The Voronoi-Quadtree: construction and visualization

I. Boada, N. Coll and J.A. Sellarès

Institut d'Informàtica i Aplicacions, Universitat de Girona, Spain

Abstract

We define a quadtree-based planar Voronoi diagram codification, the Voronoi-Quadtree, valid for generalized sites (points, line-segments, curve-arc segments, ...) and for different distance functions (Euclidean metrics, convex distance functions, ...). We present an algorithm for constructing, at a prefixed level of detail, the Voronoi-Quadtree associated to a Voronoi diagram determined by a set of sites and a given distance function. A second algorithm that, taking as input a Voronoi-Quadtree, visualizes a polygonal approximation of the boundary of the Voronoi diagram is also described.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling.

1. Introduction

The generalized Voronoi diagram [1] of a set of sites partitions the plane into regions, one per site, such that all points in a region have the same closest site according to some given distance function. Voronoi diagrams are widely used in many scientific fields and application areas, such as computer graphics, geometric modeling, geographic information systems, visualization of medical datasets, pattern recognition, robotics, shape analysis or crystal and cell growing, just to name a few (see [9]).

Algorithms for computing the Voronoi diagram, when sites are points, have been extensively developed in computational geometry and related areas [1]. These algorithms have a low computational complexity. On the contrary the computation of exact generalized Voronoi diagrams (i.e. when sites are points, line-segments, curve-arc segments, ...) use to be complicated. The algorithms that have to be applied suffer from numerical robustness problems and are time-consuming due to the numerous high precision calculations that are required. However in some applications (motion planning, geographical maps, ...) the computation of an approximate generalized Voronoi diagram within a predetermined precision is sufficient. Several algorithms have been proposed for approximating generalized Voronoi diagrams. Lavender et al.[3] use an octree representation of objects and performs spatial decomposition to compute the approximation. Vleugels et al. [8] present an algorithm for a set of disjoint convex sites and Euclidean distance in any dimension. The algorithm adaptively subdivides the space into axial cells of fixed size and computes the Voronoi diagram up to a given precision. Teichmann et al. [7] compute a polygonal approximation of Voronoi diagrams of triangles by subdividing the space into tetrahedral cells. Hoff et al. [2] visualize generalized Voronoi diagrams in the plane using interpolation-based polygon rasterization hardware.

In this paper we present an approach to visualize approximate planar Voronoi diagrams for different site shapes (points, line-segments, curve-arc segments, ...) and different distance functions (Euclidean metrics, convex distance functions, ...). Not all the sites must have associated the same distance function. The restrictions are: the Voronoi regions must be connected, the degree of a Voronoi vertex must be lower than four, and the bisectors can not contain two-dimensional pieces. The approach is based in a quadtree-based codification, the Voronoi-Quadtree. The Voronoi Quadtree (VQ) maintains in its nodes information of whether or not a Voronoi diagram is simple enough in a rectangle. If it is, the rectangular area corresponds to a terminal node. Otherwise, the rectangle is decomposed into four children, and the same process is repeated recursively. From the information encoded in the VQ terminal nodes approximation of the Voronoi diagram can be visualized at different levels of detail. Moreover, by decreasing the resolution of the VQ, the polygonal approximation tends to the exact Voronoi

diagram. The most attractive features of this VQ approach are its generality, robustness and easy implementation.

We also present two related algorithms. The first algorithm constructs the VQ associated to a Voronoi diagram defined by a set of sites and a given distance function. The second algorithm visualizes, with the desired degree of accuracy, the Voronoi region boundaries encoded in the nodes of the VQ.

The outline of the paper is as follows. In Section 2 some basic definitions are presented. Then, in Section 3 we present the VQ. Section 4 gives a detailed description of the VQ construction process, and Section 5 presents the visualization algorithm. Some results and the conclusions are provided in Section 6. Finally, Section 7 finishes with a list of research topics for the future.

