A Study on Size-Optimal Longest Edge Refinement Algorithms

Carlos Bedregal and María-Cecilia Rivara

Department of Computer Science, University of Chile
{cbedrega,mcrivara}@dcc.uchile.cl

Mesh generation and refinement are widely used in applications that require a decomposition of geometric objects for processing. Longest edge refinement algorithms seek to obtain a better decomposition over selected regions of the mesh by the division of its elements. Until now, these algorithms did not provide theoretical guarantees on the size of the triangulation obtained. In this paper we present a study of the computational cost of longest edge bisection algorithms for 2-dimensional mesh refinement and our developments in the theoretical analysis of such algorithms.

1 Introduction

Mesh generation is widely used in many applications that require a decomposition of geometric objects for processing, discretizing the continuous domain into a mesh typically composed of triangles or quadrilaterals in two dimensions, and tetrahedra or hexahedra in three dimensions. Mesh refinement techniques seek to obtain either a better, finer decomposition of the input (or a region of interest) in order to track smaller features of the mesh geometry, or to obtain a quality locally refined mesh as required by the application.

These applications are found in the areas of computational geometry, computer graphics, mechanical engineering, solid modeling, image processing, geographical information systems, and numerical simulation among others, making the problem an interdisciplinary topic.

Historically, initial algorithms for mesh generation have been developed by engineers, who were usually satisfied with the results obtained for their specific domains. Later, computational geometry researchers became interested in designing algorithms that offer mathematical guarantees on the generation of satisfying meshes [22].

Since, in practice, the time required to process a geometric object depends on the size and quality of its decomposition, an initial goal of the mesh generation and refinement is to produce size-optimal meshes that *conform* to the

geometry of the domain and satisfy the constraints on the shape and size of its elements. A second goal is to produce locally refined meshes according to an associated application, such as finite element analysis. Algorithms that offer theoretical guarantees on the shape and size of the mesh produced are thus desirable.

Longest-edge refinement algorithms for triangulations in two dimensions were specially designed to work well with numerical techniques such as adaptive finite element methods and multigrid algorithms [14, 15], ensuring the construction of good-quality irregular and nested triangulations. Longest edge bisection algorithms proposed by Rivara [14, 16, 17] perform a local refinement maintaining the geometric quality of the triangulation and increasing the area covered by nearly equilateral triangles. These algorithms offer guarantees on the minimum angle lower bound and on the number of non-similar triangles generated.

In this paper we present our recent results on the study of the computational cost of longest edge bisection algorithms. We focus on the behaviour of the algorithm when propagating the refinement to a neighbor triangle, bounding the number of points produced in order to maintain a conforming triangulation. The paper is organized as follows: Section 2 discusses the previous work in the area of mesh refinement techniques, Section 3 describes the longest edge bisection algorithm and its properties, Section 4 presents our analysis of the algorithm based on the number of point it produces; finally, Section 5 summarizes our results and current state of the research.

2 Previous Work

Most applications working with unstructured meshes require the mesh to be conforming and to have good "quality." A mesh is said to be *conforming* if adjacent elements intersect only at either a common edge or a common vertex or an entire face. Because of the domain constraints on shapes and sizes of the elements, a mesh with a good geometry quality is also desirable. The quality of a mesh is given by some measure of shape, such as a bound on the aspect ratio of all the triangles or a bound on their internal angles. A commonly used measure is the minimum angle α [19]. For example, elements with "ideal" shape are typically those nearly equilateral and equiangular for non-anisotropic applications.

A mesh generation algorithm must also consider the size of its output since *Steiner points* (vertices which are not part of the input) are generally added in order to achieve the shape bound. Adding too many points could have a negative effect on the performance of the application. Then, a mesh is considered size-optimal if the number of elements is within a constant factor of the minimum number in any mesh of the input offering the same shape bound.

2.1 Mesh Refinement

Existing methods for generation and refinement of triangular and tetrahedral meshes in 2- and 3-dimensions can be roughly classified as partitioning methods and Delaunay-based methods [19]. *Partitioning methods* refine the mesh by dividing some selected elements of the mesh. Baker et al. [1] proposed the first algorithm to give shape guarantees on the triangulation. This work was later extended by Bern et al. [3] and Melissaratos and Souvaine [11], using *quadtrees* to offer both shape and size guarantees. *Delaunay-based methods*, on the other hand, maintain a Delaunay triangulation during the refinement, adding new points to the mesh that ensure the quality of the elements [6, 19, 7, 21]. Extensive research on practical mesh generation has been also performed. See e.g. [2, 5, 10].

Our study will focus on longest edge triangulation refinement algorithms. A *triangulation* is a decomposition of the input into simplices (triangles and tetrahedrals), that meet at shared edges. Unstructured meshes are easier to generate with algorithms using simplicial elements [4]. A study on non-simplicial meshes (using quadrilateral and hexahedral elements) can be found in the survey by Schneiders [20].

