Reviewer #3: I would like to thank the authors for answering my comments in the previous round of reviews.

Minor criticism:

The authors have not mentioned a single example where memory is a bottle neck in the computation of VD or DT. Since it is an academic project, I am not that concerned that the authors are attempting to solve a problem that may have a limited impact. But it would nice of they can come up with an example.

**Answer**: Thank you for your valuable comment. We didn't find any example where memory is a bottleneck. However, as mentioned by Michael D. Adams [1], "Over the years, numerous data structures have been proposed for representing meshes, each having its own advantages and disadvantages. In the development of data structures for meshes, an important consideration is to what degree a data structure allows for efficient algorithms. In this regard, it is important to note that adjacency queries are one of the most frequent operations in most geometric algorithms. (An adjacency query simply finds the adjacent vertices/edges/faces to a given vertex/edge/face.) For this reason, it is usually critical that adjacency queries be fast (i.e., requiring constant time)." Furthermore, after the construction of VD/DT, finding neighboring faces(cells) and/or vertices(circumcircle center of Delaunay triangles) to the desired face(cell) needs additional processes [2]. While after construction of VD/DT by the proposed data structure, ID of faces(cells) and/or vertices(circumcircle center of Delaunay triangles) are accessible without further process. In fact, in many real-world applications, the information of neighboring points(cells/faces) are more useful than the information of neighboring VD edges. For more information please Cf. responses to comment 2 reviewer #4 (4th paragraph).

In addition to the above, the proposed algorithm could have a better performance in terms of the runtime compared with many well-known algorithms in compiler languages like Java. For more information please Cf. responses to comment 3 reviewer #4.

In general, the proposed algorithm by maintaining the information of VD alongside DT simultaneously, lower memory usage compared with edge-based data structures, a faster runtime performance compare with many well-known algorithms, and faster (more easily accessible) neighborhood information of faces(cells/points) in many applications, could be useful in a wide variety of applications.

\*We will add these explanations to the next version of the paper after the implementation of all three versions of the proposed algorithm in Java and a comparison of its runtime with other algorithms in the same environment.

General criticism:

In their response to my question about proofs for claiming that the algorithm terminates, authors have argued that the current method is similar to prior algorithms, so their claim still holds. I believe the authors.  I would like to see the claims in the paper rather than in the response to my review. I think they should make their claims in forms of lemmas and theorems and talk about why their algorithm is similar to prior algorithm, and thus argue that a completely new proof is not necessary. For anyone reading the paper in the future, it is important to know where to find the proofs. This is the reason why I am suggesting major revision. I would like to see those proofs in the paper. I don't think it will more than a page to the manuscript, so authors need not be concerned about the length of the paper.

**Answer**: Thank you very much for your valuable comment that increases the quality of the paper content. The following explanations are added to the new version of the article:

"**Note 2**: The proposed idea to find the cell containing the added point in this article is similar to the idea proposed by Guibas et. all in 1992 [3]. They established appropriate links between "old" and "new" triangles. Each "old" triangle is connected to two or three "newer" triangles, based on their proposed structure. Then they trace all triangles that contain the added point, in chronological order of their creation. Their proposed idea has overall time complexity in order of O(n log n). In this article, we have "old" and "new" Voronoi cells that are linked to each other based on the chronological order of their creation. Guibas et. all used the notion of the scope of each triangles, that is the number of sites contained in the interior of the circumcircle of the desired triangle, to prove the time complexity of their idea. Similarly, we define the scope of each Voronoi cell to be the number of points in the interior of the boundary of the second-order Voronoi of each Voronoi cell. To quote Guibas et. all, "A triangle X Y Z of scope k can be charged by at most k sites. It is also plain that no triangle can be charged more than once by the same site. Moreover, a necessary condition that a triangle X Y Z is charged at all is that it arises as a Delaunay triangle at some stage during the incremental construction". Similar explanations are credible for Voronoi cells in the proposed idea in this research. Interested readers could refer to reference [3], lemma 2.1, lemma 2.2, lemma 2.3, and section 3 for more details and proofs.

