A Simple GPU-based Approach for 3D Voronoi Diagram Construction and Visualization

Hsien-Hsi Hsieh and Wen-Kai Tai*
Department of Computer Science and Information Engineering
National Dong Hwa University
Tel: +886-3-8634023
Fax: +886-3-8634010
E-mail: wktai@mail.ndhu.edu.tw

Abstract

In the paper we propose a simple GPU-based approach for discrete incremental approximation of 3D Voronoi diagram. By constructing region maps via GPU. Nearest sites, space clustering, and shortest distance query can be quickly answered by lookup the region map. In addition, we proposed another representation of the 3D Voronoi diagram for visualization.

Keywords: Voronoi diagram, Visualization, GPU, Distance transform, Voxelization.

1 Introduction

The Voronoi diagram is a natural and fundamental concept related to geometric proximity that has important applications in many fields of science. Given a set of primitives called Voronoi sites, the Voronoi diagram partitions the space into convex regions so that all of the subspaces belonging to a region have the same closest site under a give distance function. There are many modified Voronoi diagrams by taking sites as different shapes or nature, applying different distance function, associating weights, or changing metrics. Voronoi diagram is also used in many fundamental problems such as nearest neighbor query, clustering, triangulation, and minimum spanning tree.

Construction of Voronoi diagram in 2D and 3D is a fundamental problem and there exist efficient and optimal algorithms running on CPU for calculating Voronoi vertices which are the intersections of the boundaries of domains. Voronoi diagram is structured by a connected segments of edges or curves in 2D or connected polygons or paraboloids in 3D. However, it needs additional computation to handle queries for nearest sites or shortest distance to sites. Therefore, we propose a discrete incremental approximation of 3D Voronoi diagram by constructing region maps by GPU. Nearest sites and shortest distance query can be quickly answered by look up the region map. In addition, we can visualize the Voronoi diagram in a different representation.

In this paper, we introduce a simple but flexible approximate construction of 3D Voronoi diagram using GPU for visualization. It could generate a map for visualization or generate a region map for query of nearest neighbor, space clustering, and shortest distance. The Voronoi diagram construction in GPU is simple. At first, we subdivide the space into subspaces by GPU. Then for each Voronoi site, distance between a subspace, called a voxel, and the site is calculated and compared with the shortest distance stored in the voxel. Each voxel keeps the shortest distance and the region ID it belongs. In current implementation, we use color as region ID and z-buffer for storing shortest distance of each voxel.

The rest of paper is organized as follows. Some related works are surveyed in the section 2. In section 3, we present the rendering framework. The experimental results are illustrated in section 4. Finally, we conclude the proposed approach and point out some future works.

2 Related Works

For computing Voronoi diagrams of points in 2D, 3D, and higher dimensions, Shamos and Hoey [14] proposed a divide-and-conquer algorithm; and Fortune [4] proposed a sweepline algorithm. Gold [6] et al. proposed a simple method to construct Voronoi diagram of line segments sites by incremen-

tally expanding line segments and using kinematic Voronoi diagram methods to maintain the Voronoi diagram. Besides, numerically robust algorithms for constructing topologically consistent Voronoi diagrams have been proposed by [8] and [16].

In Voronoi diagram approximation, Lavender et al. [9] proposed a hierarchical approach to compute an approximate Voronoi diagram of a set of general sites in arbitrary dimension. Sites are represented by an octree and the cells of the approximate diagram are obtained by considering the distance to the sites. Vleugels et al. [20, 21] have presented an approach that adaptively subdivides space into regular cells and computes the Voronoi diagram up to a given precision. Teichmann et al. [18] proposed a technique that subdivides the space into tetrahedral cells for triangle sites. Sites are inserted into a standard octree and the Voronoi diagram is computed by a wavefront propagation strategy and visualized in polygonal approximation. However, these algorithms take considerable time and memory for large models composed of tens of thousands of triangles, and cannot easily be extended to directly handle dynamic environments. Hoff et al. [7] presented an efficient method for constructing generalized Voronoi diagram in graphics hardware. They use geometry such as con and tent to approximate distance propagation of Voronoi sites. However, the precision highly depends on the quality of rendered 3D polygonal mesh that would take high cost for numerous sites in high resolution. Boada et al. [2] proposed a technique that approximate general 3D Voronoi diagrams by Voronoi-Octree and visualize the Voronoi diagram via polyhedral approximation. In constructing Voronoi diagrams of higher order sites, two broad approaches based on incremental and divide-and-conquer techniques have been summarized in [12]. Fischer and Gotsman [3] presented a GPU based approach for approximate high order Voronoi diagram and distance transform. They implemented tangent-plane algorithm for computng k-th order Voronoi diagram of a set of sites in image space.