Longest edge bisection algorithms [14] are partitioning methods suitable for the triangulation refinement problem. Given a conforming, non-degenerate triangulation with acceptable quality (e.g. angles greater than or equal to an angle α) of a polygonal region, the *triangulation refinement problem* consists in the construction of a locally refined triangulation with a desired resolution such that the smallest (or the largest) angle is bounded. For this paper, we decompose the triangulation refinement problem into the following two subproblems:

(P1) **Triangle refinement problem:** given a quality acceptable triangulation τ (of a polygonal region D) composed by N_τ triangles, all with angles greater than or equal to an angle α, and given a target triangle $t \in \tau$, construct a conforming triangulation τ' such that t has been refined once while maintaining the quality of the triangulation.

(P2) **Local refinement problem:** given a quality acceptable triangulation τ (of a polygonal region D) composed by N_τ triangles, all with angles greater than or equal to an angle α, and given a set of target triangles $S_{ref} \subseteq \tau$, construct a conforming triangulation τ' such that every $t \in S_{ref}$ has been refined once, and the quality of the triangulation is preserved.

The local refinement problem can be considered as a generalization of the triangle refinement problem, while the triangulation refinement problem can be seen as the iterative solution of the local refinement problem, with triangle set S_{ref} containing triangles over a subregion of D that do not satisfy the resolution parameter.

Longest edge bisection algorithms perform an iterative longest edge bisection on the target triangles – and some of their neighbors – in order to produce a conforming refined mesh. Additional to the non-degeneracy properties of the iterative longest edge bisection of triangles, they offer guarantees on the minimum angle lower bound and on the number of non-similar triangles generated [17].

These algorithms have been used for developing software for partial differential equations [12], generalized for 3-dimensional refinement [13], applied on finite element methods [10], and adapted to multithread parallelization [18].

3 Longest Edge Bisection Algorithms

A longest edge refinement algorithm works as follows: at each step of the process the algorithm receives an input conforming triangulation of acceptable geometric quality and a set of triangles to be refined; after performing a longest edge bisection of the selected triangles (and some of their neighbors) it then outputs a refined conforming triangulation. Algorithm 1 describes the original longest edge bisection algorithm proposed by Rivara [14].

Algorithm 1 Longest edge bisection algorithm

Input: A quality triangulation τ and a set S_{ref} of triangles to be refined
Output: A quality triangulation τ' such that each $t \in S_{ref}$ has been refined

```
while S_ref ≠ ∅ do
  for each t in S do
    Bisect t by the longest edge midpoint Q
    Find t', neighbor of t containing point Q
    if t' became non-conforming then
      Add t' to S_ref
    end if
  end for
end while
```

A practical drawback of Algorithm 1 is the management of intermediate non-conforming triangulations, which mainly depend on how the set S_{ref} is processed. Rivara introduced the Lepp-bisection algorithm [16], an elegant and efficient improvement on the original algorithm. The main contributions of this algorithm are the concepts of propagation paths and terminal edges, making it possible to navigate through the set S_{ref} performing local bisections of pairs of triangles, also maintaining the mesh valid during the whole refinement process. An advantage over Delaunay-based refinement methods is their robustness, as bisection algorithms do not depend on complex computations.

Empirical evaluations on longest edge bisection algorithms have been previously performed [9, 23, 18].

3.1 Lepp-bisection Algorithm

An edge E is called a *terminal edge* in triangulation τ if E is the longest edge of every triangle that shares E, and the triangles sharing E are called *terminal triangles* [17]. For the 2-dimensional scenario, if E is shared by two terminal triangles then E is an interior edge; if E is shared by a single terminal triangle then E is a boundary or constrained edge. Figure 1 illustrates these concepts.

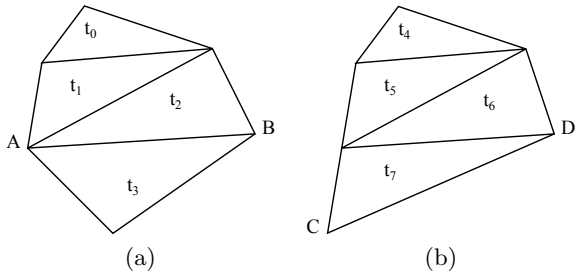


Fig. 1: (a) AB is an interior terminal edge shared by terminal triangles $\{t_2, t_3\}$ of $\texttt{Lepp}(t_0) = \{t_0, t_1, t_2, t_3\}$; (b) CD is a boundary terminal edge with terminal triangle $\{t_7\}$ of $\texttt{Lepp}(t_4) = \{t_4, t_5, t_6, t_7\}$

For any triangle t_0 in τ, the *longest edge propagating path* of t_0, $\texttt{Lepp}(t_0)$, is the ordered sequence $\{t_j\}_0^{N+1}$, where t_j is the neighbor triangle on the longest edge of t_{j-1}, and $\texttt{longest\_edge}(t_j) > \texttt{longest\_edge}(t_{j-1})$, for $j = 1, \ldots, N$. Edge $E = \texttt{longest\_edge}(t_{N+1}) = \texttt{longest\_edge}(t_N)$ is an interior terminal edge in τ and this condition determines N. Therefore, either E is shared by the couple of terminal triangles (t_N, t_{N+1}) if E is an interior edge in τ, or E is shared by a unique terminal triangle t_N with boundary (constrained) longest edge.

