Jaal:Engineering a high quality all-quadrilateral mesh generator

Chaman Singh Verma[1] and Tim Tautges[2]

[1] Department of Computer Sciences, University of Wisconsin, Madison, 53706
csverma@cs.wisc.edu
[2] Argonne National Laboratory, Argonne IL, 60439
tautges@mcs.anl.gov

Summary. In this paper, we describe the implementation of an open source code (*Jaal*) for producing a high quality, isotropic all-quadrilateral mesh for an arbitrary complex surface geometry. Two basic steps in this process are: (1) Triangle to quad mesh conversion using Suneeta Ramaswamy's tree matching algorithm and (2) Global mesh cleanup operation using Guy Bunin's one-defect remeshing to reduce irregular nodes in the mesh.

These algorithms are fairly deterministic, very simple, require no input parameters, and fully automated yet produce an extremely high quality all-quadrilateral mesh (with very few 3 and 5 valence irregular nodes) for large class of problems.

1 Introduction

There are many applications where an all-quadrilateral and all-hexahedral mesh is preferred over a triangle mesh (non-linear structural mechanics, higher order spectral methods, texture mapping etc). While many provably robust, high-quality triangle mesh generators have been described and are available as open source software (e.g. the Triangle package from [9]), to our knowledge, there are no provably robust all-quadrilateral mesh generation algorithms available in open source form. In this paper, we describe a robust, high-quality quadrilateral mesh generation algorithm freely available in source code form.

We seek to implement a robust, high-quality all-quadrilateral mesh generation algorithm capable of meshing 3D, non-simply-connected surfaces. This algorithm must be theoretically sound, to support robustness for a wide class of input domains. The algorithm must be capable of working from a pre-defined list of bounding edges, with the only constraint being that the number of edges is even (note, we do not constrain the evenness of intervals on bounding loops, only on the total number of bounding edges). The resulting meshes should have relatively few "defects" (internal nodes with degree other than four), and where possible, should be boundary-sensitive (with lower-quality elements located relatively far from boundaries). The algorithm should operate on 3D surfaces, possibly with multiple bounding loops, without reliance on an underlying 2D parameterization. Finally, the algorithm should be robust even for large relative size transitions in the bounding edges.

Quadrilateral meshing algorithms (if not unstructured meshing algorithms in general) can be classified in two groups. Indirect methods generate some other intermediate decomposition of the domain, e.g. a triangle mesh or a medial surface decomposition, afterwards converting the pieces to quadrilaterals through recombination or further decomposition. In contrast, direct methods generate quads directly, often using advancing fronts. When deciding which approach to use, two key observations guide us. First, we observe that geometric computations often interfere with the initial generation of quadrilaterals. This occurs frequently in advancing front methods, usually by missing the intersections of the advancing fronts, especially for problems with rapid size transitions. Second, virtually all unstructured meshing algorithms finish with a cleanup phase, where local topological modifications are used to improve the regularity of the mesh, sometimes making extensive changes to the initial mesh. This reduces the impact of poor initial mesh quality, and guides us to focus more on "closing" the initial mesh, no matter the initial quality. Hence, the approach we take is to generate the initial quadrilateral mesh using an indirect method, by converting triangles to quads. As we show later, this is a provably reliable method for obtaining the initial mesh. This mesh is subsequently cleaned up using topological transformations. The result is an algorithm with provable robustness that in practice generates high-quality all-quadrilateral meshes.

This paper is organized as follows. In section 3.1, we describe a tree matching algorithm and in sections 4 and 5, we describe some local and global operations for mesh cleanup. In section 6, we show results for several input problems, followed by conclusions of this work.

2 Previous work

The Q-Morph algorithm [22] transforms a given triangle mesh into a quadrilateral mesh using an *advancing front* method. In this approach, quadrilaterals are formed using existing edges in the triangulation, by inserting additional nodes, or by performing local transformations to the triangles. The final mesh quality is improved by topological clean-up and local smoothing operations. This approach has been implemented by several commercial meshing packages, but is not available in open-source form.

Marcelo's [23] *CQMesh* software converts a given triangulated mesh into an all-convex quadrilateral mesh, but the resulting mesh has many irregular nodes. QMPP is an extension to CQMesh that improves the quality at the boundaries, but topological improvements in the interior are limited. The first part of our work is greatly influenced by this approach.

Betul et al. [24] developed a direct quadrangulation algorithm with guaranteed angle bounds (18.43-171.86 degrees). A circle packing algorithm by Bern et al. [19] constructs quadrilaterals in a polygonal domain with an upper bound of 120 degrees in the interior, but provides no bounds on the smallest angle. These two theoretically proven algorithms are direct algorithms and work only for 2D planar surfaces. To the best of our knowledge, theoretically, nothing is proven for 3D surface quadrangulation.

Spectral methods[20] provide a novel approach for quadrangulating a manifold polygonal mesh using Laplacian eigenfunctions which are the natural harmonics of the surface. The surface Morse functions distribute their extrema evenly across a mesh, which connect via gradient flow into a quadrangular base mesh (known as a Morse-Smale Complex). An iterative relaxation algorithm refines this initial complex to produce a globally smooth parameterization of the surface. From this, well-shaped quadrilateral mesh with very few defects are generated. Although very elegant, this approach has many implementation problems: (1) The quality of mesh depends on an appropriate morse function, whose choice is often heuristic and, if poor, leads to high numbers of critical points; (2) calculations of the first few (about 40) eigenvectors are very expensive and sometimes impractical for a large mesh. (3) Tracing the curves from maxima to minima via saddle points may have geometric robustness issues.

Felix et al. [7] generate a high quality quadrilateral mesh using global parameterization which is guided by a user-defined frame field (often the principal curvature directions). These frame fields simplify to vector fields on the covering spaces, so that the problem of parameterization with frame fields reduces to the problem of finding a proper integrable vector field on the covering surface. Similarly, Bommes et al. [4] formulated the quadrangulation problem as two step process (cross field generation and global parameterization), both formulated as a mixed-integer problems. This scheme allows placement of singularities at geometrically meaningful locations, and produces meshes with favorable orientation and alignment properties. However, the method still requires the solution of a numerical problem, which can be expensive.

