A Simple Parallel Algorithm for the Stepwise Approximate Computation of Voronoi Diagrams of Line Segments

Christof Meigen Jörg Keller

FernUniversität in Hagen
FB Informatik — LG Parallelität und VLSI
58084 Hagen, Germany
joerg.keller@fernuni-hagen.de

Abstract: We present a PRAM algorithm to approximate the Voronoi diagram of line segments for a grid of points in the plane. The algorithm combines ideas from computer graphics and from hierarchical approximation algorithms. We analyze the complexity and report on experimental results of an implementation on a PRAM simulator. The algorithm can be extended to other types of objects and distance metrics. We sketch how to extend the algorithm for message-passing machines.

1 Introduction

Voronoi diagrams are partitions of the plane according to a given set of objects, so that the Voronoi region of a certain object consists of all the points of the plane that are closer to this particular object than to the others. While the computation of the Voronoi diagram of points in the plane is a well-understood problem, algorithms for the Voronoi diagram of line segments in the plane suffered from numerical problems for a long time. Only recently, a robust, sequential implementation based on incremental construction has been announced as part of the Computational Geometry Algorithms Library (CGAL) [Pro04, Kar04]. As for other problems, large instances call for the investigation of a parallel solution.

Parallel Algorithms for the construction of Voronoi diagrams of points in the plane are known since the eighties [ACG$^+$88]. For line segments various approaches have been published [DZ99, GOY93] but remained, as far as we know, unimplemented. From our own experiences and remarks in the literature [ACG89, BMS94, MN99] we see numerical problems and the sheer complexity of the involved data structures as the major drawback for implementations. A main reason for the numerical difficulties is the fact that, in difference to Voronoi diagrams of point sets, bisectors between regions of line segments consist of curved parts. Hence, this difficulty will also arise for other objects besides points.

Recently, focus has shifted to approximate solutions to the problem. On the one hand, one can choose a regularly spaced grid of points, and compute for each point to which Voronoi region it belongs. If Voronoi regions are assigned colors, this can be used to draw Voronoi

diagrams at different resolutions. If we consider 3D graphics, we place the plane with the line segments in the view port, and add as the third dimension the distance from objects. Then the distance function of all points from one object forms a curved surface in 3D. If each surface is assigned the color of the Voronoi region of the respective object, then a map of the Voronoi diagram can be drawn by rendering this scene with graphics processors [KEHKL$^+$99]. Also, cellular automata implemented in special-purpose hardware can be used to construct discrete approximations by dividing the plane into a finite set of points [Deh89]. While these algorithms are elegant and achieve impressive performance, they rely heavily on the enormous throughput of the particular hardware and perform rather poorly when executed on general purpose processors.

On the other hand, there are several sequential algorithms [BCS02, VO98] that approximate a Voronoi diagram by repeated hierarchical partioning of the plane, such as binary space partitioning. We combine both ideas into a PRAM algorithm. The structure of the PRAM algorithm is simple in contrast to the exact algorithms, and it is faster than sequential algorithms while it does not rely on special hardware like graphics processors. We analyse the complexity of the algorithm and present experimental results with an implementation in the FORK programming language, running on the SB-PRAM simulator [KKT01]. While we focus on the Voronoi diagram of line segments with the euclidean distance metric, the algorithm works for objects consisting of arbitrary point sets, and arbitrary distance metric, as long as we can compute the distance between a point in the plane and an object in constant time.

2 Algorithm

Our aim is, for a regularly-spaced grid of points in the plane, and for a set of n objects in the plane, to compute for each grid point to which Voronoi region it belongs. If the grid consists of $k \times k$ points, then this can be done in a trivial way in time $\mathcal{O}(k^2 \cdot n)$, by computing for each point the distances to all objects, and finding the minimum. This algorithm can be easily and work-optimally parallelized for up to k^2 processors.

