Facet-JFA: Faster computation of discrete Voronoi diagrams

Talha Bin Masood [a,*] Hari Krishna Malladi [a] Vijay Natarajan [a,b]

a Dept. of Computer Science and Automation, Indian Institute of Science, Bangalore, India
b Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore, India
Email: {tbmasood, harikrishnamalladi, vijayn}@csa.iisc.ernet.in

ABSTRACT

Jump Flooding is a method for propagating labels across a given plane from different seeds. It has been used to compute the discrete Voronoi tessellation of a given plane efficiently. We introduce a version of JFA, which optimizes the number of pixels processed by computing only the faces of the Voronoi tessellation. The pixels in the interior of the Voronoi regions are not processed resulting in a 1-skeleton representation of the Voronoi tessellation in 2D and a 2-skeleton representation in 3D. We describe an implementation of this algorithm on a GPU using CUDA and demonstrate its performance benefits on multiple data sets. As an application of the proposed algorithm, we present a GPU based method for extraction of channel centerlines in biomolecules. The fast computation of the discrete Voronoi diagram is exploited to extract channels in molecular dynamics simulation trajectories on-the-fly, thereby supporting the interactive visual analysis of static and dynamic channel structures.

Keywords

Discrete Voronoi diagram, Jump Flooding Algorithm, GPU acceleration, Bio-molecular visualization.

1. INTRODUCTION

Voronoi tessellation is one of the most widely used tools in computational geometry, with applications in computer graphics, image processing, mesh processing, robot navigation, and for data analysis in several scientific and engineering disciplines. A Voronoi tessellation partitions space into regions given a set of seed points, with each point in a particular region being closer to its corresponding seed than to any other seed. The computation of Voronoi tessellations is one of the best studied problems in computational geometry, with optimal algorithms known for computing Voronoi tessellations on the plane [1, 2].

Discrete Voronoi tessellation requires the computation of regions on a discrete grid of pixels, with seed points being pixels themselves. Due to the inherent parallelism, this problem has been approached using the GPU. The first attempts to solve this problem using the GPU were done due to Hoff *et al.* [5]. Guodong *et al.* [13] then used Jump Flooding, a parallel extension to flood-fill algorithm, which propagates label information from a pixel to the entire grid, to construct approximate Voronoi tessellation. The GPU is harnessed to enable all the labelled pixels to propagate their label information, instead of a wave-front like approach. It can be easily shown that this approach requires $\log n$ steps to flood an $n \times n$ grid of pixels. Jump Flooding can be used to generate discrete Voronoi tessellations and the resulting algorithm is referred to as JFA (Jump Flooding Algorithm). Assuming fixed grid of size $n \times n$ with seed points already placed in the grid, JFA computes the Voronoi tessellation in $\log n$ steps, and is thus independent of the number of seeds. The discrete nature of the problem introduces errors, which have been extensively documented in [13]. The algorithm has been implemented by Guodong *et al.* using textures and pixel shaders and is one of the most efficient methods for computing the discrete Voronoi tessellation till date.

This paper introduces a variant of JFA, called Facet-JFA, wherein only the pixels which are located near the Voronoi region boundaries are processed, thus immensely reducing the total amount of work done by the algorithm. The proposed approach first determines the lowest grid resolution at which seed points can be projected to the nearest grid pixel without conflict. Following this, the algorithm marks the grid pixels which are destined to lie in the interior of the Voronoi regions and refines the boundaries of these regions. This algorithm uses an intrinsic quadtree-based approach. Like JFA, the proposed approach also requires $\log n$ steps to compute the Voronoi diagram for an $n \times n$ grid of pixels. But, for larger grids with fewer seed points (and hence large Voronoi regions), almost all the pixels will be marked as interior and hence will not be processed. This strategy enables both space optimization and better running times in practice. We explore the speed-ups obtained over JFA for different grid and seed set sizes. We implement the original JFA and Facet-JFA using CUDA to compute Voronoi tessellations in two and three dimensions. We report experimental results concerning the running time and

*Corresponding author

ICVGIP '14, December 14-18, 2014, Bangalore, India

ACM 978-1-4503-3061-9/14/12 ...$15.00.
http://dx.doi.org/10.1145/2683483.2683503

other parameters across multiple GPU architectures.

As an application of FACET-JFA, we present a GPU accelerated technique for extraction of the channel network in biomolecules in two and three dimensions. The proposed method allows extraction of channels at real-time interactive rates and is thus suited for visual analysis of static and dynamic channel structures in Molecular Dynamics (MD) simulation trajectories. With examples, we demonstrate that the discrete representation and the use of FACET-JFA is appropriate for the typical resolutions required for visualizing the channels in MD trajectories at interactive rates.

2. DISCRETE VORONOI DIAGRAM COMPUTATION

2.1 Definitions

Definition 1. Let $[n]$ be the set $\{0, 1, \ldots, n-1\}$. A *grid* is defined as the Cartesian product $[n]^d$ of the set $[n]$. Here, d is the *dimension* of the grid and n is the *size* of the grid. A d-dimensional grid of size n is denoted by $[n]^d$. Any element $p \in [n]^d$ is a d-dimensional vector $(x_1, x_2, \ldots, x_d)$ and called a *pixel*.

Definition 2. Given a grid $[n]^d$ with a distance metric δ defined on it and a set of k seeds $S = \{s_1, s_2, \ldots, s_k\} \subseteq [n]^d$, the discrete Voronoi diagram is a function f defined as follows:

$$f : [n]^d \to S$$
$$\text{such that} f(p) = s_i \iff \forall j \neq i, \quad \delta(p, s_i) \leqslant \delta(p, s_j)$$

To make the above definition well defined, we will always assign lowest indexed seed to the pixel p whenever p is equidistant to multiple seeds. For any seed $s_i \in S$, $f^{-1}(s_i)$ is called the discrete Voronoi *region* of s_i.

