Generalized Voronoi Diagram Computation on GPU

Zhan Yuan*, Guodong Rong†, Xiaohu Guo†, Wenping Wang*
*Department of Computer Science
The University of Hong Kong
Hong Kong
†Department of Computer Science
University of Texas at Dallas
Richardson, TX, USA

***Abstract*—We study the problem of using the GPU to compute the generalized Voronoi diagram (GVD) for higher-order sites, such as line segments and curves. This problem has applications in many fields, including computer animation, pattern recognition and so on. A number of methods have been proposed that use the GPU to speed up the computation of the GVD. The jump flooding algorithm (to be called *JFA*) is such an efficient GPU-based method that is particularly suitable for computing the ordinary Voronoi diagram of point sites. We improve the jump flooding algorithm and apply it to computing the GVD. Specifically, instead of directly propagating the complete information of a site (i.e. the coordinates or other geometric parameters) as in the original JFA, we store the site information in a 1-D texture, and propagate only the IDs, which are short integers, of the sites in another 2D texture to generate the Voronoi diagram. This simple strategy avoids storing redundant data and leads to considerately more accurate computation of the GVD with much less memory than using the original JFA, with only moderate increase of the running time.**

***Keywords*-Generalized Voronoi Diagram; Graphics Hardware; Jump Flooding Algorithm;**

I. Introduction

Given a compact 2D domain Ω and a set of primitives S, called sites, a Voronoi diagram of S is a partition of Ω into several regions, where each region V corresponds to a particular site $s \in S$ and all the points contained in V are closer to s than to any other sites. Many efficient algorithms have been proposed to compute the Voronoi diagram with point sites. A common generalization of the Voronoi diagram is the *generalized Voronoi diagram* (GVD) for higher-order sites, such as line segments and curves. Figure 1 shows an example of a generalized Voronoi diagram. The GVD has applications in many fields, such as computer animation, shape analysis, modeling spatial structures and processes, robot motion planning, pattern recognition. Therefore fast computation of the GVD is a critical problem in real-time applications.

In recent years, rapid advances have been made in computational power and programmable capability of GPU. This has led to active research studies in computing Voronoi diagrams on GPU to gain a huge speedup.

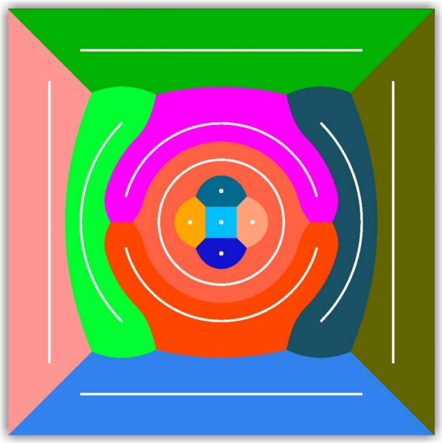
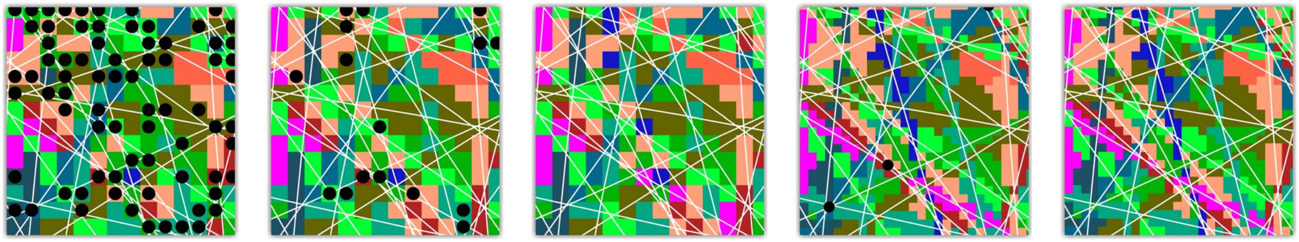


Figure 1. A generalized Voronoi diagram of five points, four line segments, four circular arcs and one circle.

The jump flooding algorithm (JFA) [1] is an efficient GPU-based algorithm for Voronoi diagram computation. It is the first known GPU implementation of Delaunay triangulation for 2D domains [2]. In this method, each site is labeled with an integer, called the ID of the site. The 2D domain Ω is discretized and represented by a 2D texture where each pixel stores the coordinates and ID of its nearest site. The JFA first maps the sites to the corresponding pixels and then propagates the site information stored in the pixels to other pixels around it. The propagation is performed in parallel on GPU. Its running time is independent of the number of the sites and only related to the resolution of the output image. Although the JFA is primarily designed for computing the ordinary Voronoi diagram of point sites, it can also be used to generate generalized Voronoi diagram by approximating the generalized sites with sampled points [1]. Several variants of the JFA are introduced in [3] that address the aspects of improving the accuracy, increasing the speed of the algorithm, and computing 3D Voronoi diagram in a slice-by-slice manner. It can also be used to compute the centroidal Voronoi tessellations on 2D planes and on surfaces [4].



