Construction of Voronoi diagrams using Fortune's method : A look on an Implementation

Čuk Roman
Faculty of Electrical Engineering and Computer Science
Laboratory for Computer Graphics and Artificial Intelligence, Center for Geometric Modeling
Smetanova 17, SI-2000 Maribor, SLOVENIA
e-mail: roman.cuk@uni-mb.si, http://www.uni-mb.si/~cgm

Abstract:

The paper deals with an implementation of Voronoi diagrams using Fortune's method. At first, Voronoi diagrams are considered briefly. The idea of Fortune's method is considered then. The implementation details and used data structure are the main focus of the paper.

Keywords: Computational geometry, Voronoi diagram, Fortune's method

1. Introduction

Voronoi diagram belong to classical problems of the computational geometry. Its origin dates back into 1850 when it was considered by Dirichlet. The rigid mathematical fundamental was given by Voronoi in 1908.

Let us have points p_i, $0 < i <= n$ in n-dimensional space P. The set of all points with property that every point inside the cell is closer to point p_i than to any other point from P represents the Voronoi cell (Figure 1). The union of all Voronoi cells is known as Voronoi diagram.

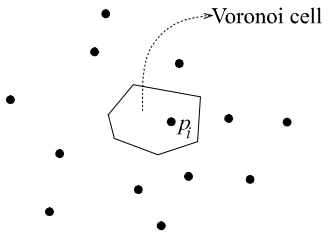

Figure 1: Voronoi cell

As seen from Figure 1, the construction of Voronoi diagram consists of repetitive subdivisions of the space determined by the input points into subspaces to meet the Voronoi criteria:

In many applications, Voronoi diagrams are already the final solution. For example, study of behavior and maintainance of live creature, which depends on number of neighbors with whom they are fighting for food and lightness is exactly what Voronoi diagram expresses. However, having a Voronoi diagram, many important geometric features like searching for the closest neighbor, Delaunay triangulation, searching for the largest empty circle, minimum spanning tree (Euclidean tree), can be determined in the linear time.

The first reliable algorithm for their construction has been published by Green and Sibson [GREE77], although the Voronoi diagrams have been known for a long time. This approach used incremental method and worked in $O(n^2)$ time. The first algorithm requiring $O(n \log_2 n)$ was suggested by Shamos and Hoey (1975) [PREP85] using divide and conquer approach. However, its implementation is complicated. In 1985, Fortune presented more elegant approach and it is described in [BERG97, O'ROU93]. In this paper, we consider primarily the data structures needed for the construction. In 1986, Edelsbruner and Seidel discovered beautiful connection between Voronoi diagram and convex hulls in one higher dimension [O'ROU93]. This method has some beautiful properties and it is becoming more and more popular.

2. Fortune's method

In 1985, Fortune invented an interesting algorithm for construction of Voronoi diagram. He used the sweep-line strategy implemented by many algorithms of computer graphics and computational geometry. The idea of the algorithm is very simple and its time complexity is $O(n \log_2 n)$ in the worst case, where n is the number of input points. In the continuation, an idea of the algorithm is shortly given; details can be find in [BERG97].

Fortune has expanded his observation in 3D. He put coins above all points from set *P*. All coins are slanted for an angle of 45°. These coins are scanned with a scanning plane π, which is also slanted for an angle of 45° to the xy plane (Figure 2).

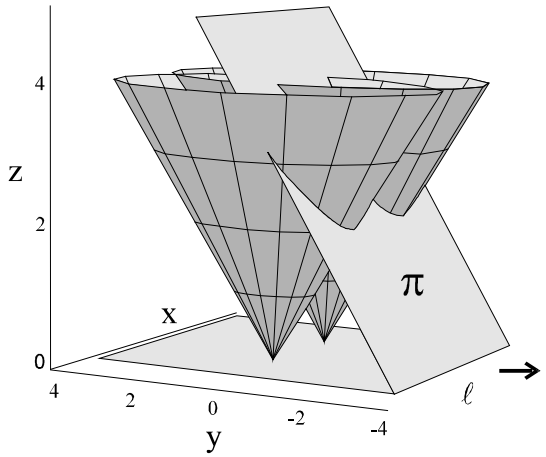

Figure 2: During the process of scanning, plane π intersects coins

In Fortune's method, we are interested in intersections of the scanning plane with coins. These intersections represent a curve of parabolic arcs or parabolic front. This curve has the property that

