Relaxed Lepp-Delaunay algorithms for the refinement / improvement of triangulations

Pedro A. Rodriguez\textsuperscript{a}, Maria-Cecilia Rivara\textsuperscript{b,}\textsuperscript{*}

\textsuperscript{a}Departamento de Sistemas de Información, Universidad del Bio-Bio, Av. Collao 1202, Concepción, 4051381, Chile
\textsuperscript{b}Departamento de Ciencias de la Computación, Universidad de Chile, Avenida Beauchef 851, Santiago, 8370456, Chile

Abstract

We discuss serial and multicore Relaxed Lepp-Delaunay algorithms for triangulation refinement, based on inserting the centroid of associated terminal triangles (that share local longest edge in the mesh), and where a neighborhood parameter $K$ is used to constrain the edge flipping propagation around the terminal edge. Empirical results on a multicore Relaxed Lepp-Delaunay centroid algorithm, show that an efficient and scalable multicore algorithm was obtained.

Keywords: centroid; Delaunay; improvement; Lepp; Lepp-Delaunay; multicore; relaxed Lepp-Delaunay; refinement; thread;

1. Introduction

Longest-edge refinement algorithms for triangulations, based on bisecting the triangles by the longest-edge, were designed to support the development of adaptive finite element software and to guarantee the construction of refined triangulations that maintain the quality of the input mesh [1]. Later the longest-edge propagating path (Lepp) concept was introduced by Rivara [2] to design both the Lepp-bisection algorithm (an efficient and simple longest-edge algorithm) and Lepp Delaunay algorithms for the automatic construction of quality triangulations.

Lepp-Delaunay algorithms combine the Lepp concept and Delaunay insertion of the selected points. Lepp-centroid algorithm has been studied by Rivara and Calderon [5] and Lepp midpoint algorithms has been studied by Bedregal and Rivara [3]. A study on multicore Lepp-bisection algorithm was presented in [6].

In this paper we propose a Relaxed Lepp-Delaunay method to refine and improve triangulations, where the delaunization step is relaxed by using a parameter $K$ that constrain the edge flipping propagation around the terminal edge. We present empirical results on a multicore relaxed Lepp-Delaunay algorithm for solving the quality triangulation problem. This method generalizes the Lepp-Centroid Delaunay method discussed in [5].

* Corresponding author. Tel.: +56-2-2978-4365 ; fax: +56-2-2689-5531.
E-mail address: mcrivara@dcc.uchile.cl (Maria-Cecilia Rivara)
2. Lepp-Delaunay centroid method

An edge $E$ is called a terminal edge [2] in triangulation $\tau$ if $E$ is the longest edge of every triangle that shares $E$, while the triangles that share $E$ are called terminal triangles [2]. Note that in 2-dimensions either $E$ is shared by two terminal triangles $t_1, t_2$ if $E$ is an interior edge, or $E$ is shared by a single terminal triangle $t_1$ if $E$ is a boundary (constrained) edge. See Figure 1 where edge $AB$ is an interior terminal edge shared by two terminal triangles $t_3, t_4$.

For any triangle $t_0$ of a conforming triangulation $\tau$, the longest-edge propagating path of $t_0$, denoted by $\text{Lepp}(t_0)$, is the finite list of increasing triangles $t_0, t_1, t_2, ..., t_{n-1}, t_n$, such that $t_i$ is the neighbor triangle of $t_{i-1}$ on a longest edge of $t_{i-1}$, for $i = 1, 2, ..., n$ [2]. Note that in general $t_{n-1}, t_n$ are terminal triangles sharing an interior terminal edge.