The Lepp-bisection algorithm [17] for the triangle refinement problem can be simply described with two basic steps: starting from an initial triangle t_0, it first finds $\texttt{Lepp}(t_0)$, reaching its two terminal triangles t_N and t_{N+1} which share terminal edge E; then, the longest edge bisection is done by the midpoint of E – refining triangles t_N and t_{N+1}. This process is repeated until initial triangle t_0 is refined. Algorithm 2 presents a generalization of the algorithm for the local refinement problem.

Figure 2 illustrates the Lepp-bisection refinement process. Terminal triangles of $\texttt{Lepp}(t_0)$ are t_2 and t_3, which are bisected on their longest edge (Fig. 2(a)). $\texttt{Lepp}(t_0)$ is recomputed with terminal triangle t_1 and triangle t_2' (a sub-triangle of t_2). These triangles are then refined and a last computation of $\texttt{Lepp}(t_0)$ occurs, with terminal triangle t_0 and triangle t_1' (a sub-triangle of t_1). This final bisection refines starting triangle t_0 (Fig. 2(b)). Since the refinement process starts at the terminal triangles (stopping at the starting triangle) we can say that the Lepp-bisection algorithm works *backwards*.

Algorithm 2 Lepp-bisection algorithm

Input: A quality triangulation τ and a set S_{ref} of triangles to be refined
Output: A quality triangulation τ' such that each $t \in S_{ref}$ has been refined
  for each t in S_{ref} **do**
    while t remains in τ **do**
      Find `Lepp`(t), terminal triangles t_1, t_2 and terminal edge l. Triangle t_2 can be null for boundary l
      Select point (P, t_1, t_2, l)
      Perform (longest edge) bisection by P of triangles t_1, t_2
      Update S_{ref}
    end while
  end for

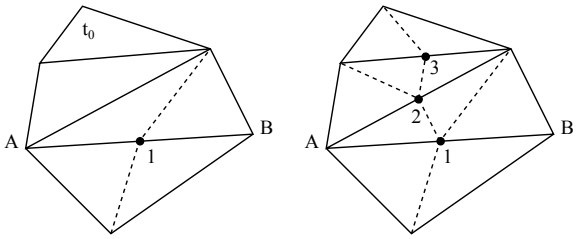


Fig. 2: Refining triangle t_0 of triangulation in Fig. 1(a) with `Lepp`$(t_0) = \{t_0, t_1, t_2, t_3\}$: (a) first vertex 1 is added by the bisection of the terminal triangles sharing edge AB; (b) final triangulation

3.2 Properties of the bisection algorithms

The main properties of the longest edge algorithms can be summarized as follows [14]:

Lemma 1. *The iterative and arbitrary use of the algorithms only produces triangles whose smallest interior angles are always greater than or equal to $\alpha/2$, where α is the smallest interior angle of the initial triangulation. Furthermore every triangle generated is similar to one of a finite number of reference triangles.*

Lemma 2. *Longest edge refinement algorithms always terminate in a finite number of steps with the construction of a conforming triangulation.*

Lemma 3. *Any triangulation τ generated by means of the iterative use of the algorithms satisfies the following smoothness condition: for any pair of side-adjacent triangles $t_1, t_2 \in \tau$ (with respective diameters h_1, h_2), it holds that $\frac{min(h_1,h_2)}{max(h_1,h_2)} \geq k > 0$, where k depends on the smallest angle of the initial triangulation.*

Lemma 4. *For any triangulation τ, the global iterative application of the algorithm (the bisection of all the triangles in the preceding iteration) covers, in a monotonically increasing form, the area of τ with quasi-equilateral triangles (with smallest angles $\geq \pi/6$).*

Proof of Lemma 2 is based on the facts that (1) the propagation of the refinement moves toward bigger triangles of the mesh, and (2) every mesh has bounded smallest angle. The smoothness property of Lemma 3 follows from the bound on the smallest angle of Lemma 1. Lemma 4 refers to the generation of more equilateral triangles with every iteration, also isolating the worst angles.

3.3 Similarity classes of triangles

The study by Gutierrez et al. [8] on the complexity of the bisection of a triangle introduced a classification of triangles based on the number of non-similar triangles produced by the iterative bisection of a triangle. This study defined six main similarity classes of triangles considering the possible geometric positions where vertex C of a triangle $t(ABC)$ may lie (see Fig. 3). Then, for any triangle t, the number of iterative bisections performed until no further non-similar triangles are generated is $O(\alpha^{-1})$, where α is the smallest angle of t.

