A REGULARIZATION APPROACH FOR AUTOMATIC QUAD MESH GENERATION

Julia Docampo-Sánchez
docampo@mit.edu

Robert Haimes
haimes@mit.edu

Aerospace Computational Design Laboratory, Department of Aeronautics & Astronautics, Massachusetts Institute of Technology

ABSTRACT

Pure quadrilateral meshes are preferred when using shell-based structural analysis solvers since they provide more accurate results if compared to triangular or mixed meshes. Triangulations of complex trimmed surfaces (as constructed in CAD) can be always generated and any triangle can be subdivided into three quadrilaterals by splitting the sides and introducing a new vertex at the centroid. Therefore, the conversion of a triangular mesh into a fully quadded conformal mesh is straightforward, and if the source triangulation is watertight, the resultant quad mesh maintains that property. However, triangle splitting implies that the quads inherit the original triangle shapes and the resulting mesh presents a very large number of irregular vertices. This paper describes a technique that recovers a significant amount of irregular vertices by performing iterative topological changes on the mesh and employs a modified Laplacian method for adjusting the vertex coordinates. The algorithm is robust, fast and produces a surface mesh of a BRep (where all vertices are on the geometry) that it is completely quadrilateral and semi-regular suitable for structural analysis and possibly other surface-based PDE solvers.

Keywords: Surface Mesh Generation, Computational Geometry, Quadrilateral, Laplacian

1. INTRODUCTION

Automatic mesh generation plays a key role in design systems when converting CAD output for use within various discipline analyses. For quad meshes, this can be done by constructing quadrilaterals directly or transforming triangular meshes into quads. The advantages of using triangular meshes as input is that for general surfaces, as for many other geometries, the mesh can always be produced using triangles. For applications in numerical simulations, a suitable mesh represents the surface geometry correctly (where all the vertices live on the surface or trimming curve) and has the appropriate element shape ensuring geometric accuracy and stability for the underlying solver.

In addition, the mesh should be as regular as possible. That is, most of its vertices should have valence four. 3-valence vertices often produce flat elements. Higher valences on the other hand, imply having highly multivalued surface points (mesh vertices) since a vertex belongs to many elements. In both cases, this has a negative effect on the performance of the numerical scheme and should be addressed.

When using a triangle splitting approach for producing quads, the new mesh has an increased number of elements as well as irregular vertices. Further, depending on the triangle anisotropy the resulting quads shape becomes unsuitable. Hence, it is necessary to perform topological changes on the mesh targeting these problems and ensuring that the final tessellation becomes suitable for the desired application.

The methodology presented here produces pure quadrilateral meshes from a given triangulation. Starting with the triangle splitting procedure [1], it applies systematic topological operations that enable a significant improvement in the mesh quality. Our

results suggest that in general we obtain semi-regular meshes with less than 5% irregular vertices. Since all the topological changes are performed in the parametric space of the geometry, its applications extend to general surfaces in 3-space. This procedure has been integrated into part of the ESP software suite [2]. The results presented here employed the software's internal tessellator, but the algorithm is designed to convert triangular meshes in general.

2. QUAD MESH GENERATION

Generating quad meshes can be done directly using advancing front techniques [3, 4, 5] or defining quad partitions on the domain [6, 7, 8, 9, 10, 11]. On the other hand, one can start with a triangular mesh and transform it into quads by merging or splitting the triangles. When merging triangles, this process requires appropriate triangle pairing [12, 13, 14, 15] since the resulting quadrangulation might leave several triangles that have to be subsequently eliminated [16].

Alternatively, any triangle can be split into three quads directly using its medians and inserting an extra vertex at the centroid [1] (see Figure 1). This approach is computationally efficient and produces pure quad conformal meshes. However, it naturally increases the total number of elements and the resulting mesh inherits the source triangle quality which in many regions generates quads that are too sharp or too flat. On top of that, the splitting step adds irregularity to the mesh. To overcome these limitations, mesh coarsening techniques [17, 18] can be applied as well as performing topological changes in the mesh. For example, the Quad Mesh Simplification [19] technique which uses poly-chords and the fundamental operation of quad collapsing. Or combining several fundamental operations, namely: swapping, splitting and collapsing [20, 21, 22, 23, 24].

Finally, most automated mesh processes require a post-processing step which ensures a suitable vertex distribution. That is, there are no sharp or flat elements and the quad's aspect ratio as well as the internal angles are within an admissible range. For quad meshes derived from triangles this is a requirement considering that the quad conversion step usually produces meshes of very poor quality. In addition, changing the mesh topology further deforms the quad shapes. Therefore, once the desired quad configuration is obtained, the final stage in the mesh generation process is to readjust the vertex coordinates. This can be cast as an optimization problem solving for the quad internal angles [25], orientation or side sizes [26]. Alternatively, one can solve variations of the Discrete Laplacian [25] or the elliptic operator [27] through an iterative scheme. Although element metrics such as skewdness, aspect ratios and the Jacobian matrix are better enforced through optimization [28, 29], iterative schemes are computationally cheaper.

The mesh regularization technique proposed in this paper builds upon the clean-up strategy from [20]. Unlike in [20, 22], the initial mesh comes directly from an unstructured triangulation consisting of nearly 50% irregular vertices. Using information from a vertex and its neighbors, we extend the set of possible two (three) step local topological operations from [20]. Each operation involves removing (adding) at most two quads and always reduces the total number of irregular vertices. Further, we adopt the approach from [22] to move irregular vertex pairs along the mesh whenever this operation preserves the number of elements.

As the algorithm evolves, vertex coordinates are recomputed using a modified Laplacian solver. At each iteration, the quad aspects ratios and relative angles about the moving vertex are used to decide the new vertex coordinates. Physical coordinates are computed from the underlying surface parametrization ensuring that mesh points remain on the surface. This technique has been tested over a wide range of complex geometries including surface cut-outs and surfaces with singular points. Although it is a heuristic approach, the algorithm consistently produces all valid meshes with isolated irregular vertices. The computational cost associated to this mesh processing step is less than a minute even for meshes consisting of 10k elements.

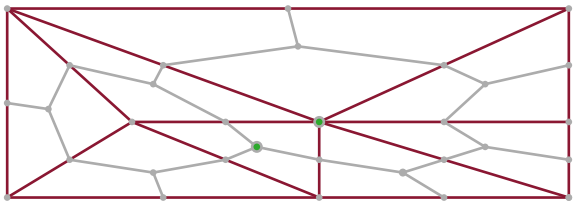


Figure 1: Catmull–Clark algorithm [1]: each triangle is split into three quads by inserting a new vertex at the centroid and creating three new sides (medians). The thicker lines (red) show the original triangles and the highlighted vertices indicate high/low valences.

