EUROGRAPHICS 2015 / O. Sorkine-Hornung and M. Wimmer
(Guest Editors)

Volume 34 (2015), Number 2

An Adaptive, Parallel Algorithm for Approximating the Generalized Voronoi Diagram

John Edwards and Nathan Vollmer and Nicholas Harrison

Idaho State University

Abstract

We present a parallel algorithm that approximates the Generalized Voronoi Diagram (GVD) on arbitrary collections of 2D or 3D geometric objects. Our algorithm is hierarchical, not uniformly gridded, enabling us to compute the GVD on datasets with extremely close object tolerances, including intersections. We demonstrate our method on a variety of data and show example applications of the GVD in 2D and 3D.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.5]: Computational Geometry and Object Modeling—Boundary representations Computer Graphics [I.3.6]: Methodology and Techniques—Graphics data structures and data types

1. Introduction

Previous paper: [EDPB15]

The generalized Voronoi diagram (GVD) is an important structure that divides space into a complex of generalized Voronoi cells (GVCs) around objects. Similar to the ordinary Voronoi diagram, each GVC contains exactly one object, or site, and every point in the GVC is closer to its contained object than to any other object. The generalized Voronoi diagram is the boundary of the cell complex, and thus every point on the GVD is equidistant from two or more closest objects. Applications of the GVD range from motion path planning to GIS analysis to mosaicking.

Ordinary Voronoi diagrams have been studied extensively and efficient algorithms exist to compute them, but the GVD is difficult to compute analytically in general [BWY06, HIKL*99] and so the majority of approaches compute an approximation. Whereas most algorithms are efficient and robust on certain datasets, all algorithms to our knowledge require inordinate amounts of memory on datasets where objects are very closely spaced relative to the size of the domain. The failures occur because the space is uniformly gridded. In such approaches, voxel size must be small enough to resolve object spacings, and if two objects are very close to each other the number of voxels can become prohibitively large.

We present an algorithm to compute a GVD approximation on arbitrary datasets, including those with closely spaced objects. The approach applies a distance transform over an octree representation of the objects. Our octree, its associated data structure, and our distance transform are novel and optimized to GVD approximation. For the remainder of the paper, "GVD" will refer to the approximated Generalized Voronoi Diagram.

This paper demonstrates GVD computation on data beyond the computational abilities of previous algorithms, unlocking interesting and important applications. Our approach allows GVD-based proximity queries and other applications using a larger class of meaningful datasets.

Main contributions The primary technical contributions described in the paper are as follows.

1. Contribution 1
2. Contribution 2
3. Contribution 3

We demonstrate various applications of the GVD...

Our GVD algorithm has a few main steps: 1) ...

2. Related work

Related work falls into two categories: algorithms that compute the GVD and algorithms that compute distance fields, many of which are adaptive.

Generalized Voronoi diagrams A theoretical framework for generalized Voronoi diagrams can be found in Boissonnat et al. [BWY06]. Ordinary Voronoi diagrams are well studied and efficient algorithms exist that compute them exactly [DBCVK08], but exact algorithms for the generalized Voronoi diagram are limited to a small set of special cases [Lee82, Kar04]. In an early work, Lavender et al. [LBD*92] define and compute GVDs over a set of solid models using an octree. Etzion and Rappoport [ER02] represent the GVD bisector symbolically for lazy evaluation, but are limited to sites that are polyhedra. Boada et al. [BCS02, BCMAS08] use an adaptive approach to GVD computation, but their algorithm is restricted to GVDs with connected regions and is inefficient for polyhedral objects with many facets. Two other works are adaptive [TT97, VO98] but are computationally expensive and are restricted to convex sites.

In recent years Voronoi diagram algorithms that take advantage of fast graphics hardware have become more common [CTMT10, FG06, HT05, RT07, SGGM06, SGG*06, HIKL*99, WLXZ08]. These algorithms are efficient and generalize well to the GVD, but most are limited to a subset of site types. More importantly, all of them use uniform grids and require an extraordinary number of voxels to resolve closely spaced objects (for example, Figs. **??** and **??** would require 2^{36} and 2^{48} voxels, respectively). To our knowledge, ours is the first fully adaptive algorithm that computes the generalized Voronoi diagram for arbitrary datasets.

Distance fields and octrees The GVD is a subset of the locus of distance field critical points, a property that we take advantage of. In that light, the GVD could be a post-processing step to any method that computes a distance field. Distance transforms compute a distance field, but most are uniformly gridded [JBS06] and are thus no more suitable than GVD algorithms that use the GPU.

Two seminal works adaptively compute the Adaptive Distance Field (ADF) on octree vertices. Strain [Str99] fully resolves the octree everywhere on the object surface, and Frisken et al. [FPRJ00] resolve the octree fully only in areas of small local feature size. Both approaches are designed to retain features of a single object rather than resolving between multiple objects, as is required for GVD computation. Qu et al. [QZS*04] implement an energy-minimizing distance field algorithm that preserves features at the expense of efficiency. Many recent works on fast octree construction using the GPU are limited to point sites [BGPZ12, Kar12, ZGHG11]. Most octree approaches that support surfaces [BLD13, CNLE09, LK11, LH07] are designed for efficient rendering, and actual construction of the octree is implemented on the CPU.