2. Basic definitions

We are going to represent a site S by $S =< G_S, D_S, P_S, L_S >$, where:

- G_S represents the *geometry* of the site S. For example, if S is a circle then G_S is composed by the center and radius of S, if S is a segment then G_S is composed by the endpoints of S;
- D_S is the function that gives the distance from any point p to S. This function depends on the geometry G_S and the distance function considered (Euclidean, convex, ...). For example, if S is the circle of center C and radius r and we consider the Euclidean distance, then $D_S(p) =\| p-C \| -r$; if S is again the circle of center C and radius r but we consider the convex distance function determined by S, then $D_S(p) =\| p-C \| /r$;
- P_S is a point such that $D_S(P_S) = 0$. We say that P_S is a *base* point of S. As we will see later, we are going to use P_S to classify the site S with respect to a given rectangular region;
- L_S is a label used to identify the site S.

Let $\mathcal{S} = \{S_1, \cdots, S_n\}$ be a set of sites. Each site $S_i \in \mathcal{S}$ has associated a Voronoi region $VR(S_i) = \{p | d_{S_i}(p) \leq d_{S_j}(p) \text{ for all } j \neq i\}$. The bisector of two different sites $S_i, S_j \in \mathcal{S}$ is defined as $B(S_i, S_j) = \{p | d_{S_i}(p) = d_{S_j}(p)\}$. The boundary of $VR(S_i)$ consists of pieces of bisectors $B(S_i, S_j)$ where $i \neq j$. The generalized Voronoi diagram of $\mathcal{S}$, denoted $VD(\mathcal{S})$, is defined as the decomposition of the plane into Voronoi regions [1].

Let R be a rectangular region and S_i a site such that $R \cap VR(S_i) \neq \emptyset$. We say that the S_i is *inside* R when $P_{S_i} \in R \cap VR(S_i)$ and we say that S_i is *outside* R otherwise. Figure 1 shows a Voronoi diagram $VD(\mathcal{S})$ determined by six sites and a rectangular region R. In the figure, each Voronoi region is represented by its base point and its label. Note that the sites labeled with C and D are inside with respect to R, the sites labeled with A, B and E are outside, and site labeled with F is neither inside nor outside.

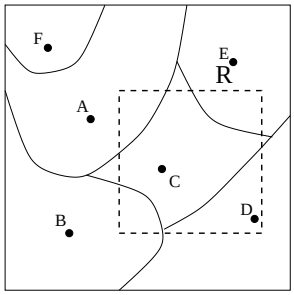


Figure 1: *Example with inside and outside sites.*

A rectangular region R is called a *basic* region when R intersects less than four Voronoi regions. In Figure 2, R_1 and R_3 are basic regions, but R_2 is not a basic region.

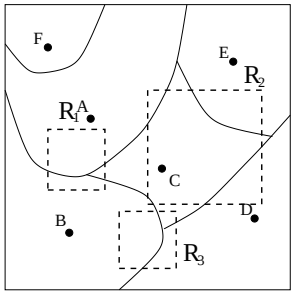


Figure 2: *Basic and not basic rectangular regions.*

3. The Voronoi-Quadtree

A quadtree [6] is a tree that encodes the recursive subdivision of a rectangular region in the plane. The root of the tree represents the region. This rectangular region is subdivided into four identical rectangular regions, called quadrants. Each quadrant is represented by one of the four nodes descendants of the root. If the information of a quadrant can not be represented in an exact way, it is labeled as a grey node. Each grey node is subdivided into another four identical rectangular regions which are represented as descendant nodes of the node representing the quadrant in question. This process is repeated recursively until quadrants contain data that can be represented exactly (named *terminal nodes*) or quadrants have a minimum edge length called resolution (*minimal resolution nodes* or *maximal subdivision nodes*).