joints of parabolic arcs lie on intersection of two neighboring coins. The parallel projection of these two coins on xy-plane represents exactly bisector on which Voronoi edge lies. At first, the method seems very complicated, but Fortune has had the fortune on his side. If the coins and sweep plane are considered in xy-plane, we get the sweep line and parabolic front. In Figure 3 it is easy to see why the curve of the parabolic front is so important. Namely, Voronoi diagram is fully constructed above the parabolic front and below we do not know anything about it.

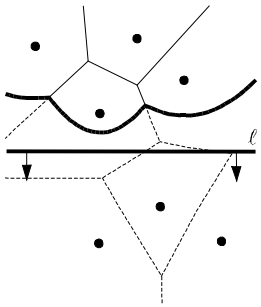

Figure 3: 2D representation of Fortune's method; Voronoi diagram is constructed only above parabolic front

From Figure 3, it can be seen that a topological structure of the parabolic front is changed by sliding the scanning line over xy-plane. Here the major problem in the construction of data structures is met. Namely: "When and how the topological structure of parabolic front is changing?" The answer is hidden in events. Two kinds of events appear. The first is a point event and the second a circle event.

2.1 Point event

The point event is handled when a new parabolic arc appears on parabolic front. This event happens exactly when a new arc appears on the parabolic front, or more precise, when the scanning line ℓ encounters a point p_i from set P. Then, the scanning plane π hits the coin defined above point p_i for the first time. Because the coin and the scanning plane are slanted for the same angle, the intersection between them appears as a line on the coin surface. This line is actually a half-edge, but if it is projected onto xy-plane we can consider it a degenerated parabola with zero width. This degenerated parabola starts to open as the scanning plane π slides over the space. When a new arc comes across, it splits the existed arc into two parts and becomes a new member of the parabolic front (Figure 4).

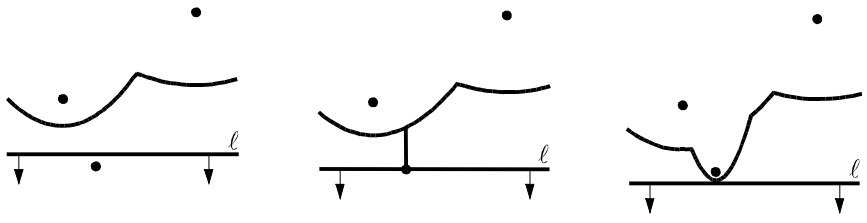

Figure 4: When point event is encountered a new arc appears on the parabolic front

What happens with Voronoi diagram when a point event is encountered? Lets remember, that a joint point of two arcs on parabolic front describes the Voronoi edge.

2.2 Circle event

Circle event happens when a parabolic arc shrinks to a point and disappears from the parabolic front (Figure 5). Let β_j be the disappearing arc and β_i and β_k are two neighboring arcs before β_j disappears. These arcs are then defined by three different points p_i, p_j and p_k. At the moment when β_j disappears, all three arcs pass through a common point q (Figure 5). The distance between q and the scanning line is the same as the distance from q to all three points p, p_j and p_k. These three points define a circle, with its center at point q which represents the Voronoi point. The lowest point of this circle touch the scanning line and represents the circle event.

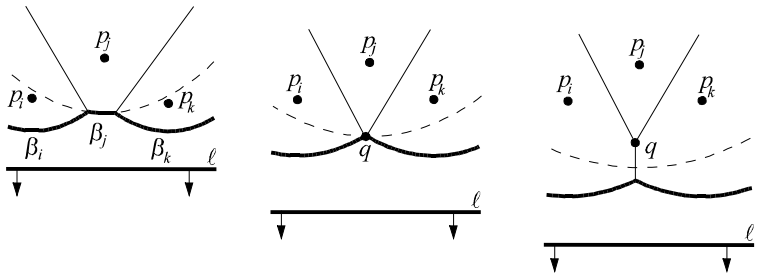

Figure 5: Parabolic arc disappears from the parabolic front

It is possible that the circle event is tried to be destroyed by some other event ("false event" or "false alarm"). Such events must be destroyed and removed from the data structure. As will be explained later, the detection of the false alarms is trivial.