Definition 3. Let $\Delta_1 = \{-1, 0, 1\}$. The *neighborhood* of a pixel $p \in [n]^d$ is the set of pixels N_p, defined as follows:

$$N'_p = \{p + \delta_1 \quad \text{such that} \quad \delta_1 \in \Delta_1^d\}$$
$$N_p = N'_p \cap [n]^d \setminus \{p\}$$

There are at most $3^d - 1$ pixels in the neighbourhood of a pixel $p \in [n]^d$. If the pixel lies on the boundary of the grid then size of neighbourhood sets is smaller than $3^d - 1$.

Definition 4. A *cell* of size l at a pixel $p \in [n]^d$ is a smaller grid of size l at origin p. The set of pixels in the cell of size l at pixel p, denoted by $C(p, l)$, is given by:

$$C(p, l) = \{p + \delta \text{ such that } \delta \in [l]^d\}$$

In 2D, a cell is a pixel in a grid of resolution $(n/l) \times (n/l)$. That is, every pixel in an $(n/l) \times (n/l)$ grid represents a cell in an $n \times n$ grid, each containing $l \times l$ pixels.

Definition 5. The *refinement* of a cell $C(p, l)$, denoted by $R(C(p, l))$, is the partition into 2^d equal sized cells. It is defined as follows:

$$\Delta_{\frac{l}{2}} = \{0, \frac{l}{2}\}$$
$$R(C(p, l)) = \{C(p + \delta_{\frac{l}{2}}, \frac{l}{2}) \text{ such that } \delta_{\frac{l}{2}} \in \Delta_{\frac{l}{2}}^d\}$$

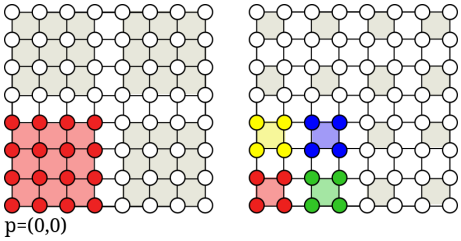


Figure 1: Left: A Cell $C(p, 4)$ in a grid $[8]^2$. Right: Its refinement into four cells of size 2.

In 2D, refinement refers to increasing the resolution of the grid form an $(n/l) \times (n/l)$ grid to an $(2n/l) \times (2n/l)$ grid.

Definition 6. For a discrete Voronoi diagram f of a grid $[n]^d$, a pixel p is called a *boundary* pixel if there exists $q \in N_p$ such that $f(p) \neq f(q)$. The other pixels in the grid are called *interior* pixels.

2.2 Jump Flooding and JFA

The idea of utilizing graphics hardware to compute geometric constructions is not new. The first attempt to compute Voronoi tessellations using graphics hardware was done by K. Hoff [5]. The algorithm described by Hoff uses projections of cones from seed points, drawing region boundaries where two cones meet. This was vastly improved in the more recent attempt by Guodong *et al.* [13], using a parallel approach to flooding, known as Jump Flooding. This algorithm is based on the observation that while flooding an area with a label, each labelled pixel can transmit its label, instead of just the ones on the boundaries of the labelled region. This ensures an exponential growth in the number of labelled pixels in a grid and thus, can be computed in $O(\log n)$ time, for an $n \times n$ grid.

The work by Guodong *et al.* [13] introduces two variants of the flooding algorithm, one with a halving step length and the other with a doubling step length. It is also shown that the former approach results in fewer errors, as compared to the latter one. The experiments in this paper use the halving approach to flood label information. The jump flooding algorithm to compute Voronoi tessellations utilize this flooding. The JFA algorithm using jump flooding in the halving step mode proceeds in the following way:

1. Initially, each pixel corresponding to a seed s records a tuple $\langle s, position(s)\rangle$ and all other pixels record $\langle nil, nil\rangle$.

2. In step l, each pixel (x, y) passes the tuple corresponding to the seed closest to it to the pixels $(x + i, y + j), i, j \in \{-l, 0, l\}$.

3. The step size halves in each iteration and the above step is repeated starting with $l = n$ till $l = 1$ when the algorithm halts.

Figure 2(a) demonstrates how JFA proceeds on a 256×256 grid. There are several variants of JFA described in [13] and [14], including JFA+1, JFA+2, JFA^2 and 1+JFA. We use JFA+1, which is JFA with one additional round of flooding for the experiments in this paper.

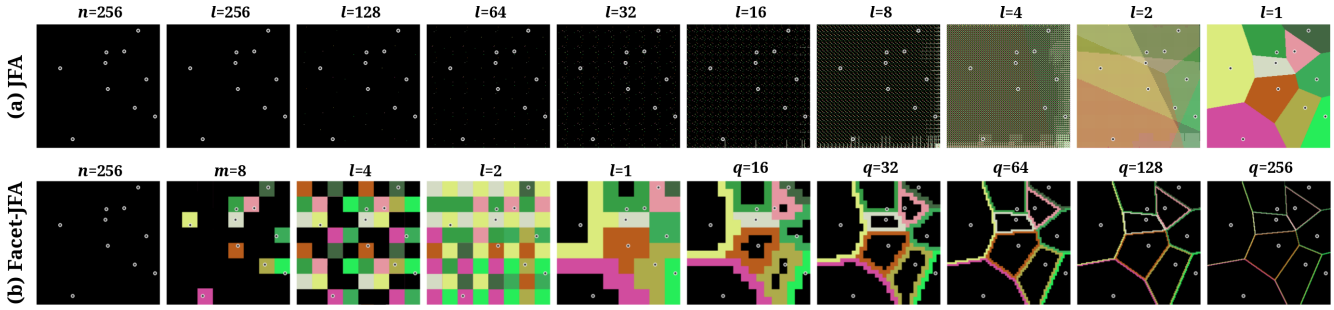


Figure 2: Comparison of execution of JFA and Facet-JFA on a 256×256 grid with 10 seed points. (a) JFA completes in 8 steps for this grid. In each step, all the pixels update their labels based on the seed closest to them. Notice that in earlier steps many pixels are coloured black as they have not received any valid seed till then. (b) Facet-JFA also takes 8 steps for completion. For this example, 8×8 grid is determined to be the initial resolution, on which JFA is executed resulting in coarse Voronoi diagram (frame with $l = 1$). In subsequent steps, the coarse boundaries are refined till the final resolution is reached. In refinement steps, the pixels coloured black are those which are marked and remain inactive, resulting in faster computation.