Instead of using the simple algorithm, we start with a low resolution (small k_s) and gradually refine the resolution until we reach the final resolution (large k_e). We regard the grid points as centers of squares (pixels) with a side length identical to the distance between points in this grid's resolution. By the hierarchical refinement, we can save work if we know that several points in a higher resolution, all being in the proximity of one point v in the lower resolution, all belong to the same Voronoi region as v. At first sight, the refinement looks trivial: if all neighbours of a pixel belong to the same Voronoi region as itself, then no refinement seems necessary. However, the decision is more difficult. Figure 1 depicts a Voronoi diagram of two orthogonal line segments, with a bisector shaped as a parabola. All nine grid points (centers of the squares) belong to the Voronoi region of the horizontal line segment. However, a part of the central square belongs to the Voronoi region of the small, vertical line segment. Hence, when increasing the resolution, one would have to split the central square although all of its neigbouring squares belong to the same Voronoi region!

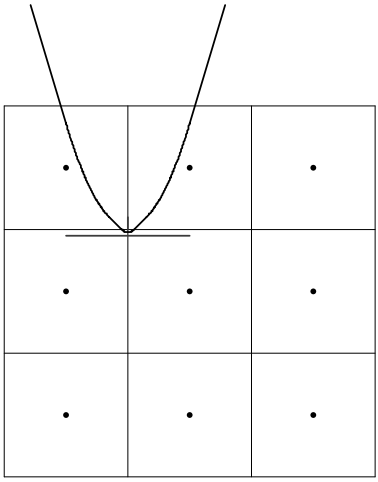

Figure 1: Voronoi diagram with all pixel centers in one Voronoi region

Our algorithm starts by dividing the square into just one pixel ($k_s = 1$) with a side length of k_e and assigning all the objects as candidates for the true Voronoi region of that pixel, with an object closest to the center of the pixel p being the so-called *favorite* $l_{\min}$.

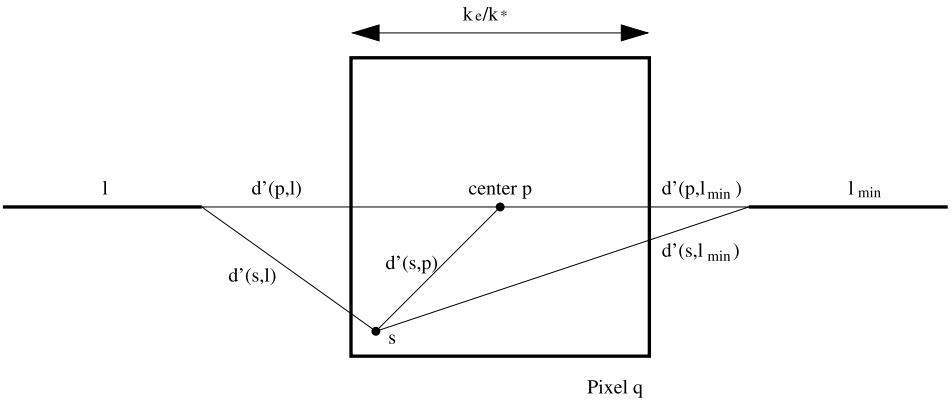

Figure 2: Sieving of a pixel's candidates

After determining the favorite, the candidates can be sieved whether they still have chances to win at least part of the pixel q against the favorite. If not, they can be discarded. We consider the situation in Figure 2. For each point s in the pixel it follows from the triangle inequality that its distance to the favorite $l_{\min}$ cannot be larger than its distance to the center p plus the distance between the center p and the favorite $l_{\min}$. The distance between s and the center p is less than or equal to $r = 0.5 \cdot \sqrt{k_e/k^*}$, when k_e/k^* is the actual side

length of the pixel. It follows that

$$d'(s, l_{\min}) \leq d'(p, l_{\min}) + r$$

where d' is the distance function. With the same argument, the distance between s and any other candidate l can be bound from below:

$$d'(s, l) \geq d'(p, l) - r \ .$$

A candidate l can be discarded if

$$d'(s, l) \geq d'(s, l_{\min}) \ .$$

for any point s in the pixel. From the above it follows that a sufficient condition, i.e. a criterion to discard candidates, is

$$d'(p, l) \geq d'(p, l_{\min}) + 2r \ .$$

If more than one candidate remains after the sieving of the candidates, the pixel is split up into an appropriate number (four) of subpixels which inherit the list of the remaining candidates from their parent. These new pixels can now be processed in parallel.