Bommes et al. [5] developed an algorithm that optimizes high-level structure of a given quadrilateral mesh, especially for helical structures. Their algorithm is able to detect helical structures and remove most of them by applying grid preserving operators.

Hormann et al. [10] presented an algorithm that converts an unstructured triangle mesh with boundaries into a regular quadrilateral mesh using global parameterization that minimizes geometric distortion. The second part of our work to reduce irregular nodes is also based on *remeshing patches*, but we do this on topologically convex patches instead of using global parameterization.

In the area of topological cleanup operations, many methods have been presented. Canann [2] and Kinney [18] present ever-growing numbers of operations, usually applied using an isomorphism approach. In contrast to those, Bunin [8] presents a very elegant technique for removing defect vertices based on patch replacement, whose results are better than those of the isomorphism approach while also being vastly simpler to implement. We make heavy use of this technique in the present work.

3 The JAAL All-Quadrilateral Meshing Algorithm

We describe an open source, all-quadrilateral mesh generator that uses an indirect approach. The input to the algorithm is any triangulated surface; the output is an all-quadrilateral mesh with the following features:

1. The mesh may have very few or no irregular nodes.
2. All the quad elements are convex.

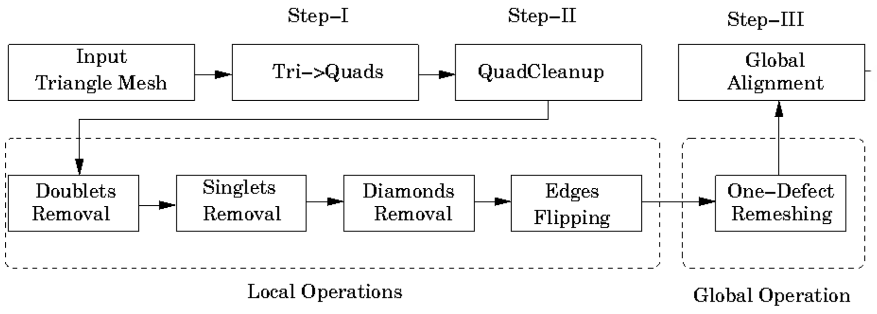

Fig. 1. Flowchart of Jaal Quadrilateral Mesh Generation

3. All the interior nodes have degree range [3-5]. The boundary nodes have degree range [1-4].

The flowchart of this algorithm is shown in Figure 1. In the step-I, triangles are combined to form quadrilaterals and in the step-II and III, geometric and topological improvements are carried out on a topologically valid quadrilateral mesh.

This algorithm relies on two fundamental ideas, tree matching and patch remeshing, which are described in following sections. When combined with other freely available components for triangle meshing, geometric evaluation, and mesh smoothing, the result is an open-source, all-quadrilateral meshing algorithm for 2D/3D surfaces of arbitrary topology. We refer to this algorithm using the name JAAL.

3.1 Tree Matching Algorithm (TMA)

Converting triangles into quadrilaterals essentially involves combining adjacent pairs of triangles to form a single quadrilateral. Greedy algorithms, although simple, may leave many unmatched triangles in the mesh, therefore some heuristics are used to maximize number and quality of the quads. For an example, *QMorph* uses advancing front method to combine triangles into quads[22].

One of the most important algorithms in combinatorial graph theory is *Edmond's perfect matching algorithm* [6]. Presently, the fastest known implementation of Edmond's algorithm is from Micali and Vazirani [17] that runs in $O(EV^{1/2})$ time complexity. For binary trees (derived from the dual graph of a triangulation), Suneeta [23] proposed the *Q-percolation algorithm* that has time complexity of $O(n)$. For many theoretical results, algorithms, and complete analysis, we recommend her original paper [23]. The simplicity in the Q-percolation is achieved by inserting *Steiner points* in the triangle mesh to allow perfect tree matching. In Table 1, we have compared the efficiency of general graph matching (available in the Boost library [1]) with our own implementation of the *Q-percolation* algorithm. These times clearly demonstrate that Q-percolation significantly outperforms the general algorithm for trees originating from a triangulated mesh. Table 2 shows that *Q-percolation* also introduces fewer than four percent steiner points.

Suneeta describes two algorithms for tree matching, a simple but non-optimal tree matching algorithm that considers local matching rules, and a theoretically optimal but complex algorithm that uses non-local patches to generate all convex quadrilaterals. We use the non-optimal approach for three reasons:

- Even with the optimal tree matching solution, in most cases, the resulting mesh will still be far from optimal in terms of defect vertices. For 3D surfaces, rules for obtaining a geometrically optimal quadrangulation may be difficult to pose, and expensive to evaluate.
- Experiments have shown that a non-optimal tree matching produces fewer than four percent steiner points, which is not a significant increase in the total number of vertices.
- The non-optimal solution is extremely fast, very easy to implement, and requires no geometric information (which is important for 3D surfaces).

Table 1. Boost Graph Matching v/s Jaal Q-Percolation Performance

# Input triangles	Graph Matching(sec)	Jaal Tree Matching (sec)
10^3	9.73E-03	5.95E-03
10^4	1.88E-01	9.8E-02
10^5	25.36	1.44
10^6	2423.23	25.64

Table 2. Experiments show that TMA introduces few steiner points

Nodes/Triangles	Nodes/Quads	# Steiner Points	% Nodes
542/1031	562/535	20	3.69
5106/10043	5306/5221	200	3.91
50433/100436	52346/52131	1913	3.79
501611/1001611	520419/519613	18808	3.74

4 Local Mesh Cleanup Operations

Although many different local mesh cleanup operations have been described [2, 18, 15, 13], we use only four local operations: *Singlet Removal*, *Doublet Removal*, *Diamond Removal*, and *Edge Flipping* (Figure 4). Unlike previous work, we do not attempt to order or prioritize these operations, as in our case, the main quality improver will be the patch-based defect removal described in the next section. We apply local mesh cleanup operations only as a means for removing well-known configurations that are always unacceptable, e.g. two quadrilaterals sharing two edges (a *doublet*). Although these local operations are simple, checks must be done so that (1) inverted elements are not created (2) doublets are not introduced. These operations are fairly inexpensive, as shown in Table 3. Among the four operations, *Edge flipping* is the most expensive for two reasons:

- The operation requires checking for inverted elements.
- Number of successful operations is, in general, very large and many sweeps over the entire mesh are essential to exhaust all the possibilities of flipping.

