




1 **votess**

2 **Samridh Dev Singh¹, Chris Byrohl², and Dylan Nelson²**

3 **1** TODO **2** Heidelberg University, Institute for Theoretical Astronomy, Albert-Ueberle-Str. 2, 69120
4 Heidelberg, Germany

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Open Journals](#) 

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright
and release the work under a
Creative Commons Attribution 4.0
International License ([CC BY 4.0](#)).

5 **Statement of need**

6 With the rise of parallel architectures, it has now become possible to solve problems that
7 would have been computationally too expensive in the past. One example would be a Voronoi
8 tessellation on large datasets. Many projects utilize such an algorithm, be it from Cosmology,
9 Earth Science, Material Science, Biochemistry, etc.

10 There isn't however, a tessellation program that can run on the variety of platforms without
11 severe modification to the source code. Thus forcing each problem requiring a voronoi
12 tessellation to have a bespoke solution.

13 **Summary**

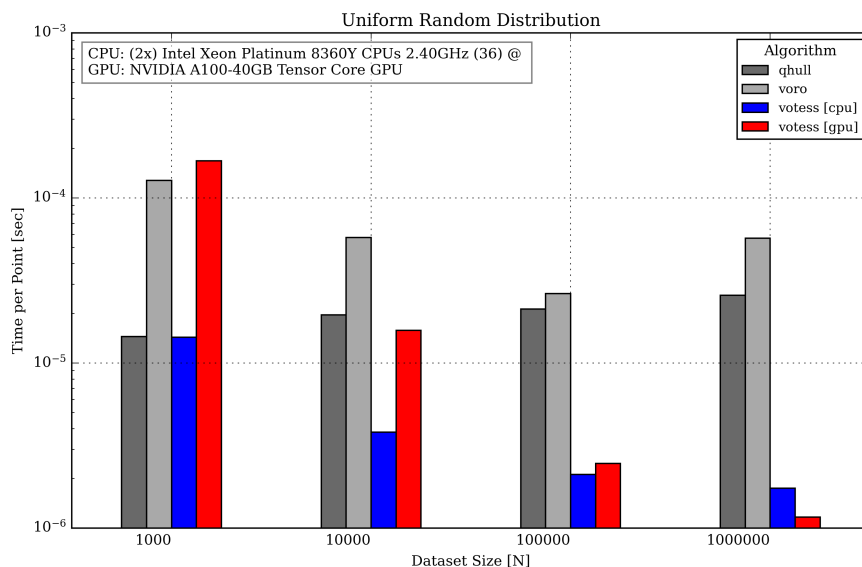
votess is a library for performing 3D Voronoi tessellations on heterogeneous platforms via
SYCL framework. votess was designed to be portable, but performant, with an easy to use
interface.

17 The underlying algorithm is based on the work of Ray et al. (2018) ([Ray et al., 2018](#)), which
18 describes how to compute a Voronoi diagram without the need for a combinatorial mesh
19 data structure, as required by classical approaches like the Bowyer-Watson algorithm. The
20 core algorithm employed by votess consists of two main steps. First, given an input set of
21 points, a k-nearest neighbors search is performed after sorting the points into a grid. With the
22 nearest neighbors identified for each point, the Voronoi cell is computed by iteratively clipping
23 a bounding box using the perpendicular bisectors of the point and its neighbors. To avoid
24 iterating through all neighbors, a security radius condition is applied. If a Voronoi cell cannot
25 be validated, a CPU fallback mechanism ensures robustness.

26 This simple, efficient algorithm allows for independent thread execution, making it well-suited
27 for GPU parallelism.

28 **Performance**

29 With a working implementation of votess, it can be seen that it outperforms several single-
30 threaded applications:



31

32 The GPU implementation of votess is designed to be high throughput, and it is apparent
33 as it is highest performing at datasets beyond a million points. The CPU implementation
34 outperforms other applications of atleast tenfold, almost reaching a hundred fold at larger
35 datasets. It must also be noted, that the benchmark was taken before votess had recieved
36 optimizations.

37 Features

38 votess provides a simple C++ interface to compute tesellations, that being a single function
39 tesellate. There also exists an interface to select ipnuts One can select the target device to
40 run the tesellation on, and currently CPU and GPU devices are supported.

41 A classvtargs is provided, with usage following closely to the `std::unordered_map` STL class.
42 It enables users to tune parameters, which could help with the runtime performance of the
43 application, if that is ever necessary.

44 The tesellate function returns a templated class dnn, as a 2 dimensional jagged array of
45 neighbors that contribute to the voronoi cell of each particle in the sorted dataset. How the
46 dataset is sorted can be tuned by class vtargs.

47 There also exists a python wrapper to tesellate, named pyvotess, with the same usage as
48 the C++ implementation.

49 Acknowledgements

50 CB and DN acknowledge funding from the Deutsche Forschungsgemeinschaft (DFG) through
51 an Emmy Noether Research Group (grant number NE 2441/1-1).

52 Ray, N., Sokolov, D., Lefebvre, S., & Lévy, B. (2018). Meshless voronoi on the GPU. *ACM*
53 *Transactions on Graphics*, 37(6), 1–12. <https://doi.org/10.1145/3272127.3275092>