Algorithm 1 FACET-JFA

Input: S: Set of seeds.
Input: n: Grid size.
Input: m: Initial grid resolution (optional).
Output: M: $n \times n$ grid where pixels on the Voronoi region boundary are set to the index of the seed.

1: If m is not provided, compute the smallest value $m = 2^p, p \in \mathbb{N}$ such that for every two seeds (i_1, j_1) and (i_2, j_2), $\frac{i_1 \cdot m}{n} \neq \frac{i_2 \cdot m}{n}$ or $\frac{j_1 \cdot m}{n} \neq \frac{j_2 \cdot m}{n}$.
2: M := Create an $m \times m$ grid.
3: Unmark each pixel in the $m \times m$ grid.
4: Run JFA on the $m \times m$ grid with seeds (i, j) replaced by $(\frac{i \cdot m}{n}, \frac{j \cdot m}{n})$.
5: Initialize q to m and repeat steps 6 to 9 doubling q after each iteration and halting when $q \geq n$.
6: M := Create a $2q \times 2q$ grid.
7: Refine each unmarked pixel (i, j) in the $q \times q$ grid to 4 pixels, $(2i, 2j), (2i+1, 2j), (2i, 2j+1), (2i+1, 2j+1)$ in the $2q \times 2q$ grid.
8: With $l = 1$, run one step of JFA on the M with seeds (i, j) replaced by $(\frac{i \cdot 2q}{n}, \frac{j \cdot 2q}{n})$. Launch threads only for the unmarked pixels.
9: Mark all pixels in M which have identically labelled neighbours. Again, launch threads only for the unmarked pixels.
10: **return** M

2.3 Facet-JFA

A great amount of work is done in the original JFA in the flooding of label information by the pixels which are in the interior of Voronoi regions. Much of this flooding can be seen as extraneous as it does not affect the boundaries of the Voronoi regions being formed. The proposed algorithm aims to eliminate this overhead by computing the region boundaries alone, instead of processing every pixel in each region. The proposed algorithm is presented as Algorithm 1 for 2D case. This algorithm can easily be extended to higher dimensions as well.

Just like JFA, the proposed algorithm also takes $\log n$ steps to finish. The algorithm marks the interior pixels, thus lowering the number of threads that need to be launched. This greatly reduces the total amount of work done by the algorithm and thus the running time, as demonstrated in the experiments. This algorithm can perform as bad as JFA if initial grid resolution m is close to or same as n. Figure 2 demonstrates the steps involved in FACET-JFA and how those steps compare with the steps involved in JFA. In the first $\log m$ steps, FACET-JFA essentially executes JFA to obtain the Voronoi diagram at a lower resolution of $m \times m$. In the next $\log n - \log m$ steps, the boundary cells are identified and refined iteratively. During refinement, a new pixel acquires a label corresponding to its parent or the label of the parent's neighbour. Executing single step of JFA with $l = 1$ accomplishes this refinement.

2.4 Runtime and space analysis

JFA and FACET-JFA both take $\log n$ steps to finish. Each of these steps involve processing multiple pixels in parallel. But the advantage of using FACET-JFA is that it processes smaller girds in initial stages, while at later stages when grid size becomes larger, many of the pixels are not processed. We perform qualitative assessment of FACET-JFA in terms of the number of pixels processed by the algorithm after performing JFA on the initial coarse grid. The lower the number of these pixels, the larger would be the performance gain, both in terms of timing benefit and potential space savings. We show that the algorithm processes $O(d \cdot k^{\lceil \frac{d}{2} \rceil} \cdot n^{d-1} \cdot \log n)$ pixels in the general case, and at most $5 \cdot (3k - 6) \cdot n \log n$ pixels in the specific case of a 2D grid. This is a factor of n improvement over JFA, which processes $\log n \cdot n^2$ pixels in its complete execution. Including the number of pixels processed in the JFA execution on the initial grid, the total number of pixels processed by FACET-JFA is at most $O(d \cdot k^{\lceil \frac{d}{2} \rceil} \cdot n^{d-1} \cdot \log n + m^2 \cdot \log m)$.

2.4.1 2D grid

LEMMA 1. *The number of boundary pixels in the discrete Voronoi diagram of a two dimensional grid $[n]^2$ with seed set of size k is linear in k and n. The number of boundary pixels is at most $5 \cdot (3k - 6) \cdot n$.*

PROOF. The Voronoi diagram is the dual of the Delaunay triangulation. So, number of edges in the 2D Voronoi diagram is equal to that in the Delaunay triangulation. The number of vertices in the Delaunay triangulation is exactly

k i.e. the number of seeds. Since the Delaunay graph is planar, the number of edges is at most $(3k-6)$.