Voronoi diagram is widely used in many applications. For example, Takahashi et al. [17] proposed moving along Voronoi boundaries to avoid obstacles. Lengyel et al. [10] also addressed the more general motion planning problem and proposed an algorithm which employs the graphics hardware for path planning computations. Vona and Rus [22] used the Voronoi diagram to perform geometric computations associated with toolpath planning for mechanical etch of printed-circuit boards.

Highly relative to Voronoi diagram, distance field can also be used for voxelization and representation of objects which are difficult to model. Frisken et al. [5] generated signed distance on grid structure by computing the minimum signed distance from the grid points to object to represent an object. Varadhan [19] used Max-Norm distance computation algorithm with hardware accelerated Voronoi diagram generation [7] to determine whether a voxel intersects a polygon for voxelization. But the model is designed for convex case, so it needs extra work to convert concave object into convex primitives. Sigg et al. [15] improved and implemented a hardware-accelerated version of the work of Mauchs [11] to generate signed distance field using bounded local distance fields and bounding polyhedrons of meshes.

A comprehensive overview of numerous works and variants of Voronoi diagrams may be found in the excellent survey by Aurenhammer [1] and book by Okabe et al. [12].

3 The Proposed GPU Framework

3.1 Basic Definition

Let $S = \{s_1, s_2...\}$ denotes a set of point sites in 3D. For two distinct sites $s_p, s_q \in S$, the space can be partitioned into two half spaces or domains. Domain of s_p over s_q can be formed by

$$Dom(s_p, s_q) = \{v \in R^3 | \sigma(v, s_p) \leq \sigma(v, s_q)\}, \quad (1)$$

where σ is a distance function. Here we use squared Euclidean distance function for lower computation cost. Then the Voronoi region of s_p can be formed by

$$R(s_p) = \bigcap_{s_q \in S - \{s_p\}} Dom(s_p, s_q). \quad (2)$$

Therefor each region $R(s_p)$ will cluster the subspace nearest to the site s_p by a given distance function.

In our approach, we use distance function to generate individual distance field for each Voronoi site. The space is then partitioned into regions by comparing domains of Voronoi sites. Due to help of advanced programming graphics hardware, Voronoi diagram can be generated by discrete approximation as region maps incrementally and visualized in different representation.

3.2 Rendering Framework

Figure 1 shows the rendering process for generating the distance field and domain region for a Voronoi

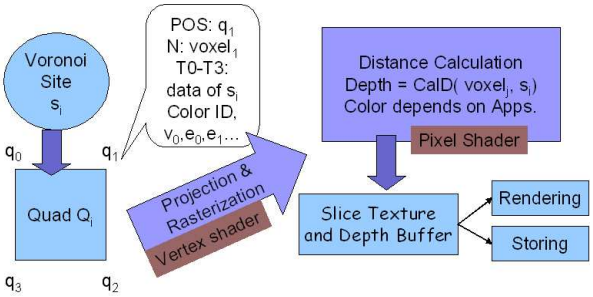


Figure 1: Rendering Pipeline for generating the distance field and Voronoi colored region of a Voronoi site. A quad including data of the Voronoi site is rendered as a computation query. Five channels of each vertex of a quad (position, normal, and 4 texture coordinates) is filled with position of quad, position of voxel, and information of a Voronoi site such as ID color and parametric coefficients of geometry of the Voronoi site.

site. A general rendering framework for distance field computation and Voronoi region construction consists three steps: Voronoi sites collection and geometry generation, space voxelization and query of distance field computation, and color rendering with distance field.