(a) JFA (b) New Method (c) Ground Truth (d) New Method (e) Ground Truth

Figure 2. The zoom-in view of GVD results of 10,000 circle sites. (a)JFA; (b)(d)New method; (c)(e)Ground truth for reference. The sites are shown in white and the erroneous pixels are marked by black dots. The resolution of the texture is 4096×4096 for (a) (b) and (c), and 8192×8192 for (d) and (e).

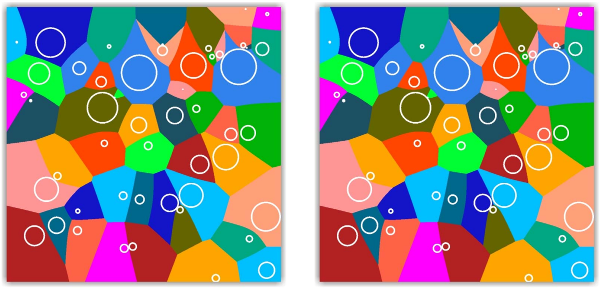
However, there are notable problems with the JFA. First, to ensure reasonable accuracy, the number of required pixels has to be much larger than the number of sites. If the information about a site (such as the coordinates of a point, or the endpoints and center of a circular arc) is directly stored in each pixel, there will be many redundant data in the texture, thus wasting the video memory.

Second, while the JFA is suitable for computing the ordinary Voronoi diagrams of point sites, it has problems with Voronoi diagrams for generalized sites. Consider two sites, each being a circular arc, intersect each other over a pixel. Since each arc is represented by sample points, the pixel can only store the information of a sample point from one arc and ignore that for the other. This can be a source of severe inaccuracy, especially when there are a large number of generalized sites. Figure 2(a) shows a zoom-in view of the GVD result generated by the JFA with many erroneous pixels within a very small area, due to this defect.

Our new method improves the jump flooding algorithm in both aspects above. As a consequence, we have a significant saving of video memory and still produce much more accurate results even for a large number of generalized sites. This is achieved with only a moderate increase of running time.

The key idea is very simple – only the IDs of the sites are stored in the 2D texture and the detailed information of the sites are stored in another 1D texture. This strategy leads to several significant improvements. We will focus on the following two improvements.

- **Accuracy Improvement.** We use IDs to access the detailed information of a site and compute the accurate site-pixel distance for determining the nearest site of each pixel. This leads to results with much higher accuracy. Moreover, our method can handle many special cases that significantly influence the accuracy. We discuss these cases in Section V-B.
- **Memory Reduction.** By propagating in the JFA only the IDs instead of all the information of the sites, we reduce the memory requirement by about 5/6. This makes it possible to use larger textures that otherwise cannot be handled by the JFA. The use of a larger texture also helps improve accuracy when dealing with a large number of sites.



(a) New Method (b) JFA

Figure 3. The GVD results of 50 circles. (a) is the result of our algorithm (4MB memory, 21.53ms, no erroneous pixel); (b) is the result of the original JFA (24MB memory, 21.46ms, 180 erroneous pixels). The texture size is 1024×1024.

Hence, our new method greatly reduces the error rate (ratio between the number of erroneous pixels and the number of total pixels) in the GVD result. For example, for a GVD of 1,000 randomly generated circular arcs in a texture of 2048×2048, the error rate of the original JFA is about 1.74% while the error rate of our method is only around 0.00196%. Furthermore, the memory reduction makes it possible to use larger textures. The JFA can only use a maximum texture of 4096×4096 on the state-of-the-art GPU, and the error rate in a GVD of 10,000 circle sites is about 30%. As a comparison, our algorithm can use a texture of 8192×8192 on the same GPU to achieve an error rate lower than 0.6%. Figure 2 compares the results of the two algorithms in these cases. The GVD is zoomed in 300 times for better visualizing the erroneous pixels.

Figure 3 shows another example of the GVD results of 50 circle sites generated by both algorithms in a texture of 1024×1024. The original JFA needs 24MB memory and costs 21.46ms, while our method uses only 4MB memory and takes 21.53ms. Furthermore, there is no error pixel in our result but 180 erroneous pixels in the result of the JFA.

II. Related Work

Existing GPU-based algorithms for computing the Voronoi diagram can be classified into two categories – linear time algorithms and constant time algorithms. The running time of the former is proportional to the number and complexity of the sites, while the latter runs in constant time and is thus independent of the sites. We will briefly review some representative algorithms of the two categories in this section.