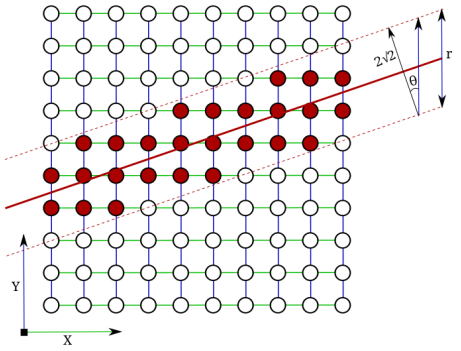


Figure 3: The set of boundary pixels (red) due to intersection of a line (red) with a $[10]^2$ grid.

Now, any line passing through $[n]^2$ can contribute at most $5n$ boundary pixels (Refer to Figure 3 for an example and Lemma 2 for proof in d dimensions). So, each Voronoi edge can contribute at most $5n$ boundary pixels. Therefore, the total number of boundary pixels in a two dimensional discrete Voronoi diagram is at most $5 \cdot (3k-6) \cdot n$. □

THEOREM 1. *Total number of pixels processed by* FACET-JFA *while computing the discrete Voronoi diagram for k seeds in a $[n]^2$ grid is bounded by $5 \cdot (3k-6) \cdot n \log n$.*

PROOF. FACET-JFA subdivides the grid in a quadtree fashion. The leaves of the quad tree are the pixels in the grid, while the internal nodes are the cells in the grid. The construction of the discrete Voronoi diagram of a grid using FACET-JFA involves reaching the boundary pixels of the Voronoi regions using this tree. By Lemma 1, we know that the number of boundary pixels in a discrete Voronoi diagram is at most $5 \cdot (3k-6) \cdot n$. In a quadtree, to reach a leaf node we need to access $\log n$ internal nodes. So, during the course of a run of FACET-JFA, the total number of pixels processed is at most $5 \cdot (3k-6) \cdot n \log n$. □

2.4.2 *d-Dimensional grid*

LEMMA 2. *The number of boundary pixels resulting because of intersection of a hyperplane in $\mathbb{R}^d$ with the grid $[n]^d$ is at most $O(d \cdot n^{d-1})$.*

PROOF. In $[n]^d$, all neighbours of a grid pixel are within a distance of $\sqrt{d}$. We consider two parallel hyperplanes at the distance of $\sqrt{d}$ from the given hyperplane. Refer to Figure 3 for an illustration. The given hyperplane is shown as bold red line while the two parallel hyperplanes are shown as dotted lines. Now, it is easy to see that boundary pixels can not lie outside the volume bounded by the two hyperplanes. So, estimating the maximum number of pixels that can lie in this volume will give an upper bound on the number of boundary pixels.

Given one of the d axes, the volume consists of at most n^{d-1} columns of pixels along this axis. We claim that for a suitable choice of axis the number of boundary pixels within each column is at most $(2d+1)$. Let r be the length of a column within the volume bounded by the two hyperplanes, see Figure 3. We choose the axis that minimizes the angle θ and maximizes $\cos\theta$. In this case, $\cos\theta \geq 1/\sqrt{d}$ where the equality holds when the hyperplane and its normal subtend the same angle with all the axes. Further, $r \cdot \cos\theta = 2\sqrt{d}$. So, the value of r is at most $2\sqrt{d} \times \sqrt{d} = 2d$. A column length of $2d$ within the volume corresponds to at most $(2d+1)$ pixels. Counting pixels within all columns results in an upper bound of $(2d+1) \cdot n^{d-1} = O(d \cdot n^{d-1})$ boundary pixels. □

For the case of $d = 2$, the number of boundary pixels due to a Voronoi edge is at most $5n$.

LEMMA 3. *The number of boundary pixels in the discrete Voronoi diagram on a grid $[n]^d$ with seed set of size k is upper bounded by $O(d \cdot k^{\lceil \frac{d}{2} \rceil} \cdot n^{d-1})$.*

PROOF. We know from Lemma 2 that intersection of any hyperplane can result in $O(d \cdot n^{d-1})$ boundary pixels in a grid. Also, it is known that there are $O(k^{\lceil \frac{d}{2} \rceil})$ Voronoi facets in a d-dimensional Voronoi diagram [4]. It follows that the total number of boundary pixels in a discrete Voronoi diagram is bounded by $O(d \cdot k^{\lceil \frac{d}{2} \rceil} \cdot n^{d-1})$. □

THEOREM 2. *Total number of pixels processed by* FACET-JFA *for computation of discrete Voronoi diagram for k seeds in a $[n]^d$ grid is $O(d \cdot k^{\lceil \frac{d}{2} \rceil} \cdot n^{d-1} \cdot \log n)$.*

PROOF. The argument given in Theorem 1 can be extended to the d dimensional case as well. By Lemma 3, we know that the number of boundary pixels in a discrete Voronoi diagram is $O(d \cdot k^{\lceil \frac{d}{2} \rceil} \cdot n^{d-1})$. In a d dimensional quadtree, to reach a leaf node we need to access $\log n$ internal nodes. Thus, the total number of pixels processed is at most $O(d \cdot k^{\lceil \frac{d}{2} \rceil} \cdot n^{d-1} \cdot \log n)$. □

2.5 CUDA Implementation

We implement jump flooding on CUDA using a gather-style approach, wherein in step l, each pixel gathers the label information from 8 neighbours l pixels away from it. It can easily be shown that this approach is equivalent to the halving mode of jump flooding. This approach is free from write-conflicts and the GPU based approach greatly benefits from this fact, as GPUs are inherently massively parallel. We used this gather-style approach to jump flooding to implement JFA and FACET-JFA.