At first, Voronoi sites are classified into three groups: point site, edge sites, and triangle sites. For each site, a full-filled quad geometry is constructed as a distance computation query to submit to GPU. Channels of texture coordinates for each quad is filled by information depending on different types of site. For example, associated data of a quad for a point site is the ID color and position of the point, for a edge site the ID color and parametric coefficients of the line segment (a point and a vector) are included, and for a triangle sites the ID color and parametric coefficients of the triangle (a point and two vectors) are associated.

Space voxelization is processed slice by slice due to limit storage of graphics hardware. As shown in Figure 1, for each Voronoi site s_i,

$$\begin{aligned} s_i &= \{P_i = (x, y, z)\}, \\ &or \quad \{E_i(t) | v_0 + t\vec{e_0}, 0 \le t \le 1\}, \\ &or \quad \{T_i(s, t) | B + s\vec{e_0} + t\vec{e_1}, s \ge 0, t \ge 0, s + t \le 1\}, \end{aligned}$$

a full-filled quad $Q_i = \{q_{i_0}, q_{i_1}, q_{i_2}, q_{i_3}\}$ is rendered and rasterized to generate the distance field from voxels on a slice to the site. Voronoi site data and voxel positions are associated within vertices of the rendering quads. Voxel positions are stored in the channel of vertex normal, the site data are separately stored in channels of texture coordinates and transmitted to GPU. Voxel positions are linearly interpolated in rasterization of rendering pipeline and pairs of site data and voxel positions are sent to pixel processors for distance computation. After distance computation, the shortest distance between a Voronoi site and a voxel is stored in the pixel depth and the pixel color is used for visualization according to the representation of Voronoi diagram. For example, ID color rendering is used for generating region map for nearest neighbor site or cluster query; assigning distance intensity to color is used for visualization of the distance field.

The rendering pseudo code can be abstracted as follows:

```
for each Voronoi site s on slice i {
  Create a quad Q for the site s
  for k = 0 to 3 {
    // assign a full-filled quad
    // q is end vertices of quad
    Q.q[k].position = ScreenBoundary.q[k];
    // assign voxel position, and Voronoi site data
    Q.q[k].normal = Slice[i].q[k];
    Q.q[k].tex0 = s.colorID;
    if (s is a point Voronoi site) Q.q[k].tex1 = s.position;
    if (s is an edge Voronoi site) {
      Q.q[k].tex1 = s.v0; Q.q[k].tex2 = s.e0;}
    if (s is a triangle Voronoi site) { Q.q[k].tex1 = s.B;
      Q.q[k].tex2 = s.e0; Q.q[k].tex3 = s.e1}
    }
  RenderQuad(Q);
}
```

Voronoi diagram is constructed incrementally by rendering quads of Voronoi sites. Unless Voronoi diagram is recalculated on different slice or a rendered Voronoi site moves, rendered quads have no need to be re-rendered again even new Voronoi sites are added in the set. There are three types of fragment programs to generate the distance field for point, edge and triangle Voronoi sites. Detail of distance computation on GPU is described in the next subsection.

3.3 Distance Function on Voronoi Sites

Different Voronoi site generates different distance field. For example, a point Voronoi site generates iso-diatnace as a sphere, and an edge Voronoi site generates iso-distance as the outer surface of the union of two balls and a cylinder. However, most geometry can be decomposite into points, lines and faces(triangles). We present distance function calculation on fragment program for the three cases as follows.

3.3.1 Distance Function on Points

For a given point Voronoi site $P(x, y, z)$, distance function $\sigma(v, P)$ is a simple squared euclidean dis-

tance between two points as follows:

$$\sigma(v,P) = ||x-P||^2 = (v_x-P_x)^2+(v_y-P_y)^2+(v_z-P_z)^2. \quad (3)$$

While using cones to approximate the distance field of points [7] has high cost for high precision, the distance field is simply generated by fragment program.