For improving a triangle $t$, the first Lepp-Delaunay algorithm [2] repeatedly selects the midpoint of the terminal edge which is Delaunay inserted in the mesh until the triangle $t$ is refined. Later Rivara and Calderon introduced the Lepp-Delaunay centroid algorithm [5] where the centroid of the terminal quadrilateral formed by a couple of terminal triangles is selected for Delaunay point insertion. For an illustration of the centroid algorithm see Figure 1, where for improving $t_0$, the centroid $P$ of the terminal triangles $t_3, t_4$ is Delaunay inserted which produces the triangulation of Figure 1 (b). Then, for improving $t_0$ (that remains in the mesh), the centroid of the terminal triangles $t_0, t'_1$ is inserted, which destroys $t_0$.

3. A serial relaxed Lepp-Delaunay algorithm

In this algorithm we use a parameter $K$ that allows to define a neighbor set of triangles $NS_k$ that constrain the edge flipping propagation. In this way, we use a quasi-Delaunay point insertion operation.

**Algorithm 1 Relaxed-Lepp-Delaunay Algorithm($\tau_0, \theta_{tol}, K$)**

Input: $\tau_0$ initial conforming mesh, threshold angle tolerance $\theta_{tol}$ and parameter $K$.
Output: An improved conforming triangulation $\tau_f$.
Find $S \subset \tau$ the set of triangles with smallest angle $< \theta_{tol}$.
while $S \neq \emptyset$ do
    Select a triangle $t$ from $S$.
    while $t$ remains in $\tau$ do
        Find Lepp of $t$ and compute the centroid $M$ of the terminal quadrilateral.
        Find set $NS_k(E)$.
        Insert the centroid $M$ by using the relaxed Lepp-Delaunay point insertion (constrained to $NS_k(E)$).
        Update $S$.
    end while
end while

**Definition.** Given a terminal edge $E$, for $K=0$, the neighbor set of triangles $NS_0(E)$ includes the terminal triangles associated to $E$. For $K > 0$, the neighbor set of triangles $NS_k(E)$ includes the triangles of $NS_{K-1}(E)$ and its exterior edge-adjacent triangles (see Figure 2).
The serial relaxed Lepp-Delaunay algorithm proceeds as follows: for each bad quality triangle \( t_0 \) to be refined, the algorithm finds \( \text{Lepp}(t_0) \), the terminal edge \( E \), the centroid \( M \) of the terminal triangles and a set \( NS_K(E) \) over which the quasi-Delaunay point insertion operation is performed. Triangulations (b), (b), (c) of Figure 2 show the \( NS_K \) sets for \( K = 0, 1, 2 \). Figure 2 (d) shows the quasi-Delaunay mesh obtained after quasi-Delaunay insertion of \( M \) for \( K=2 \).

![Fig. 2. Shadow triangles identify \( NS_K(E) \). (a) \( NS_0(E) \) includes the terminal triangles; (b) \( NS_1(E) \) includes terminal triangles (\( NS_0(E) \)) and their immediate neighbors; (c) \( NS_2(E) \) includes \( NS_1(E) \) and their immediate neighbors; (d) After the quasi-Delaunay insertion of centroid \( M \) for \( K=2 \).](image)

4. Practical behavior of the serial algorithm as a function of \( K \)

We used the serial relaxed Lepp-Delaunay algorithm for studying both the evolution of the angle distribution and the number and percentage of the non-Delaunay triangles obtained in the final mesh for different values of \( K \). Table 1 summarizes these results for \( \theta_{tol} = 30^\circ \). This includes the size of the meshes, number and percentage of non-Delaunay triangles and the execution time for different values of \( K \). Note that the final meshes have approximately the same number of elements (vertices and triangles), but the number and percentage of non-Delaunay triangles in the final meshes are different. Note that when \( K=0 \) the algorithm only inserts the centroid into a couple of terminal triangles without carrying out edge flipping operations.

Table 1. Final meshes and Percentage of Delaunay triangles for input and final meshes obtained from different values of \( K \), threshold angle 30°.