Experiments have shown that these local operations are quite effective and in a quadrilateral mesh generated from the Delaunay triangulation, as much as half of the irregular nodes are removed from the mesh. Also with the edge flipping operations, we can ensure that the output mesh with the lower bound of vertex degree three and an upper bound of vertex degree five is achieved. For the boundary nodes, interior angle is used to enforce degree.

Table 3. Performance of local operations

Quads	Singlets	Doublets	Diamonds	Edge Flipping
65000	5 0.128s	500 0.0725s	3000 0.1772s	3327 2.651s
701193	1 1.494s	4450 0.5460s	32443 2.2503s	36034 32.44s

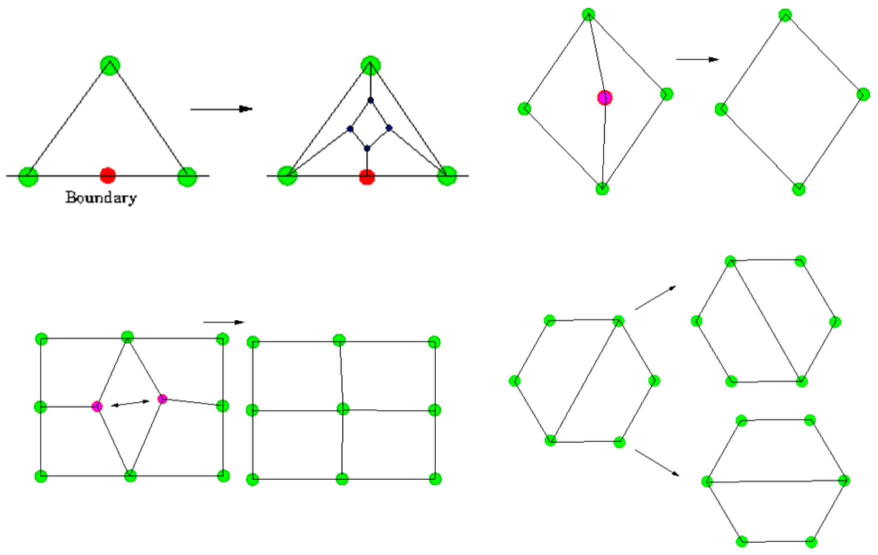

Fig. 2. Removal of a boundary singlet (top left), doublet (top right), diamond (bottom left), and edge flipping (bottom right).

5 Defect Removal by Patch Replacement

Although Bunin described algorithms for a variety of defect types, we have found that considering of only 3-, 4-, and 5-sided patch regions to be sufficient, while also being straightforward to implement.

5.1 3- and 5-sided patch remeshing

3- and 5-sided patches are broken into three and five quadrilateral regions, respectively, as shown in Figure 3, with one irregular node of degree three or five at the intersection. Since the number of intervals on the patch boundary is fixed, the defects in a given patch can be replaced by a single defect if there is an everywhere-positive solution to the corresponding interval matching problem for the patch. For a 3-sided patch, the corresponding matrix problem is

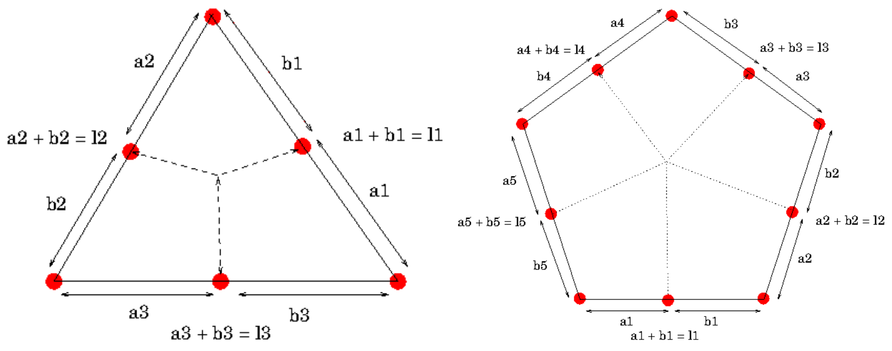

Fig. 3. Template of 3-sided (left) and 5-sided (right) patches.

$$\begin{bmatrix} 0 & 0 & -1 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ b_1 \\ b_2 \\ b_3 \end{Bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ l_1 \\ l_2 \\ l_3 \end{bmatrix}$$

The problem for 5-sided patches is similar (see [8] for details). Note that the inverse matrix for this problem can be pre-computed, since it is always the same for 3- and 5-sided patches. Figure 4 shows examples of 3- and 5-sided patch replacement.

5.2 4-sided patch remeshing

For 4-sided patches, there are two possibilities, shown in Figure 5. In a regular patch, opposite sides have equal number of nodes, and the patch can be replaced with a simple structured mesh. In an irregular patch, sides have unequal number of nodes; the patch can be broken into two quadrilateral patches and one 3-sided patch.

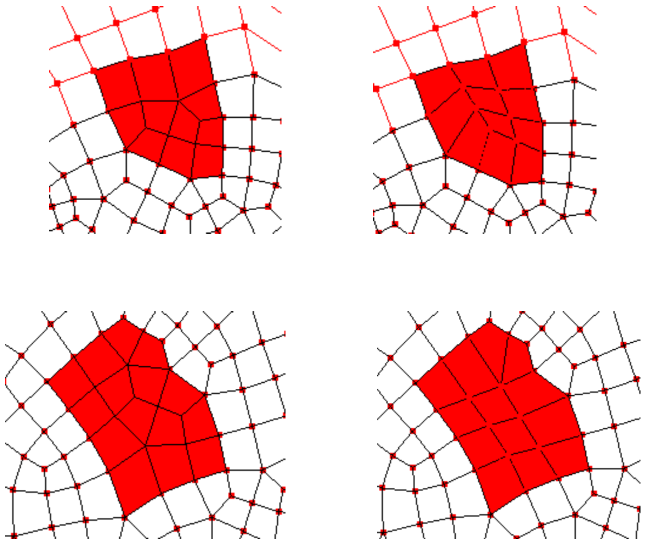

Fig. 4. Remeshing 3-sided (top) and 5-sided (bottom) patches.