3.3.2 Distance Function on Edges

For a given edge Voronoi site $E(t) = \{v_0 + t\vec{e_0}, 0 \leq t \leq 1\}$, distance function $\sigma(v, E(t))$ calculates minimum distance between point v and points on edge $E(t)$. Shortest distance point on line $E(t)$ to point v can be easily found by solving $d(||v - E(t)||^2)/dt$. And then the coefficient t of minimum distance is:

$$t_{min} = -\frac{(v_0 - v)\cdot\vec{e_0}}{||\vec{e_0}||^2}.$$

By constraint t, $\sigma(v, E(t))$ is formed as:

$$\sigma(v, E(t)) = at^2 + 2bt + c, \quad t = clamp(t_{min}) \quad (4)$$

where function $clamp(t)$ truncates value t in range [0,1], and

$$\begin{aligned} a &= \vec{e_0}\cdot\vec{e_0} \\ b &= \vec{e_0}\cdot(v_0 - v) \\ c &= (v_0 - v)\cdot(v_0 - v) \end{aligned}$$

3.3.3 Distance Function on Triangles

For a given 3D point $v(x, y, z)$ and a triangle $T(V_0, V_1, V_2)$, Hausdoff-distance is the shortest distance between the point v and any point p on the triangle. A point on triangle T can be parametrically defined by two linearly independent vector with two weights (s, t) by

$$T(s,t) = B + s\vec{e_0} + t\vec{e_1},$$

where $(s,t) \in D = \{(s,t) : s \in [0,1], t \in [0,1], s + t \leq 1\}$, and $B = V_0, \vec{e_0} = V_1 - V_0$ and $\vec{e_1} = V_2 - V_0$.

For any point on triangle T, the distance from T to v is

$$\|T(s,t) - v\|,$$

or we can instead use the squared-distance function

$$Q(s,t) = \|T(s,t) - v\|^2,$$

where exists a point $p' = (\bar{s}, \bar{t})$ which makes $Q(\bar{s}, \bar{t})$ minimum.

Therefore, the computation of distance can be reduced into a minimization problem. For an efficient computation, we can expand $Q(s,t)$ as

$$Q(s,t) = as^2 + 2bst + ct^2 + 2ds + 2et + f,$$

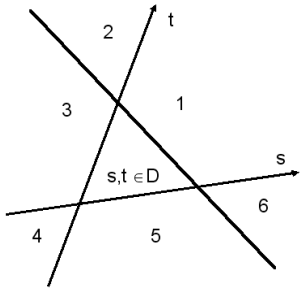


Figure 2: Six regions in s, t coordinate. Space is partitioned by range of parameters s and t for efficient shortest position classification and calculation.

where

$$\begin{aligned} a &= \vec{e_0}\cdot\vec{e_0} \\ b &= \vec{e_0}\cdot\vec{e_1} \\ c &= \vec{e_1}\cdot\vec{e_1} \\ d &= \vec{e_0}\cdot(B - v) \\ e &= \vec{e_1}\cdot(B - v) \\ f &= (B - v)\cdot(B - v) \end{aligned}$$

From analyzing the gradient of $Q(s,t)$, the minimum $\bar{s}$ and $\bar{t}$ happens only when ∇Q is zero, where

$$\bar{s} = \frac{be - cd}{ac - b^2} \qquad \bar{t} = \frac{bd - ae}{ac - b^2}$$

If $(\bar{s}, \bar{t}) \in D$, the minimum distance is the distance between p' and v; otherwise, according to the sign of $\bar{s}$ and $\bar{t}$, there are six possible regions that the shortest distance point p'' may lie on triangle T, as shown in Figure 2. Efficient solutions are well addressed on the book [13] for CPU computation with simple calculation with logic classification.