A. Linear Time Algorithms

Hoff et al.'s cone algorithm [5] is one of the earliest algorithms which utilize GPU to compute the Voronoi diagram. It represents by 3D geometries the shapes of the distance functions measuring the distance of a point from a site. Specifically, the distance function of a point site is represented by a cone whose apex is at the site. And the distance function of a line segment site is represented by two planes extending away from the line at 45 degrees, and two semi-cones at the endpoints. For more complex sites such as curves, it approximates them by sampled points and line segments. The Voronoi diagram is generated by orthogonally rendering these geometries. The number of required geometries is proportional to the number of sampled points and line segments. Each cone is approximated by triangles. In order to guarantee the accuracy of the result, the number of triangles used to approximate each cone should be proportional to the resolution of the output image. So a large number of triangles need to be rendered when many complex sites are used.

A similar algorithm is proposed by Denny [6] where the cones in the above algorithm are replaced by quads with pre-computed depth textures. This eliminates the possible errors introduced by the approximated cones. The number of quads in this algorithm is also same as the number of sites. Fischer and Gotsman [7] introduce another way to avoid the approximated cones. They lift all the point sites to a paraboloid, and build a tangent plane for every lifted point. The planes are then rendered orthogonally to get the Voronoi diagram of the sites. This method uses no approximated geometries, but the number of planes is same as the number of sites. Both of these two algorithms are designed for point sites, and they need to approximate higher-order sites with sampled points to compute a generalized Voronoi diagram.

Hsieh and Tai [8] use a similar idea to compute the 3D Voronoi diagram in a slice-by-slice manner. But their method performs the distance computation in running time, and thus is too slow for large number of sites.

B. Constant Time Algorithms

There are two widely used constant time algorithms on CPU to compute the Voronoi diagram – Chamfer distance transform [9] and sequential Euclidean distance mapping [10]. Both algorithms use a pre-defined mask to sequentially traverse the pixels in a texture – first from left to right and top to bottom, then from right to left and bottom to top – and update the distance information for all pixels covered by the mask. Since every pixel is processed exactly twice, the running time of these algorithms is only dependent on the resolution of the texture, and is independent of the number of sites. Weber et al.[11] noticed that such traverses can be performed in parallel for some pixels, and migrated this method from CPU to GPU. By rotating the scanning paths by 45 degrees, the parallelism is greatly increased and thus fits the GPU structure better. A similar algorithm is also proposed by Schneider et al.[12].

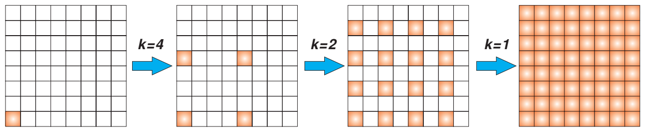


Figure 4. The flooding rounds of the JFA for an 8×8 texture with an initial site at the bottom left corner.

The jump flooding algorithm (JFA) [1] is another efficient constant time algorithm. It propagates the site information outwards using various step lengths in $\log n$ passes, where n is the dimension of the texture. The step length starts from $n/2$ and is halved in every subsequence pass until 1. Cuntz and Kolb [13] gave a modification of JFA to extend it to 3D space but with more errors. Recently, Cao et al.[14] presented the parallel banding algorithm (PBA) which is also a constant time algorithm but is faster than JFA and has no errors. However, it relies on some properties of point sites and it is not clear whether the generalized sites have similar properties.

All the above algorithms deal with point sites only, and approximate the higher-order sites as sampled points for generalized Voronoi diagram. As discussed before, this may lead to very high error rates in GVD results due to the overlaps of generalized sites. We propose in this paper a modification of the JFA to directly compute the distance between a pixel and a higher-order site, and thus achieve results with higher accuracy while using much less memory.

III. Jump Flooding Algorithm

The jump flooding algorithm (JFA) computes a Voronoi diagram in a discrete 2D space represented by a 2D texture. It propagates the information (2D coordinates and ID) of all sites to all the pixels in the texture. In the first stage, called mapping stage, each site is mapped to its corresponding pixel in the texture. It can be done in parallel by the GPU rasterizer. The second stage, flooding stage, consists of $\log n$ rounds of flooding, where n is the dimension of the texture. All pixels are processed in parallel by the GPU. In each round, the site information stored in each pixel $p(x, y)$ is propagated to at most eight other pixels at $(x + i, y + j)$ where $i, j \in \{-k, 0, k\}$, and k is the step length of the

