MARS: Adaptive Simplicial Mesh Refinement

July 20, 2018

1 Introduction

MARS is a N-dimensional mesh (tested up to 6D) refinement C++ library. Currently the refinement algorithms are only available for serial run-times, but an MPI-based parallel implementation is in development.

2 Edge bisection algorithm

The general refinement strategy implemented in this library is based on recursive edge bisection. The algorithm consists of the following steps:

1. Mark elements for refinement.
2. For each marked element select the edge to bisect.
3. Refine the element by bisect the edge and mark the incident elements for refinement.
4. Go back to 2 and repeat until the mesh is conforming.

Figure 1 shows consecutive refinement steps on a 2D mesh, and Figure 2 shows the final mesh for a hyper-spherical refinement pattern for both 2D and 3D meshes.

3 Edge selection

We have implemented the following strategies:

- Recursive longest-edge bisection.
- Newest vertex bisection.

4 Recursive edge selection

When refining an element S incident to an edge e_1 that has been selected for refinement, we first select the appropriate edge e_2 (which does not have to respect $e_1 = e_2$). For the algorithm to terminate we have to finally select e_1 from one of the descendants of S. Figure 3 shows the difference between non-recursive and recursive edge selection.

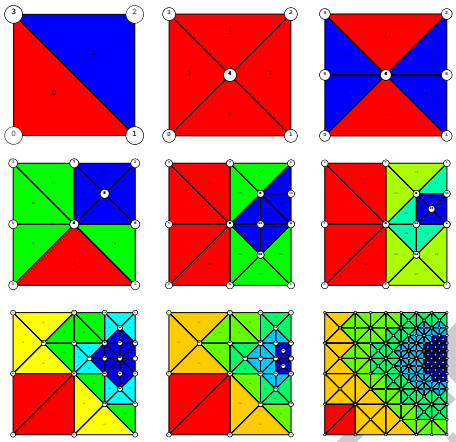

Figure 1: Steps of the bisection algorithm.

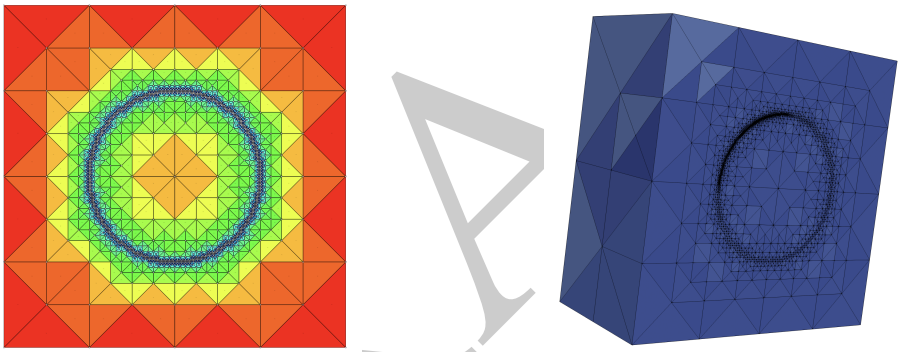

Figure 2: Result for sphere refinement pattern. Left: 2D. Right: 3D.

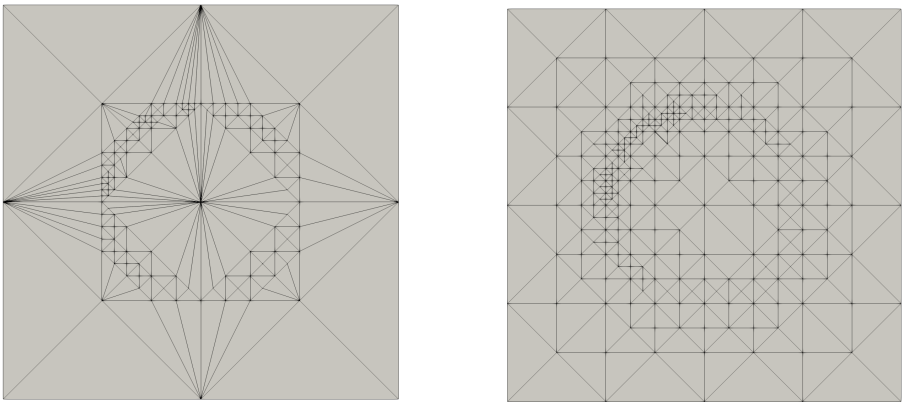

Figure 3: Non-recursive vs. recursive (3D).