The resulting quadrilateral patches are remeshed using a regular 4-sided algorithm, with the 3-sided region replaced using the algorithm described in Section 5.1. 4-sided patches can be remeshed only if the interior 3-sided patch is remeshable, as described in Section 5.1. An example of this type of patch replacement is shown in Figure 6.

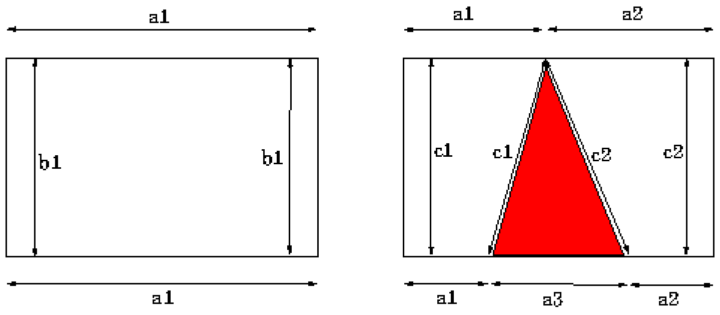

Fig. 5. Remeshing a 4-sided loop. Regular quad patch (left); irregular quad patch (right).

5.3 Patch Identification

Therefore, the crux of the problem is to identify all the remeshable patches. We now present pseudo-codes to explain the entire one-defect remeshing.

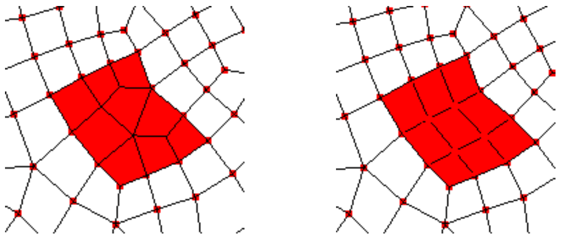

Fig. 6. Remeshing 4 Sided loop

```
     void remesh_defective_patches( Mesh *mesh)
         begin
1.              identify_boundary_nodes(mesh);
2.              for( i = 4; i < 12; i++)
3.                  max_faces_allowed = pow(2,i);
4.                  while(1)
5.                      ncount = 0;
6.                      IR = build_irregular_nodeset(mesh);
7.                      while( !IR.empty() )
8.                          Vertex *vertex = IR.top();
9.                          IR = IR - vertex;
10.                         patch = build_one_defect_patch( mesh, vertex)
11.                         if( patch )
12.                             if( patch.remesh() == 0)
13.                                 IR = IR - patch.get_all_irregular_nodes();
14.                                 IR = IR + patch.get_new_defective_node();
15.                                 ncount++;
16.                     if( ncount == 0) break;
17.      end
```

Line 1: Here we identify all the boundary nodes and tag them as *boundary*. Global operations are valid only for the internal nodes.

Line 2: In this loop we set the maximum number of faces that a patch can have. We start with small number of faces, and once all the possibilities have exhausted, we increase the patch size. Many experiments have shown that 5000 elements are more than sufficient for reasonably good quality mesh. Large patches are generally avoided on highly curved 3D surfaces.

Line 4: We start the loop of global operation. This while loop exit only when no global operation is performed inside the loop. For this, ncount is initialized to zero at line 5 and checked at line 16.

Line 6: We build all the irregular nodes (internal) having valency of 3 or 5. (Assuming that the input nodes have valency 3,4 or 5).

Line 7: Loop starts for every irregular node in the set.

Line 10: Try to build a remeshable patch at the given irregular node. (This is explained in the next pseudo code).

Line 12: Try to remesh the patch. Remeshing may fail, if it results in any inverted element.

Line 13: A successful patch may contain many irregular nodes, remove all of them from the set.

Line 14: There will be at the most one irregular node in the remeshed region, insert it into the set and continue.

```
    Patch build_one_defect_patch( Mesh *mesh, Vertex *v)
        begin
1.          initPath = dijkstra_shortest_path(mesh,v);
2.          faces = initial_blob( initPath );
3.          while(1)
4.              if( faces.size() > max_faces_allowed) return NULL;
5.              bn = get_boundary_nodes( faces);
6.              topo_convex_region = 1;
7.              for( i = 0; i < bn.size(); i++)
8.                  if( is_internal_node( bn[i] )
9.                      topo_angle = get_topological_outer_angle( bn[i] )
10.                     if( topo_angle < 0)
11.                         expand_blob( bn[i] )
12.                         topo_convex_region = 0
13.             if( topo_convex_region)
14.                 if patch345_meshable( bn ) == 0)
15.                     return new Patch( faces, bn);
16.                 for( i = 0; i < bn.size(); i++)
17.                     expand_blob( bn[i] )
```

Line 1: Starting from a given irregular node, search for another irregular node in the mesh using *Dijkstra's shortest path* algorithm and record all the nodes in the path. This initial path is the skeleton from where expansion will start.

Line 2: Collect all the faces around the skeleton, these form an initial *blob*.

Line 3: Blob expansion loop starts.

Line 4: If the blob size is more than specified, we just return with no patch found.

Line 5: Collect all the outer nodes of the boundary of the current patch.

Line 9: For each boundary node, topological outer angle (as defined in Guy Bunin's paper) is calculated, and if it is negative (line 10), more outer elements surrounding the vertex are added to the blob (line 11). The continues till all the boundary nodes of the patch have valid topological angle.

Line 14: Identify the corners of the valid patch. We support patches having 3, 4 or 5 sides only. If a valid patch is found, check if is remeshable by solving linear equations. If the patch is remeshable, return the patch with all the faces, boundary nodes and corners. This patch will be remeshed by the callee program.

Line 15: If the patch is not remeshable (i.e. no integer solution to the linear equations) then expand the blob by including more elements at the boundary and continue.