With $l_1 \ldots l_n$ being the n objects, $k_e \times k_e$ being the desired resolution of the final picture with a distance of 1 between grid points, and dividing a pixel always into four parts, the algorithm can be more formally stated as follows. Here, Q is the set of pixels, and each pixel is represented as a tuple consisting of resolution, center point, list of candidates, and favorite.

Algorithm 1 (Stepwise construction of a discrete Voronoi diagram)

$Q \leftarrow \{(k = 1; p = (\frac{k_e}{2}, \frac{k_e}{2}); L = \{l_1 \ldots l_n\}; l_{\min} = \emptyset)\}$
$k^* \leftarrow 1$
while $k^* \leq k_e$ **do**
  for all $q \in Q : q.k = k^*$ **do**
    Set $q.l_{\min}$ to a $l \in q.L$ satisfying $d'(q.p, l) = \min_{l^* \in q.L} d'(q.p, l^*)$
    $q.L \leftarrow \{l \in q.L | d'(q.p, l) < \frac{k_e}{k^*}\sqrt{2} + d'(q.p, q.l_{\min})\}$
    if $|q.L| > 1 \wedge k^* \neq k_e$ **then**
      $Q \leftarrow Q \backslash q$
      $Q \leftarrow Q \cup \{(k = 2k^*; p = q.p + (\frac{k_e}{4k^*}, \frac{k_e}{4k^*}); L = q.L; l_{\min} = \emptyset),$
        $(k = 2k^*; p = q.p + (\frac{k_e}{4k^*}, -\frac{k_e}{4k^*}); L = q.L; l_{\min} = \emptyset),$
        $(k = 2k^*; p = q.p + (-\frac{k_e}{4k^*}, \frac{k_e}{4k^*}); L = q.L; l_{\min} = \emptyset),$
        $(k = 2k^*; p = q.p + (-\frac{k_e}{4k^*}, -\frac{k_e}{4k^*}); L = q.L; l_{\min} = \emptyset)\}$
    end if
  end for
  $k^* \leftarrow 2k^*$
end while

k_e	Total no. of tests	Tests in different rounds
4	20	4/16
32	516	4/16/64/140/292
128	1 728	4/16/64/140/292/480/732
1 024	22 368	4/16/64/140/292/480/732/1 392/2 812/5 480/10 956

Table 1: Number of tests during step-wise computation of Voronoi diagram in Fig. 3

In the end, the set Q contains all the pixels, large or small, with the favorites $l_{\min}$ being the object they belong to, as far as the resolution k_e is concerned.

While this algorithm still has work in $\mathcal{O}(k_e^2 \cdot n)$ for the worst case, and therefore seems no improvement over the trivial solution to test all the distances for every pixel in the final picture, this happens only if the objects are clumped together beyond the resolution, which suggests that the area of interest was not chosen appropriately. For the case of uniformly distributed, non-intersecting line segments, the work seems to be sublinear with respect to n, as the experiments in the next section reveal. The reason is that the list of candidates is reduced to two for almost all pixels after a few steps, so that the algorithm is mainly concerned with drawing the border between two Voronoi regions more smoothly.

3 Experiments

We implemented the algorithm with the PRAM programming language FORK and ran it on the SB-PRAM simulator, see [KKT01]. The main data structure is the set Q. It is maintained as a parallel FIFO queue, which can be efficiently implemented on a PRAM with the help of parallel prefix commands, see [KKT01, Chap. 7]. Figure 3 depicts the Voronoi diagram of five line segments in different resolutions. Pixels that get split up are colored with a darker shade. Table 1 reports the number of tests (distance computations between points and objects) in the different steps of the computation of that Voronoi diagram. We clearly see that for higher resolutions, the number of tests does not increase with factor 4 (which would be the case if all pixels would be split up), but with a factor of about 2. Figure 4 depicts the number of tests for a fixed target resolution $k_e = 512$, and sets with an increasing number n of line segments. The line segments were chosen randomly, equally distributed in the plane, and without intersections. We see here that the number of tests increases less than linear with n, which supports our hypothesis from the previous chapter.