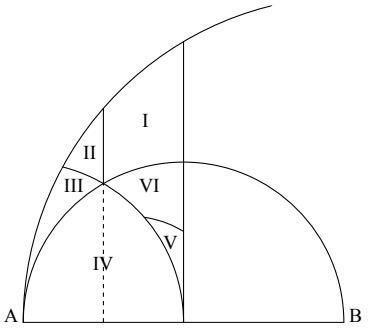


Fig. 3: Regions defining the classes of triangles $t(ABC)$. Virtual vertex C lies in one of the regions defining a triangle t with $AB \geq BC \geq CA$

It is proved in [8] that the iterative bisection of Class I triangles will generate at most four non-similar triangles (Lemma 2 in [8]). Figure 4 shows a Class I triangle $t(ABC)$ and the triangles generated after five bisections. The initial bisection generates two non-similar triangles $t_1(ADC)$ and $t_2(BCD)$. The bisection of t_1 generates one non-similar triangle, $t_{1,1}(DCH)$, because $t_{1,2}(ADH)$ is similar to initial triangle t. The bisection of $t_{1,1}$ generates triangles $t_{1,1,1}(CHF)$ and $t_{1,1,2}(HDF)$, which are similar to triangles t_1 and t_2

respectively. Finally, iterative bisections on t_2 generate only one non-similar triangle, $t_{2,1}(CDE)$.

It is also proved in [8] that for triangles of Classes II to VI, some new non-similar triangles are produced until triangles of Class I are created, which finishes the creation of new triangles. Note that for some of these cases the bisection of triangle $t(ADC)$ in Fig. 4 needs to be performed either on the edge CD or AD, while the bisection of $t_{2,1}(CDE)$ can be performed on edge CE.

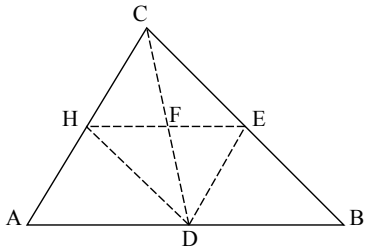


Fig. 4: Iterative bisections on a Class I triangle

The analysis of longest edge bisection refinement algorithms, described in Sec. 4, uses a similar classification to estimate the number of bisections needed to obtain a conforming refined mesh, specifically, the number of bisections needed to obtain "good" triangles on difficult regions.

4 Analysis of the Longest Edge Bisection Algorithm

Consider that the algorithm introduces a number of N_{ref} new mandatory points needed to achieve the required triangle resolution, and a number of N_{prop} new propagation points required for the mesh to remain valid. The computational cost of the algorithm will depend on (1) $N = N_{ref} + N_{prop}$, the number of new points inserted into the mesh, and (2) the cost of inserting these points. For (1), N_{ref} is bounded by the refinement parameters, so our study will focus on bounding N_{prop}. For (2), using appropriate data structures that allow easy access to a triangle's neighborhood information, the cost of insertion becomes linear in N, independently of the size of the triangulation and the number of iterations performed [14].

In the rest of the section we analyze the number of new points (and triangles) generated by the longest edge bisection algorithm. We start with an analysis of the number of points inserted by the propagation of refining a single triangle; then we extend the results to the local refinement problem and the case of iterative refinement around a vertex.

4.1 Refining a non-conforming triangle

Our goal is to analyze the behavior of the original longest edge bisection algorithm described in Algorithm 1 over individual triangles. Considering that the algorithm consistently selects the longest edge of a triangle for non-unique longest edges, the order in which set S_{ref} is processed does not affect the output of the algorithm. This also ensures that the original bisection algorithm and the Lepp-bisection algorithm obtain the same results. To study the refinement propagation we will analyze the behavior of the iterative bisection on a triangle $t \in S_{ref}$ in order to get a conforming triangulation; according to Algorithm 1, we will consider that the next triangles t^* to be refined are the ones generated by the bisection of the last triangle.

After the bisection of a target triangle t, whenever a non-conforming triangle t' is produced, the mesh is made conforming according to one of the three scenarios of Fig. 5:

Case 1: Q lays on the longest edge of t'. This case is trivial since t and t' share the longest edge. The bisection of t' by the midpoint of l is enough to obtain a valid mesh (see Fig. 5(a)). This case is analogous to refining the two terminal triangles of the Lepp.

Case 2: Q lays on the second longest edge of t'. In this case a second bisection is required to obtain a valid mesh. The first bisection corresponds to the longest edge bisection of t', inserting point Q_1 and splitting t' in two triangles. Consider t'_1 as the non-conforming triangle containing Q. Since Q lays on the longest edge of t'_1 (which corresponds to Case 1), the mesh is made conforming by bisecting t'_1 by the midpoint of l (see Fig. 5(b)).