3. MESH REGULARIZATION

Let us begin by briefly describing the Catmull – Clark [1] algorithm. Given a triangular tessellation, we split each triangle's side using the medians and insert a new vertex at all the triangle centroids. Then, as shown in Figure 1, three new quads are produced by linking the centroid location where the median intersects each side. The result is a fully quadded mesh but with many irregular vertices (valence $\neq 4$): the original vertices whose valence does not change, and the new vertices added at the centroids, which end up all having valence

three. On the other hand, note that the new vertices that were inserted along triangle sides nicely end up with a valence of 4 (2 from the side split and 2 from the connection of the centroids of the triangle neighbors).

Hence, once the initial quadrangulation has been produced, we apply a regularization technique that recovers as many valence 4 vertices as possible. To motivate the discussion, in Figure 2 we have produced an idealized mesh showing the mesh manipulation pipeline. The first step (local operations) which is discussed in Section 3 consists of applying the basic element operations of swapping, collapsing and splitting (and combinations of the same such as swap-collapse or double splitting) systematically. As it will be discussed later, it is necessary to have at least 3 irregular vertices for an overall improvement. Hence, for pairs of irregular vertices surrounded by regular quads, a type of vertex translation is applied allowing quads from far regions to interact with each other. As [22] demonstrated, it is possible to identify the shortest path between two isolated irregular vertices and bring them together by inserting a number of elements proportional to their graph distance. For complex geometries, it is not always possible to generate space for inserting new vertices whilst preserving mesh validity so the vertex translation is restricted to vertex pairs with valences $3, 5$ and in a particular configuration. This is discussed at the end of this section.

Once there are no more possible operations (or the mesh is fully regular), we apply a Laplacian based iterative scheme for recomputing the vertex locations (in the parameter space of the surface) and produce the final mesh.

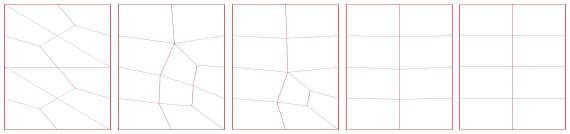


Figure 2: A simple quadded mesh obtained by triangle splitting and the evolution of the regularization algorithm including a final vertex coordinates computation.

We will start by detailing the basic operations followed by examples of compositions that allow for recovering regular vertices within a neighborhood (not all of them necessarily connected). We will use the following notation: v_i are quad vertices, $val(v_i)$ their valences and the pair $(a,\ b)$ means vertices with valences a and b respectively. Similarly, the group $\{a, b, c, d\}$ denotes the valences of an ordered quad.

Swapping: this process allows for exchanging high and low valence pairs by changing the vertex pair forming the common side of any two adjacent quads. If (v_1, v_2) was the common side to both quads which has now become (v_3, v_4), then:

$$val(v_i) = val(v_i) - 1,\ i = 1, 2, \tag{1}$$
$$val(v_j) = val(v_j) + 1,\ j = 3, 4. \tag{2}$$

Hence, as shown in Figure 3, the perfect swap occurs for a $(5, 5)$ and $(3, 3)$ pair, producing two regular quads. In practice, we don't require them to be perfect since, for example, swapping a $(5, 5)$ pair with a $(3, 4)$ gives $(4, 4)$ and $(4, 5)$ pairs, thus improving the overall mesh regularity.

Collapsing: eliminating a quad by merging two of its opposite vertices (see Figure 3). Given an ordered quad with vertices $\{v_1, v_2, v_3, v_4\}$, collapsing v_1 with v_3 results in:

$$val(v_2) = val(v_2) - 1, \quad val(v_4) = val(v_4) - 1 \tag{3}$$
$$val(v_{13}) = val(v_1) + val(v_3) - 2 \tag{4}$$

Therefore, the ideal collapse occurs for a $\{3, 5, 3, 5\}$ quad since $val(v_2) = val(v_4) = 5 - 1$ and $val(v_{13}) = 3+3-2 = 4$. Like for the previous case, this operation is suitable as long as there are three irregular vertices.

Splitting: inverse operation to collapsing and it is applied whenever there are high valence vertices linked to low valence vertices. Figure 3 shows a split which goes from three irregular vertices to one. In this case, the quad distance between the vertices dictate the final valence distribution. In Figure 4 we illustrate a split for a valence 6 vertex which can result perfect or not depending on the $(3, 3)$ pair configuration.

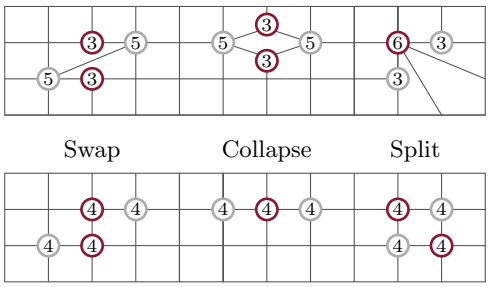


Figure 3: The three basic vertex operations showing how vertices gain / lose valences and become regular ($= 4$).

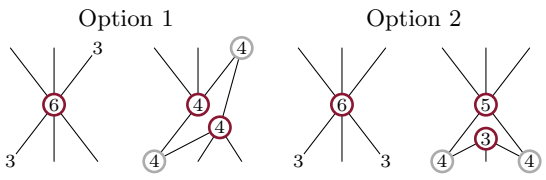


Figure 4: Splits depending on quad distances. Option 1 is perfect since vertices are 3 quads away but Option 2 produced a $(3, 5)$ pair because they are 2(4) quads away.

Using these operations alone leaves many irregular vertices behind. This is due to the fact that the remaining irregular vertices are either too far away or in a configuration for which none of the basic operations are suitable. However, by combining several operations as well as translating vertices, it is possible to improve the mesh quality even further, finally resulting in few or no irregular vertices. To illustrate this Figure 5 shows two stages of the regularization process: without vertex translation (b) and using vertex translations together with compositions (c). Notice that the final mesh has fewer elements but this is acceptable since we started with 3× the number of triangles.

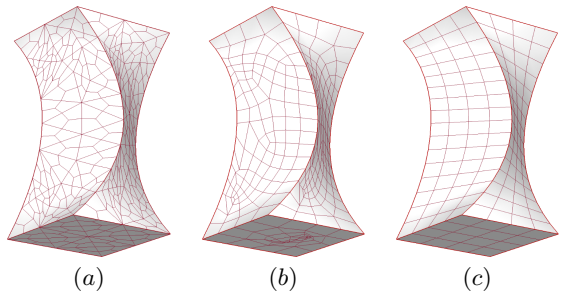


Figure 5: A twisted surface at the initial mesh (a), without transfering the vertex valences (b) and after applying full regularization with vertex translation (c).