4 Conclusions and Extensions

We have presented a PRAM algorithm to compute an approximation of a Voronoi diagram of line segments in the plane. The algorithm works for any distance measure obeying the triangle inequality, and for any type of object, as long as the distance between a point and

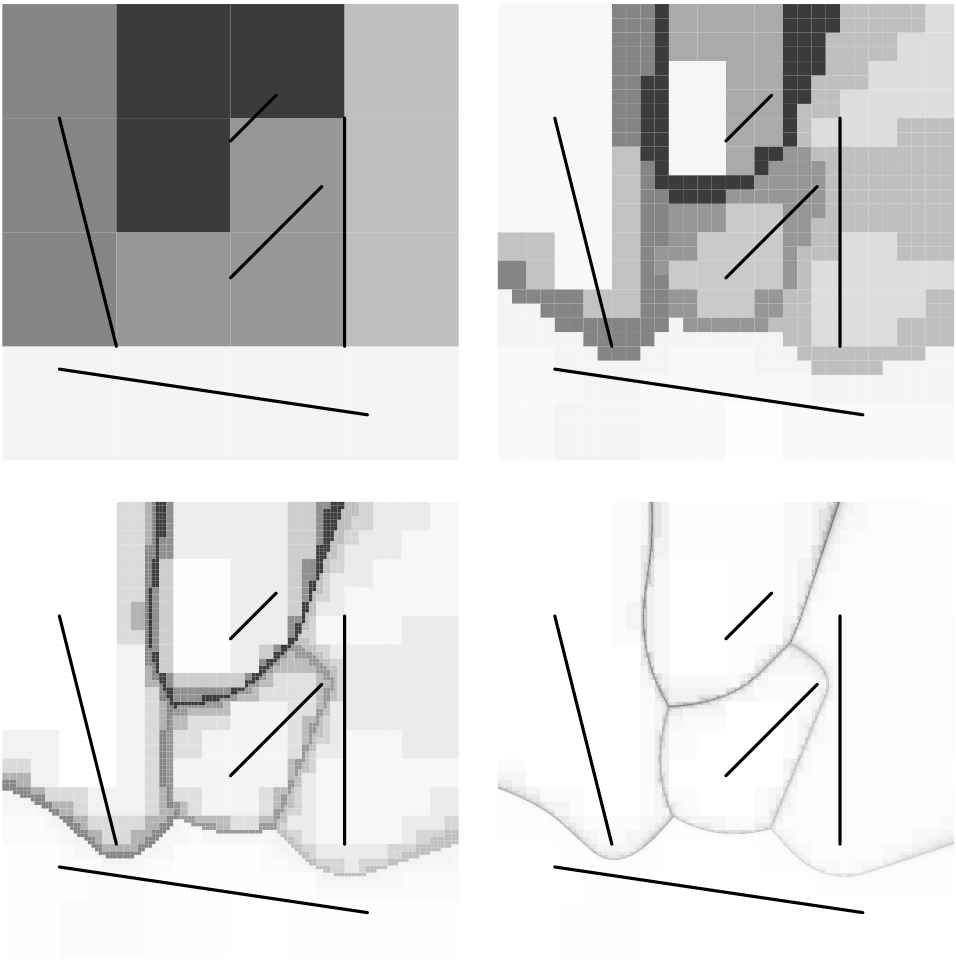

Figure 3: Step-wise computation of a Voronoi diagram of 5 line segments, with resolutions of 4, 32, 128 and 1024 pixels