In the implementation of JFA, the grid is copied only once to the device memory. A total of $\log n$ calls are made to the kernel, each call corresponding to a step of the flooding algorithm. The implementation of FACET-JFA involves multiple kernels as the algorithm consists of several phases. The computation of m in the first step of FACET-JFA is done on the CPU and is considered as a preprocessing step (a short discussion on this step is included in the supplement). In practice, m can be directly provided by the user, determined based on a user-defined error tolerance, or it can be domain specific and based on closest possible pair. Once m has been determined, FACET-JFA first computes Voronoi diagram at the coarsest resolution of $m \times m$ using JFA. Then the algorithm proceeds by repeatedly launching the cell refinement, JFA and marking steps in succession till the final resolution $n \times n$ is reached. At an intermediate grid resolution of $q \times q$, q^2 threads are launched, which return immediately after launching if the pixel is marked. Otherwise, they update the Voronoi region using one step ($l = 1$) of JFA on that grid. The cell refinements to obtain the $2q \times 2q$ grid and the marking of interior pixels is also done using

CUDA kernels. The main overhead in this implementation lies in the launching of threads for every pixel, irrespective of whether it is marked or not. A better and thus more efficient approach would be to launch threads only for the unmarked pixels, thus greatly reducing the launch overhead. Selective launching of threads only for unmarked pixels is non-trivial and not supported on all CUDA architectures.

It should be noted that space requirement in the current implementation is n^2 which is used for storing the $n \times n$ matrix. The refinement step which results in doubling of resolution is handled carefully within the allocated space so that no extra space is required even temporarily. Theoretically, FACET-JFA requires much less space than JFA as discussed in 2.4, but achieving that limit would require special indexing structures.

3. EXPERIMENTAL RESULTS

3.1 Experimental Setup

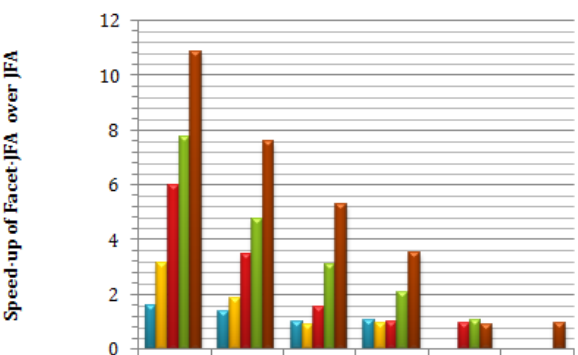
The experiments have been performed simultaneously on two different GPU architectures from nVidia, namely the latest Kepler and the older Fermi. The GPUs used for the experiments are the GTX-660 Ti and Tesla C2050. The CPU in both cases was an Intel Xeon octa core, running at 2GHz and with 16 GB of main memory. The following experiments were done.

1. Time comparisons for JFA and FACET-JFA in a 2D case on the Kepler and Fermi architectures.
2. Time comparisons for JFA and FACET-JFA in a 3D case on the Kepler architecture.
3. Comparison of the number of threads launched in JFA and FACET-JFA.

The experiments were run at resolutions ranging from 256×256 to 4096×4096, with the resolution doubling in each run. The number of seeds was increased from 10 to 100,000 incrementally for each resolution. The default random number generator, rand() was used to generate the seed locations. Running times are reported in milliseconds and are calculated separately for the actual kernel execution and copying of the grid from the host to the device and back. In the case of FACET-JFA, the time taken to generate the starting resolution is taken as a preprocessing step and is not included in the timing computation. In each experiment, the time reported is the average taken over 100 runs of the algorithm.

3.2 Observations

The speed-up obtained on the GTX-660Ti GPU has been plotted in Figure 4. The performance of FACET-JFA was superior to that of JFA in most cases, with speed-ups as high as 10x in the case of 10 seeds on a 4096×4096 grid on the GTX-660Ti. It was observed that the number of seeds and the speed-up observed are inversely related, as the presence of a larger number of seeds would generally result in smaller regions, resulting in fewer interior points. This could also be due to the fact that, in FACET-JFA, the starting resolution of a grid would increase in proportion to the number of seeds, thus necessitating larger numbers of threads to be launched in the initial call to JFA. Nevertheless, in cases where JFA outperformed Facet JFA, the slow down was marginal (0.9x).



	10	100	500	1000	10000	100000
256 x 256	1.61	1.40	0.99	1.06		
512 x 512	3.17	1.86	0.88	0.95		
1024 x 1024	5.99	3.45	1.54	0.98	0.97	
2048 x 2048	7.77	4.76	3.12	2.06	1.06	
4096 x 4096	10.87	7.59	5.28	3.50	0.90	0.98

Figure 4: Facet-JFA speed-ups in 2D case.

The threshold on the number of seeds for which JFA starts to outperform FACET-JFA was experimentally observed to be approximately n seeds for an $n \times n$ grid.

The pattern of speed-up observed on the Tesla C2050 almost exactly mirrored the observations on the GTX-660Ti, albeit at a slightly lower factor. This can be attributed to fewer number of cores on the Tesla (448, as compared to 1344 on the GTX-660Ti). We also observe similar speed-ups in the 3-dimensional case. For example, for a 512^3 grid with 1000 seeds, we observed a speed-up of around 6x. The time taken to copy the contents of the host memory to the device and back dominated the total running time in the 256×256 and 512×512 grids, whereas the time taken to execute the kernel assumed prominence from the 1024×1024 grid onwards. The speed-up was most pronounced, as expected, for the largest grid. The detailed results of these three experiments are included in the supplementary material.

Also included in the supplementary material, is an experiment where we note a marked improvement in the number of threads launched (fewer than JFA) in most of the cases under consideration. The trend we observed in this experiment almost exactly mirrors the trends observed in the speed-up, implying that speed-up is a direct consequence of the number of threads launched. However, we observed that the speed-up is a scaled down value of the ratio of the improvements in the number of launched threads (*e.g.* 88 times fewer active threads translated into 11x speed-up).

