Fortune's Voronoi Diagram algorithm and an application in MRI reconstruction.

Themistoklis Haris

Abstract

In this report, the fundamentals of Voronoi diagrams are presented and an elegant and efficient sweep-line algorithm for their construction is described and analyzed (Fortune's algorithm). Specifically, the usage of the required data structures (priority queues and balanced binary search trees) is justified, detailed pseudo-code is given and the most important scenarios of the algorithm are graphically illustrated. Then, an application of Voronoi diagrams to the problem of Magnetic Resonance Image (MRI) reconstruction is sketched. Preliminary results on synthetic images are shown.

1 Introduction

The way space is partitioned around a fixed set of points has always been a field of scientific interest. Nature itself arranges molecular structures in a way that the least possible amount of energy is required to achieve stability. When it comes to biology, the main and most fundamental autonomous entity that can sustain life, the cell, organizes itself so that the operations and chemical reactions it needs to survive happen fast and efficiently. In addition, space partitioning around points of interest has many more applications in science. It can be used in Astronomy, Geology, Forestry, Robotics, Manufacture and Computer Graphics to mention a few. In this report, we present and analyzed a very special partitioning of space, the so called *Voronoi diagram*. Although there is a huge number of algorithms for the construction of Voronoi Diagrams, the algorithm we present here is a sweep line algorithm proposed by Fortune in 1986 [4] and still remains one of the fastest and simplest ones.

The rest of the report is organized as follows. In Section 2, we introduce and analyze the topic geometrically, from a mathematician's point of view, and in Section 3, we present Fortune's algorithm that successfully produces the partitioning of a plane (the Voronoi Diagram), based on specific input points. In Section 4, we present an application of Voronoi diagrams to a problem of medical imaging, namely, the problem of Magnetic Resonance Image (MRI) reconstruction. The problem is briefly introduced and a simple use of Voronoi diagrams to achieve

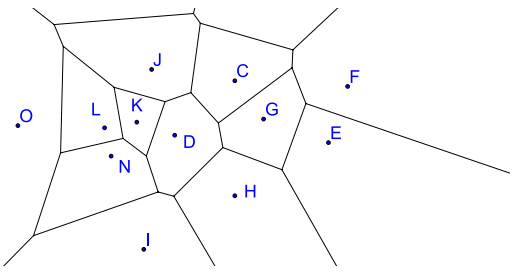


Figure 1: A Voronoi Diagram

generic sampling independent reconstructions is described. Preliminary results on 3D synthetic images are shown. Finally, in Section 5, we conclude our findings and take a broader look at the subject of Voronoi Diagrams and MRI reconstruction.

2 Voronoi diagrams: basic properties

A formal definition Let P be a set of $n \geq 2$ points in a plane. The points in P are referred to as **sites**. The *Voronoi diagram* of P is a partitioning of the plane into n cells, one for each point. The cell corresponding to point $p_i \in P$ is denoted as $V(p_i)$. The fundamental property of this partitioning is that every point $q \in V(p_i)$ is closer to p_i than it is to any other site.

More formally, we define a relation $V : \mathbb{R}^2 \rightarrow P$ so that $dist(x, s) \leq min_{z \in P}\,(dist(x, z)) \Leftrightarrow x \in V(s)$, $\forall x \in \mathbb{R}^2$. We can easily infer that some points belong to two or more Voronoi cells, being equidistant from two or more sites. The collection of these points is what we call the Voronoi Diagram. An example is shown in Fig. 1.

In the following, an analysis of the geometrical structure of a Voronoi diagram is presented, and some basic observations are reported.

Observation 2.1 *Every Voronoi cell is a closed, convex and possibly unbounded polygonal chain, whose sides are parts of the perpendicular bisectors of the segments connecting the sites (see Fig. 2). Thus, each Voronoi cell has at most $n-1$ vertices and edges.*

Observation 2.2 *The Voronoi diagram is a connected planar embedding. The sides of each Voronoi cell are either half-lines or line segments. They cannot be full lines, except the case where all points are collinear. In that degenerate case, the diagram consists of $n-1$ parallel lines (See Fig. 3).*

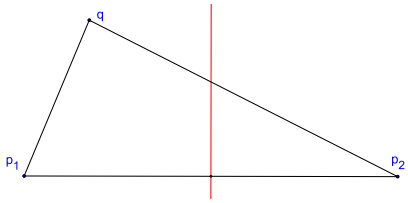


Figure 2: The perpendicular bisector of the segment p_1p_2 divides the plane in two. q is closer to p_1 than to p_2.

