Delaunay triangulation using a parallel architecture

Andras P. Findt, Willem A. Clarke
School of Electrical and Electronic Engineering,
University of Johannesburg,
Auckland Park, South Africa
Email: andras@hypervisionlab.com

Abstract—A method for generating the Delaunay triangulation of a given set of points efficiently in parallel is explored. Although Delaunay triangulation is well-defined for sequential processing architectures, a parallel implementation could improve performance and allow for larger problems to be computed than a serial architecture would ordinarily handle. Various approaches are evaluated and a suitable method is chosen and implemented. A further aim is then to implement the solution on modern graphics hardware (GPU).

Index Terms—Delaunay Triangulation, Parallel Computing, GPGPU

I. INTRODUCTION

The Delaunay triangulation is a well-defined and well-documented concept in computational geometry [1]. In many areas of research, it is often important to triangulate a set of points and a number of algorithms have been developed for serially computing the Delaunay triangulation [2]. Since the generation of the Delaunay triangulation is such a complex problem, it would greatly benefit from an efficient parallel implementation. The aim of this project is to determine which, if any, of these serial algorithms best lend themselves to parallelisation, and then to implement that algorithm on a parallel processing architecture. A GPU (Graphics Processing Unit) implementation of the Delaunay will also be attempted.

II. DELAUNAY TRIANGULATION

A. Conditions

The Delaunay triangulation is defined by the following rules:

- For a set of \( P \) points in \( n \)-dimensional Euclidean space, \( DT(P) \) is a triangulation such that no point \( D \) in \( P \) is inside the circumsphere of any triangle \( \triangle ABC \) in \( DT(P) \).
- There is no Delaunay triangulation \( DT(P) \) for set \( P \) if all points in \( P \) are collinear.

B. Algorithms

There are a number of different approaches to the problem of Delaunay triangulation which will be discussed here.

1) Triangle Flipping: This process involves creating an arbitrary triangulation of the points in \( P \), and then flipping edges of the triangles until no triangle is non-Delaunay. Triangle flipping is not an ideal solution since in the worst case it requires \( O(n^2) \) flips [3]. It is therefore often used simply as a final step to ensure that all triangles adhere to the Delaunay conditions.

2) Divide and Conquer: This approach involves recursively drawing a line that splits the points into two sets. Each set then has its Delaunay triangulation computed and the sets are then merged along the splitting line[s] [4] [5]. Divide and conquer is typically used for triangulations in two dimensions and has been shown to be the fastest \( DT(P) \) generation technique [6].

3) Incremental: Incremental generation of \( DT(P) \) involves repeatedly adding a point from the set \( P \) to the triangulation and retriangulating only the part of the graph affected by the addition of the point [7].

4) Sweepline: This algorithm uses a line that moves from one side of a plane to the other and adds each point as it passes it. When it reaches each point in the plane, a parabolic beach line forms that is used to calculate the Voronoi diagram [2].

5) Sweep-hull: Sweep-hull is a hybrid algorithm which uses a radially propagating sweep-hull, which is generated from a radially sorted set of points in two dimensions. This, coupled with a final triangle flipping step provides the Delaunay triangulation for the set of points [8].
III. PARALLEL ALGORITHM DESIGN

With recent advances in parallel processing the Delaunay triangulation may be computed efficiently. Furthermore, a parallel implementation will allow for better performance and allow for larger problems that serial architectures cannot handle [9].

There have been a few attempts at parallelising the Delaunay triangulation with generally worse results than expected [10]. Blelloch et al. propose an approach which involves using the the reduction of 2D triangulation to 3D convex hull of points [11]. Their approach roughly follows the style of Divide and Conquer. An incremental Delaunay triangulation was proposed by Kohout [12].

We chose to develop an initial parallel algorithm using MPI (Message Passing Interface) for multi-process execution. By following an approach loosely based on the Divide and Conquer algorithm discussed above, we sorted the problem set into a number of subsets equal to the number of processes to be used in execution. The Delaunay triangulation is then performed on each individual subset, whereafter all triangulations are combined to generate the final triangulation as seen in Figure 1. The results of this initial implementation are displayed in the next section.

IV. RESULTS AND FUTURE WORK

In the results below, each

A. Testing Parameters

Computer used: AMD Athlon II X2 250, 4GB DDR3 RAM, NVIDIA GTX260 896MB. Number of points used: 512 random points. Number of processes: 1-16.

B. Preliminary Results

The table below displays the time it took to run the program on different numbers of processes. The final column shows the speedup of each number of processes over one process. Each test was repeated 10 and the average taken to ensure accuracy of results.

<table>
<thead>
<tr>
<th># Processes</th>
<th># Points</th>
<th>Average Time Taken (s)</th>
<th>Times Faster</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>512</td>
<td>8.024</td>
<td>—</td>
</tr>
<tr>
<td>2</td>
<td>512</td>
<td>1.375</td>
<td>5.835</td>
</tr>
<tr>
<td>3</td>
<td>512</td>
<td>0.917</td>
<td>8.750</td>
</tr>
<tr>
<td>4</td>
<td>512</td>
<td>0.592</td>
<td>13.600</td>
</tr>
<tr>
<td>6</td>
<td>512</td>
<td>0.394</td>
<td>20.441</td>
</tr>
<tr>
<td>8</td>
<td>512</td>
<td>0.362</td>
<td>22.289</td>
</tr>
<tr>
<td>16</td>
<td>512</td>
<td>0.346</td>
<td>23.191</td>
</tr>
</tbody>
</table>

C. Future Work

We are currently in the process of creating a GPU-based implementation of this algorithm. The parallel architecture of the GPU should allow for even greater performance gains, and even allow for real-time triangulation of very large problem sets. This will be explored in future work.

REFERENCES


András Findt received his undergraduate degree in 2010 from the University of Johannesburg. After participating in the 2009 UJ Image Processing Challenge, he was invited to join the HyperVision Research Lab as an assistant researcher. He is currently studying towards his Honours in Computer Science. His research interests include Feature Tracking, Texture Analysis and GPGPU Programming.