6 Results

All the development, testing, and evaluation were done on Ubuntu11.04 running on Intel Quad-Core processors with 4GM RAM. For mesh quality [11], we plotted Min-Max angles and aspect ratio distribution. We used the Delaunay triangulation as an input triangle mesh using *Triangle* [9] software. Experiments have shown that the Delaunay triangulation generate approximately half irregular nodes. In most of the cases, local operations are capable of reducing irregular nodes to almost half as shown in the table . After the global operation, very few irregular nodes remain in the mesh. Some experimental results are shown in 9, 10, 11, and 12.

6.1 Mesh smoothing

Our implementation is conservative in the sense, that we do not allow creation of any inverted element and then use expensive untangling algorithms. Convex regions increases the chances of successful operations. Experiments have shown that standard Laplacian smoothing is less effective in correcting concave regions as shown in Figure 8, therefore we use *Quasi-Newton* non-linear mesh optimization supported by Mesquite software.

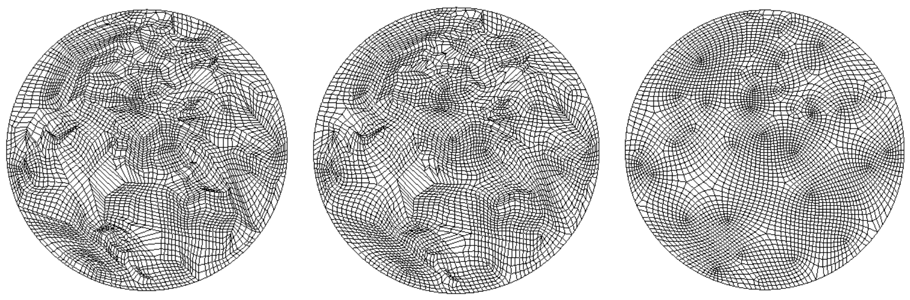

Fig. 7. (A) Mesh after one global operation (B) Standard Laplacian smoothing (C) Non-linear smoothing by Mesquite software

6.2 Power of simple global operation

Converting a triangle into three quadrilateral is probably the simplest algorithm for quadrangulation (known as Berg's algorithm). Although this has complexity of $O(n)$, in literature, this approached has been shunned because it (1) increases the mesh complexity by three times (2) createas large number of irregular nodes. We used this algorithm for stress testing for both local and global operations.

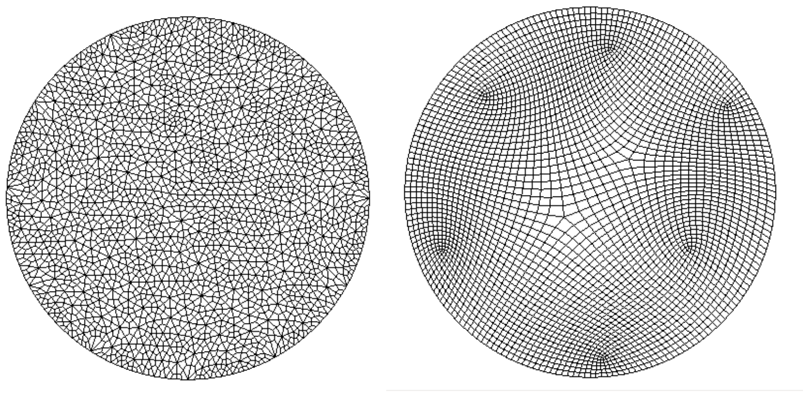

Fig. 8. (A) Quadrangulation using Berg's algorithm(B) Meshclean up using JAAL

Table 4. Performance results of Quadrangulation operations(F:#faces, I:# irregular nodes)

Operations	Dataset I	Dataset II	Dataset III
Quads/Irregular nodes	F:5088 I:2535	F:10316 I:5097	F:51323 I: 25317
Local Operations			
Doublet Removal	1.68E-03	9.993E-03	5.24E-02
Singlet Removal	3.44E-03	1.073E-02	9.71E-02
Diamonds Removal	1.91E-03	4.032E-03	2.05E-01
Edge Flipping	2.38E-01	3.014E-03	2.9296
Shape Optimization	0.2407	1.0	33.0
Quads/Irregular nodes	F:4732 I: 1091	F:9596 I:2280	F:48093 I: 11262
Global Operations			
# 3-Sided Patches	273	441	25875
# 4-Sided Patches	57	104	5507
# 5-Sided Patches	88	195	11305
Time for searching Patches	3.363	9.9462	800
Time for remeshing patches	8.80E-03	1.9E-02	30.0
Shape Optimization Time	2.92E-02	0.50	120.0
Quads/Irregular nodes	F:4144 I:4	F:9779 I:4	F:38036 I:4

7 Software Implementation

The entire software is written in C++ and part of the **MeshKit** software. (Toolkit for mesh generation) which can be freely downloaded from:
https://trac.mcs.anl.gov/projects/fathom/wiki/MeshKit. In addition, we used the following modules (all source codes are freely available).