Case 3: Q lays on the smallest edge of t'. This case is more complex to solve as the number of points inserted inside t' will depend on the triangle's similarity class (see Fig. 5(c)). The number of points inserted in t' is bounded by $O(\log \frac{1}{\alpha})$, where α represents the smallest angle of t'. [1]

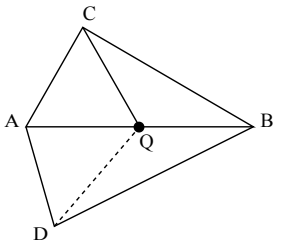
For Cases 1 and 2, triangle t' is made conforming independently of the triangle's similarity class. The analysis of Case 3 is further described in Sec. 4.2.

4.2 Refining triangles by the smallest edge

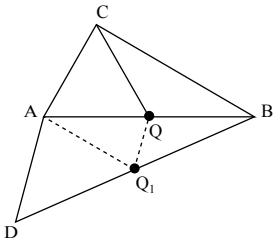
When a non-conforming triangle is produced after adding a point on its smallest edge (as described in Case 3 of Sec. 4.1), the triangulation is made valid after performing a few additional longest edge bisections inside the non-conforming triangle, i.e. until the hanging point is connected using one of the new edges.

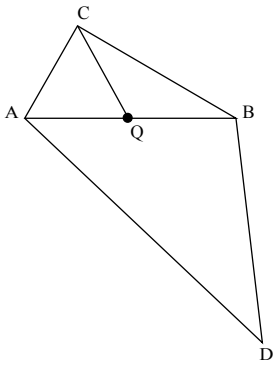
The number of bisections (and triangles) required to obtain a conforming triangle can be determined by the similarity class [8] of the triangle being processed, as described by the following lemma.

[1] Our logarithms are base 2.



(a) Case 1





(b) Case 2 (c) Case 3

Fig. 5: Longest edge bisection of triangle a $t(ABC)$ producing non-conforming neighbor triangle $t'(ADB)$: (a) AB is the longest edge of t'; (b) AB is the second longest edge of t'; (c) AB is the smallest edge of t'

Lemma 5. *In order to make valid a non-conforming triangle (with smallest angle α) when a point is hanging at the midpoint of its smallest edge, the number of new points (and triangles) produced by the longest edge bisection algorithm is:*

(1) For a Class I triangle it produces 3 triangles and 1 new point.
(2) For a Class II triangle it produces 6 triangles and 3 new points.
(3) For a Class III triangle it produces at most $6 + 3k$ triangles and $3 + 2k$ points, for a constant $k = \lceil \log(\frac{\pi}{6\alpha}) / \log(\frac{3}{2}) \rceil$.
(4) Class IV triangles behave no worse than Class III triangles.
(5) Class V and Class VI triangles present the same behavior as Class I triangles.

Proof. Consider a triangle $t(ABC)$, with smallest angle α and $AB \geq BC \geq CA$, and point Q located at the midpoint of its smallest edge CA making the triangle non-conforming. Also consider point Q_1 inserted after the first bisection of t by the midpoint of its longest edge AB, producing triangles

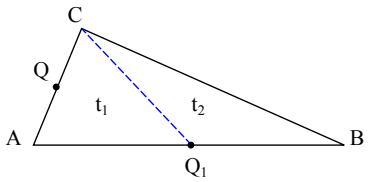


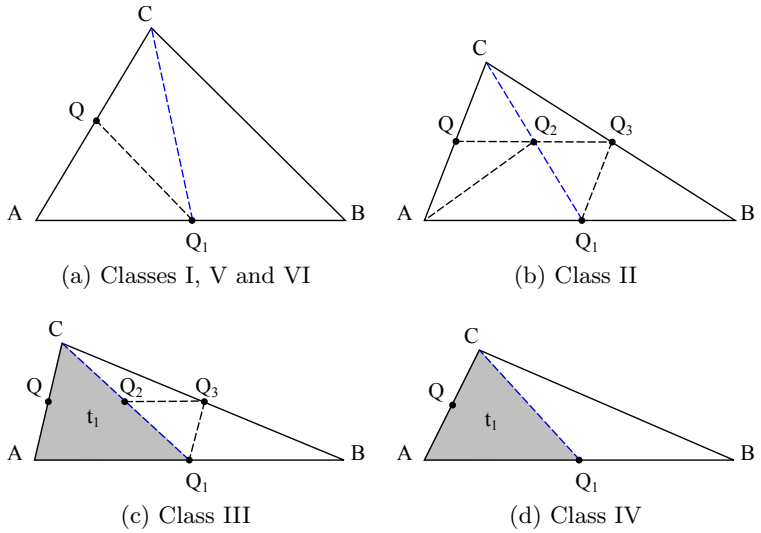
Fig. 6: Non-conforming triangle $t(ABC)$ produced by the introduction of point Q on its smallest edge CA. Triangles $t_1(AQ_1C)$ and $t_2(CQ_1B)$ are obtained after the initial longest edge bisection of t