In our case, the quadtree data structure is used to maintain the information required to generate a generalized Voronoi diagram. With this purpose we have defined the Voronoi-Quadtree. The charaterization of terminal VQ nodes (section 3.2) is based on the principle presented in section 3.1.

3.1. Main idea

Let R be a rectangular region containing the sites of $\mathcal{S}$. To obtain a quadtree representation of the part of the Voronoi diagram $VD(\mathcal{S})$ contained in R we use a recursive process that, starting with R (represented in the root node of the quadtree), subdivides each rectangular region in four new identical rectangular regions (denoted $R0$, $R1$, $R2$ and $R3$ and represented in the descendants nodes of R). This subdivision process is repeated until the actual region is a basic region or its resolution is the minimal. This subdivision process is composed of the following steps (see Figure 3) :

1. *Labeling.* At each vertex of the $R0$, $R1$, $R2$ and $R3$ regions of the actual rectangular region R we assign the label of the nearest site belonging to the set of sites classified as inside or outside with respect to R;
2. *Inside sites Identification.* At each son node $R0$, $R1$, $R2$ and $R3$ we attach the sites classified as inside with respect to them. In Figure 3 $R3$ has the sites labels A and F as inside;
3. *Propagation.* For each vertex v of $R0$, $R1$, $R2$ and $R3$ consider the set $\mathcal{N}$ of nodes that represents rectangular regions whose boundary contains v. Assume that vertex v has been labeled with label L_S in step (1). For each node $N \in \mathcal{N}$, if the site S is not an inside site of N then attach the site S to N as an outside site. As it is illustrated in Figure 4 this step might involve rectangles at different subdivision levels. Observe that the son node $R0$ has the site of label A as outside site after the subdivision of the son node $R3$.

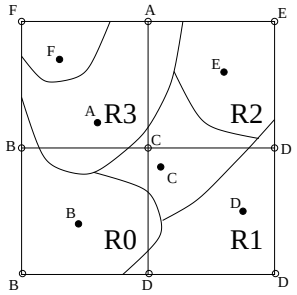


Figure 3: *Rectangle subdivision.*

3.2. Characterization of the terminal nodes

Each node N of VQ is associated to a rectangular region. Basic regions are associated to terminal nodes. To identify terminal nodes we maintain for each node:

- The labels of the vertices of the rectangular region represented by N. Each vertex label corresponds to the (unique) site of $\mathcal{S}$ closest to the vertex. We denote $V(N)$ the set of these four labels attached to N;
- The set of labels of the sites of $\mathcal{S}$ that are classified as inside with respect to the rectangular region represented by N. Let $INsites(N)$ denote this set of labels;
- The set of labels of the sites of $\mathcal{S}$ that are classified as outside with respect to the rectangular region represented by N. Let $OUTsites(N)$ denote this set of labels.

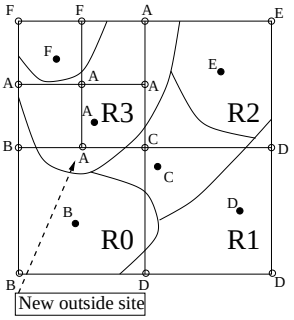


Figure 4: *Outside propagation.*

We define $L(N)$ as the total number of different labels in $V(N)$, $INsites(N)$ and $OUTsites(N)$. N is a *terminal node* if $L(N) \leq 3$.

The possible terminal configurations are characterized by the distribution of the labels of $V(N)$ and the number $L(N)$. Concretely, terminal patterns, represented in Figure 5, are:

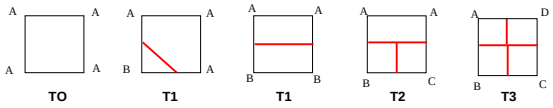
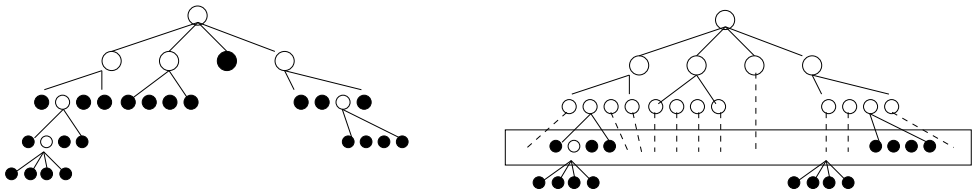