2.3 Fortune's algorithm

Up to now, we have met all fundamental parts of Fortune's method and so we can think about an implementation. However, let us say just a word about termination condition. When all points are behind the scanning line and all events have been handled, Voronoi diagram is not fully constructed yet. We have to construct the remaining half-lines. For this purpose, the whole diagram is surrounded by a box (Figure 6) on which border all half-edges terminate.

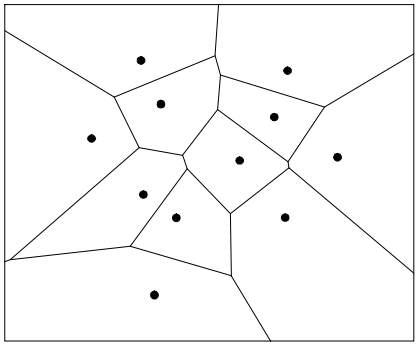

Figure 6: Completely defined computer represented Voronoi diagram does not contain any half-edges

The following algorithm summaries the consideration done up to now and shows the core of the Fortune's method.

```
VORONOI_DIAGRAM(P)
Input:  A set P of point sites in the plane
Output:  The Voronoi diagram Vor(P) given inside a bounding box in a doubly-connected edge list structure
    {
      Initialize the event tree Q with all point events
      while Q is not empty do
      {
        Consider the event with largest y-coordinate in Q
        if the event is a point event, occurring at site p_i
          then HANDLE_POINT_EVENT( p_i )
          else HANDLE_CIRCLE_EVENT( p_ℓ )
        Remove the event from Q
      }
      Put Voronoi diagram into a box
    }
```

3. Data structures used in Fortune's method

Fortune's method is the most popular method to construct the Voronoi diagrams today. However, the construction of the most suitable data structures and the functions operating on them are still the tack of a programmer. In general, three data structures are needed: one for keeping Voronoi diagram and the other two for implementation of the sweeping process (point and circle events and parabolic front).

3.1 Data structure of Voronoi diagram

Data structure for storing Voronoi diagram is combined from three data structures. Voronoi diagram consists of Voronoi cells (faces), Voronoi edges and Voronoi points [BERG97]. Each group of these elements is stored in its own data structure that is, of course, connected among them. The graphical representation of the Voronoi diagram is shown Figure 7.