However, in GPU, there is no efficient dynamic flow control to determine the shortest point on a triangle. Therefore, instead of directly computing the point of shortest distance on a triangle, we compute the distance from the 3D point to four possible points which may be inside the triangle or on the three boundaries and then the minimum scalar is the shortest distance. These four points are

$$\begin{aligned} (s_0,t_0) &= (\frac{be - cd}{ac - b^2}, \frac{bd - ae}{ac - b^2}) \\ (s_1,t_1) &= (0, -\frac{e}{c}) \\ (s_2,t_2) &= (-\frac{d}{a}, 0) \\ (s_3,t_3) &= (\frac{c + e - b - d}{a - 2b + c}, \frac{a + d - b - e}{a - 2b + c}), \end{aligned}$$

where position (s_0, t_0) assumes point p' is inside the triangle, positions (s_1, t_1), (s_2, t_2) and (s_3, t_3) assume point p'' is on boundaries of $s = 0$, $t = 0$, and $s+t = 1$. All calculated points are truncated to

make $(s,t) \in D$ so that three end vertices of the triangle T are also in consideration and it guarantees these points are on the triangle for distance computation. Therefore, the minimum distance is the shortest distance from the point v to the triangle T. Then distance function $\sigma(v, T(s,t))$ is formed by

$$\begin{aligned} \sigma(v, T(s,t)) &= min\{Q_0(s'_0, t'_0), Q_1(s'_1, t'_1), \\ &\quad Q_2(s'_2, t'_2), Q_3(s'_3, t'_3)\} \end{aligned} \quad (5)$$

where $(s'_i, t'_i) = clamp((s_i, t_i))$, and

$$\begin{aligned} Q_0(s,t) &= V(s+t)(as^2 + 2bst + ct^2 + 2ds + 2et) + f \\ Q_1(s,t) &= ct^2 + 2et + f \\ Q_2(s,t) &= as^2 + 2ds + f \\ Q_3(s,t) &= as^2 + 2bst + ct^2 + 2ds + 2et + f \end{aligned}$$

where function $V(x)$ returns 1 if $x \leq 1$, otherwise it returns 0.

3.4 Visualization of Voronoi Diagram

Generally a region map is constructed by assigning ID color of the Voronoi site to region color. It is enough to answer the query of nearest Voronoi site site or the cluster of neighborhood by look up the region map. However, there is no enough information about the geometry of Voronoi sites and the distance from the Voronoi sites to their regions. Directly visualize the distance field of the Voronoi sites will lose the information of distinct regions. Therefore, for visualizing the Voronoi diagram with information about the regions, distance, and geometry of Voronoi sites, we combine ID color and the distance field of a Voronoi site as the new color of its region, by the following function:

$$\begin{aligned} Color_{new} &= f * Color_{ID} + (1-f) * Color_{DF} \\ Color_{DF}.Red &= clamp(D * w_R - C_R) \\ Color_{DF}.Green &= clamp(C_G - D * w_G) \\ Color_{DF}.Blue &= clamp(D * w_B - C_B) \\ f &= clamp(C_G - D * w_G) + D * w_C \end{aligned}$$

where color in display $Color_{new}$ is blended by the $Color_{ID}$ of the Voronoi site and the colored distance field $Color_{DF}$ with a weight f. D is the distance value of a voxel. w_R, w_G, w_B and w_C are scale factors to scale distance value to a noticeable range. C_R, C_G, and C_B are three parameters to modify the mapping from distance to color. In our experiment, (C_R, C_G, C_B) and (w_R, w_G, w_B, w_C) are (0.2, 1.0, 0.5) and (24, 64, 16, 6) respectively so that regions of distance from low to high will be shaded into four levels: green blending ID color of sites, red, blue, and ID color of sites. Geometry of Voronoi sites and the propagation of distance field of Voronoi sites can be easily seen from the regions in ID color. Figure 3 shows the visualization of 3D Voronoi region on a slice for two distinct points, edges, and triangles.