Composition: we have seen that three or more irregular vertices need to exchange valences effectively in order to improve the mesh overall. However, the basic operations require that all the irregular vertices have to be contained within a quad and its adjacent neighbors, and in a precise configuration; suitable collapses need a vertex distribution where opposite vertices have low-low and high-high valences respectively. Swaps on the other hand, need the vertex pairs to be three vertex counts apart (clockwise or counter-clockwise). In Figure 6 we illustrate a vertex star centered at v_0 with all the quads and vertices that are stored within this datatype. Star groups are employed to detect three or more irregular vertices around a particular vertex (S). For example, the pair (S, v_2) cannot see vertex v_6 from quad q_1 but star S sees the three irregular vertices. This information is used to perform the operations shown in Figures 7, 8, 9 ,10, 11.

```
struct {
int nV, nQ;
int *v, *q;
} star;
```

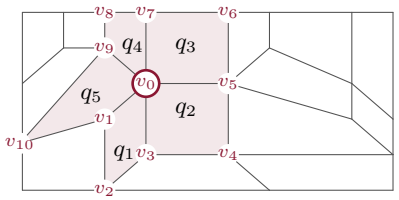


Figure 6: Vertex star $S(v_0)$ highlighting its surrounding quads and vertices stored clockwise in `*v, *q`.

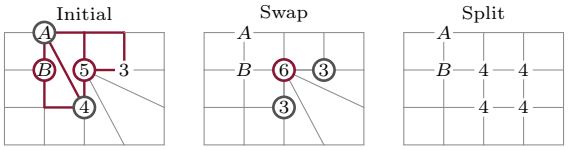


Figure 7: Swap-split process using three quads. Swapping increases the number of irregular vertices but produces an ideal scenario for a split. This operation is useful when $A = 5$ and $B \leq 4$ or when $A \geq 4$ and $B = 3$.

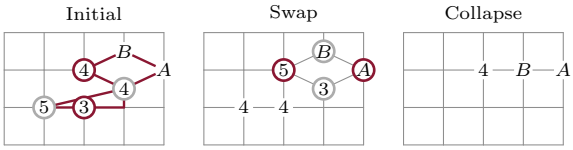


Figure 8: Swap-collapse using two irregular quads. We require that either $B = 3$ and $A \geq 4$ or $A \geq 5$ and $B \leq 4$. Then, after swapping, the top quad is almost fully irregular and can be collapsed.

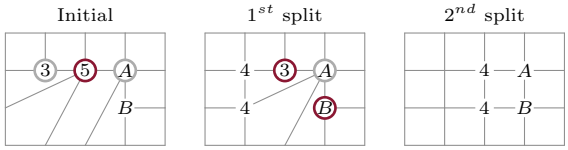


Figure 9: Double split: first split (5) using $(3, A)$ as links. Then, perform a second split in A through $(3, B)$ which are three quads distance. For this operation we need $A = 5$ and $B \leq 4$ or $A = 4$ and $B = 3$.

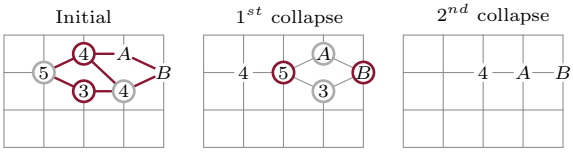


Figure 10: A double collapse: collapse the left quad $\{5, 3, 4, 4\}$. Now the adjacent quad has $\{5, 3, B, A\}$ valences and a second collapse is performed. We require that $A = 3$ and $B \geq 4$ or $B \geq 5$ and $A \leq 4$.

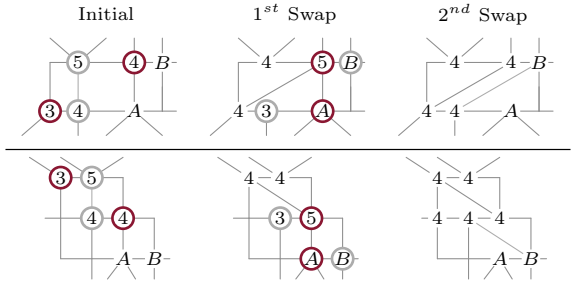


Figure 11: Double swaps. Adjacent quads (top) need $A = 5$ and $B \leq 4$ or $B = 3$ and $A \geq 4$. Diagonal swaps (bottom) need $A \geq 5$ and $B \leq 4$ or $B = 3$ and $A \geq 4$.

Translation: the swapping operation can be used to move irregular vertices along the mesh without varying the total number of elements (as opposed to collapse and split that remove/ add quads). For example, a $(4, 5)$ pair swapped with a $(3, 4)$ pair results in $(3, 4)$ and $(4, 5)$ valences. The total number of irregular vertices has not changed but their position has. In Figure 12 we show how we can take advantage of this operation to bring irregular vertices that were not in close proximity towards each other resulting in a regular region after two consecutive swaps. Vertex pairs of valences $(3, 5)$ can be moved indefinitely along the mesh. If the vertices $(3, 5)$ are linked, together they move *forward* (*backwards*). On the other hand, when they are opposite vertices (like in Figure 12) they move *upwards* (*downwards*). For pairs of vertices with valences $(3, 3)$ or $(5, 5)$, the swapping operation is more limited. It can be applied once to change their locations as shown in Figure 13. Unlike the $(3, 5)$ pairs, a second swap of $(3, 3)$ or $(5, 5)$ pairs would return to the original configuration.

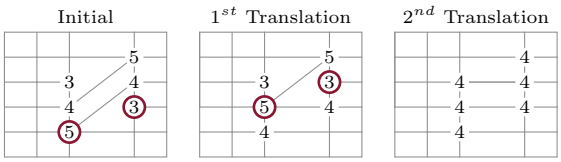


Figure 12: Translation of an irregular vertex pair $(3, 5)$ upwards along the mesh. After two movements, a suitable swap was found producing a regular region.

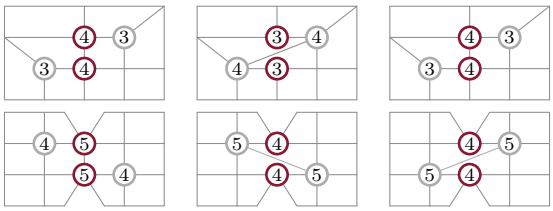


Figure 13: Possible swaps during vertex translation for vertex pairs with valences $(3, 3)$ and $(5, 5)$. The $(3, 3)$ pair has only one option whereas the other one has two.

4. MESH QUALITY

Changing the mesh topology can lead to vertex configurations with unacceptable element shapes: too flat or sharp and even self-intersecting. Hence, vertex coordinates are recomputed during regularization ensuring that the entire affected local region remains valid. The new vertex locations are computed using a modification of the Laplacian operator which is detailed in Algorithm 2 and will be discussed later. Note that this procedure is applied after each topological operation.