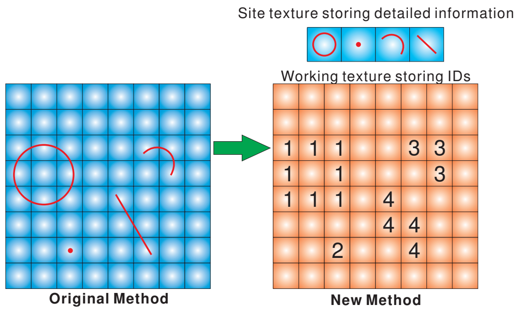

Site texture storing detailed information
Working texture storing IDs
1 1 1 3 3
1 1 3
1 1 1 4
4 4
2 4
Original Method
New Method

Figure 5. Storing IDs and coordinates separately.

current round. In the first round, we use $n/2$ as the initial step length to ensure that each pixel is reached by at least one site. Here we assume n is a power of 2. If not, the initial step length is set as $2^{\lceil \log n \rceil - 1}$. The step length k is halved in each of the following round. When more than one sites are propagated into the same pixel, the nearest site to the pixel will be chosen to update it. After the round with the step length of 1, the result is the computed Voronoi diagram. Figure 4 illustrates flooding rounds of the JFA for one point site in an 8×8 texture.

The Voronoi diagram generated by the JFA may contain some errors. Although the error rate is quite low, it can be further lowered by simply adding one additional flooding round with step length of 1 at the very beginning of the flooding stage. This variant is called 1+JFA and it generates almost no errors in most real applications [3]. In this paper, we use 1+JFA for all experiments and will refer it as JFA too.

The JFA can also be used to generate the GVD by approximating the higher-order sites by sampled points, each of which shares the same ID of the original site. Each sample point is treated as a point site and the JFA is used to compute the Voronoi diagram of them. The ID information in the final result helps to merge the Voronoi cells of the same generalized site into one Voronoi cell to form the GVD. The approximation step can be done efficiently on GPU by rendering the generalized sites into the texture.

IV. Improved Jump Flooding Algorithm

In this section, we explain the strategy we used to improve the jump flooding algorithm and the pros and cons of this method. For the ease of understanding, we explain the main idea using ordinary Voronoi diagrams of point sites in this section and extend it to generalized Voronoi diagram in Section V.

The jump flooding algorithm propagates all the attributes of the sites, including coordinates and IDs, together in a 2D texture. Since the number of pixels in the texture is usually much more than the number of the sites, there are a lot of redundant information being stored and taking part in the propagation, thus wasting memory. We improve the jump flooding algorithm by storing the detailed information and IDs separately (see Figure 5). The detailed information such as coordinates is stored in a 1D texture, where the number of pixels is same as the number of sites. We call this 1D texture the *site texture*, and the 2D texture in which the JFA is performed the *working texture*. We only propagate in the working texture the IDs of the sites indicating where they are saved in the site texture.

Using this strategy, the detailed information of each site, which takes more memory space than an ID, is stored only once, resulting in great savings in memory. For example, suppose that we use a working texture of 8192×8192 to generate a 2D Voronoi diagram of 1K point sites. Ping-pong textures are used for propagation in both algorithms. In the original JFA, if we store the coordinates and IDs in RGB channels of a float texture, the memory requirement is 1.5GB, which cannot be handled by a single state-of-the-art GPU. But our method only needs 256MB for the working texture, where each pixel uses two bytes to store an ID, and 8KB for the site texture, which can be processed by most current GPUs easily. In this example, our method reduces the memory requirement by about 5/6. This ratio depends on the data types representing the ID and coordinates and is independent of the texture size. By reducing the memory requirement, this simple strategy makes it possible to use a larger texture to increase the accuracy of the Voronoi diagram. Hence, with larger texture, we can deal with more sites for a given error bound.

Now we discuss the efficiency issues of the new algorithm. The efficiency of the flooding stage is affected by two major factors. The first one is that extra texture fetching operations are needed for accessing the information of a site from its ID. In the original JFA, each thread for a pixel p needs to do 9 texture fetching operations to get the coordinates of the sites stored in p and 8 other pixels around it. After choosing the nearest site s from the (at most) 9 different sites, we need to write the coordinates and the ID of s to pixel p in the result texture. There are 9 texture fetching operations and 1 writing operations in total. In our method, each thread needs to do 18 texture fetching operations and 1 writing operation for every pixel. The extra fetching operations will slow down the program. Besides, when the texture fetching operation is applied to the 2D working texture, the memory accesses can be well coalesced. However, the IDs fetched from adjacent pixels may be scattered throughout the 1D site texture. In consequence, the memory accesses cannot be coalesced. It will affect the performance by an amount related to the number of sites, because if there are more sites, the IDs fetched from the adjacent pixels tends to be more scattered throughout the site texture and the probability of required site information being pre-fetched tends to be lower, which means that the fetching operations are more likely to take longer time.