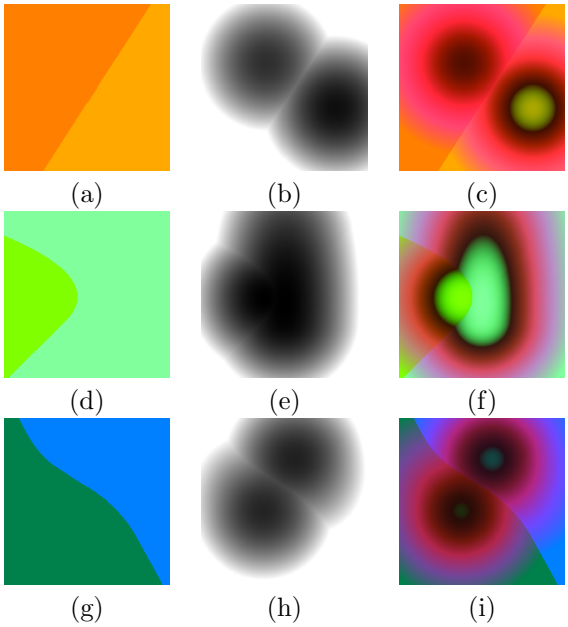
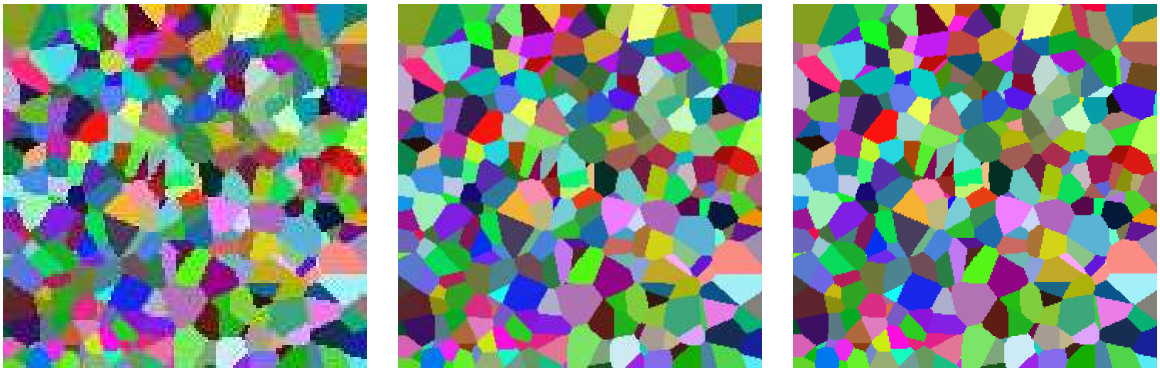


Figure 3: Visualization on simple Voronoi diagram and the distance field. (a) A Voronoi diagram constructed by 2 points. (b) Distance field of 2 points. (c) Visualization of (a). (d) A Voronoi diagram constructed by 2 edges. (e) Distance field of 2 edges. (f) Visualization of (d). (g) A Voronoi diagram constructed by 2 triangles. (h) Distance field of 2 triangles. (i) Visualization of (g).

4 Experimental Results

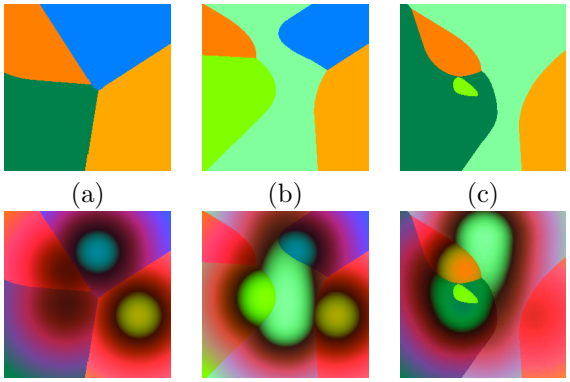
We implement our fragment program using HLSL on a Pentium 4 3.0 MHz PC with 1G RAM with a nVidia Geforce FX5800 graphics card running on Windows XP with DirectX 9.0c. We use Vertex Shader 1.1 and Pixel Shader 2.0 to implement fragment program in scattering pairs of voxel positions and Voronoi site data and in distance calculation and Voronoi diagram visualization.