Two works [BC08, PLKK10] implement the ADF using GPU parallelism to compute the distance value at sample points, but building the octree itself is done sequentially. Yin et al. [YLW11] compute the distance field entirely on the GPU using a bottom-up approach by initially subdividing into a complete octree, resulting in memory usage that is no better than using a uniform grid. A method by Kim and Liu [KL14] computes the octree and a BVH entirely on the GPU. However, octree construction is performed on barycenters of triangles, and so a leaf octree cell can have an arbitrary number of triangle intersections as long as it contains no more than one triangle's barycenter. We have found no GPU octree construction method that can resolve between objects.

3. Build octree

Our algorithm works in both 2D and 3D. Lacking a dimension-independent term, we use "octree" as a general term to refer to both quadtrees and octrees.

In the algorithm below, we use the Karras algorithm as a subroutine. It takes vertices of the objects as data points. Given enough parallel units, it runs in $O(\log N)$ time, where N is the number of vertices.

We use $container(l, O)$ to denote the smallest octree cell in octree O that fully contains the geometric object l. The complexity of the $container(l, O)$ function is $O(\log N)$. Thus, step 2 of the algorithm (lines 2-10) runs in $O(\log N)$ time.

Step 3 of the algorithm (lines 11-29) runs in $O(L)$ time, where L is the number of lines in all objects. Even though the loop is doubly-nested, each line is stored in a unique internal node, so no more than L lines will be visited in the loops. In practice, far fewer than L lines will be checked for each leaf cell, because most datasets have lines that are completely contained in internal cells that are reasonably low in the tree.

Steps 1-3 are $O(\log N + L) = O(2L) = O(L)$. However, average case is $O(\log N)$, considering that most lines are contained entirely in a cell at reasonably low depth.

In Step 4, Stack is preallocated to size $M \cdot 2^D$ where M is the maximum octree depth and D is the dimension. A conflict cell is a cell that intersects at least two different objects, or two lines of different labels.

In Fig. 1, R (Root) is the smallest containing cell for lines A, B, and C, cell 20 contains line D, and cell 2 contains lines E and F. After Step 3 of the algorithm, line A is stored in leaf cells 202, 203, 21, and 3. Conflict cells, which are the only cells that are subdivided, are 203 and 21.

4. Compute GVD surface

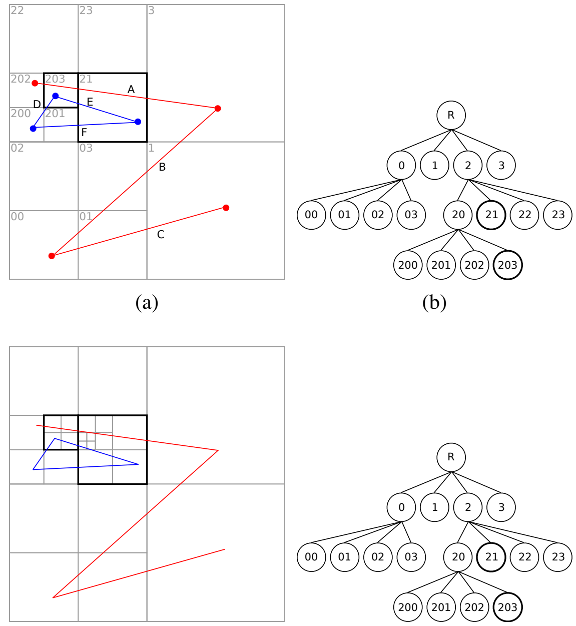
5. Results and applications

Our implementation[†] of the algorithm supports polygons and triangulated objects, and our wavefront initialization

[†] Source code is available at `http://cedmav.org/research/project/33-gvds.html`.

Algorithm 1: BUILD_OCTREE

```
Input: Objects, Vertices
// 1. Build initial octree IOctree
IOctree := karras(Vertices)
// 2. Find containing internal cells
for line l in Objects do in parallel
    c := container(l, IOctree)
    c.numLines++
end
Allocate c.lines using parallel prefix sums (scan)
for line l in Objects do in parallel
    c := container(l, IOctree)
    c.lines.append(l)
end
// 3. Find line-leaf cell
   intersections
for leaf cell c in IOctree do in parallel
    foreach cell a in direct_ancestors(c) do
        foreach line l in a.lines do
            if l intersects c then
                c.numLines++
            end
        end
    end
end
Allocate c.lines using parallel prefix sums (scan)
for leaf cell c in IOctree do in parallel
    foreach cell a in direct_ancestors(c) do
        foreach line l in a.lines do
            if l intersects c then
                c.lines.append(l)
            end
        end
    end
end
// 4. Octree refinement
for leaf cell c in IOctree do in parallel
    c' := c
    while c' is conflict cell do
        (c'_0, c'_1, ..., c'_{2^D-1}) := subdivide c'
        push (c'_0, c'_1, ..., c'_{2^D-1}) onto Stack
        c' := Stack.pop
    end
end
```