4. APPLICATION TO BIO-MOLECULAR CHANNEL EXTRACTION

Biomolecules, such as proteins, are the fundamental building blocks of living systems. It has been observed that structures including cavities, channels, protrusions, and depressions in biomolecule play an important role in defining its function [9]. Thus, geometric techniques have been used widely to facilitate study of this structure-function relationship [3].

A *channel* is a pathway through the empty space within a molecule that connects an internal point and the molecular exterior [11]. Channels are crucial for the migration of ions,

solvent, and small molecules through proteins, and their ultimate binding to the functional sites. Channels through transmembrane proteins selectively transport ions and small molecules across cell membrane. In recent years, continuous Voronoi diagrams have been used to determine center-lines of the channels in biomolecules [8, 12, 15]. The set of all channel center-lines is usually referred to as the *channel network* of the biomolecule.

Biomolecules are not static entities, they undergo various structural changes dynamically which are important for their function. Molecular Dynamics (MD) simulation trajectories are series of snapshots of the biomolecule as it undergoes changes over time. This simulation data has proved crucial in understanding dynamic behaviour of biomolecules. Recently, there has been great interest in development of fast techniques for analysis of MD trajectories [6, 10]. Lindow *et al.* [7] addressed the problem of channel extraction in MD trajectories using continuous Voronoi diagrams with focus more on the accuracy of detected channels rather than interactive analysis. Discrete Voronoi diagrams can be utilized to overcome the time-consuming step of computing continuous Voronoi diagram, and the extracted discrete channel network is sufficient for visual analysis. Further, here we are interested in computation of Voronoi edges only, which makes FACET-JFA an ideal candidate.

In this section, we present a GPU accelerated technique for computation of channel center-lines in biomolecules. We motivate this problem and present our technique using 2D synthetic data. However, the method can be easily extended to 3D. We show results both for 2D and 3D data. We exploit the fast computation of Voronoi facets by FACET-JFA for extracting channel network in the biomolecule. The Voronoi edges provide the locus of points which are locally farthest from the closest pair of atoms. The Voronoi edges can be restricted to empty regions inside the molecule to obtain the channel network. The discrete Voronoi diagram is computed using the GPU accelerated FACET-JFA. We additionally propose parallel methods for all other stages of the channel extraction algorithm. This implementation can process molecules of considerable size at a grid size suited for fast volume rendering on modern GPUs. Thus, using the proposed method, it is possible to interactively analyse static as well as dynamic channel structures in MD trajectories.

4.1 Channel extraction algorithm

Algorithm 2 describes the proposed parallel approach for fast extraction of channel network. To simplify notation, we refer to disks and union of disks in the 2D data also as atoms and molecules, respectively. The working of this algorithm is demonstrated with a 2D example in Figure 5. Brief demonstration for a 3D example is provided in Figure 6.

4.2 Discussion

As mentioned earlier, the proposed GPU accelerated extraction of channels is particularly suited for gaining a quick overview of channels in Molecular Dynamics (MD) trajectories. With fast channel extraction, the user can view the evolution of channels over time in MD data which typically consists of thousands of steps, and identify critical time steps. Further, channels can be computed for different solvent (probe) radii at interactive rates. An example use of this technique for study of dynamic channels in a 2D synthetic MD trajectory is shown in Figure 8.

Algorithm 2 Extract Channel Network

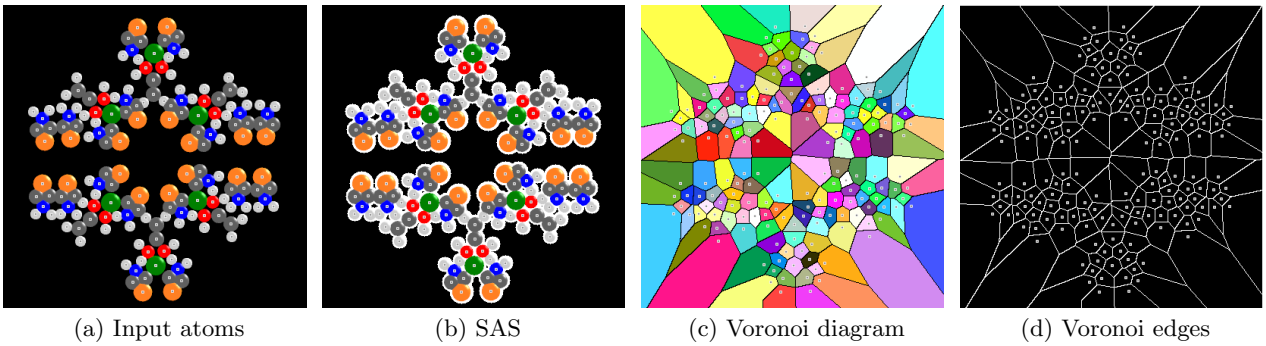
Input: S: Set of atoms.
Input: r_s: Solvent radius.
Input: n: Grid size.
Output: CN: $n \times n$ grid where pixels on channel center-lines are set to 1 while other pixels are 0.
Output: AM: $n \times n$ grid where pixels occupied by atoms of biomolecule are set to 1. This is an optional output.