Figure 5: *Terminal node patterns of a Voronoi-Quadtree.*

- PATTERN $T0$. Obtained when $L(N) = 1$ (i.e. all the vertices of the rectangular region represented by N have the same label). In this case N is contained in a unique Voronoi region. Therefore no Voronoi region boundary intersects the edges of the rectangular region represented by N;
- PATTERN $T1$. Obtained when $L(N) = 2$. The rectangular region represented by N is intersected by two Voronoi regions. The boundary of these regions intersects two edges of the region;
- PATTERN $T2$. Obtained when $L(N) = 3$ and the rectangular region represented by N is intersected by three Voronoi regions. In this case the region is intersected by three Voronoi region boundaries;
- PATTERN $T3$. Given when $L(N) \geq 4$ and the node N is of Maximal resolution. In this case the rectangular region is intersected by four or more Voronoi region boundaries.

4. The VQ construction algorithm

In this section we are going to present, first, the algorithm that has to be applied to construct a VQ. Then, we are going



(a) (b)

Figure 6: *(a) Distribution of terminal nodes at the end of the basic algorithm. (b) Distribution of terminal nodes when a level of refinament has been introduced.*

to describe how this algorithm can be extended to obtain approximations of the Voronoi diagram at a determined degree of accuracy, which in the extreme situation tend to the exact solution.

4.1. The basic algorithm

The basic construction algorithm of the VQ requires as input $\mathcal{S}$. The algorithm is based on a top-down strategy. It starts assigning to the root of the quadtree a rectangular region containing $\mathcal{S}$ and then computing the information to be encoded in this node, i.e $V(N), INsites(N)$ and $OUTsites(N)$. Unless the root is terminal, its corresponding rectangular region is subdivided into four identical rectangular regions, each one represented by one of the four descendants of the root, and the information to be encoded in each of them is computed (see Section 3.1). This subdivision process is repeated recursively until the nodes obtained during the subdivision process are terminal nodes or are nodes of minimal resolution.

To carry out this subdivision process a priority queue, Q, is used. The queue Q maintains a pointer to non-terminal nodes. The priority of a node is the level of the node in the quadtree. In this way, the nodes in Q with upper quadtree level are always the first to be processed.

At the end of the process, for each terminal node of VQ we determine the exact set of Voronoi region boundaries intersecting the rectangular region represented by the node. Note that terminal nodes might be distributed at different levels of the quadtree. See Figure6(a), where terminal nodes have been represented as black circles.

This basic algorithm can be summarized as follows:

```
ALGORITHM ConstructVQ(INPUT:S,OUTPUT:VQ)

  N:=DefineRootNode(set of input_sites);
  INsites:=S;
  OUTsites:=NULL;
  SetVerticeLabels(N);
  if !(Terminal(N)) then Q.In(N,0);
  while !(Q.Empty()) do
   N:=Q.Out();
   if !(Terminal(N)) then
      N.DefineSons(N_sons);
      for i:=1 to 4 do
        SetVerticeLabels(N.son[i]);
        ComputeINsites(N.son[i]);
        ComputeOUTsites(N.son[i]);
        PropagateLabels(N.son[i]);
        if !(Terminal(N.son[i])) then
          Q.In(N.son[i],N.son[i].level);
        end_if
       end_for
    end_if
  end_while
```

where functions: `SetVerticeLabels(N)` computes the label of the vertices of node N; `ComputeINsites(N)` computes the inside sites of node N; `ComputeOUTsites(N)` computes the outside sites of node N; `PropagateLabels(N)` performs the propagation step described in section 3.1.