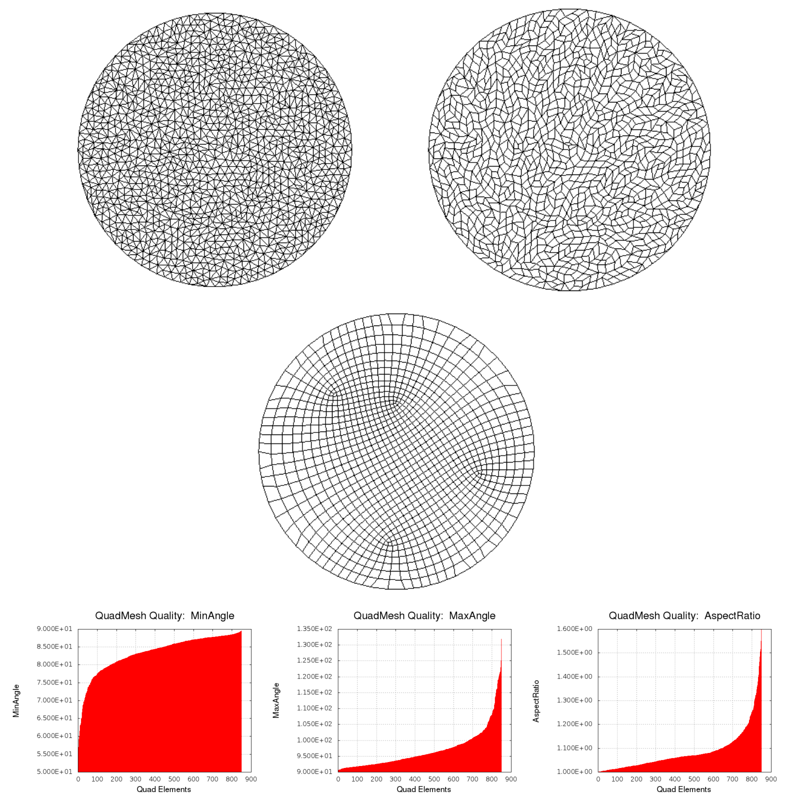
QuadMesh Quality: MinAngle
MinAngle
Quad Elements
QuadMesh Quality: MaxAngle
MaxAngle
Quad Elements
QuadMesh Quality: AspectRatio
AspectRatio
Quad Elements

Fig. 9. Result-I: Quadrangulation in a disk produces 4 irregular nodes. (A) Initial triangulation (B) Quadrangulation after tree matching (C) Quadrilateral mesh after mesh cleanup.

1. **MOAB:** This is a component for representing and evaluating mesh data. With this software, various relationships among the mesh entities can be stored, modified and queried very efficiently.
2. **CGMA:** This software provides all the geometric functionalities that are needed for mesh generation. We used this software for geometric queries and projecting vertices on the model.
3. **Mesquite:** This software provides sophisticated non-linear algorithms for mesh shape optimization. This is the most critical component in the Jaal software pipeline.
4. **Verdict:** This software is used for measuring various mesh elements qualities.

8 Conclusion

We have engineered two algorithms to develop an open-source code for all-quadrilateral mesh for 2D/3D surfaces using indirect approach. We have done extensive exper-

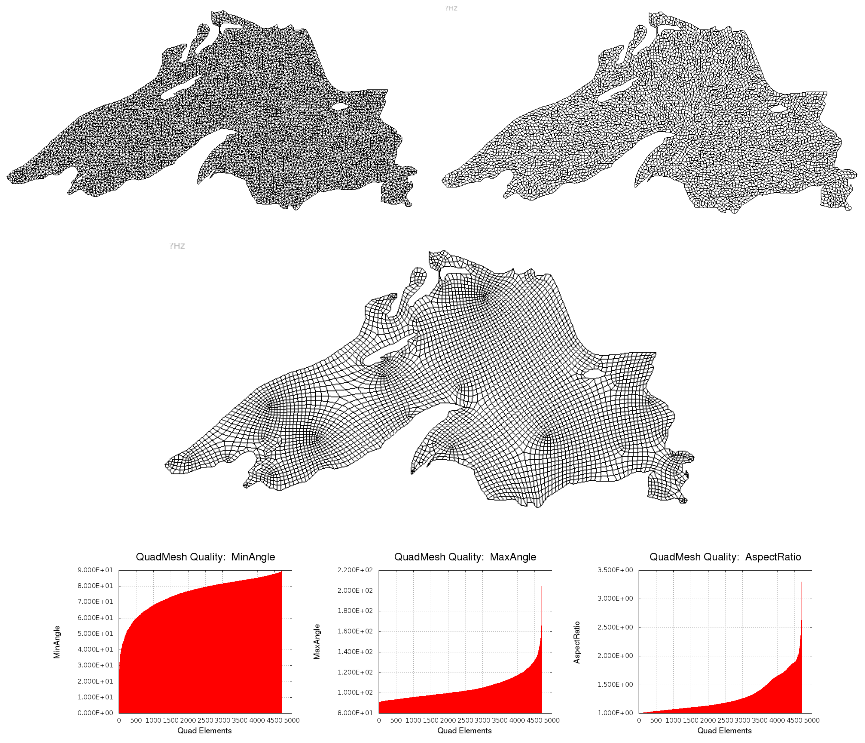
QuadMesh Quality: MinAngle
MinAngle
Quad Elements
QuadMesh Quality: MaxAngle
MaxAngle
Quad Elements
QuadMesh Quality: AspectRatio
AspectRatio
Quad Elements

Fig. 10. Result-II: Quadrangulation of Lake Superior model. (A) Initial triangulation (B) Quadrilateral mesh after tree matching (C) Quadrilateral mesh after mesh cleanup.

iments with the present software and conclude that a large class of surfaces can be quadrangulated with high quality using very simple operations (four local and one global). The entire pipeline is fully automatic and require no user's parameters. Both geometrically and topologically, the final mesh quality, in all cases, exceeded our initial expectations. Still, there are scopes for further improvements:

- The biggest performance bottleneck in the pipeline is *Patch Searching* which is purely heuristic. It is quite possible that other approaches and data structures may have less time complexity than what we have presented.
- The present code has no control over spatial distribution of irregular nodes. For good quality mesh, irregular nodes must be deep inside the domain and evenly distributed.
- At present, the code does not support constrained quadrangulation, which are essential for feature preservation.

9 Future Work

We plan to extend the current work in the following directions:

- *Parallelization :* The presented algorithm is inherently parallelizable on both shared memory and distributed memory architecture machines. In the immediate future, we will implement this algorithm using Intel Thread Building Blocks(TBB) library for multi-core machines and Message Passing Interface (MPI) for large scale problem on distributed memory machines. Transactional memory paradigm have also been successful in parallelization of incremental Delaunay triangulation(IDT) [16]. Since our global cleanup algorithm is very similar to IDT, therefore, scalable implementation will be quite interesting development.
- *Non-geometrical models :* The present method will be extended to handle arbitrary polygonal models where the underlying geometry is not known.
- *Feature preservation:* Not preserving user defined edges is one of the biggest weakness of the present algorithm. This can be done in either step-I or step-II, which we will explore.