$t_1(AQ_1C)$ and $t_2(CQ_1B)$, where t_1 is the non-conforming triangle containing point Q (see Fig. 6). Then, for each similarity class we can prove that:

1. **Class I ($AQ_1 \leq CQ_1 \leq CA$):** Edge CA becomes the longest edge of triangle t_1, so the longest edge bisection on t_1 is enough to make the triangulation valid, while triangle t_2 remains unaffected (see Fig. 7(a)). In total, one new point is produced.
2. **Class II ($AQ_1 \leq CA \leq CQ_1$):** Triangle t_1 belongs to Class I, so two bisections are performed on it. Point Q_2 inserted on edge CQ_1, making neighbor triangle t_2 non-conforming. Since $AQ_1 = Q_1B \leq CQ_1$, point Q_2 is affecting the second longest edge of t_2 (recall Case 2), two extra bisections are needed to make the triangulation valid (see Fig. 7(b)). In total, three new points are produced.
3. **Class III ($CA \leq AQ_1 \leq CQ_1$):** Triangle t_1 will either belong to Class I, II or III (see Fig. 7(c)). For Classes I and II the number of bisections and points produced inside t_1 is constant as described above. In either case point Q_2 is inserted on edge CQ_1, making triangle t_2 non-conforming. Since CQ_1 is again the second longest edge of t_2 (corresponding to Case 2), two bisections are required to make the triangulation valid. On the other hand, if t_1 still belongs to Class III, it is iteratively bisected until a Class I or II triangle is produced which contains Q in no more than $\lceil \log(\frac{\pi}{6\alpha}) / \log(\frac{3}{2}) \rceil$ steps. Consider i the number of steps performed, in the i-th iteration triangle t_1^i inserts point Q_1^i, affecting the second longest edge of neighbor t_2^i. Since $AQ_1 = Q_1B \leq CQ_1$, two bisections on t_2^i are needed to make the triangulation valid. This 2-bisection pattern is repeated on each t_2^j, for $j = 1, \ldots, i-1$. Finally, at most $3 + 2\lceil \log(\frac{\pi}{6\alpha}) / \log(\frac{3}{2}) \rceil$ new points are produced.
4. **Class IV ($CA, CQ_1 \leq AQ_1$):** Triangle t_1 could still belong to Class IV, but after no more than $\lceil \log(\frac{\pi}{6\alpha}) \rceil$ iterative bisections, the non-conforming triangle containing Q will either belong to Class I, II, III, V or VI (see Fig. 7(d)). Considering i the number of steps until producing a Class I, V or VI triangle containing Q, additional bisections are propagated on each t_2^j for $j = 1, \ldots, i$ only if there was a triangle t_1^j that belonged to Class II or III, otherwise no propagation is required. Although this class should not

perform worse than Class III, for an α close to 0 the propagation could produce an extra point in each step, so the number of new points would be bounded by $O(\log^2(\frac{\pi}{6\alpha}))$.

5. **Class V and VI ($CQ_1 \leq AQ_1 \leq CA$):** The behavior for these triangles is the same as in Class I since the affected edge CA becomes the longest edge of t_1. □



(a) Classes I, V and VI (b) Class II

(c) Class III (d) Class IV

Fig. 7: Behaviour of the longest edge bisection algorithm for each similarity class when a point Q is affecting the smallest edge of non-conforming triangle $t(ABC)$

Note that the number of new triangles produced is one more than the number of bisections performed. Thus, for the triangle refinement problem the algorithm performs $k = O(\log \frac{1}{\alpha_0})$ bisections on each triangle refined by propagation, with α_0 the smallest angle of the triangles affected. Considering m the number of triangles effectively refined, the longest edge refinement algorithm produces $O(km)$ new points. Working with quality acceptable triangulations, the value of k becomes negligible.

For $|S_{ref}| > 1$, the local refinement problem, consider $M = M_{ref} + M_{prop}$ the number of triangles effectively refined, where $M_{ref} = |S_{ref}|$ is the number of triangles initially selected for refinement, and M_{prop} is the number of triangles refined due to propagation. The following lemma summarizes the cost of the algorithm for the local refinement problem.

Lemma 6. *A longest edge bisection refinement algorithm inserts at most $O(kM)$ new points into a triangulation with smallest angle α_0, where M is the number of triangles effectively refined and $k = O(\log \frac{1}{\alpha_0})$.*

The algorithm produces the most points when the refinement propagates to the smallest edge of the neighbor triangle as discussed in Sec. 4.1. Then, the proof of the lemma follows from Lemma 5. □

For a quality acceptable triangulation the value of k is negligible, and the number of new points produced becomes linear in M. For example, in a triangulation with $\alpha_0 \geq \pi/6$ the algorithm inserts at most $3M$ new points, since no triangle belongs to Classes III or IV. On the other hand, if we allow Class III or IV triangles in the triangulation, for example with $\alpha_0 \geq \pi/18$, these triangles could not be iteratively refined more than three times.