4.2. The improved algorithm

Once the VQ has been constructed we are able to visualize an approximation of the Voronoi diagram visualizing the information in the terminal nodes (see Section 5). However, in order to exploit the capabilities of the VQ hierarchical data structure we have extended the basic construction algorithm to obtain Voronoi diagrams at any level of detail. The proposed modification consists in the introduction of a new parameter: the level of accuracy, denoted L. This parameter needs also be introduced as input, and is used to force the subdivision of $T1$, $T2$ and $T3$ terminal VQ nodes whose level is lower than L. In Figure 6(b) the rectangle represents level L. To obtain the Voronoi diagram at accuracy L all terminal nodes obtained during the basic algorithm are subdivided until they reach this level. In the case that a node corresponding to the level L is not a terminal node, the subdivision continues until terminal nodes are reached.

5. The VQ visualization algorithm

Given a quadtree representation we define a *cut* on the quadtree as a set of nodes such that, for each possible root-

leaf path, one and only one of its nodes is contained in the cut.
The VQ visualization algorithm, used to obtain a visualization of the Voronoi diagram codified in the VQ, is divided into two steps: (1) the selection of the cut; and (2) the visualization of the cut using the node visualization function.

5.1. Cut selection

The first visualization step determines which nodes of VQ compose the cut C. To select these nodes a criteria that determines when a VQ node is considered or not member of C is required. In our case we consider a node member of C when it is a terminal node.

To select C a top-down traversal of the quadtree is performed. Each node N is evaluated: if N is terminal we add N to C; conversely, if the node is not terminal we check the children of N. At the end of this process C is composed for all the terminal nodes.

5.2. Node visualization

To visualize C we apply a visualization function to each one of its nodes. This function visualizes the set of segments that define, in an approximated manner, the part of the Voronoi region boundaries encoded in the node.

To generate these segments the next considerations are taken into account:

- The Voronoi region boundaries encoded in a terminal node can be approximated by a set of segments. The number of segments required for the visualization varies with the pattern encoded in the terminal node. Figure 5 shows the pattern:number of segments relation T0:0, T1:1, T2:3 and T3:4;
- The segments used for the visualization of the Voronoi region boundaries are forced to meet certain positions. There is a limitation on the number of possible segment incidences. This strategy is based on the isosurface reconstruction technique, the Discretized Marching Cubes, proposed by Montani et al. [5] algorithm;

The node visualization function applied to each node of C is composed of the three following steps:

(1) *Identification of intersected edges*. The edges of the rectangular region represented by the node intersected by the segment that approximates the Voronoi region are identified. An edge is intersected by the segment when the labels of its vertices are different;
(2) *Selection intersection point position.* The position of the intersection point is determined by the level of the node and the level of its adjacent terminal nodes. Observe that nodes in the cut C may belong to different levels of the quadtree. Therefore, the selection of the intersection point depends on the level of the processed node and on the levels of its adjacent terminal nodes:
   - if the adjacent node is in the same or in an upper level we select the midpoint as the intersection point;
   - if the adjacent node is in a lower level, first we detect its level and then we set the intersection point to match with the point computed for the adjacent node (i.e. a higher level node is forced to match with the lower one).

   This selection strategy guarantees the continuity of the visualized Voronoi region boundaries. Note that if we select the midpoint of the edge as the intersection point independently of the level of the node, then the final visualization might contain discontinuities. This undesirable situation has been illustrated in Figure 7);
(3) *Visualization of the segments*. The selected intersection points are connected, according its terminal configuration, and the corresponding segment is visualized.