<table>
<thead>
<tr>
<th>( K )</th>
<th>Vertices</th>
<th>Triangles</th>
<th>Non-Delaunay Triangles (NDT)</th>
<th>Percentage of NDT</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init Mesh →</td>
<td>2,999,998</td>
<td>5,999,953</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Final mesh, ( K=0 )</td>
<td>10,929,370</td>
<td>21,841,912</td>
<td>2,552,911</td>
<td>11.69</td>
<td>330,511</td>
</tr>
<tr>
<td>Final mesh, ( K=1 )</td>
<td>10,885,375</td>
<td>21,753,875</td>
<td>40,004</td>
<td>0.1839</td>
<td>370,841</td>
</tr>
<tr>
<td>Final mesh, ( K=2 )</td>
<td>10,867,452</td>
<td>21,718,098</td>
<td>44</td>
<td>0.000203</td>
<td>331,952</td>
</tr>
<tr>
<td>Final mesh, ( K=3 )</td>
<td>10,864,202</td>
<td>21,711,627</td>
<td>12</td>
<td>0.000055</td>
<td>362,422</td>
</tr>
<tr>
<td>Final mesh, ( K=4 )</td>
<td>10,863,910</td>
<td>21,711,078</td>
<td>10</td>
<td>0.000046</td>
<td>368,190</td>
</tr>
<tr>
<td>Final mesh, ( K=7 )</td>
<td>10,863,878</td>
<td>21,711,007</td>
<td>0</td>
<td>0.000000</td>
<td>445,158</td>
</tr>
<tr>
<td>Final mesh, ( K=10 )</td>
<td>10,863,826</td>
<td>21,710,922</td>
<td>0</td>
<td>0.000000</td>
<td>589,493</td>
</tr>
</tbody>
</table>

As expected the percentage of non-Delaunay triangles obtained decreases when the value of \( K \) increases. However this remains very low for \( K \geq 3 \), which suggests that either \( K=2 \) or \( K=3 \) is a good parameter value.

Table 2 summarizes the distribution of the smallest angles (between 0 and 60 degrees) in triangle percentage for the initial and final meshes, for \( K=0,1,2,3,4,7,10 \). Note that good quality meshes formed by triangles with good internal angles (threshold 30°) are obtained even when the mesh is not fully Delaunay. Note that even for \( K=0 \) all the bad quality triangles (needle, cap, etc) are eliminated from the mesh.
Table 2. Distribution (in triangle percentage) of smallest angles for different values of \( K, \theta_{tol} = 30^\circ \).

<table>
<thead>
<tr>
<th>Degrees</th>
<th>( 0^\circ - 10^\circ )</th>
<th>( 10^\circ - 20^\circ )</th>
<th>( 20^\circ - 30^\circ )</th>
<th>( 30^\circ - 40^\circ )</th>
<th>( 40^\circ - 50^\circ )</th>
<th>( 50^\circ - 60^\circ )</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>Degrees</th>
<th>( 0^\circ - 10^\circ )</th>
<th>( 10^\circ - 20^\circ )</th>
<th>( 20^\circ - 30^\circ )</th>
<th>( 30^\circ - 40^\circ )</th>
<th>( 40^\circ - 50^\circ )</th>
<th>( 50^\circ - 60^\circ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>42.0955</td>
<td>44.6932</td>
<td>13.2113</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>36.9587</td>
<td>48.7855</td>
<td>14.2558</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>37.0057</td>
<td>48.7315</td>
<td>14.2627</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>37.0166</td>
<td>48.7239</td>
<td>14.2595</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>37.0170</td>
<td>48.7235</td>
<td>14.2595</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>37.0169</td>
<td>48.7235</td>
<td>14.2595</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>37.0169</td>
<td>48.7236</td>
<td>14.2595</td>
<td></td>
</tr>
</tbody>
</table>