The second factor is due to the reduction of data that need

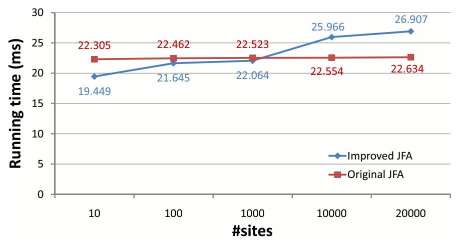


Figure 6. The running time of the flooding stage for the two methods using the texture of 1024×1024 and different number of point sites.

to be written into a texture. In the original JFA, we need to write both the coordinates and the ID of a site to each pixel. But in our new method we only need to update the ID stored in each pixel. The data required to be updated in our method is only one sixth of the original algorithm, thus leading to speedup.

To investigate how the total running time is affected by the combination of these two factors, we did experiments on NVIDIA Quadro FX 4800 using a texture of 1024×1024 to compare the speed of the original JFA and the new method. The result is shown in Figure 6.

The experiments show that, when the number of sites is below 1,000, our method is a little faster than the original JFA. When the number of sites is larger than 1,000, our method is a little slower than the original JFA. Hence, the running times of the two methods are comparable in this setting.

V. Computing Generalized Voronoi Diagram

It is straightforward to apply our new method to computing generalized Voronoi diagram, since we can represent any higher-order site with an ID indicating the position where the real detailed information being stored. In this section we will outline the main steps of our method.

A. Implementation Details

The method to extend our algorithm to GVD computation is as follows. In our experiments, we use four types of sites: points, line segments, circles, and circular arcs, which are widely used in applications. We store them in the site texture with four float channels. They are organized as follows where RGBA represent the four channels sequentially.

- **Point.** R and G channels are used to store its x and y coordinates respectively.
- **Line segment.** The four channels are used to store the coordinates of its two end points.
- **Circle.** The coordinates of the center are saved in R and G channels and the radius is saved in the B channel.
- **Circular arc.** We use the R and G channels to store the center of the arc, B channel to store the radius. Moreover, we use short integers to represent the two angles between the x-axis and the lines connecting the end points and the center of the arc. The two short integers are packed into the A channel.

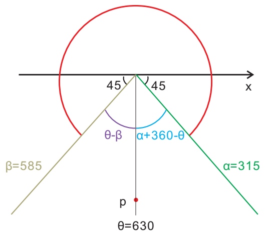


Figure 7. An illustration of the pixel-arc distance computation.

The sites of the same type are stored together as a group in the site texture. And we use three integers to record the index of the last site in each group so that we can determine the type of a site by simply checking in which interval of addresses its index falls. This avoids storing the type of each site and using texture fetching operations to get them. In the mapping stage, we draw the sites once to map their IDs to corresponding pixels in the working texture. Then we run our algorithm for the ordinary Voronoi diagram. We implement a distance computation method for each type of sites and use their indices to choose their corresponding method. The distance between a pixel and point, line segment and circle is easy to compute. But for the circular arcs, we need the following scheme to make the computation more efficient.

Our main concern is to reduce the execution paths of the program, since the execution branch has a significant impact on the efficiency of the program running on the GPU. Before storing a circular arc, we first preprocess it such that its two angles α and β satisfy that the arc starts from α to β along the counterclockwise direction and $0 \leq \alpha \leq 360$, $\alpha \leq \beta \leq 720$. This preprocessing step involves many execution paths and is performed efficiently on the CPU. Then the algorithm to compute the arc-pixel distance on GPU becomes very simple. We get the angle θ between x-axis and the line connecting the arc center and the pixel and transform it to $[\alpha, \alpha + 360]$. If θ is smaller than β, it means that $\theta \in [\alpha, \beta]$ and the pixel is nearer to the arc than to the end points. Otherwise, we can determine which endpoint is nearer to the pixel by comparing $\theta - \beta$ and $\alpha + 360 - \theta$ as shown in Figure 7.

B. Accuracy Improvement

In the original JFA, each sampled point carries a unique part of information of the higher-order site and is mapped to only one corresponding pixel in the mapping stage. If the information stored in this pixel is lost, we cannot recover it in the following flooding stage. But in our method, each ID carries the whole information about a site and is mapped to multiple pixels by drawing the site in the mapping step. Even if we lose some of pixels, we can recover the information from other pixels storing the same ID. There are three major

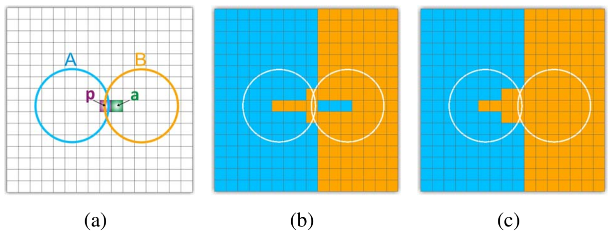


Figure 8. An example of missing pixels due to more than one sites being mapped to a common pixel. (b) is the result of our method; (c) is the result of the original JFA.