Figure 14 illustrates a valid vertex split; in the first image we show the three vertices involved in this operation. The middle image shows the resulting split where the highlighted vertices will be (possibly) moved when considering the step valid. The local region *in play* are the vertices involved in the split directly together with all their neighbors. Notice that surface boundary vertices are not moving since we require the tessellation to be watertight. The quads affected by a movement of any of these vertices ares the ones that will be checked and are highlighted as well. By restricting the number of moving vertices to this group, the operation remains local ensuring that we don't create invalid quads that will be overlooked. The rightmost image shows the final vertex distribution after moving all the necessary vertices. In the event that invalid elements remain, the topological operation is rejected and the mesh is restored from its previous valid configuration. In the following, we discuss how to detect invalid elements along the surface.

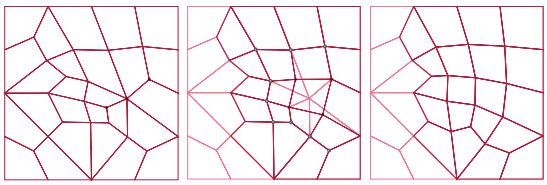


Figure 14: A vertex split highlighting the moving points and the quads that are checked when testing that the mesh remains valid.

4.1 Detecting Invalid Elements

Here we discuss how we ensure mesh validity throughout the regularization process. At the very least, a suitable mesh should be made of quads whose internal angles are less than 180° [29]. Preserving this property ensures that there are no folds (self-intersecting quads). The idea is the following: use the underlying parametrization to obtain the surface normal an use it to generate the tangent plane at the centroid. Then, as shown in Figure 15, project all the quad coordinates onto that plane and calculate the vertex orientations relative to each other.

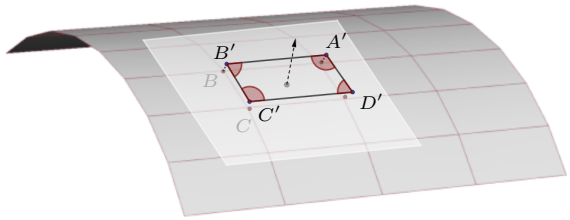


Figure 15: A cylindrical surface showing the tangent plane centered a particular quad and all its vertices projected onto such plane.

By construction, vertices forming a quad are always stored clock-wise (positive) so we can detect obtuse or self-intersecting elements by looking at the orientations relative to every vertex. For every three consecutive vertices, the orientations are computed as follows:

$$ori(ABC) = sign(< \vec{AB} \times \vec{AC}, \vec{n} >). \quad (5)$$

Here $<,>$ is the dot product and $\vec{n}$ the normal vector to the surface at the quad center (see Figure 15). All positive orientations indicate a valid quad. If there are at least two pairs with positive orientations, the quad is obtuse and finally when there are no vertices with positive orientation, then the quad is self-intersecting.

In Figure 16 we show an obtuse quad together with each vertex orientations. Notice that at A and C the orientations change sign $(+,-)$ $(-,+)$ whereas at B and C we obtain $(+,+)$. For quads that have at least one boundary vertex, we also check that none of its sides have crossed the domain bounds. This happens frequently when performing operations on surfaces with internal holes that have sharp corners.

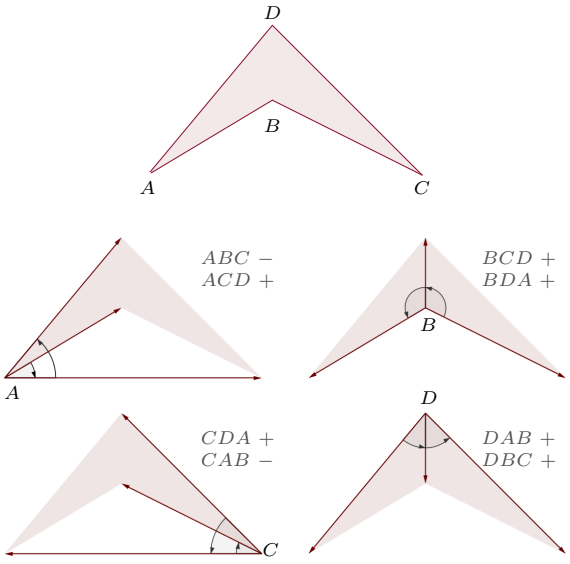


Figure 16: An obtuse quad showing each vertex orientations with respect the other vertices. Quads are valid only when the orientations are $(+,+)$ at every vertex.

4.2 Laplacian using Angles and Ratios

The new coordinates of a particular vertex are chosen depending on the quality of its surrounding quads which are categorized as shown in Figure 17. In addition, the vertex relative position to the mesh is also taken into account; vertices linked to boundary vertices (especially surface holes) have more restricted movement than those that are internal.

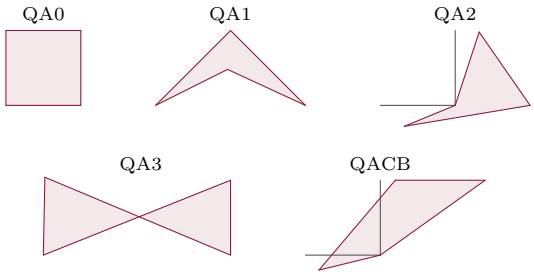


Figure 17: Possible quad configurations in descending order of validity: convex (QA0), concave (QA1), concave at an internal boundary (QA2), self-intersecting (QA3) and crossing the domain boundary (QACB).

For each vertex, we use its star to compute all the angles as well as the quad triangle ratios [30] which are illustrated in Figure 18. The vertex angles are computed at the normal plane using the same methodology described in Section 4.1 (see Figure 15) but in this case we use the surface normal at the central vertex to generate the plane to which all the other quads are projected. Irregular vertices (*i.e.* valence $\neq 4$) don't have an obvious orientation and the internal angles are computed just to detect detect flat or sharp elements. For regular vertices, we compute the angle distance between opposite vertices.

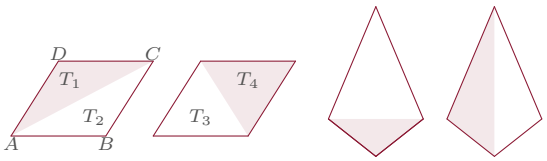


Figure 18: Two set of quads showing for each quad its two triangle ratios ($\{T_1, T_2\}$, $\{T_3, T_4\}$). Quads should have both ratios close (equal) to one like the left pair.