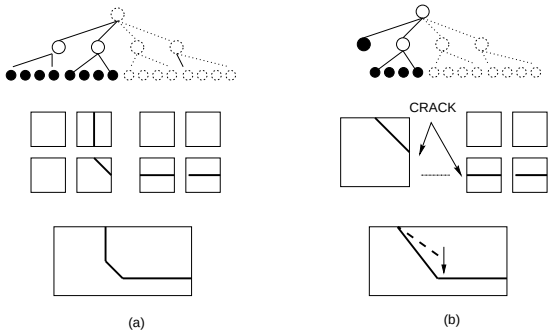


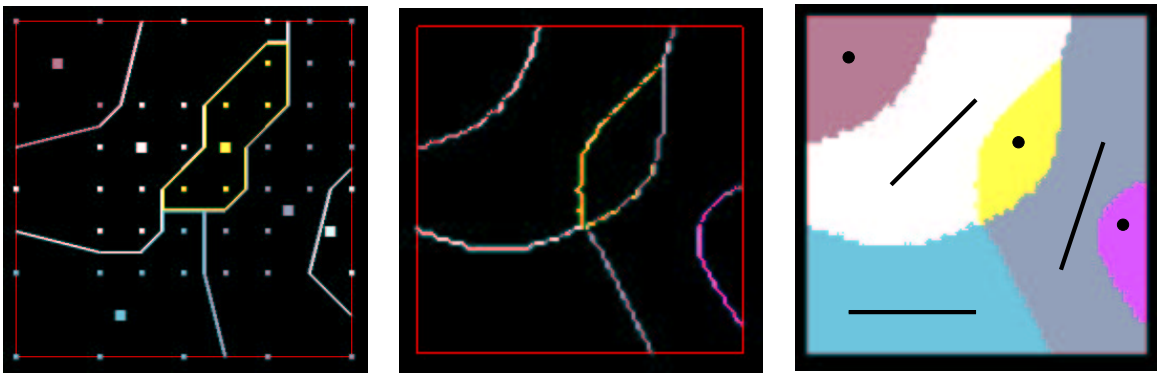
Figure 7: *Black circles identify terminal nodes of the quadtree. (a) When all the nodes are at the same level the edge/segment intersection point coincides with the midpoint of the edge. The Voronoi region boundaries are continuous. (b) When terminal nodes correspond to different quadtree levels, the midpoint generates a discontinuity. The intersection point of the upper node is forced to match with the intersection point position of the lower node.*

The inputs used by the visualization algorithm are the VQ and the criteria used to select the nodes of the cut, in our case the terminal nodes. However, note that this criteria can vary according the user or the application parameters, and that in this manner the capabilities of a quadtree to support multiresolution can be exploited.

6. Experimental results and conclusions

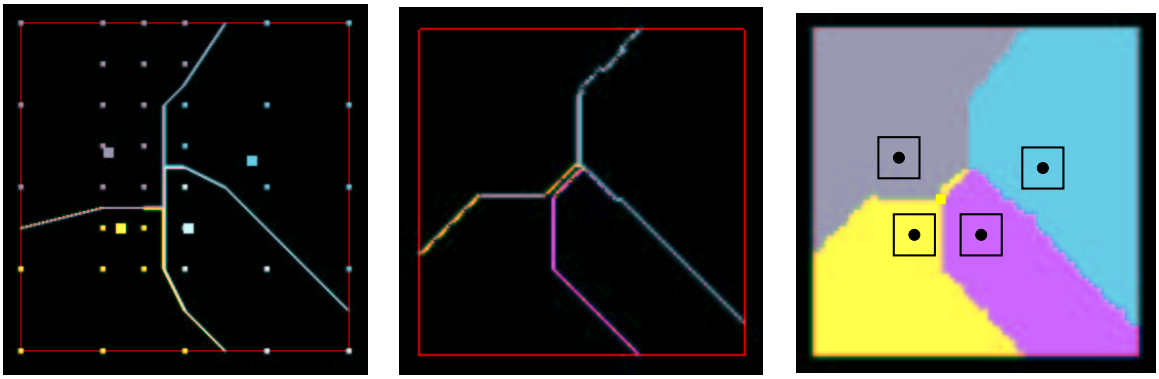
The algorithms of construction and visualization of the VQ have been implemented. Figures 8 and 9 show some of the results obtained with these implementations considering different distances and resolutions.