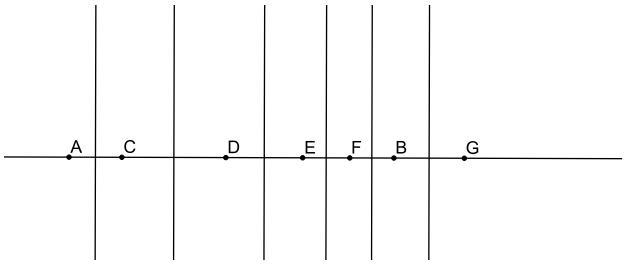


Figure 3: Degenerate case of Voronoi diagram with all points collinear

Theorem 2.1 *As a planar embedding, the complexity of the Voronoi diagram is **linear**. More specifically, for $n \geq 3$, a Voronoi diagram can have at most $2n-5$ nodes and $3n-6$ edges. Therefore, on average, each Voronoi cell has less that six sides.*

Proof: The statement follows immediately from *Euler's formula* for connected embedded planar graphs, which states for the number of nodes m_n, the number of faces m_f and the number of sides m_e that: $m_n - m_e + m_f = 2$ (See Fig. 4). Of course, our graph has some infinite edges, so the formula cannot be applied the way it is. We have to create a dummy vertex v_∞, to which every infinite edge will be incident to. Then, we get that $(m_n+1)-(m_e)+n = 2$. At the same time, we also know that every edge is incident to exactly two vertices and every vertex has a degree of at least 3, so the relation $2m_e \geq 3(m_n + 1)$ holds. From that, we arrive at the desired conclusion $\square$

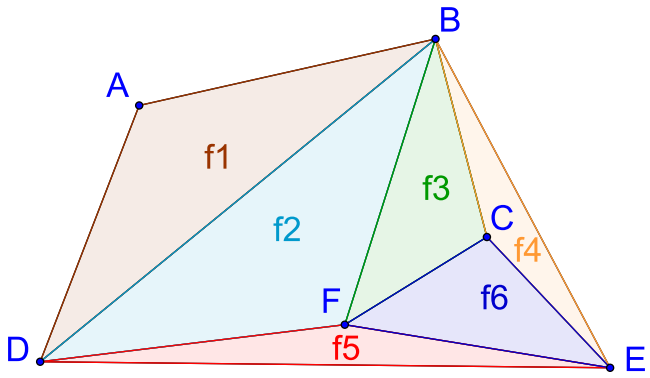


Figure 4: In this planar graph, $m_n = 6$, $m_e = 11$ and $m_f = 7$, so Euler's formula holds

Observation 2.3 *Since we have a quadratic amount of perpendicular bisectors and only linear complexity, not all bisectors contribute to the sides of the Voronoi diagram. For a bisector to be part of a side between cells $V(p_i)$ and $V(p_j)$, it must contain at least one point whose largest empty circle in respect to set P passes through only p_i and p_j (See Fig. 5) Also, a point q is a vertex of the Voronoi diagram if and only if its largest empty circle in respect to P contains three or more sites on its boundary.*

3 Constructing the Voronoi diagram

Many algorithms have been proposed to compute the Voronoi diagram of a set of points P. First of all, one could simply examine all possible perpendicular bisectors defined by the points in P in $O(n^2)$. That is a complete search algorithm, which runs in quadratic time and is sufficient if execution time limits are not very demanding. However, constructing the Voronoi diagram can be done faster in $O(n \log_2 n)$. The first algorithm to achieve this complexity was proposed by Shamos and Hoey in 1975 [6] and uses the divide and conquer paradigm. However, this idea uses very complex data structures, contains many non-automatic steps and is very difficult to implement. A simpler algorithm for Voronoi diagrams was proposed by Steven Fortune in 1986 [4, 1]. This algorithm is analyzed below.

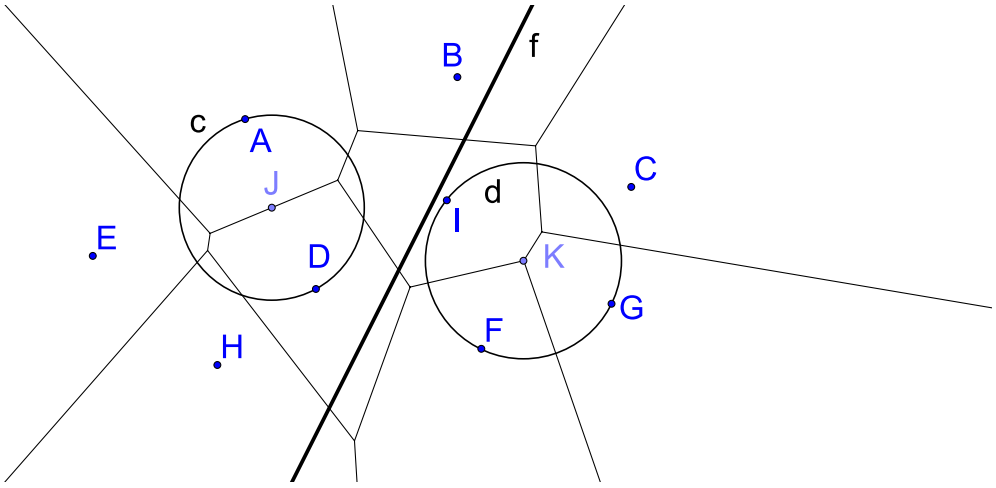


Figure 5: J is a point in a side, so circle c only passes through sites A and D. Point K is a vertex, so circle d passes through sites I,F and G. Even though line f is a perpendicular bisector of segment AG, it is not part of the Voronoi diagram.