In the worst possible scenario, the propagation affects every triangle of a bad quality triangulation (e.g. a triangulation covered by spikes of Class III or IV triangles), so every triangle of the triangulation is refined during a first iteration. It is important to note that the algorithm produces Class I, V and VI triangles to solve hanging points in non-conforming triangles. The effect of this is that "better" quality triangles tend to cover the propagation path, so future iterations would be processed faster and their propagations reduced as the smallest angles are isolated.

When multiple hanging points are inserted over the same edge (e.g., due to the iterative bisection of Class III or IV triangles), the algorithm processes non-conforming neighbor solving the hanging points in the same order they were produced. Again, as better triangles are produced in this area, solving the following hanging points would insert a constant number of new points.

4.3 Iterative refinement around a vertex

Iterative refinement of the mesh reduces the number of bisected triangles since the propagation is reduced. The area covered by affected triangles is also reduced. In practice, the number of triangles refined by propagation is less than five. Additionally, iterative refinement produces more quasi-equilateral triangles (isolating the bad triangles), so non-conforming issues are solved more quickly.

This is related to the *fractal* property, observed after successive iterations of the refinement around a vertex. After a number of iterations the algorithm performs repetitive patterns of bisections – called fractals – as it gets "closer" to the vertex being refined. In this sense, the iterative refinement maintains a *locality* on the new triangles generated, reducing the propagation in each iteration. The following lemma address the fractal property as stated in [17].

Lemma 7. *For any vertex Q, use the algorithm to repeatedly refine each triangle of vertex Q. Then after a finite number of triangle refinements around Q, a fixed angle molecule is obtained (the angles of vertex Q are not partitioned if the refinement follows). In addition, further refinement around Q reproduces the same fractal geometry.*

The proof of the lemma is given by the fact that if the mesh has smallest angles greater than or equal to α, then vertex Q is shared by at most $2\pi/\alpha$

triangles. Then, from the results of Sec. 3.3 we can note that the number of non-similar triangles is finite, so after a finite number of partitions the triangle geometry around vertex Q is reproduced. □

Consider triangle t with α_1, α_2 and α_3 the biggest, second biggest and smallest angle of t respectively. The speed by which the fractal property is observed inside t will depend on the angle in which Q is located:

- If Q corresponds to the smallest angle α_3, iterative refinement will generate a zig-zag pattern toward the vertex (see Fig. 8(a)). This happens because every two bisections, the triangle containing Q is similar to t.
- If Q corresponds to the biggest angle α_1 or second biggest angle α_2, iterative refinement will split the triangles into smaller ones, until the new triangles starts generating the aforementioned zig-zag pattern (see Fig. 8(c) and Fig. 8(b)).

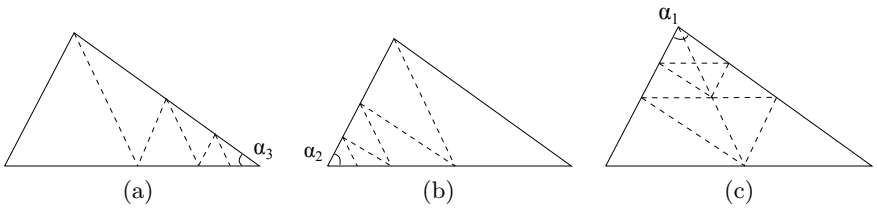


Fig. 8: Examples of the fractal property after the iterative refinement over a vertex Q: (a) over the smallest angle; (b) over the second biggest angle; (c) over the biggest angle

The current stage of our research is focused on using the results of Lemma 5 in Sec. 4.2 to bound the cost of the iterative refinement around a vertex. Based on Lemma 7, our hypothesis is that the propagation is reduced in each iteration until the work of refining the triangles around the vertex becomes constant (i.e., the fractal pattern appears). Since only initial iterations would propagate to more triangles, an amortized cost analysis would be required to establish the cost of the algorithm.

5 Conclusions and Future Work

In this paper we presented a new study on the size optimality of the longest edge bisection algorithm for the refinement of triangulations in two dimensions. In practice, longest edge algorithms for iterative refinement perform well, refining only a constant number of triangles in order to maintain a conforming triangulation. These algorithms currently offer guarantees only on the shape of the output triangulation, so our study aimed to establish a theoretical bound on its size.