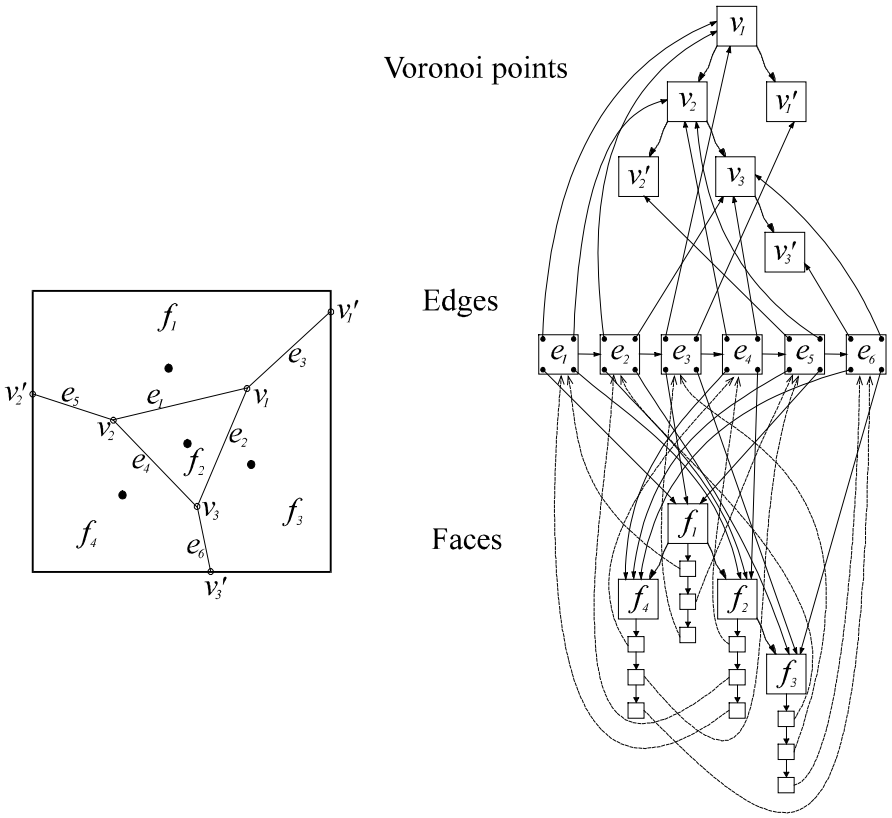

Figure 7: Data structure for keeping Voronoi diagram

Let us just consider the final tasks of constructing the Voronoi diagram after all input points have been processed. The intersection points of half-edges with the box surrounding the Voronoi diagram are calculated. These points are proclaimed as the Voronoi points. This is done by help of the leaves of parabolic front tree, which stay in the tree after all events have been processed. Those leaves, which stay in the tree, represent exactly those Voronoi edges, which are not defined completely after the construction is done.

3.2 The data structure for managing the parabolic front

The binary tree as a data structure for keeping the parabolic front is not chosen just because of quick updating, but also because the topological structure of the parabolic front can be represented the most naturally by the binary tree (Figure 8).

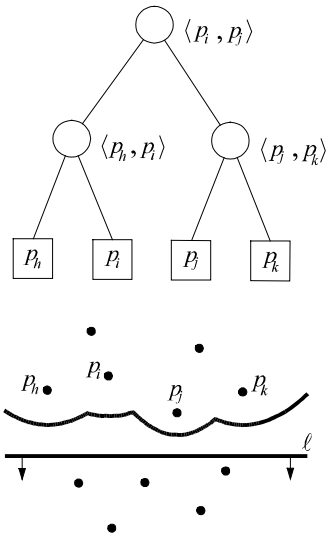

Figure 8: The representation of a topological structure of the parabolic front

The arcs on the parabolic front are represented by the tree leaves while joints between arcs are stored by the interior tree nodes. The most left arc on the parabolic front is represented by the most left leaf in the tree, the second left most arc is represented by the next left most leaf and so on.

Of course, the most interesting operation is inserting and deleting the parabolic arcs from the data structure. As we mentioned in section 2.1, when a point event happens, new arc appears also on the parabolic front by dividing the existed arc upon the event point. In the data structure representing the parabolic front, this is manifested as replacing the leaf with a sub-tree. As shown in Figure 10, the sub-tree consists of three nodes; the middle one represents the new arc and the other two the divided parts of the old arc.

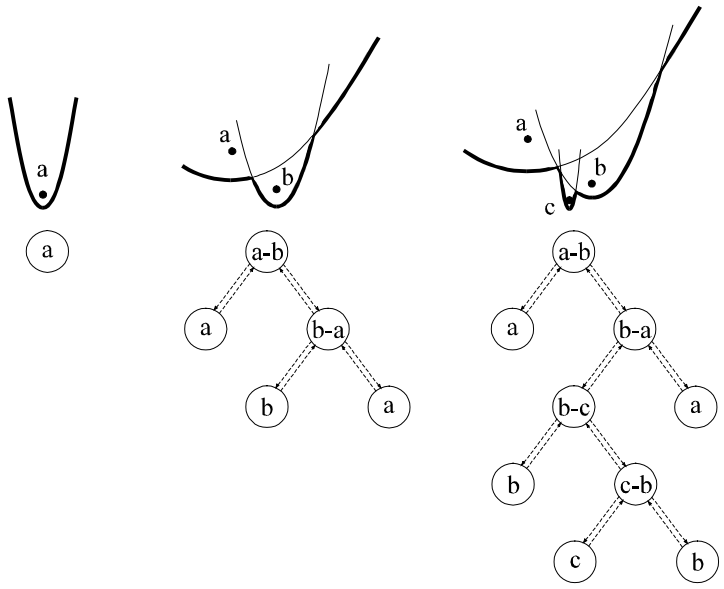

Figure 9: Inserting a new arc into the tree representing the parabolic front

As we said in section 2.2, an arc shrinks to a point and disappears from the parabolic front at the circle event. In the tree representing the parabolic front, this reflects as deleting a leaf from the tree. However, with the leaf we must delete also the upper internal node and the second internal node