3D Voronoi diagrams in the experiment are calculated on a slice under voxel resolution of 512^2. All Voronoi sites are randomly assigned. Figure 4 shows 3D Voronoi diagrams constructed by union of 2 vertices, 2 edges, and 2 triangles. In Figure 4f, a light green line Voronoi region appears on a dark green triangle Voronoi region especially inside the low distance region of the triangle Voronoi region. It happens when the line intersects the triangle. However, in Figure 4c, it is difficult to realize the situation. Figure 5 shows 3D Voronoi diagrams of 50, 2000, and 10000 point Voronoi sites. When many Voronoi sites contribute their distance fields to a slice, individual Voronoi region becomes



(a) (b) (c)

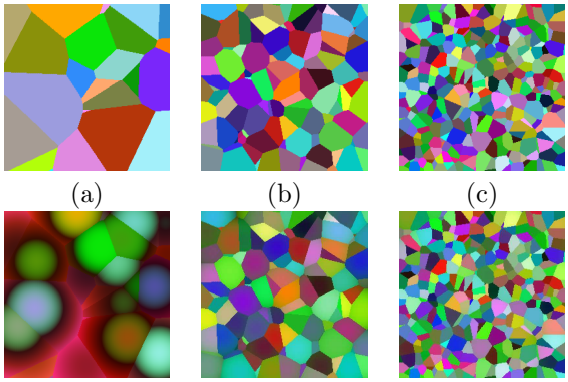
Figure 6: Effectiveness comparison on the same Voronoi diagram constructed by 10k point Voronoi sites on proportional slices under voxel resolution of (a) 128^2,(b) 256^2 and (c) 512^2.



(a) (b) (c)

(d) (e) (f)

Figure 4: Visualization on Voronoi diagrams constructed by Voronoi sites in different shapes. (a) A Voronoi diagram constructed by 2 points and 2 triangles. (b) A Voronoi diagram constructed by 2 points, 2 edges and 2 triangles. (c) The same 3D Voronoi diagram as (b) but on a different slice. (d) Visualization of (a). (e) Visualization of (b). (f) Visualization of (c).



(a) (b) (c)

(d) (e) (f)

Figure 5: Visualization on Voronoi diagrams constructed by point Voronoi sites. (a), (b) and (c) are Voronoi diagrams constructed by 50, 2000, and 10000 points respectively. (d), (e) and (f) are visualization of (a), (b) and (c) respectively.

small and the stored distance value in each voxel is small too. Thus the image resulted by our visualization method is almost identical to visualization by ID color, which is better in visualizing many small Voronoi regions.

Performance of Voronoi diagram construction depends on both voxel resolution and the number of Voronoi sites. 3D Voronoi diagram can be visualized in real time or interactively with hundreds of Voronoi sites. While visualizing 3D Voronoi digram of 10000 point Voronoi site, construction of Voronoi diagram on a slice is processed in a second under voxel resolution of 128^2. However, it still produces a reliable map of Voronoi diagram because it is based on pixel-wise distance computation. Figure 6 shows a Voronoi diagram of 10k point sites on proportional slices under voxel resolution of 128^2, 256^2 and 512^2.

5 Conclusion

In this paper, we introduce a GPU-based approach for 3D Voronoi diagram construction and visualization. We calculate minimum distance between pairs of sampled voxels and Voronoi sites in different shapes for guarantee of Hausdorff distance. With programmable hardware vertex/pixel processors, Voronoi diagram can be incrementally constructed by rendering simple quad geometry with

associated data of Voronoi site and visualized with different representation.

However, in current implementation, performance of pixel shader is the bottleneck in overall processing speed. Area of rasterization also has a significant influence on the loading of pixel shader. Therefore, in the near feature, searching a better computational methodology for GPU is a way to improve performance of construction of Voronoi diagram. In addition, a sophisticated Voronoi sites culling for reduce the number of computation query of the distance field on Voronoi sites will also be a solution in demand. We would also like to extend our work for more types of Voronoi diagrams and find better or more meaningful visualization on Voronoi diagram for applications.

References

[1] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Computer Surveys*, 23(3):686–695, 1991.