3.1 Prerequisites

In order to understand Fortune's algorithm, the reader must be familiar with the following mathematical and computational ideas and concepts:

Parabolic curves

A **parabola** is the locus of the points in the plane that are equidistant from a point F(called the focus) and a line l(called the directrix) (Fig. 6). If $p = dist(F, l)$, (a, b) is the vertex of the parabola and the equation of the directrix is $y = k \in \mathbb{R}$, then the equation of the parabola in the plane is: $y = \frac{1}{2p}(x - a)^2 + b$.

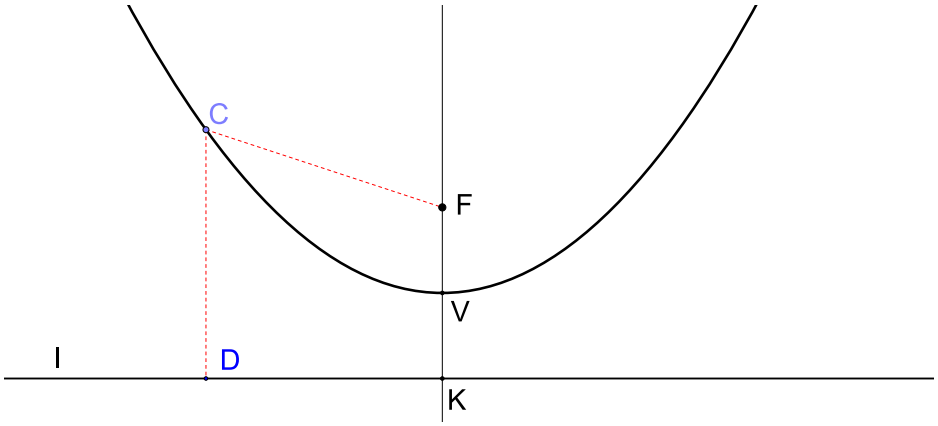


Figure 6: $|FK| = p, V = (a, b), l : y = k \in \mathbb{R} \Leftrightarrow c : y = \frac{11}{2p}(x - a)^2 + b$

Binary Search Trees and Red Black Trees

Binary search trees are binary trees that satisfy the **binary-search-tree property**: If x is a node in the tree and y an node in the left subtree of x, then $y.key \leq x.key$. If y is a node in the right subtree of x, the $y.key \geq x.key$. The order relations $\leq$ or $\geq$ are of our choosing. This simple property allows us to search for, insert or delete elements in the tree in $O(h)$ time, where h is the height of the tree, using very simple algorithms, like inorder tree walks. For the implementation, we use pointers from parent nodes to child nodes. If a child node is missing the pointer is *NIL*.

Sometimes however, binary search trees receive values in such a way that the height becomes very large and the complexity of insertion or deletion queries becomes linear and the binary search tree becomes practically a linked list data structure [2]. To avoid that and to keep the complexity of quering binary search trees logarithmic, we introduce a notion called "balancing", i.e. restructuring the tree to facilitate some operations. There are many versions of balanced BSTs, but we will be using Red-Black Trees in our implementation of Fortune's algorithm.

Red-Black Trees give a "color" attribute to each node and use that to perform balancing operations called "rotations" [2]. We will not analyze in depth these rotations, but we will be using them to maintain balance in our trees.

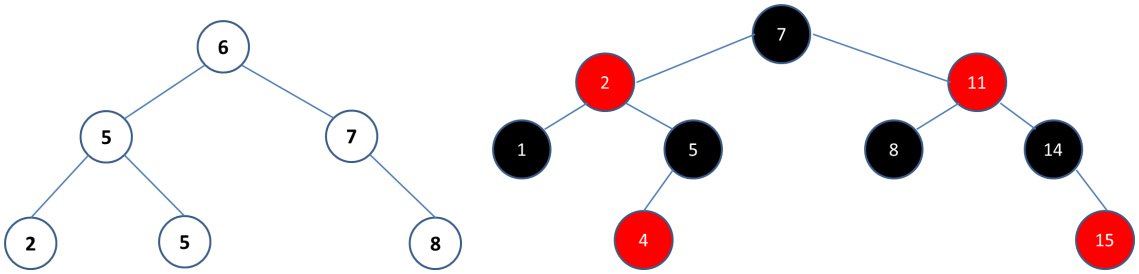
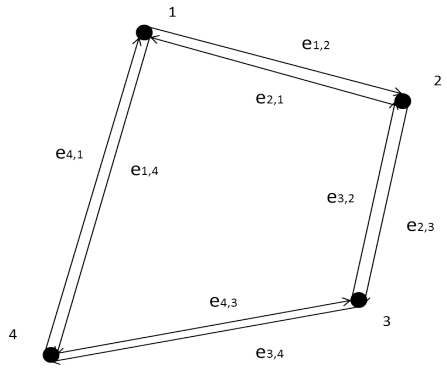


Figure 7: A Binary Search Tree and a Red-Black Tree. Notice the BST property which holds in both trees

Doubly-Connected Edge Lists

The Doubly-Connected Edge List (DCEL) is a data structure that efficiently stores information about a planar subdivision induced by a planar embedding of a graph. In the general case, a DCEL is a complex data structure that stores a lot of information. In our case however, since the Voronoi planar graph is connected, the DCEL will consist only of two records, which will be implemented as arrays that store pointers. The first record will contain information about the vertices of the Voronoi diagram and the second one will store information about the edges of the Voronoi diagram in the form of a doubly-linked list. Please refer to 8 for more clarifications.