Figure 8 has been obtained using the point and segment sites represented in 8(c) and the Euclidean distance. Figure 8(a) corresponds to the visualization of the VQ obtained from the basic construction algorithm. Note that the cut used



(a) (b) (c)

Figure 8: *Visualization of a VQ obtained with Euclidean distance.*



(a) (b) (c)

Figure 9: *Visualization of a VQ obtained with a convex distance determined by an square.*

for the rendering is composed of nodes distributed at different levels. To illustrate this fact we have assigned the color of the closest site to the vertices of the rectangular region represented by the node. Observe that the segments representing the boundary of the Voronoi regions are continous. Figure 8(b) has been obtained using the improved algorithm with a higher degree of accuracy.

Figure 9 has been obtained using a convex distance function (see Figure 9(c)). As in the previous case, Figures 9(a) and 9(b) correspond to the visualization of the VQ obtained from the basic and the improved construction algorithms respectively.

As it can be seen from the previous figures, preliminary results are very encouraging. However, we are conscious that a more accurate analysis of the VQ has to be done. A comparative study (precision, run time, ...) with previous methods for visualizing generalized Voronoi diagrams remains to be done.

7. Future work

We are extending the construction algorithm so that the VQ can also be constructed in some special cases: Voronoi vertices of degree greater than three, bisectors that contain two-dimensional pieces, etc.

Among future developments, we consider of high interest to evaluate the capabilities of the VQ to add and to delete sites. Due to the hierarchical nature of the VQ, this data-structure is well-suited to support multiresolution, therefore an other promising line of research is the design of adaptive algorithms to generate the VQ at different chosen levels of detail in selected regions.

Our final aim is to extend our approach to three-dimensional space for approximating generalized Voronoi diagrams, defining a Voronoi-Octree structure and visualizing the polyhedral approximation of the corresponding Voronoi diagram.

Acknowledgements

This work was supported by DURSI 2001SGR-00296. The first author was supported in part by grants TIC2000-1009 and TIC2001-2226-C02-02. The second and third authors were supported in part by grants MEC-DGES-SEUID PB98-0933, and TIC2001-2392-C03-01.

References

1. F. Aurenhammer and R. Klein. *Voronoi diagrams. In Handbook of Computational Geometry. J.-R. Sack, J. Urrutia, editors*. Elsevier Science Publishers, 2000.
2. K. Hoff, T. Culver, J. Keyser, M. Lin, D. Manocha, Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware. *Proc. SIGGRAPH'99, ACM Press*, pp 277-286,(1999).
3. D. Lavender, A. Bowyer, J. Davenport, A. Wallis and J. Woodwark, Voronoi diagrams of set-thoretic solid models. *IEEE Comput. Graph. Appl.*, 12(5):69-77,(1992).
4. D. Meagher, Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129-147,(1982).
5. C. Montani, R. Scateni and R. Scopigno. Discretized Marching Cubes. *Visualization '94 Proceedings*, R.D. Bergeron and A.E.Kaufman, Eds. 1994, pp 281-287. IEEE Computer Society Press.
6. H. Samet. *Applications of Spatial Data Structures*. Addison Wesley, Reading, MA, (1990).
7. M. Teichmann and S. Teller, Polygonal approximation of Voronoi diagrams of a set of triangles in three dimensions. *Technical Report 766*, Laboratory of Computer science, MIT,(1997).
8. J. Vleugels and M. Overmars, Approximating Generalized Voronoi Diagrams in Any Dimension. *Technical Report UU-CS-1995-14*, Utrech University,(1995).
9. Voronoi Diagrams page on the Web: Applications. *http://www.voronoi.com/ApplFrames.htm*