[2] I. Boada, N. Coll, N. Madern, and J. A. Sellareś. Approximations of 3d generalized voronoi diagrams. In *Proc. European Workshop on Computational Geometry*, pages 163–166, 2005.

[3] I. Fischer and C. Gotsman. Fast approximation of high order voronoi diagrams and distance transforms on the GPU. *Technical Report TR-07-05*, 2005. Computer Science Group, Harvard University.

[4] S. Fortune. A sweepline algorithm for voronoi diagrams. In *Proc. 2nd Annual ACM Symp. on Computational Geometry*, pages 313–322, 1986.

[5] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proceedings of ACM SIGGRAPH '00*, pages 249–254, 2000.

[6] C. M. Gold, P. R. Remmele, and T. Roos. Voronoi diagrams of line segments made easy. In *Proc. Canadian Conference on Computational Geometry*, pages 223–228, 1995.

[7] K. E. Hoff III, J. Keyser, M. Lin, D. Manocha, and T. Culver. Fast computation of generalized voronoi diagrams using graphics hardware. In *Proceedings of ACM SIGGRAPH '99*, pages 277–286, 1999.

[8] H. Inagaki, K. Sugihara, and N. Sugie. Numerically robust incremental algorithm for constructing three-dimensional voronoi diagrams. In *Proc. Canadian Conference on Computational Geometry*, pages 334–339, 1992.

[9] D. Lavender, A. Bowyer, J. Davenport, A. Wallis, and J. Woodwark. Voronoi diagrams of set-thoretic solid models. *IEEE Computer Graphics and Applications*, 12(5):69–77, 1992.

[10] J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg. Real-time robot motion planning using rasterizing computer graphics hardware. In *Proc. 17th annual conference on Computer graphics and interactive techniques*, pages 327–335, 1990.

[11] S. Mauchs. Effcient algorithms for solving static hamilton-jacobi equations. *Ph.D. Ihesii, California Inst. of Tech*, 2003.

[12] A. Okabe, B. Boots, K. Sugihara, and S.N. Chiu. *Spatial Tessellations: Concepts and Application of Voronoi Diagrams*. John Wiley & Sons, 2000. 2^{nd} Ed.

[13] P. Schneider and D. H. Eberly. *Geometry Tools for Computer Graphics*. Morgan Kaufmann, 2003.

[14] M.I. Shamos and D. Hoey. Closest-point problems. In *Proc. 16th Annual IEEE Symp. on Foundations of Computational Science*, pages 151–162, 1975.

[15] C. Sigg, R. Peikert, and M. Gross. Signed distance transform using graphics hardware. In *Proceedings of IEEE Visualization*, pages 19–24, 2003.

[16] K. Sugihara and M. Iri. A robust topology-oriented incremental algorithm for voronoi diagrams. *Int. J. Computational Geometry and Applications*, 4:179–228, 1994.

[17] O. Takahashi and R. J. Schilling. Motion planning in a plane using generalized voronoi diagrams. *IEEE Transactions on Robotics and Automation*, 5(2):143–150, 1989.

[18] M. Teichmann and S. Teller. Polygonal approximation of voronoi diagrams of a set of triangles in three dimensions. *Technical Report 766*, 1997. Laboratory of Computer science, MIT.

[19] G. Varadhan, S. Krishnan, Y. J. Kim, S. Diggavi, and D. Manocha. Efficient max-norm distance computation and reliable voxelization. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 116–126, 2003.

[20] J. Vleugels, V. Ferrucci, M. Overmars, and A. Rao. Hunting voronoi vertices. *Int. J. Computational Geometry and Applications*, 6:329–354, 1996.

[21] J. Vleugels and M. Overmars. Approximating generalized voronoi diagrams in any dimension. *Int. J. Computational Geometry and Applications*, 8:201–221, 1998.

[22] M. A. Vona and D. Rus. Voronoi toolpaths for pcb mechanical etch: Simple and intuitive algorithms with the 3d GPU. In *Proc. International Conference on Robotics and Automation*, 2005.