Vertex	Coordinates	Incident Edge
v_1	$(1, 5)$	$\vec{e}_{1,2}$
v_2	$(4, 4)$	$\vec{e}_{3,2}$
v_3	$(3, 1)$	$\vec{e}_{3,4}$
v_4	$(0, 0)$	$\vec{e}_{1,4}$

Half-Edge	Origin	Twin	Next	Prev
$\vec{e}_{1,2}$	v1	$\vec{e}_{2,1}$	$\vec{e}_{2,3}$	$\vec{e}_{4,1}$
$\vec{e}_{2,3}$	v2	$\vec{e}_{3,2}$	$\vec{e}_{3,4}$	$\vec{e}_{1,2}$
$\vec{e}_{3,4}$	v3	$\vec{e}_{4,3}$	$\vec{e}_{4,1}$	$\vec{e}_{2,3}$
$\vec{e}_{4,1}$	v4	$\vec{e}_{1,4}$	$\vec{e}_{1,2}$	$\vec{e}_{3,4}$
$\vec{e}_{2,1}$	v2	$\vec{e}_{1,2}$	$\vec{e}_{1,4}$	$\vec{e}_{3,2}$
$\vec{e}_{3,2}$	v3	$\vec{e}_{2,3}$	$\vec{e}_{2,1}$	$\vec{e}_{4,3}$
$\vec{e}_{4,3}$	v4	$\vec{e}_{3,4}$	$\vec{e}_{3,2}$	$\vec{e}_{1,4}$
$\vec{e}_{1,4}$	v1	$\vec{e}_{4,1}$	$\vec{e}_{4,3}$	$\vec{e}_{1,2}$

Figure 8: The DCEL we will be using for Fortune's algorithm.

3.2 An overview of Fortune's algorithm

Fortune's algorithm for computing the Voronoi diagram of a set of points P is a **sweep line algorithm**, i.e. it uses a horizontal line to scan the points in P, gradually collecting information and building the diagram. That line, which we denote as l, scans the points from top to bottom and as it moves downwards, the algorithm assumes that a fixed set of points in the field above the line have already been assigned to their respective Voronoi cells. That is the **loop invariant** of Fortune's algorithm. More specifically, the points that have already been assigned to a Voronoi cell when l reaches some position are all points that are certainly closer to the sites above the line than to the sites below it. Every site p_i along with line l defines a unique parabola c_i, with p_i being the focus and l being the directrix. All points inside that parabola are closer to p_i than they are to line l. When viewed together, all such parabolas form a unique parabolic and x-monotone borderline sequence, which we call a **"beachline"**, with the property that all points above the beachline are closer to the sites above l than they are to the sites below l. The movement and structure of that beachline is determined by the position of the sites and the movement of line l.

Lemma 3.1 *Let the points in the beachline where two different parabolic arcs intersect be called **"breakpoints"**. A breakpoint belongs to an edge in the Voronoi diagram. As the beachline moves downwards, its breakpoints trace the Voronoi diagram.*

Lemma 3.2 *The structure of the beachline changes as line l moves downwards and discovers new points. There are two possible changes, called **"events"**, that can happen to the structure of the beachline: a new parabolic arc may appear or an arc may disappear.*

A) When a parabolic arc appears in the beachline, there is a *site event*. This happens only when l encounters a new point. As a consequence, the beachline can have at most $2n-1$