References

1. Boost Graph Libary: http://www.boost.org/doc/libs/1_37_0/libs/graph/doc/maximum_matching.html
2. S. Canann, S. Muthukrishnan, and R. Phillips (1998). Topological Improvement Procedures for Quadrilateral Finite Element Meshes. *Engineering with Computers*, 14:168177,
3. J. Daniels, M. Lizier, M. Siqueira, C. T. Silva, and L. G. Nonato. Template Based Quadrilateral Meshing. Computers and Graphics, Volume 35, Issue 3, June 2011, Pages 471-482 Shape Modeling International (SMI) Conference 2011
4. David Bommes, Henrik Zimmer, Leif Kobbelt (2009), Mixed-Integer Quadrangulation, *ACM Transactions on Graphics (TOG)*, 28(3), Article No. 77.
5. David Bommes, Timm Lempfer, and Leif Kobbelt(2011), Global Structure Optimization of Quadrilateral Meshes, Eurographics 2011
6. Edmonds, Jack (1965). "Paths, trees, and flowers". Canad. J. Math. 17: 449467
7. Felix Klberer and Matthias Nieser and Konrad Polthier (2007) QuadCover Surface Parameterization using Branched Coverings. EUROGRAPHICS 2007, Volume 26, Number 3
8. Guy Bunin (2006) Non-Local Topological Cleanup ,15th International Meshing Roundtable.
9. Jonathan R. Shewchuk. Triangle: A Two dimensional quality mesh generator and Delaunay triangulator, http://www.cs.cmu.edu/ quake/triangle.html
10. Kai Hormann, and Gunther Greiner. Quadrilateral Remeshing Proceedings of Vision, Modeling, and Visualization 2000, pages:153-162, Editors: B. Girod and G. Greiner and H. Niemann and H.-P. Seidel
11. Knupp, P. (2000): Achieving Finite Element Mesh Quality via Optimization of the Jacobian Matrix Norm and Associated Quantities. International Journal for Numerical i Methods in Engineering 48(8), 11651185.
12. Marcin Mucha and Piotr Sankowski (2004), Maximum matchings in planar graphs via Gaussian elimination. in Algorithmica.
13. Mark W. Dewey, Steven E. Benzley, Jason F. Shepherd, and Matthew L. Staten, Automated Quadrilateral Coarsening by Ring Collapse.
14. Marshall Bern (1997) Quadrilateral meshing by circle packing, International Journal of Computational Geometry and Applications,7–20

15. Matthew Staten and Scott A. Canann (1997) Post Refinement Element Shape Improvement For Quadrilateral Meshes, 220 Trends in Unstructured Mesh Generation, ASME,9–16
16. Milind Kulkarni, L. Paul Chew, Keshav Pingali. Using Transactions in Delaunay Mesh Generation. https://engineering.purdue.edu/ milind/docs/wtw06-slides.pdf
17. Micali, Silvio and Vazirani, Vijay V., An $O(v|v|c|E|)$ algoithm for finding maximum matching in general graphs, Foundations of Computer Science, 1980., 21st Annual Symposium on.
18. Paul Kinney (1997) CleanUp: Improving Quadrilateral Finite Element Meshes,6th International Meshing Roundtable, 449–461
19. Marshall Bern, David Eppstein, Quadrilateral Meshing by Circle packing. International Journal of Computational Geometry and Applications,1997, pages7–20
20. Shen Dong and Peer-timo Bremer and Michael Garland (2006), Spectral surface quadrangulation, ACM Transaction on Graphics, vol25, 1057–1066
21. Stefan Arnborg and Jens Lagergren (1991), Easy Problems for Tree-Decomposable Graphs. http://www.informatik.uni-trier.de/ ley/db/journals/jal/jal12.html#ArnborgLS91
22. Steven J. Owen and Matthew L. Staten and Scott A. Canann and Sunil Saigal (1998),Advancing Front Quadrilateral Meshing Using Triangle Transformations,
23. Suneeta Ramaswami and Pedro Ramos and Godfried Toussaint (1998) Converting triangulations to quadrangulations, Computational Geometry: Theory and Applications, 9:257-276.
24. F. Betual Atalay, Suneeta Ramaswami and Dianna Xu (2011) Quadrilateral Meshes with Provable Angle Bounds.
25. Tian Zhou and Kenji Shimada (2000) An angle-based approach to two-dimensional mesh smoothing, In Proceedings, 9th International Meshing Roundtable, 373–384