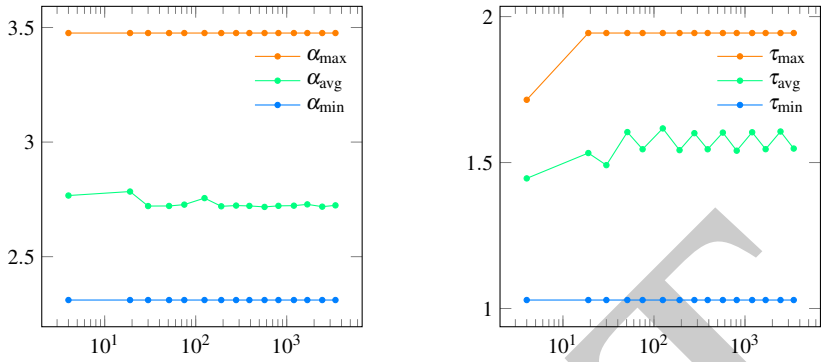

Figure 4: Quality of recursive longest edge refinement (lower is better). The x-axis represents the number of elements in the mesh (2D), and the y-axis the quality metric. Left α, Right τ.

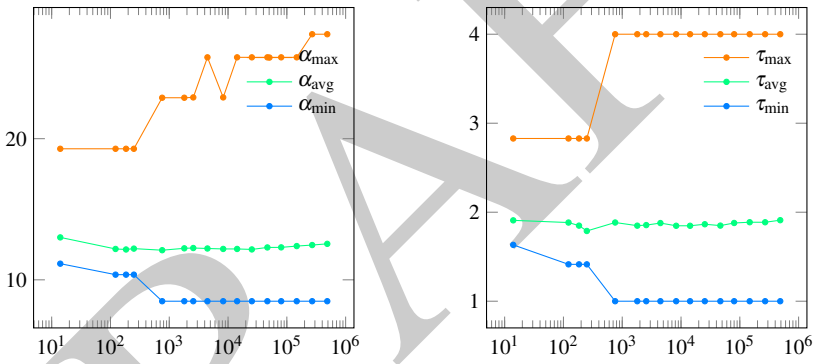

Figure 5: Quality of recursive longest edge refinement (lower is better). The x-axis represents the number of elements in the mesh (3D), and the y-axis the quality metric. Left α, Right τ.

5 Quality

For measuring the condition of an element S we use the following metrics:

$$\alpha(S) = \frac{\left(\sum_{i=1}^{D+1} \text{vol}(\text{side}_i(S))/(D+1)\right)}{\text{vol}(S)}$$

$$\tau(S) = \frac{\max_{i=1}^{D+1} \text{vol}(\text{side}_i(S))}{\min_{i=1}^{D+1} \text{vol}(\text{side}_i(S))}$$

Longest-edge bisection is known to generate stable and high quality refinements as shown includegraphics Figure 4, Figure 5, and Figure 6.

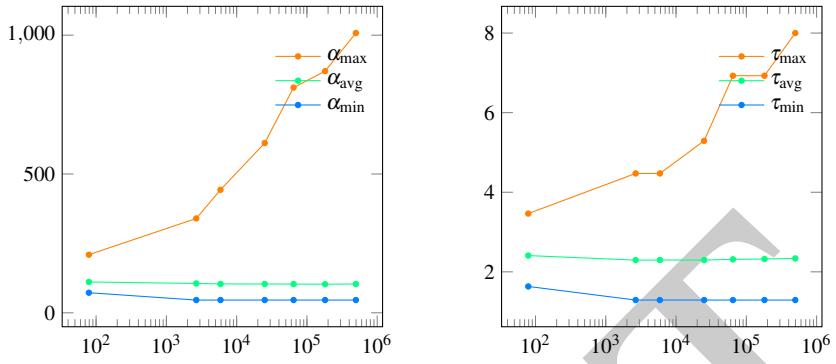

Figure 6: Quality of recursive longest edge refinement (lower is better). The x-axis represents the number of elements in the mesh (4D), and the y-axis the quality metric. Left α, Right τ.