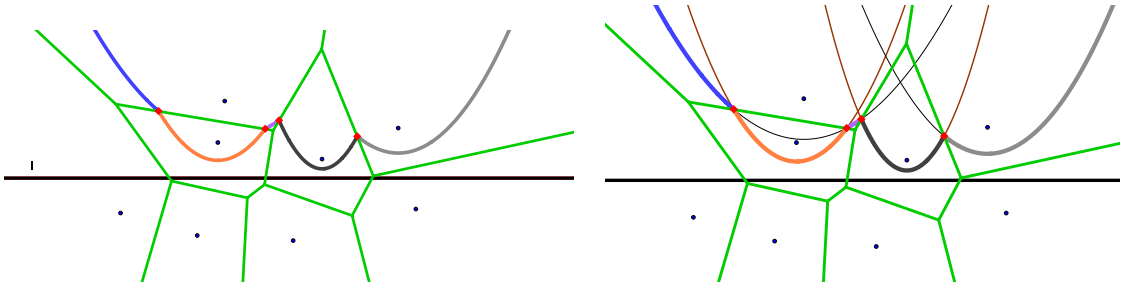
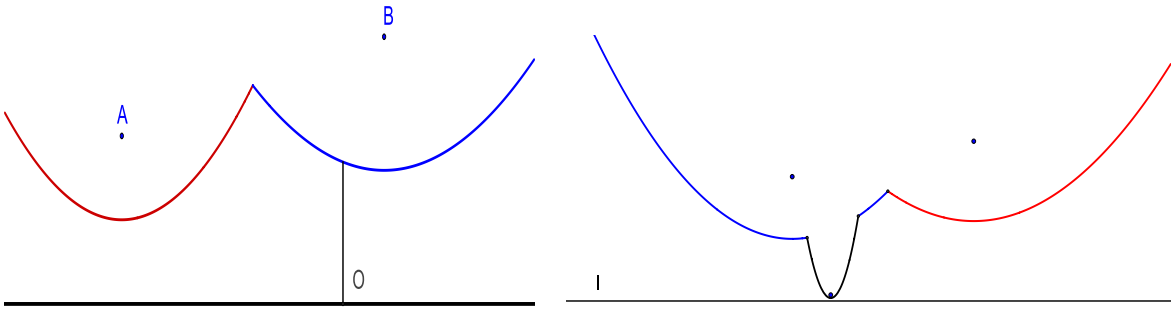
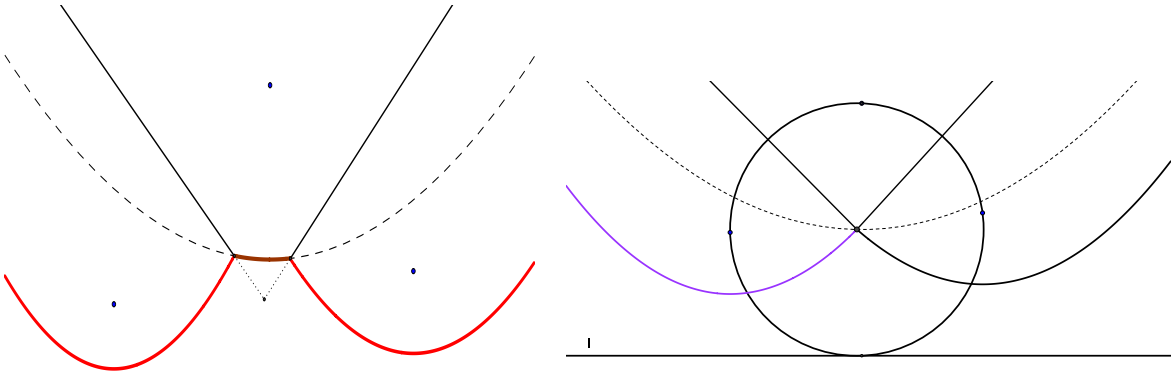


Figure 9: Beachline and Breakpoints

parabolic arcs. In a site event, the outline of the edges is differentiated and drawn.



B) When a parabolic arc or a part of an arc disappears from the beachline, there is a *circle event*. This can only happen when the sweep line reaches the lowest point of a circle through three sites defining consecutive arcs. During circle events we spot all the vertices of the Voronoi diagram.



3.3 An implementation in detail

Fortune's algorithm as described above basically traverses through all possible events in an order from top to bottom (or left to right) and based on the information it collects, it constructs the beachline and subsequently the Voronoi diagram.

The algorithm maintains three data structures as it progresses through all events:

The Voronoi diagram itself is represented by a DCEL D, which stores records of the edges and the vertices (See 8).

The beachline is stored implicitly in a Red-Black Tree T. At all times, the points stored in T are either the sites of P that act as foci to the parabolic arcs of the beachline or the breakpoints between parabolic arcs. So T contains at most $2n - 1$ nodes.

The leaves of the tree represent the sites that act as foci of the current parabolas. The internal nodes represent the breakpoints of the beachline. The points in T are ordered from left to right. When a site event appears, we search T for the part of the parabola that needs to be replaced by a new arc. This happens in logarithmic time. When a circle event appears, we delete a specific parabolic arc from our beachline. That also happens in logarithmic time. Of course, these operations will be accompanied by the necessary modifications to the structure of the Tree and will also induce progress in the construction of the Voronoi diagram in D. For example, at a site event, we will add edges to D and at a circle event we will form vertices in D.