n	Tests
4	10 979
8	18 119
16	26 503
32	44 599
64	66 181
96	82 559

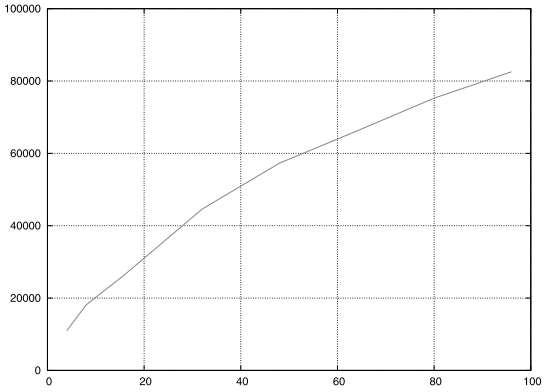

Figure 4: Number of distance computations (tests) for resolution $k_e = 512$ depending on n , for n randomly chosen line segments without intersections

an object can be computed in constant time. From the point of view of implementation, this algorithm can be directly formulated in any language providing a general *forall*-facility, such as PMLS; it is not restricted to Fork.

Even an implementation of the algorithm on message-passing systems seems possible, if the set Q can be maintained in a distributed way. One possibility would be to execute the algorithm sequentially up to a certain resolution, and then distribute the pixel set over all processors. Each processor can then work on its local part of the set, and only when some processors get idle, a load balancing has to be performed, which provides idle processors with pixels from busy processors. The object data would have to be replicated on each processor in this case.

References

[ACG+88] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. Yap. Parallel Computational Geometry. *Algorithmica*, 3:293–327, 1988.

[ACG89] M. J. Atallah, R. Cole, and Michael T. Goodrich. Cascading divide and conquer: A Technique for Designing Parallel Algorithms. *SIAM Journal on Computing*, 18:499–532, 1989.

[BCS02] I. Boada, N. Coll, and J. A. Sellarès. Hierarchical Planar Voronoi Diagram Approximations. In *Proceedings of the 14th Canadian Conference on Computational Geometry*, pages 40–44, 2002.

[BMS94] Christoph Burnikel, Kurt Mehlhorn, and Stefan Schirra. How to Compute the Voronoi Diagram of Line Segments: Theoretical and Experimental Results. In *Proceedings of the Second Annual European Symposium on Algorithms*, pages 227–239, 1994.

[Deh89] Frank Dehne. Computing digitized Voronoi diagrams on a systolic screen and applications to clustering. In *Proceedings of the International Symposium on Optimal Algorithms*, pages 14–24, 1989.

[DZ99] Xiaotie Deng and Binhai Zhu. A Randomized Algorithm for the Voronoi Diagram of Lines Segments on coarse-grained Multiprocessors. *Algorithmica*, 24:270–286, 1999.

[GOY93] Michael T. Goodrich, Colm O'Dunlaing, and Chee K. Yap. Constructing the Voronoi Diagram of a Set of Line Segments in Parallel. *Algorithmica*, 9:128–141, 1993.

[Kar04] Menelaos I. Karavelas. A Robust and Efficient Implementation for the Segment Voronoi Diagram. In *Proc. Internat. Symp. on Voronoi diagrams in Science and Engineering (VD2004)*, 2004.

[KEHKL+99] III Kenneth E. Hoff, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. Fast computation of generalized Voronoi diagrams using graphics hardware. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 277–286, New York, 1999. ACM Press/Addison-Wesley Publishing Co.

[KKT01] Jörg Keller, Christoph W. Keßler, and Jesper Larsson Träff. *Practical PRAM Programming*. John Wiley and Sons, Inc., New York, 2001.

[MN99] Kurt Mehlhorn and Stefan Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[Pro04] The CGAL Project. CGAL User and Reference Manual Release 3.1, Chapter 43, December 2004. http://www.cgal.org/Manual.

[VO98] Jules Vleugels and Mark Overmars. Approximating Voronoi Diagrams of Convex Sites in Any Dimension. *Int. Journal on Computational Geometry and Applications*, 8(2):201–221, April 1998.