**Note 3**: The algorithm proposed in this paper for the second step is similar to the procedure proposed by Green and Sibson in [4] for updating the Voronoi diagram locally. The main difference is that in the proposed method of Green and Sibson, the locations where perpendicular bisectors meet the edges of the Voronoi cells are calculated and from them, the list of the nearest neighbors is obtained. But in our proposed method, which is based on a non-edge-based data structure, first, the neighboring points are found counterclockwise and the vertices of the Voronoi cell of added point are calculated from the perpendicular intersection of two consecutive points in the list of nearest neighbors. Usually, similar previous research check if the added point is inside the circumference of neighboring triangles. On the contrary, in the proposed method, relying on the proposed baseline data structure and considering the dual relationship between Delaunay triangulation and the Voronoi diagram, we check if the center of the circumferential circle of the surrounding triangles are inside the Voronoi cell of the added point. Checking is also carried out in such a way that the list of the nearest neighbors to the added point of the triangles that have lost their Delaunay property is extracted counterclockwise. Thus, all the properties of Delaunay triangulation and its dual (i.e., Voronoi diagram) are preserved and this dual relationship is exploited. Moreover, checking the center of the circumference of each triangle is done through a point that is one of the vertices of the triangle itself. Hence, checking the center of the circumference of each triangle is performed with high accuracy and reliability. Thus, the proposed idea finds the list of the nearest neighbors of the added point alongside the id of the triangles that lost their Delaunay property, correctly."

Reviewer #4: Although an innovative incremental insertion algorithm for 2D Delaunay is proposed in this paper, the value of it is in doubt because the performance of the algorithm (such as the time efficiency) has not been well compared and explained. Therefore, I cannot agree with this paper published on Advances in Engineering Software. However, if the author can make further explanations or experiments on the following questions I have raised, I will consider whether to publish it:

1. The author designed an ingenious data structure to save memory, but the memory of 2D Delaunay is not an important problem. If the author thinks it has strong application, please explain its importance.

**Answer**: Thank you for your valuable comment. Cf. responses to comment 1 reviewer #2.

2. Similarly, although the algorithm can obtain VD and DT at the same time, I think it's relatively easy to obtain VD after DT has been obtained, because the adjacencies of triangles are often recorded in the process of generating DT. I think it had better explain the significance of obtaining both VD and DT at the same time in the paper.

**Answer**: Thanks for your valuable comment. Delaunay triangulation and Voronoi diagram are used together in some applications [5–8] and usually one structure is built from the other structure. As stated in [9], after creating the Delaunay triangulation or Voronoi diagram with the help of the DCEL data structure, the other structure can be adapted with calculations with complexity of O(n). Therefore, after creating Delaunay triangulation or the Voronoi diagram with complexity O(nlogn), a separate process with complexity O(n) is required to extract another structure. But according to the proposed idea in this study, the Delaunay triangle and the Voronoi diagram are created together with the complexity of O(nlogn).

The memory required to store Voronoi diagram information alone is at least 27n-25 (for winged edge and DCEL data structures) through other edge-based data structures (as mentioned in the article). So extracting and storing Delaunay triangulation information will take up even more memory than this value. Occupied memory is also of the 36n-36 order according to the quad-edge structure, which stores Voronoi diagram information and Delaunay triangulation together. While the data structure proposed in our study requires 24n-15 (at most) / 12n-15 (at least) integer memory (which is less than the memory required to store Veronoi diagram information in other data structures alone) to hold Voronoi diagram information and Delaunay triangulation together.

Furthermore, simultaneous construction of VD and DT by the proposed algorithm does not affect its performance in terms of runtime. In fact, it constructs both diagrams based on the dual relationship between DT and VD. Comparison of the runtime of the proposed algorithm with some of the well-known algorithms shows the good performance of the proposed algorithm in terms of the run time.

 In addition to what has been said, it is better to compare proposed data structure with edge-based data structures in practice: The Voronoi diagram is used in a variety of areas that require point neighborhood information without the need to create spatial indexes including clustering, nearest neighbor search (e.g. post office problem), k-nearest neighbors, robotics, and so on. In order to better determine the performance of the proposed face-based structure relative to the edge-based structures presented in practice, two important and widely used applications of the Voronoi diagram, which are the point location problem and the problem of all nearest neighbors of points (ANN), are discussed. These two problems have many applications in biology, ecology, geography, and physics [10].

**Point location problem:** in these problems, we are looking for a site (Voronoi cell) of a desired location. So we find the closest neighbor to the desired point. Some of the proposed solutions (such as those proposed in [3–5]), are optimal in terms of complexity (O(logn)), but difficult in terms of implementation. Some research including [3] has provided a simple solution with a non-complex implementation but non-optimal time complexity ($O(logn)^{2}$). While in the proposed algorithm, finding the Voronoi cell of the added points is part of the algorithm itself. Also, for each new point, which is incrementally added to the structure of the Voronoi diagram, a Voronoi cell containing that point is found through the connection between the old-vm and vm points in the first step. Therefore, there is no need for additional complex structures and, the point location problem can be solved as part of its routine with the solution presented in this research. In addition to the simplicity of implementation, the proposed solution also has the required memory of O(n) order and is optimal in terms of complexity (i.e., O(logn)).