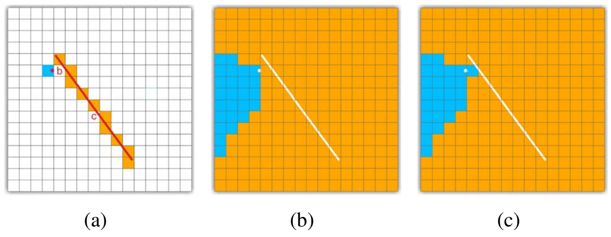


Figure 9. An example of missing pixels due to rasterization. (b) is the result of our method; (c) is the result of the original JFA.

factors which may lead to missing pixels (pixels that are lost during the mapping stage).

First, it occurs when more than one sites are mapped to a same pixel. For example, as shown in Figure 8(a), two circle sites A and B are both mapped to the pixel p. Suppose A was mapped first, then B will overwrite the information of A stored in p. In this case, the original JFA lose the sampled point P_A of site A which should be stored in p. So the pixels which are nearest to P_A may be assigned to other Voronoi regions in the JFA result. For example, the pixel a in Figure 8(a) is nearer to A but will be assigned to B by the JFA. In contrast, our method can still get the correct result in this case, since all the pixels with the ID of A have the information of the whole circle, and a can get this information from pixels other than p. Figure 8(b) and Figure 8(c) show the results of the two methods in this case. To better visualize the missing pixels, we use a texture of 16×16.

Second, the method we used to choose the sample points in the mapping step may also lead to missing information. In our implementation we simply draw the sites once and use the built-in GPU rasterizer to get the sample points. As shown in Figure 9(a), some pixels such as b and c in the figure which have a small intersection area with the site will not be chosen by the rasterizer. Figure 9(b) and Figure 9(c) show the results of the two methods in this case using a texture of 16×16. In the example, one pixel is misclassified in the result of the original JFA.

Third, when a part of a site is out of the texture, the original JFA will lose the information of this part, and the pixels which are nearest to that part may be assigned to other sites. Our method can handle this case very well. Figure 10 demonstrates an example of this case where the pixels a, b and c are assigned to the wrong site by the original JFA.

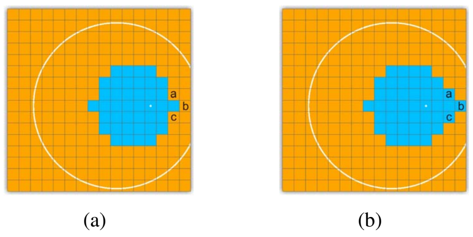


Figure 10. An example of missing pixels due to a part of the site being out of the texture. (a) is the result of our method; (b) is the result of the original JFA.

As discussed above, our method outperforms the original JFA by using less memory and generating more accurate result, while it is slower than the original JFA by a factor of 1.2 to 2.0. Clearly, more complex sites will take longer time. We believe in many applications this moderate increase of the running time is acceptable, given the advantages the new method brings about.

VI. Experimental Results

We implemented the original jump flooding algorithm and our new algorithm using Microsoft Visual C++ and GLSL (OpenGL Shading Language). The hardware platform is Intel Xeon X5460 3.16GHz, 8G DDR3 RAM and NVIDIA Quadro FX 4800 with 1G available DDR3 video memory.

In our experiments, we have four kinds of sites: points, line segments, circles and circular arcs. The following experiments address on different aspects of the algorithm.

A. *Accuracy*

We compute an accurate GVD and calculate the error rates in the results of the two algorithms to compare their accuracy. To generate the accurate GVD, for each pixel p, we compute the distance between p and all the sites and store in p its nearest site. Considering the probability of missing pixels, which we discussed in Section V, depends on the type of sites, we test each type of sites separately. Since the third factor leading to missing pixels causes too many errors of the original JFA, we exclude its effect by restricting all the sites in the range of texture. Figure 11 shows the experimental results for line segment sites, circle sites, and circular arc sites, and it is obvious that there is no difference between the two algorithms for point sites. In each experiment, 1,000 sites are used. The experimental results show that the result of our method is much more accurate than that of the original JFA.

