief introduction. *SIAM Review*, 31(4):614–627, 1989.
17. H. Yserentant. Hierarchical bases. In *ICIAM91*. SIAM, 1991.
18. B. von Herzen and A. Barr. Accurate triangulations of deformed, intersecting surfaces. In *Proceedings of SIGGRAPH'87 conference*, pages 103–110, 1987.
19. H. Samet. *Applications of spatial data structures*. Addison-Wesley, 2nd edition, 1995.
20. G. Booch, I. Jacobson, and J. Rumbaugh. *Unified Modeling Language for Object-Oriented Development. Documentation set version 0.91*. Rational Software Corporation, Santa Clara, CA, 1995.
21. M. W. Beall and M. S. Shephard. An object-oriented framework for reliable numerical simulation. *Engineering with Computers*, 15:61–72, 1999.
22. T. J. R. Hughes. *The Finite Element Method: Linear static and dynamic analysis*. Dover, 2000.
23. B. Zhu and P. Krysl. Data fitting and shape approximation with CHARMS mesh refinements. *Computer Methods in Science and Engineering*, 2002. in preparation.
24. O. C. Zienkiewicz and R. L. Taylor. *The finite element method: The basis*, volume 1. Butterworth and Heinemann, 5 edition, 2000.
25. L. Endres and P. Krysl. Octasection-based refinement of finite element approximations on tetrahedral meshes that guarantees shape quality. *International Journal for Numerical Methods in Engineering*, 2003. to appear.