The triangle ratios (see Figure 18) are computed as follows. Let Q_{ABCD} be an ordered quad, then:

$$\left.\begin{aligned} T_1 &= ||AB \times AC|| \\ T_2 &= ||AC \times AD|| \end{aligned}\right\} D_{12}(Q) = \min\left(\frac{T_1}{T_2}, \frac{T_2}{T_1}\right). \quad (6)$$

Similarly, defining $T_3 = ||BC \times BD||$ and $T_4 = ||BD \times BA||$ gives $D_{34}(Q)$. The final quad ratio is given by

$$R(Q) = \min(D_{12}, D_{34}). \quad (7)$$

The vertex contribution of an obtuse quad is measured in the following way. Identifying A with the vertex, if $T_1 + T_2 > T_3 + T_4$ then vertex A can improve / worsen the large angle and so we take $R_A(Q) = -D_A(Q)$. Otherwise $(T_1 + T_2 < T_3 + T_4)$, we use the following:

$$R_A(Q) = -T_4/T_3 \quad \text{if } \angle DAB > 180^\circ \quad (8)$$

$$R_A(Q) = D_{12}(Q) \quad \text{otherwise.} \quad (9)$$

In Algorithm 1 we provide details on how the vertex area function is implemented.

Algorithm 1 Vertex Area

INPUT: Vertex ID (p)
$tot = 0,\ a = QA0,\ nr = 1;\ pr = 1;\ t = 1;$
for $i = 1$ to $S(p)$->nQ **do** ▷ S = star
  $q = S(p)$->$q(i)$
  $count =$ `positive_orientation`(q)
  $e =$ `get_bound_edges`(q)
  if $e > 0$ *and* $(e \cap BC \neq \emptyset$ *or* $e \cap CD \neq \emptyset)$ **then**
    $a = QACB$
  else if $count\ =\ 0$ **then** $a = QA3$
  else
    if $count\ >\ 0$ **then** $a = QA1$
    end if
    $ratio =$ `quad_ratio`(q, v)
    if $ratio <$ `EPS` **then** $a = QA1$
    end if
    $angle(i) = \angle(DAB)$
    if $angle(i) > 180$ *and* $e > 0$ **then** $a = QA2$
    end if
    if $ratio < 0$ **then** $nr = \min(nr, ratio)$
    else $np = \min(nr, ratio$
    end if
  end if
  $tot\ +=\ a$
end for
for $i = 1$ to $S(p)$->nQ **do**
  if $S(p)$->$totQ\ = 4$ **then**
    $t = (180 - |angle(i) - angle(i+2)|)/180$
  else if $angle(i) >$ `ANGPASS` **then**
    $t = (1.0 - penalty(t))$
  end if
  $pr\ *=\ t$
end for
return $tot,\ nr,\ pr$

The actual vertex coordinates are computed using the centroid (Laplacian), midpoints between two opposite links and a *biased* Laplacian where a particular quad (two vertices) is excluded. This is useful for unfolding elements or reducing large angle sizes arising as a result of a swap, split, etc. Notice that the average surface parametric coordinates uv does not necessarily result in the average physical coordinates. The centroid or mid points are computed in the following way. First evaluate the physical average (which may not live on the surface). Then, using an inverse evaluate function from the geometry kernel, obtain the *closest* surface point with its corresponding uv pair.

Vertices linked to internal boundaries (surface holes) are treated differently. This is done in order to avoid the Laplacian element shrinking effect; a surface cut-out that is convex (*e.g.*, a circle) or a sharp corner is generally incompatible with an averaging method since such point may be driven towards the hole or create very large angles. Therefore, boundary vertices forming sharp angles are flagged and the Laplacian option is deactivated for any internal vertex linked to such vertices. This technique is described in Algorithm 2. Recall that $S(p)$ denotes the star at vertex p and has the structure defined in Figure 6.

Algorithm 2 New Coordinates

INPUT: Vertex and Laplacian indeces (p, t)
$(u, v) \leftarrow p(u, v)$
for $m = i = 0$ to $S(p)$->nQ **do**
  if $S(p)$->$v(2i+1) =$ `obtuseBound` **then**
    $m = 2i + 1$
  end if
end for
if $t = -1$ **then**
  if $m = 0$ **then return**
  else $p(u, v) \leftarrow$ `Laplacian`$(S(p))$
  end if
else if `area`$(S(p)) \neq QA0$ **then**
  $(u,\ v) \leftarrow$ `Laplacian`$(S(p) \setminus S(p)$->$q(t))$
else if $S(p)$->$totQ = 3$ **then return** (u, v)
else if $m = 0$ **then**
  $a = S(p)$->$v(2t+1)$
  $b = S(p)$->$v(2t+3)$
  $p(u, v) \leftarrow$ `mid_point`(a, b)
else
  $a = S(p)$->$v(m)$
  $b = S(p)$->$v(2t+1)$
  $p(u, v) \leftarrow$ `mid_point`(a, b)
end if
return (u, v)

Finally, in Algorithm 3 we show how the actual vertex coordinates are chosen. This function is called iteratively at the highest level; during regularization, only few iterations are allowed since a vertex may undergo several manipulations and even get deleted. Once the regularization process is complete we invoke this function for 50 iterations to improve the final mesh.

The idea is the following: for each vertex, we compute all the possible vertex coordinates (Algorithm 2) and recompute the areas, triangle ratios and angles. Then we assess if the new coordinates have improved the ratios and angle differences and if so, the coordinates are stored as best candidate. Since unless the mesh is fully regular, it is impossible to obtain equal angles and ratios. Hence, we accept Laplacian coordinates whenever these values are above a certain tolerance which we have set to 0.25. In Figure 19 we show two examples of surfaces before and after regularization using Algorithm 3 for recomputing coordinates. Surfaces with sharp internal corners like the star cut-out can produce large angles and even lead to element folding. In addition, the average coordinates may even pro-

duce quads crossing the domain boundaries. Notice that the resulting mesh has quads with angles close to 90° at each star vertex. Further, the sphere hole in the second image produces a circular edge where all the boundary vertices have angles > 180° with respect to each other. When taking averages, this can lead to element shrinking and eventually move points outside the surface, *i.e*, towards the hole. Notice that the algorithm overcomes the element shrinking effect and produces a satisfactory result.

Algorithm 3 Update Vertex

INPUT: Moving vertices (n, p)
for $i = 1$ to n **do**
  $(a_0,\ nr_0,\ pr_0) \leftarrow$ `vertex_area`$(p(i))$
  $p_{new}(u,v) \leftarrow p(i)(u,v)$
  $p_{old}(u,v) \leftarrow p(i)(u,v)$
  for $update = j = 0$ to $2 \cdot S(p(i))$`->`nQ **do**
    $p(i)(u,v) \leftarrow$ `new_coordinates`$(p(i), j-1)$
    $(a_1,\ nr_1,\ pr_1) \leftarrow$ `vertex_area`$(p(i))$
    $k = 0$
    if $a_1 < a_0$ **then** $k = 1$
    else if $a_1 = a_0$ **then**
      if $nr_0 < 0$ or $nr_1 < 0$ **then**
        if $nr_1 > nr_0$ **then** $k = 1$
        end if
      else if $pr_1 > 0.25$ and $j = 0$ **then** $k = 2$
      else if $pr_1 > pr_0$ **then** $k = 1$
      end if
    end if
    if $k = 1$ **then**
      $update = 1$
      $p_{new}(u,v) \leftarrow p(u,v)$
      $(a_0,\ nr_0,\ pr_0) \leftarrow (a_1,\ nr_1,\ pr_1)$
    end if
    if $k = 2$ **then break**
    end if
  end for
  if $update = 1$ **then** $p(i)(u,v) \leftarrow p_{new}(u,v)$
  end if
end for