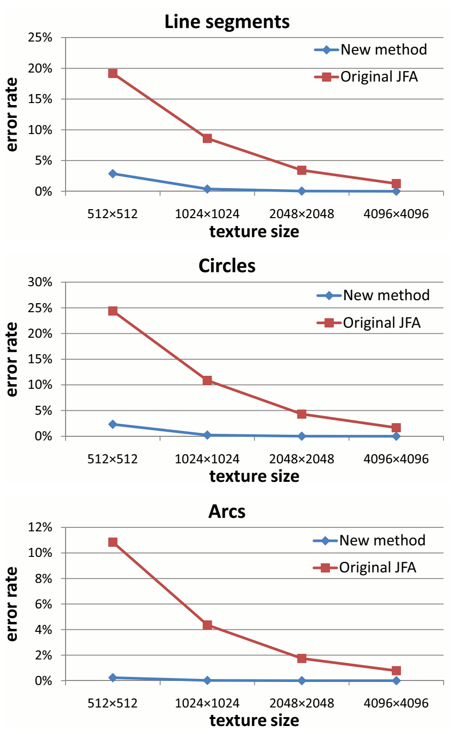


Figure 11. The error rates in the GVDs of 1,000 sites generated by the two methods.

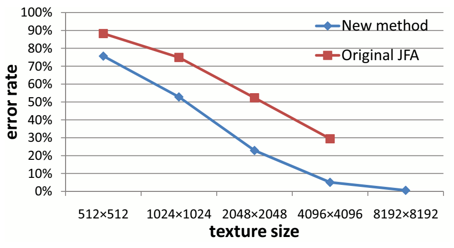


Figure 12. The error rates in the GVDs for 10,000 circles generated by the two methods.

B. *Memory Requirement*

As mentioned in Section IV, the memory requirement of our method is around one sixth of the original JFA's requirement. We did an experiment to show that our method can reach a higher accuracy by using larger texture size which cannot be handled by the original JFA. The maximum texture size supported by our hardware platform is 8192×8192. If this texture size is used, the original JFA requires about 1.5GB video memory while our method only needs about 256MB video memory. In this case, the original JFA cannot run on our hardware platform, but our method doesn't have such a problem. Figure 12 shows the error rates in the GVDs for 10,000 circles generated by the two methods. Since the circles are generated randomly, many of them overlap with each other. Therefore, the error rate is very high when a small texture is used. The result shows that the error rate in the result of our method with the texture of 8192×8192 is smaller than 0.6% which is accurate enough for most applications while the error rate in the result of the original JFA with the texture of 4096×4096 is about 30% which is unacceptable.

C. *Running time*

In this section we compare the running time of the two algorithms. If we use the same sites and texture size, the speed difference between the two algorithms depends on two major factors: the different texture accessing operations and different distance computation routines. We first illustrate the effect of the first factor. In this experiment, we only use point sites, and thus the distance computation is the same in the two algorithms. We generate the point sites randomly and get the average time of each algorithm for 100 running times. The result is shown in Figure 6. It shows that when a small number of sites are used, our method is faster than the original JFA, and as the number of sites increases, our method is slower than the original JFA.

In the next experiment, we use different kinds of sites to observe the overall effect of the two factors. All the sites are generated randomly. Figure 13 shows the results of the different situations. For the experiment with mixed sites, the running time seems to be proportional to the number of sites. That is because the running time of the distance computation depends on whether the adjacent pixels deal with the same type of sites. If they process different types of sites, they will take different branch of the program and the parallelism will be affected. And the probability of adjacent pixels processing different types of sites is related to the number of sites. The more the sites, the higher the probability. In these experimental results, the difference between our algorithm and the original JFA is bigger than the point sites case, but our algorithm is still fast enough for most real-time applications, and it uses much less memory and generates more accurate GVD results.

VII. Conclusion and Future Work

This paper improves the jump flooding algorithm (JFA) to compute Voronoi diagrams for generalized sites, such as line segment and circles. Compared with the original JFA, our method is more accurate due to using the accurate site-pixel distance and the good property that it is not sensitive to missing pixels. Furthermore, it reduces the memory requirement by around 5/6. Therefore it can reach a higher accuracy by using larger texture which cannot be handled by the original JFA. Although our method is slower than the original JFA when dealing with complex higher-order sites, due to additional texture fetching operations and more complicated distance computation, it is still fast enough for most real-time applications.