becomes new joint point of two neighboring arcs. The circle event determines which arc has to be deleted. Both neighbors can be found as left and right leaf of disappearing leaf. The first node is trivial to find, because it is the father of the disappearing leaf. But finding the second node is not easy. We have to know on which side of the point, defining the disappearing arc the second node is. The second node is determined as an intersection node of two searching directions (see Figure 10). The first starts from the disappearing arc and the second from the left (if we divide arc on the left side) or the right neighbor (if the arc on the right side is being divided). The intersection node represents joint of two neighboring arcs.

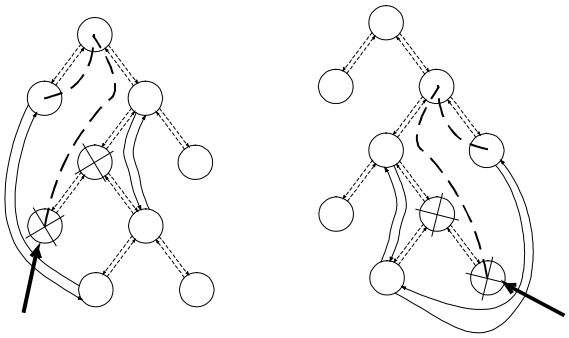

Figure 10: Two possibilities of deleting an arc from the tree of the parabolic front

To complete the explanation of this structure, a few words is needed to explain the attributes stored in the nodes and the leaves. We are interested especially in pointers between the tree of the parabolic front and other structures. We are also interested in how the parabola is stored in the tree. Let us remember that there is no need to store a real parabola; it is enough to store just a point, which defines the parabola. This can be seen from the following equation [BERG97]:

$$\beta_j := y = \frac{1}{2\left(p_{j,y} - \ell_y\right)}\left(x^2 - 2p_{j,x}x + p_{j,x}^2 + p_{j,y}^2 - \ell_y^2\right)$$

where ℓ_y represents the scanning line at the moment of observation and $p_j := \left(p_{j,x}, p_{j,y}\right)$ represents the point from *P*, which defines the arc β_j.

Internal nodes of the tree the pointers pointing to the edge in the data structure of Voronoi diagram are stored. In this way, each edge belonging to the Voronoi point is directly accessible, when the circle event is serviced and so there is no need to perform any search operations in the data structure of Voronoi diagram. At the leaves of the tree a pointer to the circle event is stored, if the arc defines a circle event. If not, pointer is set to NULL. By maintaining this pointer, we do not have to perform any search after encountering false events.

3.3 Data structure for keeping events

The event queue Q is arranged regarding the events y-coordinates. It stores the upcoming events that are already known. For the point event, the point is simply stored. For the circle event, the lowest point of the circle is stored. In the last case, the pointer to the leaf in the tree of parabolic front that represents the arc disappearing in the event is stored, too.

4. Conclusions and future work

It is known from literature, the Fortune's algorithm works in $O(n \log_2 n)$ time and that it is one of the simplest algorithms for constructing the Voronoi diagrams. Table 1 shows the obtained measurements of spent CPU time for different number of points. It can be concluded that the theoretical time complexity estimation is achieved also in practice. The algorithm is implemented in C++ and the measurements have been done on PC Pentium 133 MHz and 64 Mbytes RAM. The results of construction can be seen in Figure 11.