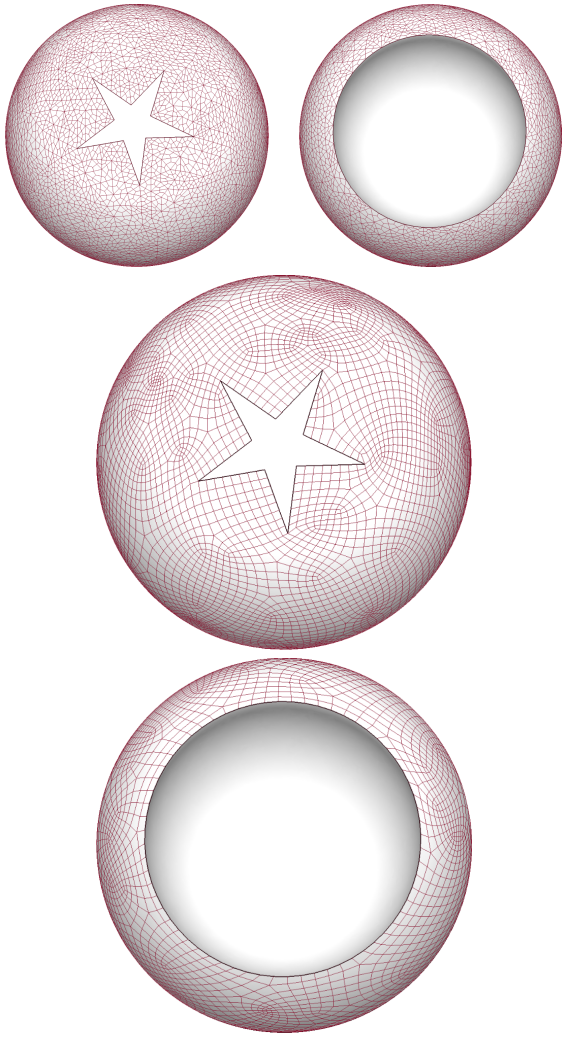


Figure 19: A sphere with two type of cut-outs, a star and a second sphere. Notice that all the angles at the star vertices are close to 90° and that all the quads around the sphere hole have not collapsed towards the boundary. The top figures show the initial meshes.

5. VALIDATION

The target for these quadrilateral meshes is structural analysis (specifically *Built-up Element Models*), and these results should not be viewed through a CFD lens. In a real sense, the task at hand is more difficult; it is harder to produce valid meshes for curved geometry when its applications require the element size to be coarser. We begin by looking at the algorithm performance over basic surfaces including surfaces with cut-outs. Figure 20 shows a cylinder and a composition of spheres. For the cylinder case, the final mesh that resulted was fully regular. We readily admit that such surface can be tessellated directly into fully regular quads and in this case, the triangulation was forced just for validation purposes. The spherical body is included to highlight the strength of the regularization technique: for example, the concave face contains a boundary vertex which has valence eighteen.

In Figure 21 we study several surfaces. The first two consist of a flat and a spherical surfaces with several cut-outs. The third one was produced by revolving a spline curve where the pole (top vertex) is singular. The coloring is based on the quad's largest angle. We haven't colored the initial mesh since we already know that these meshes are unsuitable. Notice that in all three cases, only a few angles are over 150°.

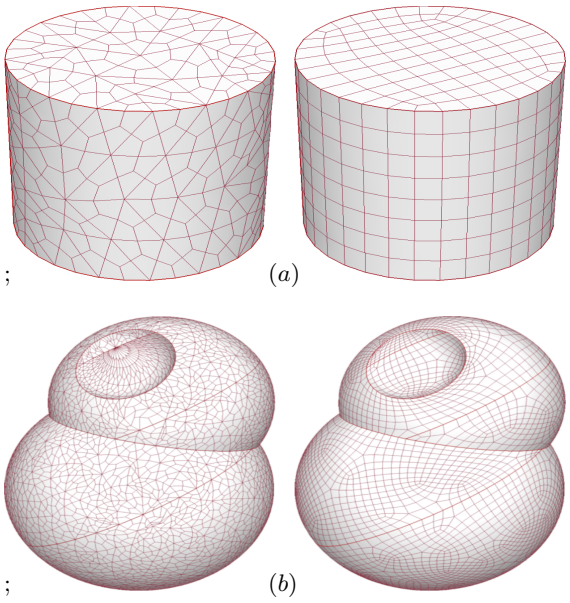


Figure 20: A cylinder and spherical body showing the meshes before and after regularization.

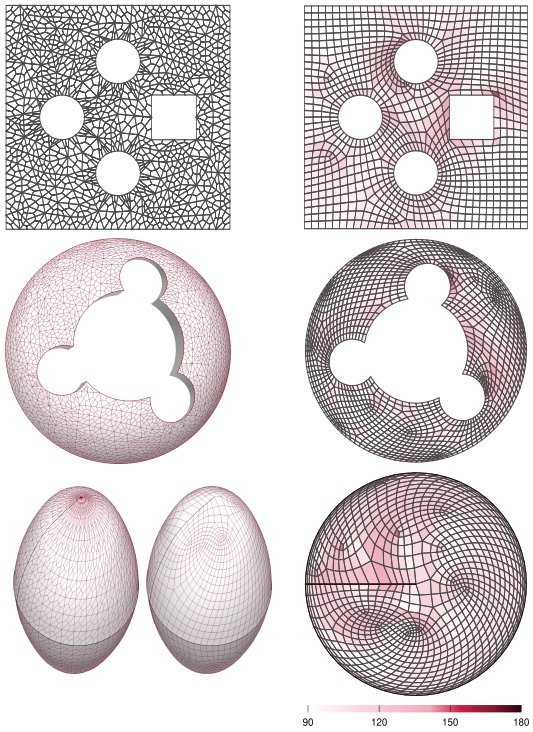


Figure 21: Three surfaces before and after regularization colored by the quads largest angles. The top (flat) and middle (spherical) present several cut-outs. The third is a surface of revolution which has a singularity at the pole. The highlighted lines denote the surface edges (fixed vertices) and the color map corresponds to the top face.