The study was based on the behavior of the algorithm propagation. When propagating to the longest edge of a neighbor triangle, the propagation stops. For the second longest edge, propagation moves in only one direction. We focused the analysis on the non-trivial scenario where propagation moves to the smallest edge of a triangle, as it might insert more points. Even for this scenario, the number of points produced by the algorithm remains within a constant factor of the number of mandatory points required by the application. This factor will depend on the smallest angle in the triangulation.

Currently we are working on a tighter bound on the size of the propagation produced by the refinement. Our hypothesis is that iterative refinement reduces the propagation to a constant factor, corresponding to fractal patterns, so that the cost associated with the propagation would be amortized.

We are also working on a more detailed classification of the triangles based on the behavior of the propagation. For example, there are triangles from subregions of Class IV that behave similarly to Class I triangles.

For future work we plan to extend the results to the vertex refinement problem introduced in Sec. 4.3. The analysis of this problem is similar to the iterative refinement of bad quality triangles reviewed in Sec. 4.2. We also plan to extend these theoretical results to other longest-edge refinement algorithms such as Lepp-Delaunay [16].

References

1. B. S. Baker, E. Grosse, and C. S. Rafferty. Nonobtuse triangulation of polygons. *Discrete & Computational Geometry*, 3:147–168, 1988.
2. T. Baker. Automatic mesh generation for complex three-dimensional regions using a constrained delaunay triangulation. *Engineering with Computers*, 5:161–175, 1989. 10.1007/BF02274210.
3. M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. *Journal of Computer and System Sciences*, 48(3):384–409, June 1994.
4. M. W. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D.-Z. Du and F. K.-M. Hwang, editors, *Computing in Euclidean Geometry*, number 1 in Lecture Notes Series on Computing, pages 23–90. World Scientific, 1992.
5. H. Borouchaki and P. L. George. Aspects of 2-d delaunay mesh generation. *International Journal for Numerical Methods in Engineering*, 40(11):1957–1975, 1997.
6. L. P. Chew. Constrained delaunay triangulations. In *Proceedings of the third annual symposium on Computational geometry*, SCG '87, pages 215–222, New York, NY, USA, 1987. ACM.
7. H. Edelsbrunner. Triangulations and meshes in computational geometry. *Acta Numerica*, 9:133–213, 2000.
8. C. Gutierrez, F. Gutierrez, and M.-C. Rivara. Complexity of the bisection method. *Theoretical Computer Science*, 382(2):131–138, 2007.
9. M. T. Jones and P. E. Plassman. Computational results for parallel unstructured mesh computations. Technical report, Knoxville, TN, USA, 1994.

10. M. T. Jones and P. E. Plassmann. Adaptive refinement of unstructured finite-element meshes. *Finite Elements in Analysis and Design*, 25(1-2):41–60, 1997.
11. E. A. Melissaratos and D. L. Souvaine. Coping with inconsistencies: a new approach to produce quality triangulations of polygonal domains with holes. In *Proceedings of the eighth annual symposium on Computational geometry*, SCG '92, pages 202–211, New York, NY, USA, 1992. ACM.
12. S. N. Muthukrishnan, P. S. Shiakolas, R. V. Nambiar, and K. L. Lawrence. Simple algorithm for the adaptive refinement of three dimensional problems with tetrahedral meshes. *AIAA Journal*, 33(5):928–932, 1995.
13. R. V. Nambiar, R. S. Valera, K. L. Lawrence, R. B. Morgan, and D. Amil. An algorithm for adaptive refinement of triangular element meshes. *International Journal for Numerical Methods in Engineering*, 36(3):499–509, 1993.
14. M.-C. Rivara. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *International Journal for Numerical Methods in Engineering*, 20(4):745–756, 1984.
15. M.-C. Rivara. Design and data structure of fully adaptive, multigrid, finite-element software. *ACM Transactions on Mathematical Software*, 10(3):242–264, 1984.
16. 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.
17. M.-C. Rivara. Lepp-bisection algorithms, applications and mathematical properties. *Applied Numerical Mathematics*, 59(9):2218–2235, 2009.
18. M.-C. Rivara, P. Rodriguez, R. Montenegro, and G. Jorquera. Multithread parallelization of lepp-bisection algorithms. *Applied Numerical Mathematics*, 62(4):473–488, 2012.
19. J. Ruppert. A delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548 – 585, 1995.
20. R. Schneiders. Quadrilateral and Hexahedral Element Meshes. In J. Thompson, B. Soni, and N. Weatherill, editors, *Handbook of Grid Generation*. CRC Press, 1999.
21. J. R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(1-3):21–74, 2002.
22. J. R. Shewchuk. Unstructured mesh generation. In U. Naumann and O. Schenk, editors, *Combinatorial Scientific Computing*, chapter 10, pages 259–298. CRC Press, 2011.
23. J. P. Suárez, A. Plaza, and G. F. Carey. The propagation problem in longest-edge refinement. *Finite Elements in Analysis and Design*, 42(2):130–151, Nov. 2005.