1: VD := Construct discrete Voronoi diagram for S using FACET-JFA.
2: VE := Extract Voronoi edges by processing each pixel in VD in parallel. Set such edge pixels to 1. This step is not required in 2D, as VD already consists of only edges.
3: AM := Process each atom $a \in$ S in parallel and set pixels lying inside the ball of radius $r_a + r_s$ to 1. AM is called *atomic region mask*.
4: Shoot n rays in X direction in parallel into AM and determine their first and last intersections with AM. Repeat the same procedure for Y direction.
5: Construct MM, the *molecular region mask* where the pixel is set to 1 if they lie inside at least one of the intersection intervals determined in previous step.
6: IM := AM XOR MM. IM is called *molecular inside mask*.
7: CN := VE AND IM. Restrict Voronoi edges to inside region of molecule to obtain the *channel network* for the molecule.
8: **return** CN, AM

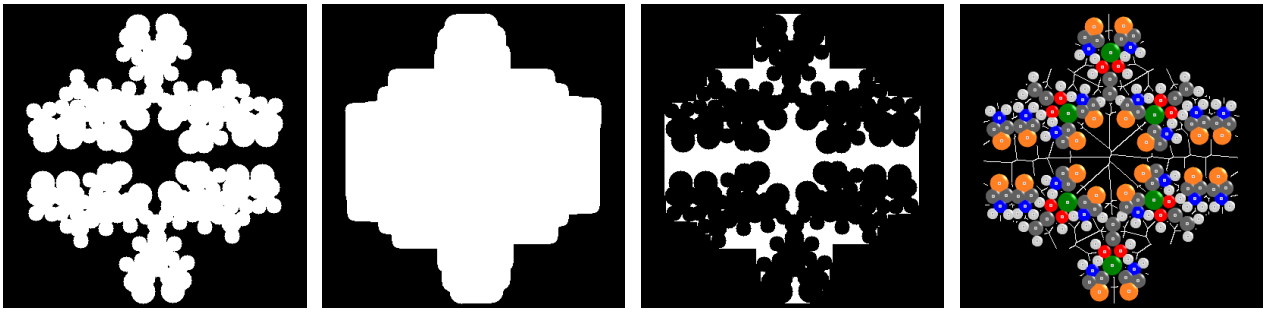
Figure 7 shows the quality of results obtained for biomolecules with number of atoms ranging from few hundreds to thirteen thousand. The grid resolution used is $128 \times 128 \times 128$ which is enough for computing a good overview of the channel network in the molecule. This resolution is ideal for high quality volume rendering at interactive rates on modern graphics hardware. The results shown in Figure 7 can be computed and simultaneously visualized at interactive speed of 10 FPS. Currently we don't support analysis of the extracted channels, like identification of connected components or pruning of small channels. This facility can be introduced to give the user a richer experience. We believe this minimal analysis can also be performed at interactive rates.

5. CONCLUSIONS

We propose a variant of JFA, FACET-JFA, to compute discrete Voronoi tessellations using a GPU, which optimises on the number of threads launched, and hence the running time. The proposed algorithm has been studied both theoretically and via experiments on large data sizes. Several improvements are possible over the proposition. FACET-JFA uses a quadtree to refine the starting grid. The starting grid could have been non-uniform, thus starting with varying levels of refinement in various regions. This would require a divide and conquer approach to merge regions with the same quadtree level, resulting in the launch of fewer threads, as compared to FACET-JFA. Also, in most real world applications, the computation of approximate Voronoi boundaries might suffice. Hence, fewer iterations of the refinement steps of FACET-JFA might provide an acceptable resolution for the application.

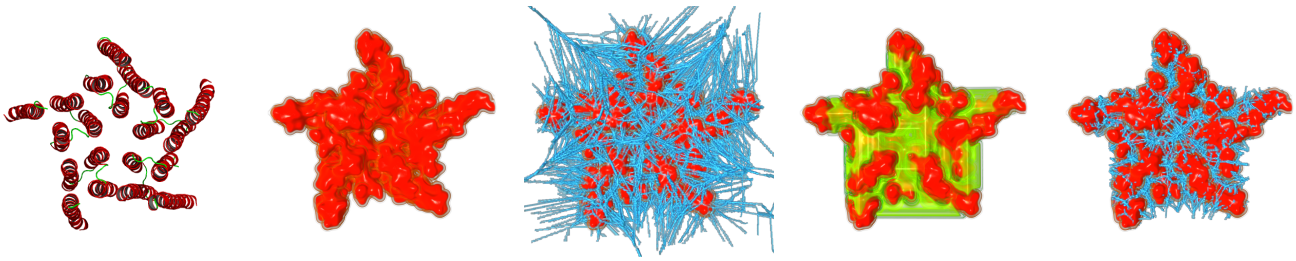


(a) Input atoms (b) SAS (c) Voronoi diagram (d) Voronoi edges



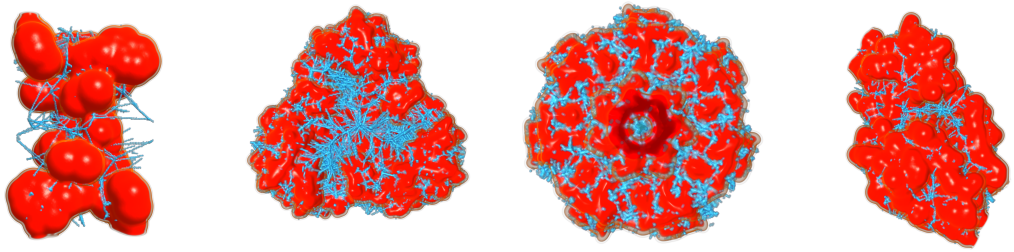
(e) Atomic Region (f) Molecular Region (g) Interior (h) Channels

Figure 5: Demonstration of GPU accelerated extraction of bio-molecular channel network. (a) A synthetic molecule in 2D. (b) Solvent accessible surface constructed by incrementing the atomic radii by the solvent radius. (c) Voronoi diagram for set of atom centres. Each Voronoi region is given a random colour. (d) Voronoi edges. (e) The *atomic region mask* obtained by projecting each atom on the grid. This mask is computed by processing the atoms in parallel. (f) The *molecular region mask* obtained by shooting rays on atomic region mask. We shoot n rays from the bottom and the left, and determine each ray's first and last intersection points with the atomic region mask. Again parallelism is exploited by processing rays in parallel. (g) Pixels in the *molecular interior* are determined by performing XOR operation on atomic region mask and molecular region mask. The pixels in molecular interior are exactly those which lie inside the molecular region but are not occupied by atoms of the molecule. (h) Finally, the Voronoi edges are restricted to the region inside the molecule to obtain the *channel network*. This is accomplished by simply performing AND operation on Voronoi edge mask and molecular interior mask.