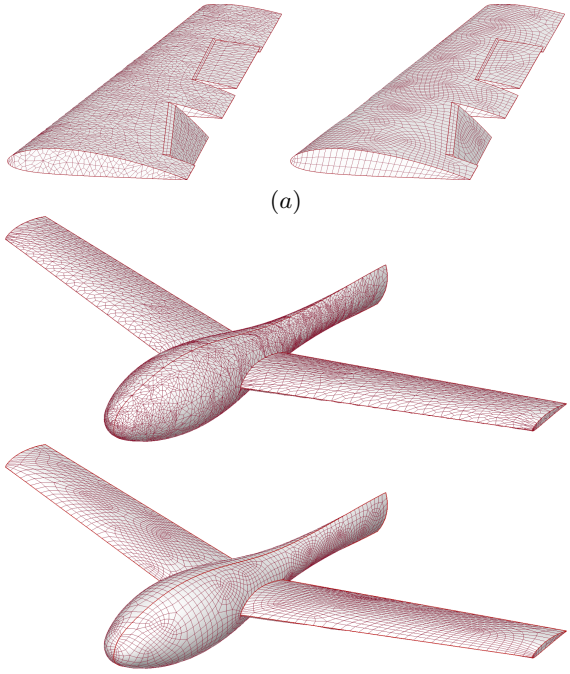


Figure 22: A wing body with two flaps (a) and a full aircraft (b) before and after regularization.

The wing and aircraft body shown in Figure 22 study the performance over more curved surfaces. The wing flaps are introduced to illustrate how the scheme works over trimmed surfaces. In both cases, the scheme is able to recover a large number of regular vertices producing good quality meshes. Notice that the mesh quality of the aircraft is especially high at the fuselage and at the nose where there is high curvature.

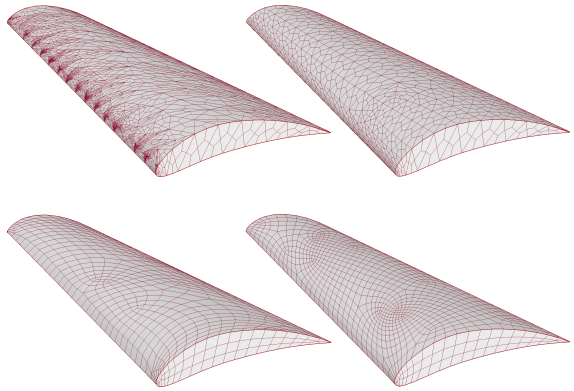
5.1 A note on the Triangulation

The performance of our technique clearly depends on the original tessellation. In particular, since boundary edges are fixed, the final mesh quality is strongly dictated by the side spacing at the bounds. The initial triangulations were generated by the tessellator in in EGADS [31].

It produces watertight triangulations of BReps by first discretizing the BRep Edges and then performing the trimmed surface (BRep Face) triangulations. There is no notion of grading of spacings because the tessellator is driven by being able to best represent the geometry with the fewest number of vertices/triangles. The technique used simply bifurcates regions that don't meet the user input criteria, which can obviously display abrupt spacing changes of a factor of 2 or more. In addition, large interior angle deviations between neighboring triangles as well as large side spacing are

allowed (especially when using coarse tessellation parameters), thus producing triangulations that are far from equilateral.

In Figure 23 we show two wing profiles using the EGADS tessellator directly and applying the quadrilateral templating scheme described in [32]. Both tessellations used the same length criteria producing the same discretization at the surface bounds (BRep Edges). Observe that without templating (left), the initial tessellation has an excessive number of quads with highly irregular vertices at the leading edge. The result is that most of those quads are collapsed during regularization. The right images on the other hand, have a more realistic starting point and require much less mesh manipulation.



(a) (b)

Figure 23: A comparison of two tessellations with (a) and without (b) the quadding template [32] and the resulting meshes after regularization (bottom).

5.2 CPU Time and Regularity

Finally, we focus on the actual performance both in terms of the number of recovered regular vertices as well as the computational costs. In Table 1 we show the results for some of the surface bodies used in the validation Section. The results are very consistent: the initial meshes start with $\sim 49\%$ irregular vertices and after regularization, there are only $< 4\%$ left. Also it is important to note that the number of irregular vertices relative to the mesh doesn't grow as the mesh becomes finer nor is the performance affected as the surface geometry becomes more complex. In Table 2 we have gathered several surfaces and show results for the minimum quad triangle ratios [30] as well as minimum and maximum angles. It should be noted that angles are computed at the surface tangent plane and this may not always be accurate. In high curvature regions such as the wing leading edge, the projected coordinates can increase (reduce) the angle magnitude.

	Total Vertices		Irregular Vertices			
Figure	Initial	Final	Initial	(%)	Final	(%)
20 2^{nd}	5799	5401	2843	49.0	188	3.5
22 (a)	4174	1029	1994	47.8	38	3.7
22 (b)	1876	1869	926	49.4	64	3.7
23 (a)	3259	3035	1631	50.0	108	3.6
23 (b)	10347	6169	5105	49.3	226	3.7

Table 1: Results in terms of number of irregular vertices for several bodies considered in this manuscript.

The computational times are shown in Table 3. It should be noted that the implementation operates on a BRep Face at a time and has been threaded providing scalability based on the number of cores available (and number of Faces to process). The results are very promising: even for the finer meshes the total simulation time remains in less than a minute.

Figure	Min Ratio	Min Angle	Max Angle
21 top	0.32	38	151
21 middle	0.29	35	160
21 bottom	0.36	41	147
22-(a) top face	0.31	18	162
22-(b) fuselage	0.18	34	150
23-(a)	0.32	21	158

Table 2: Global minimum ratios and minimum and maximum angles for several bodies shown in this paper.

6. CONCLUSIONS AND ONGOING WORK

We have presented an automatic mesh generation technique that produces almost regular quadrilateral meshes. Starting with a triangulation and using the splitting approach from [1], we produce a fully quadded mesh which then undergoes topological changes in order to recover regularity. Our results show that in general, it is possible to reduce the number of irregular vertices from around 49% to $< 4\%$.

The regularization process is coupled with a Laplacian based iterative scheme for computing the move-

Figure	Faces	Time (secs)
20 2^{nd}	6	5
22 (a)	4	1
22 (b)	4	12
23 (a)	10	11
23 (b)	8	46

Table 3: Simulation times using a Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz machine for the examples in Table 1 showing also the number of faces composing each body.

ment of vertex coordinates. The resulting mesh is always valid but there is no notion of optimal sizing. Rather, we just attempt to reduce the angle deviation as well as produce quads with aspect ratio close to one and updating the new coordinates based on the "best candidate". This results in low scalar computational times (less than one minute) even for meshes starting with around $10k$ quads. However, we understand the tradeoff in this approach and for specific applications, a quad distribution based on a specific metric should be introduced.

The initial triangulations were produced using the EGADS internal tessellator. This tool was designed for visualization purposes and therefore can produce poor quality meshes for simulations (see Figure 23 (a)) with excessive number of elements as a starting point. Since our technique is independent of the tesselator, it could use as input other triangulations designed specifically for applications in numerical simulations.