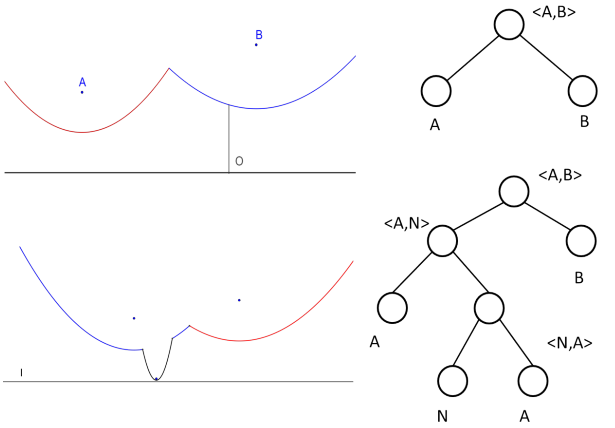


Figure 10: Red-Black Tree operations during Site Event handling

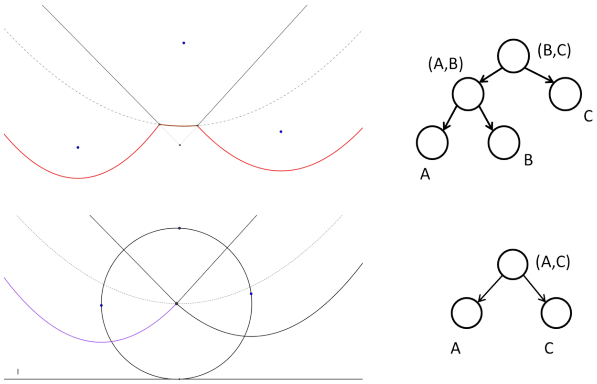


Figure 11: Red-Black Tree operations during Circle Event handling

Lastly, the algorithm examines events in an order from top to bottom and from left to right. To achieve this, we must maintain a *priority queue* Q, which will contain all the events whose probable appearance at some point can be predicted at a current position of the sweep line. Of course, not all such events will actually take place. Many circle events which we will predict from each site event or circle event will not happen, because there can be consecutive triples of foci whose breakpoints do not converge. We will refer to this phenomenon as a **false alarm**.

In the pseudocode implementation of Fortune's algorithm that follows, we shall assume that the following generic classes have been constructed:

point the coordinates of a point $p : (x, y)$. This class will also be used to represent

the vertices of the Voronoi diagram.

edge: The fields are designed to fit in a DCEL and are all public: $e.start, e.end, e.next, e.prev$ and some other attributes needed to make rough algebraic calculations.

line: a startard *"straight line"* class, defining all lines as equations of the form: $ax + by = c, |a| + |b| \neq 0$.

event: an event is defined by a point($ev.point$), a parabolic arc($ev.parabola$) and a boolean value $ev.se$, which identifies if an event is a site event(se) or not).

parabola: This is the most important class in the algorithm. An object instantiating the "parabola" class may not necessarily represent a parabolic conic curve. These objects are used as nodes in the Red-Black Tree T, so their functions are numerous. If a "parabola" object is a leaf of T, then it represents an arc. Otherwise it stands for a breakpoint. Every "parabola" object p has the following public fields: $p.site$ (the focus, if it is a leaf), $p.edge$ (the breakpoint, if it is an internal node, $p.ce$ (the circle event corresponding to the parabola), $p.parent$ (p's parent in T) and $p.isLeaf$ (a boolean value which is self-explanatory).

Algorithm 3.3.1 Main structural compartment of Fortune's $O(n \log_2 n)$ Voronoi diagram computation algorithm.

1: **procedure** VoronoiDiagram
2: `INPUT :` a set of points $P := \{p_1, p_2, p_3, ..., p_n\}$
3: `OUTPUT :` a doubly-connected edge list D, representing the Voronoi diagram of P (8)
4: (* Initialization *)
5: Initialize an empty *Red-Black Tree* **T**, an empty *DCEL* **D** and a *priority queue* (heap data structure) **Q** with all site events.
6: (* End of Initialization *)
7: **while** $Q \neq \emptyset$ **do**
8: $e \leftarrow Q.top()$
9: $Q.pop()$
10: **if** e.se == true **then**
11: $InsertParabola(e.point)$
12: **else**
13: $DeleteParabola(e.parabola)$
14: **end if**
15: **end while**
16: Compute the bounding box containing all vertices of the Voronoi diagram and attach the half infinite edges that are left in T to it.
17: Update and output D.
18: **end** VoronoiDiagram

The methods to handle site and circle events are implemented as follows:

Procedure 3.3.2
InsertParabola procedure, for handling site events.

1: **procedure** InsertParabola
2: `INPUT :` a point u that is the focus of the new parabolic arc being inserted.
3: *parabola par* = the parabola vertically above u. We search for *par* in T.
4: **if** *par.ce* is in Q **then**

5: *par.ce* is a false alarm. Delete it from Q.
6: **end if**
7: Create 3 new *parabola* objects: a, b, c.
8: $b.site \leftarrow u$
9: $a.site \leftarrow c.site \leftarrow par.site$
10: $a.edge \leftarrow$ bisector of $|a.site \leftrightarrow b.site|$ and $b.edge \leftarrow$ bisector of $|b.site \leftrightarrow c.site|$.
11: Insert *par* in T (10), while performing the necessery rebalancing operations.
12: FindCircleEvent(a)
13: FindCircleEvent(c)

Procedure 3.3.3
DeleteParabola procedure, for handling circle events.