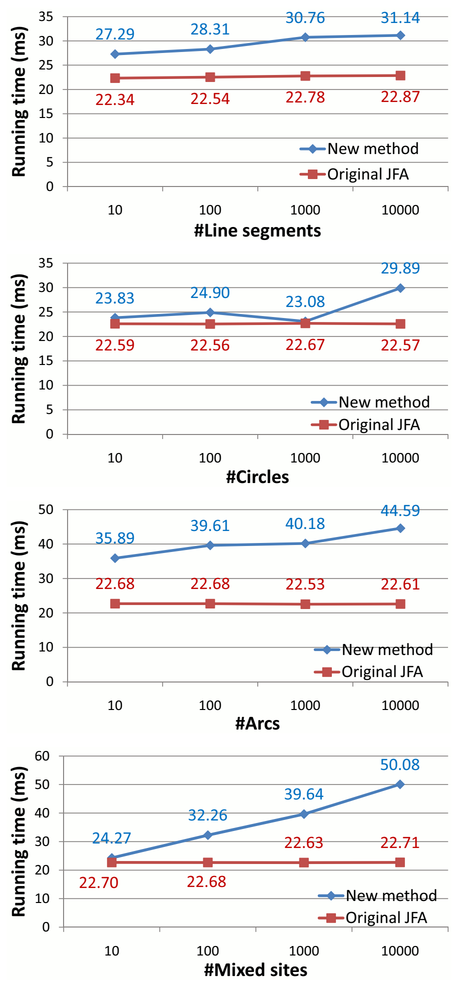


Figure 13. The running time of the flooding stage using the texture size of 1024×1024 and different kinds of sites.

One possible future research problem is to extend the work in this paper to the 3D case where the memory requirement is more demanding than the 2D case. So the memory limitation will easily become a critical bottleneck for the application of the original JFA. Our method can be extended to 3D and would reduce the memory requirement by around 6/7. This would make it possible to use larger texture to reach a higher accuracy or deal with more sites.

Acknowledgment

The authors would like to thank the anonymous reviewers for their constructive comments. Xiaohu Guo are partially supported by the US National Science Foundation (NSF) under Grant No. CCF-0727098. Wenping Wang is partially supported by the General Research Funds (718010, 718209, 717808) of Research Grant Council of Hong Kong, NSFC-Microsoft Research Asia co-funded project (60933008), and National 863 High-Tech Program of China (2009AA01Z304).

References

[1] G. Rong and T.-S. Tan, "Jump flooding in GPU with applications to Voronoi diagram and distance transform," in *Proceedings of the Symposium on Interactive 3D Graphics and Games*. ACM Press, 2006, pp. 109–116.

[2] G. Rong, T.-S. Tan, T.-T. Cao, and Stephanus, "Computing two-dimensional Delaunay triangulation using graphics hardware," in *Proceedings of the Symposium on Interactive 3D Graphics and Games*. ACM Press, 2008, pp. 89–97.

[3] G. Rong and T.-S. Tan, "Variants of jump flooding algorithm for computing discrete Voronoi diagrams," in *Proceedings of the 4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD'07)*, 2007, pp. 176–181.

[4] G. Rong, Y. Liu, W. Wang, X. Yin, X. Gu, and X. Guo, "Gpu-assisted computation of centroidal voronoi tessellation," *IEEE Transactions on Visualization and Computer Graphics*, pp. 345–356, 2010.

[5] K. E. Hoff III, T. Culver, J. Keyser, M. Lin, and D. Manocha, "Fast computation of generalized Voronoi diagrams using graphics hardware," in *SIGGRAPH '99*, 1999, pp. 277–286.

[6] M. O. Denny, "Algorithmic geometry via graphics hardware," Ph.D. dissertation, Universität des Saarlandes, 2003.

[7] I. Fischer and C. Gotsman, "Fast approximation of high order Voronoi diagrams and distance transforms on the GPU," *Journal of Graphics Tools*, vol. 11, no. 4, pp. 39–60, 2006.

[8] H. Hsieh and W. Tai, "A simple GPU-based approach for 3D Voronoi diagram construction and visualization," *Simulation Modelling Practice and Theory*, vol. 13, no. 8, pp. 681–692, 2005.

[9] G. Borgefors, "Distance transformations in arbitrary dimensions," *Computer Vision, Graphics, and Image Processing*, vol. 27, pp. 321–345, 1984.

[10] P.-E. Danielsson, "Euclidean distance mapping," *Computer Graphics and Image Processing*, vol. 14, pp. 227–248, 1980.

[11] O. Weber, Y. S. Devir, A. M. Bronstein, M. M. Bronstein, and R. Kimmel, "Parallel algorithms for approximation of distance maps on parametric surfaces," *ACM Transactions on Graphics*, vol. 27, no. 4, pp. 1–16, 2008.

[12] J. Schneider, M. Kraus, and R. Westermann, "GPU-based real-time discrete Euclidean distance transforms with precise error-bounds," in *International Conference on Computer Vision Theory and Applications (VISAPP)*, 2009, pp. 435–442.

[13] N. Cuntz and A. Kolb, "Fast hierarchical 3D distance transforms on the gpu," Institute for Vision and Graphics, Universität Siegen, Technical report, 2006.

[14] T.-T. Cao, K. Tang, A. Mohamed, and T.-S. Tan, "Parallel banding algorithm to compute exact distance transform with the GPU," in *Proceedings of the Symposium on Interactive 3D Graphics and Games*. ACM Press, 2010, to appear.