**All nearest neighbors problem:** Suppose we want to extract a list of the nearest neighbors of points through the provided edge-based data structure. In this case, in the first step, we have to create a Voronoi diagram of the points, and then during a procedure in the next step, we extract the list of the nearest points for each point separately through the edge information. For example, [2] provides a solution for extracting the neighboring polygons of each polygon in the Voronoi diagram (extracting the nearest neighbors at any point) through the winged-edge data structure. Rather than the edge-based data structure, in the proposed data structure, the list of the nearest points of each point in the first or second column of the VM matrix of points is available counterclockwise from top to bottom after completing the construction of the Voronoi diagram and no additional processing is required, because the proposed data structure in this study is the vertex-based and neighborhood information is easily available.

3. Most importantly, the author claims that the time complexity can reach O (nlogn). However, I think most of the current incremental insertion algorithms have a time complexity of O (nlogn). In addition, the efficiency of algorithms with the same time complexity can vary greatly. The author said that the reason why the low time efficiency shown in the experiment is the row shift operation of Matlab, but I think the operation of each step of the algorithm is complex and probably inefficient. If the author doesn't think so, I still suggest realizing it in other programming languages, and comparing it with other algorithms (for example, running 2D Delaunay code of CGAL and the proposed algorithm code with the same compilation method on the same device) to prove its time efficiency. After all, an algorithm with relatively low memory footprint but very low time efficiency is not enough to be published in this journal.

**Answer**: Thank you for your valuable comment that increases the quality of the paper content. We've implemented the first version of the proposed algorithm in Java (IntelliJ IDEA software). Unfortunately, we didn't find any version of CGAL for java. However, for comparison of the proposed algorithm with some of the well-known algorithms we used the following implementations available in GitHub:

* Fotune's algorithm [14]: Available online at <https://github.com/aschlosser/voronoi-java>.
* Guibas et. all [3]: Available online at <https://github.com/jdiemke/delaunay-triangulator>
* A. Nocaj and U. Brandes [15]: It is implemented for weighted Voronoi diagrams by computation of 3D convex hull. Available online at <https://github.com/ArlindNocaj/power-voronoi-diagram>.

As mentioned by the authors, to get an ordinary voronoi diagram, we set the weight of all points equal to zero.

The results are shown in Table 1 and Table 2. We've compared the runtime of the mentioned algorithms with the proposed algorithm in this research. Table 1 shows the runtime of the four mentioned algorithms for $2^{12}$ to $2^{16}$ uniform distributed points. Fortune's algorithm has the worst runtime among other algorithms. Although the proposed idea in our research is similar to Guibas et. all [3] idea for walking operation, the difference in runtime is noticeable and the proposed idea alongside the proposed face-based data structure shows a better performance than that of Guibas et. all. From Table 1 it is clear that the proposed algorithm has a better performance compared with others. To avoid overhead effects, we compared the runtime of the two pioneer algorithms (the new proposed algorithm and Nocaj-Brandes algorithm) for $2^{17}$ to $2^{21}$ number of uniform distributed points. It can be seen from Table 2 that the performance of the proposed algorithm in this research is better than Nocaj-Brandes algorithm.

In comparison with Guibas et. all [3], the proposed idea in this research has not only fewer memory costs but also a better run time performance. Heretofore, we didn’t claim that our algorithm has a better performance than all previous research. But now, it is possible to claim that the proposed algorithm (alongside the proposed data structure) could have a better performance, compared with some of the well-known algorithms, at least.