1: **procedure** DeleteParabola
2: `INPUT :` a circle event e.
3: *parabola* $left, right$ = the parabolas left and right of the arc being deleted.
4: If $left$ or $right$ are predicted to have circle events, then it is a false alarm. Delete these events from Q.
5: $s \leftarrow$ the center of the circle through points $left.site, p.site, right.site$.
6: x: new edge from s through the bisector of $|left.site \leftrightarrow right.site|$.
7: Delete p from T (11), while performing the necessary re-balancing operations.
8: FindCircleEvent(left)
9: FindCircleEvent(right)

Procedure 3.3.4
FindCircleEvent procedure

1: **procedure** FindCircleEvent
2: `INPUT :` a parabola p, whose circle event we will find and add to Q.
3: *parabola* $left, right$ = the parabolas left and right of the arc we predict is going to disappear.
4: **if** either one of $left$ or $right$ doesn't exist `OR` $left.site == right.site$ **then**
5: `RETURN` //that means p is at the edge of the beachline
6: **end if**
7: xl, xr: bisectors of $|left.site \leftrightarrow p.site|$ and $|p.site \leftrightarrow right.site|$.
8: $s := xl \cap xr$
9: **if** s doesn't exist **then**
10: `RETURN` //edges xl and xr diverge, so we have no circle event
11: **end if**
12: $r :=$ radius of circle with center $p.site$ that passes through s.
13: **if** $s.y - r$ strictly above sweep line l **then**
14: `RETURN` //we have already predicted this circle event.
15: **end if**
16: *event* $e :=$ a new circle event has been detected.
17: $e.ce \leftarrow$ `true`
18: $e.parabola \leftarrow p$
19: $e.point.y = s.y - r$
20: $Q.push(e)$

Theorem 3.1 *Fortune's algorithm for computing the Voronoi Diagram runs in $O(n \log_2 n)$ time and uses $O(n)$ storage.*

Proof: The result is derived directly from Theorem 2.1, which implies that there are $O(n)$ total events to process. Processing each event takes $O(\log_2 n)$ time, because of the efficient Red-Black Balanced BSTs we use in the implementation. Therefore, the total running time is $O(n \log_2 n)$. □

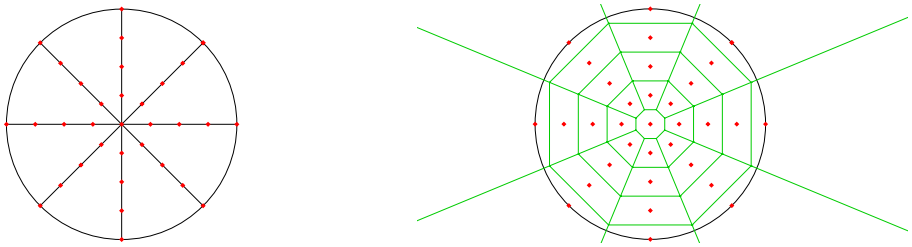


Figure 12: *Left*: Two dimensional radial sampling (four profiles). Each profile contains 9 samples. *Right*: Corresponding Voronoi diagram of the radial sampling points.

4 Voronoi in MRI

In this section, we present an application of Voronoi diagrams to the MRI reconstruction problem. More specifically, we portray a way to use Voronoi diagrams in 2 or 3 dimensions as a tool to estimate the parameters required by non-uniform FFT (NUFFT) algorithms for the solution of the MRI reconstruction problem.

In MRI, high quality images of a selected body part, like the heart, the brain or the liver are produced via the exploitation of the *nuclear magnetic resonance* phenomenon. Strong magnetic fields (1.5 T - 11 T) are used to excite hydrogen nuclei (protons) present in almost every cavity of the human body. When the magnetic fields return to normal, the excited protons emit radio waves which are captured by special receiving coils and turned into signals. The acquired signals are represented in the frequency domain (Fourier), which is referred to as ***k*-space**, in the form of points in the plane or in space. In what is called *"radial sampling"*, the collected points reside along radial profiles covering a circle (see Fig. 12(*Left*)) or a sphere.

In order to construct the final MRI image from 2-D or 3-D k-space data, we employ specific image reconstruction algorithms, that convert the acquired samples to $2D$ or $3D$ images, respectively [3]. A crucial step in this process is the sampling density compensation [5]: each sample point is assigned a specific value (**weight**) which depends on the area (or volume in 3D) of the voronoi cell of the point (see Fig. 12(*Right*)). Since the points are non-uniformly sampled, the derivation of closed form mathematical formulas for the computation of the weight of samples is not always possible. This is why we have to use the Voronoi diagram, either in $2D$ or $3D$.

```
function [w]=density_weights2(X,N,M)
    numPoints = size(X,1);
    [V,C]=voronoin(X);
    vcellVolumes = zeros(numPoints,1);
    for vcell = 1 : numPoints
        if all(C{vcell} ~= 1)
            vpoly = V(C{vcell},:);
            [~,vcellVolumes(vcell)] = convhulln(vpoly);
        end
    end
    w=reshape ( vcellVolumes, N,M );
    w=w./max(w(:));
```

```
end
```

5 Conclusions

To conclude this report, we have to acknowledge the immense research endeavors and scientific breakthroughs that Voronoi Diagrams have inspired in the latest years. Nevertheless, the progress we mentioned earlier is even more pronounced in the field of Magnetic Resonance, where efforts to understand the human body and pinpoint traits of its functionality endlessly fuel research projects. And with MRI images becoming clearer, better and more versatile, we can preform diagnoses of various diseases with much more accuracy. Of course, this is all in an experimental level as of today, but we hope that technology advances quickly, so that the people in the future can enjoy a more advanced healthcare and live more comfortably. All in all, Voronoi Diagrams have many more properties to explore and their applications seem to be more than we can currently imagine.