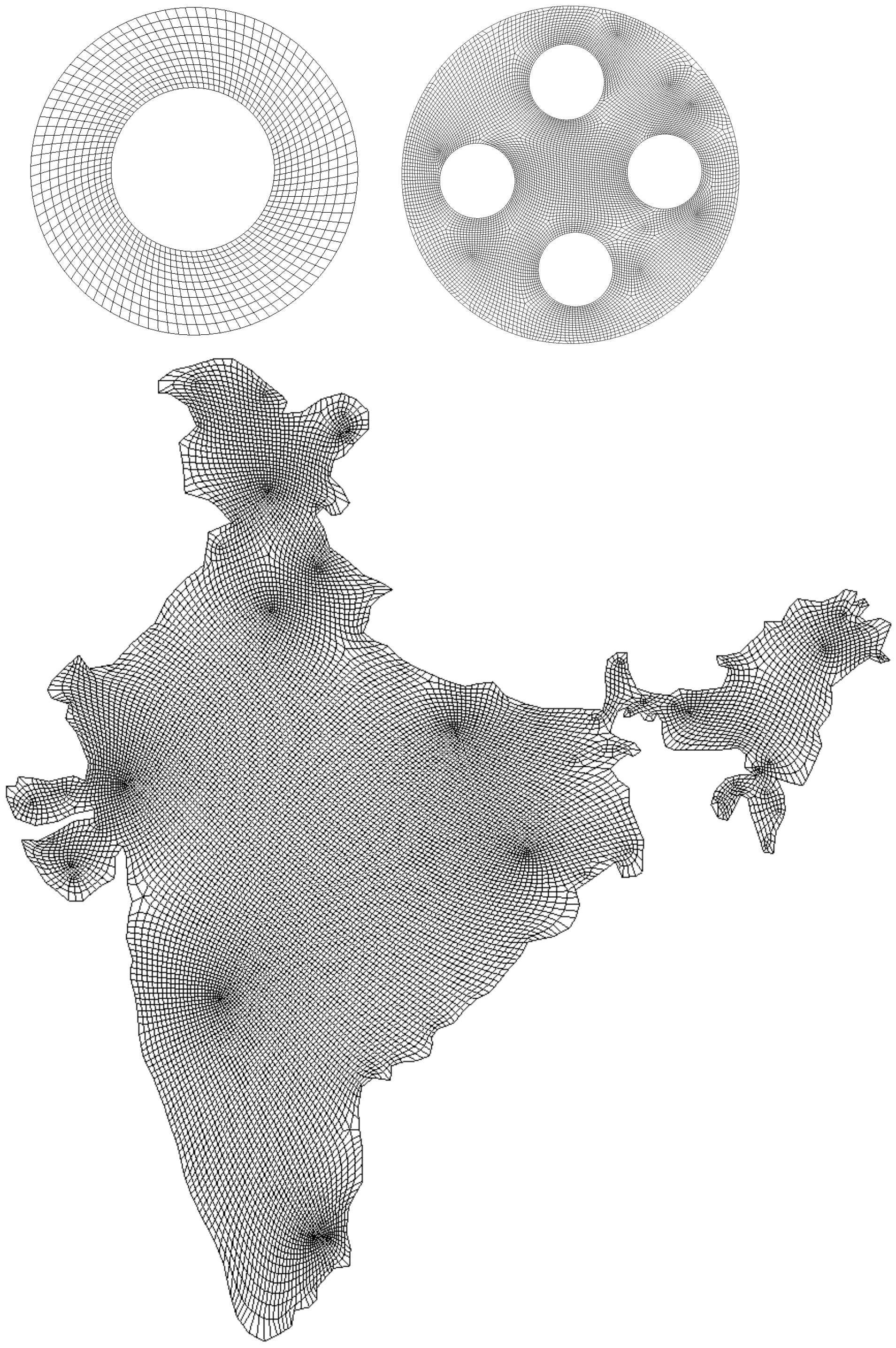

Fig. 11. Quadrangulation of 2D regions

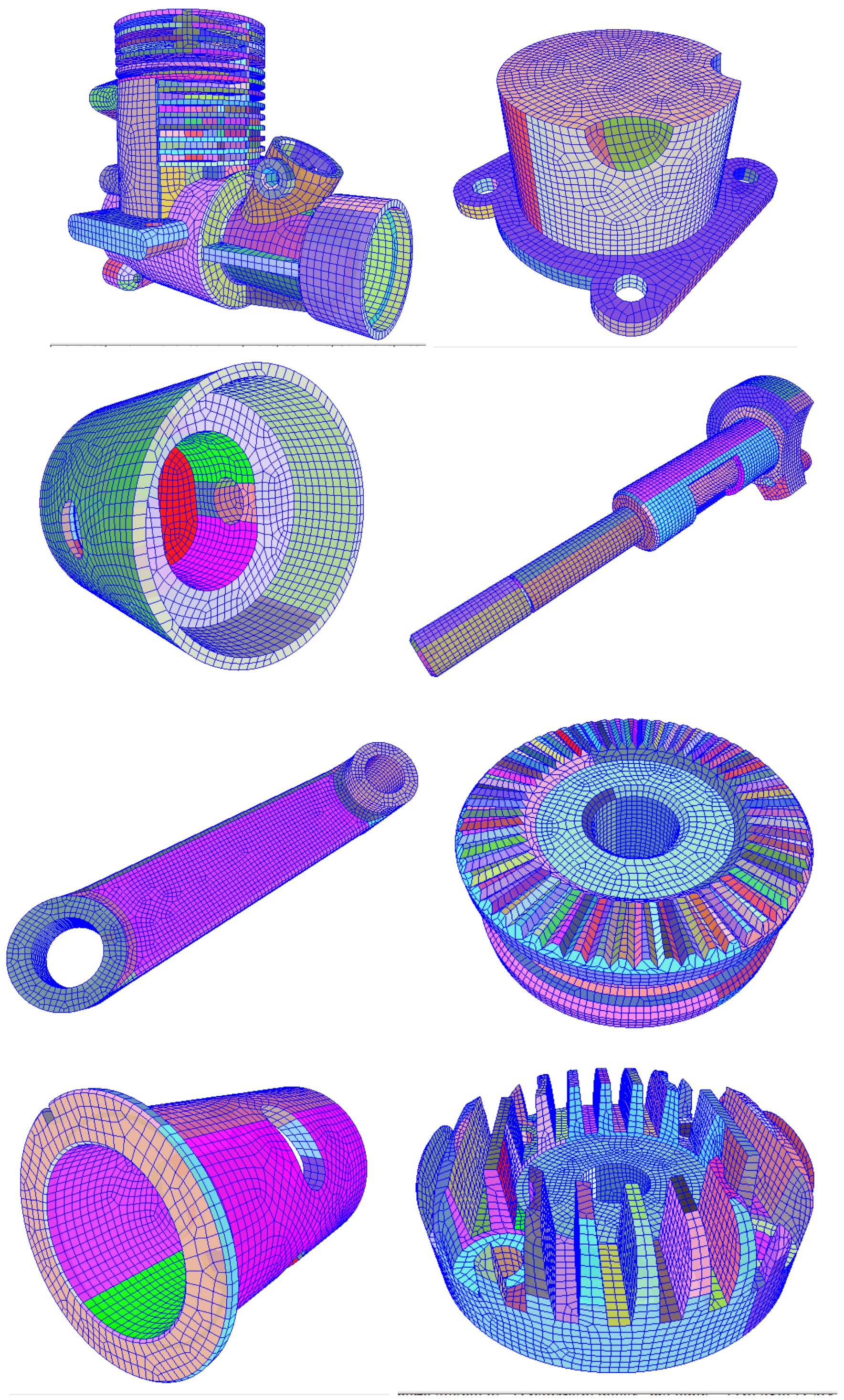

Fig. 12. Quadrangulation of 3D surfaces