Finally, since the computational cost associated to our regularization scheme is rather low, our results suggest that the initial input mesh should be relatively coarse. In this case, the resulting mesh would consist on few irregular vertices. If a smaller sizing is desired, mesh refining can be applied directly by subdividing quads without introducing new irregular vertices.

ACKNOWLEDGEMENTS

This work was funded by the CAPS project, AFRL Contract FA8050-14-C-2472: "CAPS: Computational Aircraft Prototype Syntheses"; Dean Bryson is the Technical Monitor. We would also like to thank the anonymous reviewers for their feedback and useful suggestions.

References

[1] Catmull E., Clark J. "Recursively generated B-spline surfaces on arbitrary topological meshes." *Computer-aided design*, vol. 10, no. 6, 350–355, 1978

[2] Haimes R., Dannenhoffer J. "The Engineering Sketch Pad: A Solid-Modeling, Feature-Based, Web-Enabled System for Building Parametric Geometry." *21st AIAA Computational Fluid Dynamics Conference*. 2003

[3] Talbert J.A., Parkinson A.R. "Development of an automatic, two-dimensional finite element mesh generator using quadrilateral elements and Bezier curve boundary definition." *International Journal for Numerical Methods in Engineering*, vol. 29, no. 7, 1551–1567, 1990

[4] Blacker T.D., Stephenson M.B. "Paving: A new approach to automated quadrilateral mesh generation." *International Journal for Numerical Methods in Engineering*, vol. 32, no. 4, 811–847, 1991

[5] White D.R., Kinney P. "Redesign of the paving algorithm: Robustness enhancements through element by element meshing." *6th International Meshing Roundtable*, pp. 323–335. 1997

[6] Tam T., Armstrong C. "2D finite element mesh generation by medial axis subdivision." *Advances in Engineering Software and Workstations*, vol. 13, no. 5, 313 – 324, 1991

[7] Dong S., Bremer P.T., Garland M., Pascucci V., Hart J.C. "Spectral surface quadrangulation." *Transactions on Graphics (TOG)*, vol. 25, no. 3, 1057–1066, 2006

[8] Tarini M., Puppo E., Panozzo D., Pietroni N., Cignoni P. "Simple quad domains for field aligned mesh parametrization." *Transactions on Graphics (TOG)*, vol. 30, 142, 2011

[9] Kowalski N., Ledoux F., Frey P. "A PDE Based Approach to Multidomain Partitioning and Quadrilateral Meshing." X. Jiao, J.C. Weill, editors, *21st International Meshing Roundtable*, pp. 137–154. 2013

[10] Bommes D., Campen M., Ebke H.C., Alliez P., Kobbelt L. "Integer-grid maps for reliable quad meshing." *Transactions on Graphics (TOG)*, vol. 32, no. 4, 98, 2013

[11] Fogg H.J., Armstrong C.G., Robinson T.T. "Automatic generation of multiblock decompositions of surfaces." *International Journal for Numerical Methods in Engineering*, vol. 101, no. 13, 965–991, 2015

[12] Lo S., Lee C. "On using meshes of mixed element types in adaptive finite element analysis." *Finite Elements in Analysis and Design*, vol. 11, no. 4, 307 – 336, 1992

[13] Lee C.K., Lo S. "A new scheme for the generation of a graded quadrilateral mesh." *Computers & Structures*, vol. 52, no. 5, 847–857, 1994

[14] Borouchaki H., Frey P.J. "Adaptive triangular–quadrilateral mesh generation." *International Journal for Numerical Methods in Engineering*, vol. 41, no. 5, 915–934, 1998

[15] Remacle J.F., Lambrechts J., Seny B., Marchandise E., Johnen A., Geuzainet C. "Blossom-Quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm."

International Journal for Numerical Methods in Engineering, vol. 89, no. 9, 1102–1119, 2012

[16] Owen S.J., Staten M.L., Canann S.A., Saigal S. "Q-Morph: an indirect approach to advancing front quad meshing." *International Journal for Numerical Methods in Engineering*, vol. 44, no. 9, 1317–1340, 1999

[17] Daniels II J., Silva C.T., Cohen E. "Localized quadrilateral coarsening." *Proceedings of the Symposium on Geometry Processing*, pp. 1437–1444. Eurographics Association, 2009

[18] Staten M.L., Benzley S., Scott M. "A methodology for quadrilateral finite element mesh coarsening." *Engineering with Computers*, vol. 24, no. 3, 241–251, 2008

[19] Daniels J., Silva C.T., Shepherd J., Cohen E. "Quadrilateral Mesh Simplification." *Transactions on Graphics (TOG)*, vol. 27, no. 5, 148, 2008

[20] Kinney P. "Cleanup: Improving quadrilateral finite element meshes." *6th International Meshing Roundtable*, pp. 437–447. 1997

[21] Tarini M., Pietroni N., Cignoni P., Panozzo D., Puppo E. "Practical quad mesh simplification." *Computer Graphics Forum*, vol. 29, pp. 407–418. 2010

[22] Peng C.H., Zhang E., Kobayashi Y., Wonka P. "Connectivity editing for quadrilateral meshes." *Transactions on Graphics (TOG)*, vol. 30, 141, 2011

[23] Bozzo A., Panozzo D., Puppo E., Pietroni N., Rocca L. "Adaptive Quad Mesh Simplification." *Eurographics Italian Chapter Conference*, pp. 95–102. 2010

[24] Verma C.S., Suresh K. "A robust combinatorial approach to reduce singularities in quadrilateral meshes." *Procedia Engineering*, vol. 124, 252–264, 2015

[25] Zhou T., Shimada K. "An Angle-Based Approach to Two-Dimensional Mesh Smoothing." *9th International Meshing Roundtable*, pp. 373–384. 2000

[26] Freitag L.A. "On combining Laplacian and optimization-based mesh smoothing techniques." Tech. rep., Argonne National Lab., IL, 1997

[27] Knupp P.M. "Winslow smoothing on two-dimensional unstructured meshes." *Engineering with Computers*, vol. 15, no. 3, 263–268, 1999

[28] Knupp P.M. "Algebraic mesh quality metrics." *SIAM journal on scientific computing*, vol. 23, no. 1, 193–218, 2001

[29] Robinson J. "Some new distortion measures for quadrilaterals." *Finite Elements in Analysis and Design*, vol. 3, no. 3, 183 – 197, 1987

[30] Lo S. "Generating quadrilateral elements on plane and over curved surfaces." *Computers & structures*, vol. 31, no. 3, 421–426, 1989

[31] Haimes R., Drela M. "On the construction of aircraft conceptual geometry for high-fidelity analysis and design." *50th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, p. 683. 2012

[32] Haimes R., Aftosmis M.J. "Watertight Anisotropic Surface Meshing Using Quadrilateral Patches." *13th International Meshing Roundtable*, pp. 311–322. 2004