6 Acknowledgements

The author would like to thank Dr. K. Haris for the discussions about the application of Voronoi diagrams to MRI reconstruction, the technical writing assistance and the guidance about the usage of LaTeX and MATLAB.

References

[1] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[3] Jeffrey A Fessler and Bradley P Sutton. Nonuniform fast fourier transforms using min-max interpolation. *IEEE Transactions on Signal Processing*, 51(2):560–574, 2003.

[4] S Fortune. A sweepline algorithm for voronoi diagrams, in proceedings of the second annual symposium on computational geometry, pp. 313-322, 1986.

[5] Volker Rasche, Roland Proksa, R Sinkus, Peter Bornert, and Holger Eggers. Resampling of data between arbitrary grids using convolution interpolation. *IEEE transactions on medical imaging*, 18(5):385–392, 1999.

[6] Michael Ian Shamos and Dan Hoey. Closest-point problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pages 151–162. IEEE, 1975.

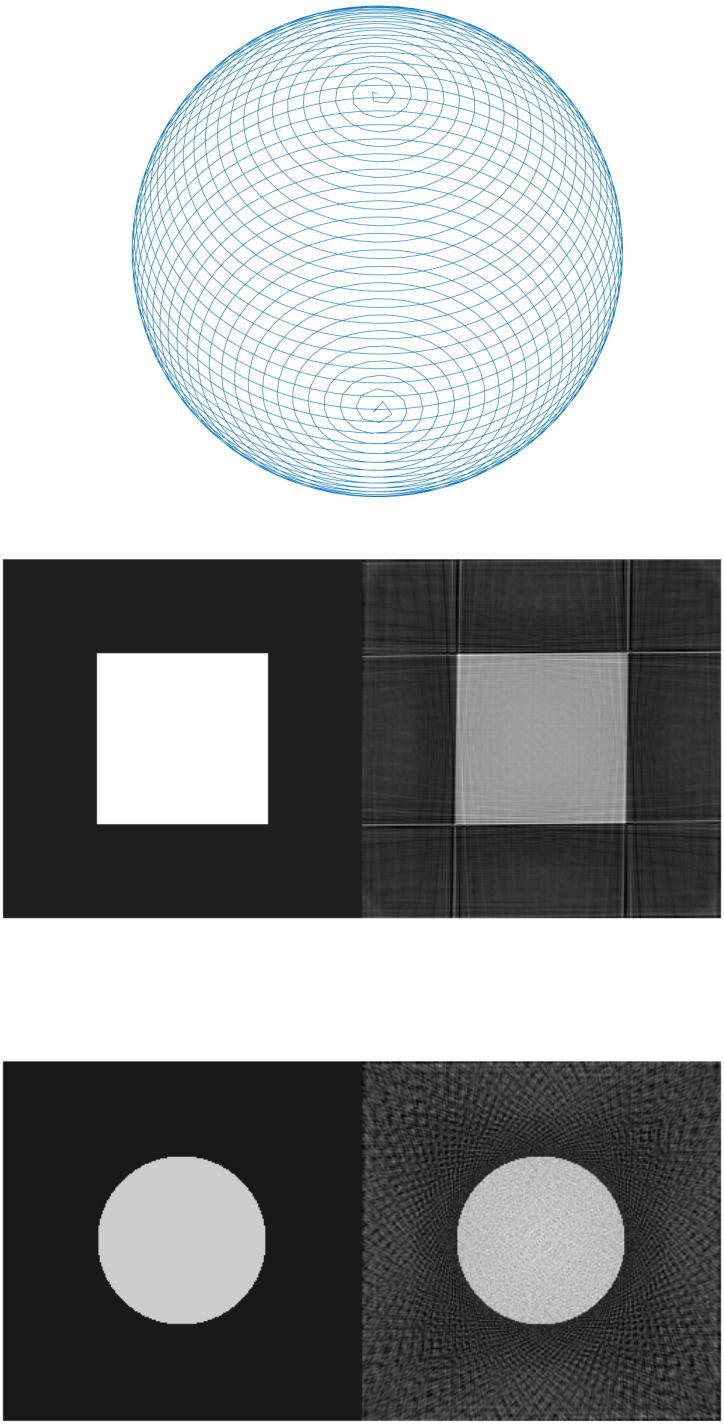


Figure 13: Synthetic image reconstruction preliminary results using the Voronoi diagram for sampling density compensation. (*Top:*) The trace on the sphere of the 3D radial profile end points. (*Middle and Bottom:*) A middle slice of the original $3D$ image (a cube and a sphere) [*left*] and the corresponding reconstructed versions [*right*].