Another important point is that unlike MATLAB that the shift operation of VM matrices has a high time cost, in java we don't have a high amount of the row shift operation impact. As mentioned before, that is because MATLAB is an interpreter language, while java is a compiler one.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  Number of pointsAlgorithm | $$2^{12}$$ | $$2^{13}$$ | $$2^{14}$$ | $$2^{15}$$ | $$2^{16}$$ |
| Fortune | 1.45989 s | 5.1921 | 19.4035 | 88.26695 | 349.83348 |
| Guibas et. all | 0.40692 | 1.62936 | 6.67799 | 29.73235 | 157.65163 |
| Nocaj-Brandes | 0.17976 | 0.28577 | 0.44975 | 1.51971 | 2.64529 |
| Proposed algorithm | **0.13809** | **0.26869** | **0.36207** | **0.65435** | **1.02162** |

Table 1: Total runtime of four algorithms

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Number of pointsAlgorithm | $$2^{17}$$ | $$2^{18}$$ | $$2^{19}$$ | $$2^{20}$$ | $$2^{21}$$ |
| Nocaj-Brandes | 5.38715 | 11.34868 | 25.25682 | 51.25547 | 113.17969 |
| Proposed algorithm | **2.29009** | **4.33315** | **7.47251** | **16.23786** | **39.30965** |

Table 2: Runtime of Nocaj and Brandes algorithm and the proposed algorithm for higher number of points

\*To implement all three versions of our algorithm in java and for a better and complete comparison with other algorithms (and maybe increase the performance of the proposed algorithm), we need more time. We would be grateful if you give us more time to re-compute the runtime of all three versions of the proposed algorithm in java, compare it with other algorithms, and replace the results with previous results of the paper, as well.

**References**:

[1] M. D. Adams, *Multiresolution Signal and Geometry Processing: Filter Banks, Wavelets, and Subdivision (Version: 2013-09-26)*. Michael Adams, 2013.

[2] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, 2009.

[3] Guibas LJ, Knuth DE, Sharir M. Randomized incremental construction of Delaunay and Voronoi diagrams. Algorithmica. 1992 Jun;7(1):381-413.

[4] Green PJ, Sibson R. Computing Dirichlet tessellations in the plane. The computer journal. 1978 May 1;21(2):168-73.

[5] Z. Miao, Y. Chen, X. Zeng, and J. Li, “Integrating Spatial and Attribute Characteristics of Extended Voronoi Diagrams in Spatial Patterning Research: A Case Study of Wuhan City in China,” *ISPRS Int. J. Geo-Inf.*, vol. 5, no. 7, p. 120, Jul. 2016, doi: 10.3390/ijgi5070120.

[6] A. Santamaria-Ibirika, X. Cantero, S. Huerta, I. Santos, and P. G. Bringas, “Procedural Playable Cave Systems Based on Voronoi Diagram and Delaunay Triangulation,” in *2014 International Conference on Cyberworlds*, Santander, Cantabria, Spain, Oct. 2014, pp. 15–22. doi: 10.1109/CW.2014.11.

[7] R. Soleymani and M. Chehel Amirani, “A hybrid fingerprint matching algorithm using Delaunay triangulation and Voronoi diagram,” in *20th Iranian Conference on Electrical Engineering (ICEE2012)*, Tehran, Iran, May 2012, pp. 752–757. doi: 10.1109/IranianCEE.2012.6292454.

[8] J. Chen, C. Luo, M. Krishnan, M. Paulik, and Y. Tang, “An enhanced dynamic Delaunay triangulation-based path planning algorithm for autonomous mobile robot navigation,” San Jose, California, Jan. 2010, p. 75390P. doi: 10.1117/12.838966.

[9] F. Aurenhammer, R. Klein, and D. T. Lee, *Voronoi diagrams and Delaunay triangulations. World Scientific Publishing Company.* 2013.

[10] J. O’Rourke, *Computational geometry in C*, 2nd ed. Cambridge, UK, ; New York, NY, USA: Cambridge University Press, 1998.

[11] D. Kirkpatrick, “Optimal Search in Planar Subdivisions,” *SIAM J. Comput.*, vol. 12, no. 1, pp. 28–35, Feb. 1983, doi: 10.1137/0212002.

[12] H. Edelsbrunner, L. J. Guibas, and J. Stolfi, “Optimal Point Location in a Monotone Subdivision,” *SIAM J. Comput.*, vol. 15, no. 2, pp. 317–340, May 1986, doi: 10.1137/0215023.

[13] R. J. Lipton and R. E. Tarjan, “Applications of a Planar Separator Theorem,” *SIAM J. Comput.*, vol. 9, no. 3, pp. 615–627, Aug. 1980, doi: 10.1137/0209046.

[14] S. Fortune, “A sweepline algorithm for Voronoi diagrams,” *Algorithmica*, vol. 2, no. 1, p. 153, Nov. 1987, doi: 10.1007/BF01840357.

[15] A. Nocaj and U. Brandes, “Computing Voronoi Treemaps: Faster, Simpler, and Resolution‐independent,” 2012, vol. 31, no. 3pt1, pp. 855–864.