Table 1. Measurements of spent CPU time

No. of points	2000	4000	6000	8000	10000
Spent CPU time [s]	3.86	7.61	11.73	16.84	22.31

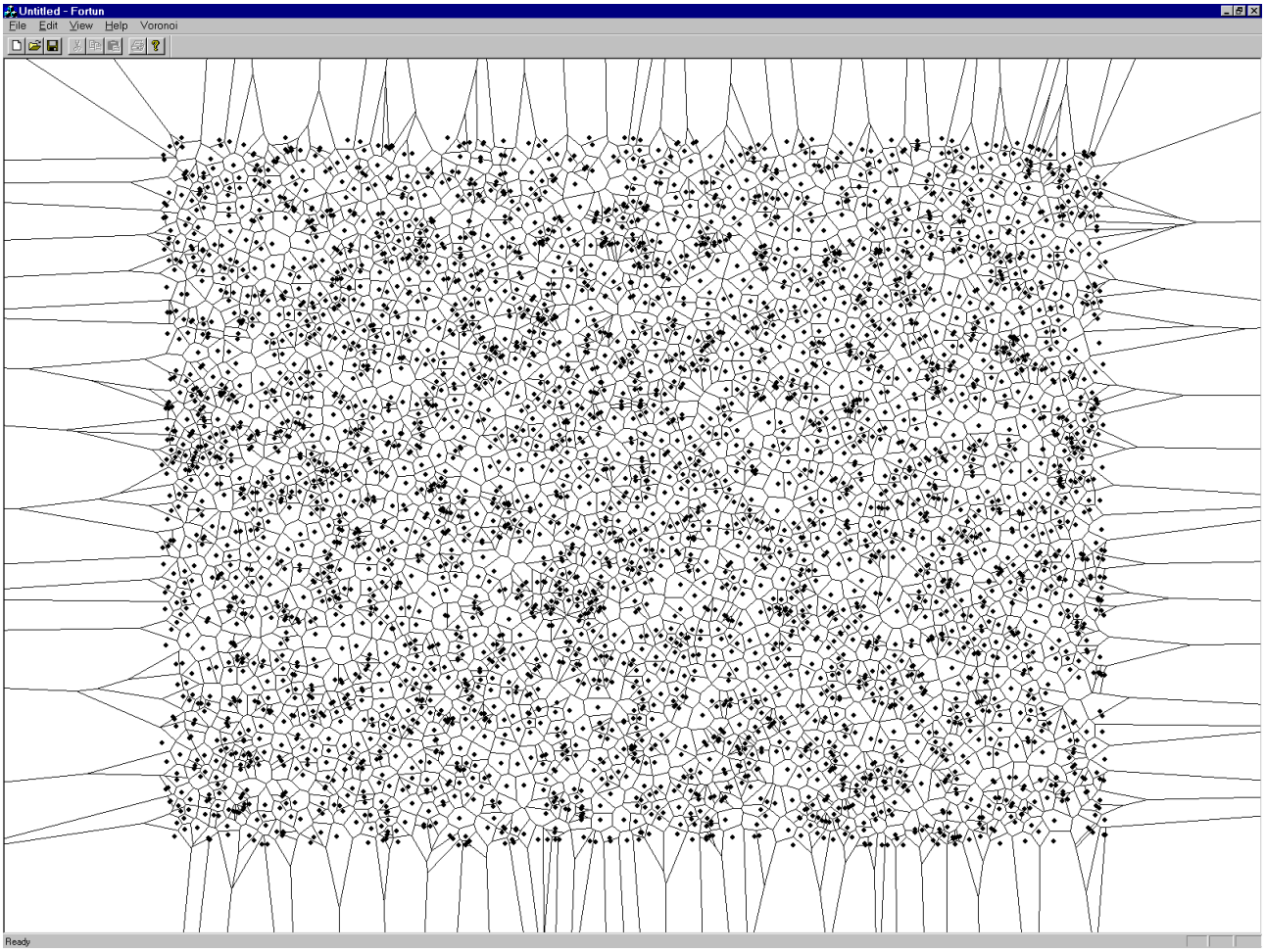

Figure 11: Voronoi diagram constructed on 4000 points

5. References

[GREE77] Green, P., and R. Sibson, "Computing Dirichlet tessellations in the plane", Computational Journal, Vol. 21, 1977, pp. 168-173.

[BERG97] Berg de, M., M. van Kreveld, M. Vermars, O. Schwarzhopf, "Computational Geometry: Algorithms and Applications", Springer-Verlag, Berlin, 1997.

[O'ROU93] O'Rourke, J., "Computational Geometry in C", Cambridge University Press, Cambridge, 1993.

[PREP85] Preparata, F. P., and Shamos, M.I., "Computational Geometry: an Introduction", Springer-Verlang, 1985.