6 Parallelization (work in progress)

Le E be an element with vertex indices $I(E) = \{v_1, v_2, ...v_{D+1}\}$.

A way of producing a conforming mesh is to ensure that there exists only one sequence for refining an element and its facets. We call two neighboring simplices having a facet in common "mates". Mates can be refined independently as long as the sequence of bisected edges is the same in their common facet.

The element refinement rule is based on the recursive longest-edge. However, the (rare) corner case where two edges have the exact same lenght is to be handled. For this reason we use the global identifier of the vertices of the mesh. Let $e = (a,b)$ be and edge where $I(e) \subset I(E)$ $I_g(a), I_g(b)$ are global vertex identifiers and $I_g(a) < I_g(b)$.

The set of edges associated with the element E are sorted with respect to their length and lexicographically using $I_g(e)$. Hence, a facet is refined the same sequence of bisections for both the "mates" because the selection will be performed with the exact same order.

This implies that set E can be split only once without requiring synchronization. Which leads us to the following parallel algorithm

1. Mark elements for refinement
2. Refine all marked elements and satisfy the compatibility chain as long as the elements in the chain have valid global indexing.
3. Σ is the edges for which the compatibility chain was not completed.
4. Determine global identifiers for the newly added vertices (global communication) and communicate edges that have been bisected in the interface between two processes (point-to-point communication).
5. If $\Sigma \neq \emptyset$ mark the elements incident to each $e \in \Sigma$ and go to 2.

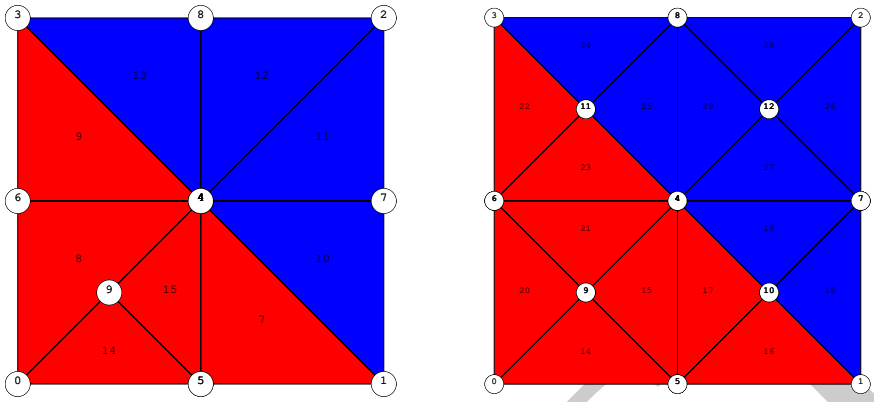

Figure 7: Two steps of the parallel algorithm. Left: refinement of marked elements. Right: completion of the compatibility chain.

7 Implementation

MARS is developed using object oriented design and meta-programming. All the important classes are parametrized with compilation time dimensions. With `Dim` we define the coordinate space dimension, and with `ManifoldDim`, with `ManifoldDim` $\leq$ `Dim`, we define the dimension of the manifold.

Here are the main classes

- `template<Integer Dim, Integer ManifoldDim> class Mesh`. The `Mesh` class stores the element and point information, and can construct side and boundary information on the fly.
- `template<Integer Dim, Integer ManifoldDim> class Simplex`. The `Simplex` class represent any type of elements supported by the library. For instance a `Simplex<3, 3>` is a tetrahedron volume element, and `Simplex<3, 2>` is a triangle surface element.
- `template<class Mesh> class Bisection`. The `Bisection` class allows to refine the mesh both uniformly and adaptively.

Here is an example work-flow:

Listing 1: Example refinement code

```
using namespace mars;

Mesh4 mesh;
read_mesh("path_to_4D_mesh.MFEM", mesh);
mesh.renumber_nodes();

Bisection<Mesh4> b(mesh);
b.set_edge_select(std::make_shared<LongestEdgeSelect<Mesh4>>());
```

```
//uniform refinement
b.uniform_refine(1);

//adaptive refinement
b.refine({marked_el_1, marked_el_2, ...});
```