5. Practical performance of the parallel Relaxed Lepp-Delaunay algorithm

Given an input triangulation \( \tau \), a set \( S \subset \tau \) of bad quality triangles and a parameter \( K \). Then for each triangle \( t \) in \( S \), the parallel relaxed algorithm proceeds as follows: (1) Lepp\( (t) \), and centroid \( M \) of the terminal quadrilateral are computed; (2) The \( NS_{K}(E) \) set is found; (3) If \( NS_{K}(E) \) is computed without detecting collisions, then the centroid \( M \) is Delaunay inserted into the mesh. Otherwise, the computation is stopped and the core proceeds to pick up a new triangle from \( S \). Algorithm 2 summarizes the parallel relaxed Lepp-Delaunay centroid algorithm:

**Algorithm 2** Multicore Lepp-Delaunay algorithm

Input: \( \tau_0 \) an initial mesh, threshold angle tolerance \( \theta_{tol} \), parameter \( K \).  
Output: An improved conforming triangulation \( \tau_f \).  
Find \( S \subset \tau \) the set of triangles with smallest angle < \( \theta_{tol} \).  
while \( S \neq \phi \) do  
    Take a triangle \( t \) from \( S \).  
    while \( t \) remains in \( \tau \) do  
        Find Lepp\( (t) \).  
        Find \( NS_{K}(E) \) and compute centroid \( M \) of the terminal quadrilateral.  
        if Collision is detected while computing a Lepp\( (t) \) or \( NS_{K} \) \then  
            Destroy Lepp\( (t) \) and take a new triangle \( t \) from \( S \).  
        else  
            Lock the triangles of \( NS_{K} \).  
            Insert the centroid \( M \) into the mesh by using relaxed Lepp-Delaunay point insertion.  
            Update \( S \).  
        end if  
    end while  
end while

We have used a computer with two Intel Xeon E5-2660 processors (20 physical cores, 10 core per socket) for testing the algorithm behavior. We used several triangulations of sets of randomly generated points over a rectangle. The input domain was divided in a grid of rectangles in order to distribute the triangles and the workload between the threads. Empirical work shows that the multicore algorithms (for \( K \leq 3 \)) have good scalable behavior until 20 processors are used,
In Table 3 and Figure 3 we present results for the meshes of Table 1 (input and final meshes of approximately 6 millions and 21.7 millions of triangles respectively, for $\theta_{tol} = 30^\circ$). Table 3 shows the efficiency behavior and Figure 3 shows the speedup behavior.

Table 3. Performance measure: efficiency; threshold angle 30°; Intel Xeon E5550.

<table>
<thead>
<tr>
<th>K</th>
<th>1P</th>
<th>2P</th>
<th>4P</th>
<th>8P</th>
<th>10P</th>
<th>16P</th>
<th>20P</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0</td>
<td>0.79</td>
<td>0.73</td>
<td>0.68</td>
<td>0.67</td>
<td>0.77</td>
<td>0.66</td>
</tr>
<tr>
<td>2</td>
<td>1.0</td>
<td>0.76</td>
<td>0.68</td>
<td>0.64</td>
<td>0.61</td>
<td>0.75</td>
<td>0.64</td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td>0.68</td>
<td>0.65</td>
<td>0.61</td>
<td>0.60</td>
<td>0.72</td>
<td>0.60</td>
</tr>
<tr>
<td>4</td>
<td>1.0</td>
<td>0.66</td>
<td>0.58</td>
<td>0.53</td>
<td>0.52</td>
<td>0.53</td>
<td>0.44</td>
</tr>
</tbody>
</table>

Fig. 3. Speedup for $K=0,2,3,4$, threshold angle 30°. Intel Xeon E5-2660, 2, 4, 8, 10,16 and 20 cores.

Acknowledgements

Work partially supported by Departamento de Ciencias de la Computación, Universidad de Chile, Departamento de Sistemas de Información and Research Group GI150115/EF, Universidad del Bio-Bio. Powered@NLHPC: This research was partially supported by the supercomputing infrastructure of the NLHPC (ECM-02)

References