(a) 1OED (b) Atomic region (c) Voronoi edges (d) Interior (e) Channels

Figure 6: Extension of 2D channel extraction algorithm to 3D. (a) A trans-membrane channel protein, PDB id: 1OED, is shown in cartoon representation. (b) The atoms are projected onto the 3D gid to determine atomic region. (c) The Voronoi edges are computed using Facet-JFA. (d) The region inside the molecule (shown in yellow) is determined by shooting rays in 3 orthogonal directions. (e) The Voronoi edges are restricted to the molecular inside region resulting in highlighting the molecular channel network.



(a) 1GRM (184) (b) 2J1N (8,142) (c) 2OAU (13,573) (d) 200L (1,289)

Figure 7: A few channel networks extracted are shown in blue along with the atomic region (orange). These images were generated using 128^3 grid resolution at interactive speeds. Number of atoms are in the brackets.

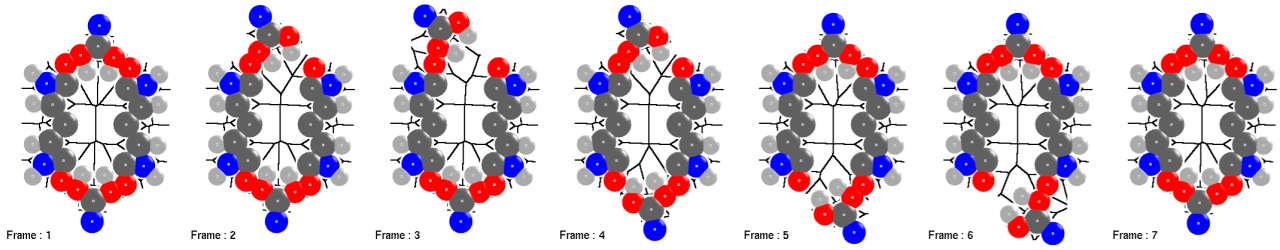


Figure 8: An example of dynamic channel in Molecular Dynamics simulation trajectory. The molecule in this synthetic dataset has two gates which are both closed initially. In Frame 2, the gate at the top opens resulting in channel to central cavity. In Frame 5, the top gate closes completely while bottom gate starts to open revealing a dynamic channel from top to the bottom. By Frame 7, the molecule regains its closed state.

Acknowledgments. Talha Bin Masood was supported by Microsoft Corporation and Microsoft Research India under the Microsoft Research India PhD Fellowship Award. This work was partially supported by the Department of Science and Technology, India, under Grant SR/S3/EECE/0086/2012 and the DST Center for Mathematical Biology, IISc, under Grant SR/S4/MS:799/12. We would like to thank Sathish Govindarajan for helpful discussions.

6. REFERENCES

[1] F. Aurenhammer, R. Klein, and D.-T. Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific, 2013.

[2] Aurenhammer, Franz. Voronoi Diagrams – A Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

[3] J. Brezovsky, E. Chovancova, A. Gora, A. Pavelka, L. Biedermannova, and J. Damborsky. Software tools for identification, visualization and analysis of protein tunnels and channels. *Biotechnol. Adv.*, 31(1):38–49, 2013.

[4] M. De Berg, M. Van Kreveld, M. Overmars, and O. C. Schwarzkopf. *Computational geometry*. Springer, 2000.

[5] 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 the 26th annual conference on Computer graphics and interactive techniques*, pages 277–286. ACM Press/Addison-Wesley Publishing Co., 1999.

[6] M. Krone, M. Falk, S. Rehm, J. Pleiss, and T. Ertl. Interactive exploration of protein cavities. In *Computer Graphics Forum*, volume 30, pages 673–682, 2011.

[7] N. Lindow, D. Baum, A. Bondar, and H. Hege. Dynamic channels in biomolecular systems: Path analysis and visualization. In *Proc. IEEE Symposium on Biological Data Visualization (BioVis)*, pages 99–106, 2012.

[8] N. Lindow, D. Baum, and H. Hege. Voronoi-based extraction and visualization of molecular paths. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2025–2034, 2011.

[9] L. Lo Conte, C. Chothia, and J. Janin. The atomic structure of protein-protein recognition sites. *J. Mol. Biol.*, 285(5):2177–2198, Feb 1999.

[10] J. Parulek, C. Turkay, N. Reuter, and I. Viola. Implicit surfaces for interactive graph based cavity analysis of molecular simulations. In *Biological Data Visualization (BioVis), 2012 IEEE Symposium on*, pages 115–122, 2012.

[11] M. Petrek, P. Kosinova, J. Koca, and M. Otyepka. MOLE: A Voronoi diagram-based explorer of molecular channels, pores, and tunnels. *Structure*, 15(11):1357–1363, 2007.

[12] M. Petrek, M. Otyepka, P. Banas, P. Kosinova, J. Koca, and J. Damborsky. Caver: a new tool to explore routes from protein clefts, pockets and cavities. *BMC bioinformatics*, 7(1):316, 2006.

[13] 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*, pages 109–116. ACM Press, 2006.

[14] 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)*, pages 176–181, 2007.

[15] E. Yaffe, D. Fishelovitch, H. J. Wolfson, D. Halperin, and R. Nussinov. MolAxis: efficient and accurate identification of channels in macromolecules. *Proteins*, 73(1):72–86, Oct 2008.