(a) (b) (c) (d)

Figure 1: The $container(l,O)$ function finds the smallest octree cell that completely contains line l. **??** Cartesian view of the octree with a red and a blue object. **??** Hierarchical view of the octree.

step is implemented on the GPU using OpenCL. All tests were run on a MacBook Pro laptop with a dual-core 2.9 GHz processor, 8 GB memory, and Intel HD 4000 graphics card. Figure **??** shows our implementation of the GVD computation pipeline, and Figure **??** shows the computed GVD on a more challenging dataset. We compare our method with other work and then show examples in three application settings: path planning, proximity queries, and exploded diagrams.

5.1. Comparison to other methods

6. Conclusions

References

[BC08] Bastos T., Celes W.: Gpu-accelerated adaptively sampled distance fields. In *Shape Modeling and Applications, 2008. SMI 2008. IEEE International Conference on* (2008), IEEE, pp. 171–178. 2

[BCMAS08] Boada I., Coll N., Madern N., Antoni Sellares J.: Approximations of 2d and 3d generalized voronoi diagrams. *International Journal of Computer Mathematics 85*, 7 (2008), 1003–1022. 2

[BCS02] Boada I., Coll N., Sellares J.: The voronoi-quadtree: construction and visualization. *Eurographics 2002 Short Presentation* (2002), 349–355. 2

dataset	objects	object Δs ($\times 10^3$)	octree depth	octree cells ($\times 10^3$)	octree memory (Mb)	GVD (sec)	GVD Δs ($\times 10^3$)
Fig. **??**	3	7	8	54	3	0.9	83
Fig. **??**	4	15	12	146	9	3.9	232
Fig. **??**	470	5	24	158	8	2.0	151
Fig. **??**	448	4015	8	2716	151	195	8100
Fig. **??**	35	1500	8	496	70	19	2700

Table 1: Table of octree/GVD computation statistics and timings on datasets that are unmanageable using other methods. Columns are: *objects* - the number of objects in the dataset; *object* Δs - the number of line segments (2D) or triangles (3D) of all objects in the dataset; *octree depth* - required octree depth in order to resolve objects; *octree cells* - total number of leaf octree cells; *octree memory* - amount of memory used by the octree; *GVD (sec)* - seconds to perform all steps of GVD computation; *GVD* Δs - number of line segments (2D) or triangles (3D) in the GVD.

[BGPZ12] Bédorf J., Gaburov E., Portegies Zwart S.: A sparse octree gravitational< i> n</i>-body code that runs entirely on the gpu processor. *Journal of Computational Physics 231*, 7 (2012), 2825–2839. 2

[BLD13] Baert J., Lagae A., Dutré P.: Out-of-core construction of sparse voxel octrees. In *Proceedings of the 5th High-Performance Graphics Conference* (2013), ACM, pp. 27–32. 2

[BWY06] Boissonnat J.-D., Wormser C., Yvinec M.: Curved voronoi diagrams. In *Effective Computational Geometry for Curves and Surfaces*. Springer, 2006, pp. 67–116. 1, 2

[CNLE09] Crassin C., Neyret F., Lefebvre S., Eisemann E.: Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games* (2009), ACM, pp. 15–22. 2

[CTMT10] Cao T.-T., Tang K., Mohamed A., Tan T.-S.: Parallel banding algorithm to compute exact distance transform with the gpu. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games* (2010), ACM, pp. 83–90. 2

[DBCVK08] De Berg M., Cheong O., Van Kreveld M.: *Computational geometry: algorithms and applications*. Springer, 2008. 2

[EDPB15] Edwards J., Daniel E., Pascucci V., Bajaj C.: Approximating the generalized voronoi diagram of closely spaced objects. *Computer Graphics Forum 34*, 2 (2015), 299–309. URL: `http://dx.doi.org/10.1111/cgf.12561`, `doi:10.1111/cgf.12561`. 1

[ER02] Etzion M., Rappoport A.: Computing voronoi skeletons of a 3-d polyhedron by space subdivision. *Computational Geometry 21*, 3 (2002), 87–120. 2

[FG06] Fischer I., Gotsman C.: Fast approximation of high-order voronoi diagrams and distance transforms on the gpu. *Journal of Graphics, GPU, and Game Tools 11*, 4 (2006), 39–60. 2

[FPRJ00] Frisken S. F., Perry R. N., Rockwood A. P., Jones T. R.: Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 249–254. 2

[HIKL*99] Hoff III K. E., Keyser J., Lin M., Manocha D., Culver T.: Fast computation of generalized voronoi diagrams using graphics hardware. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), ACM Press/Addison-Wesley Publishing Co., pp. 277–286. 1, 2

[HT05] Hsieh H.-H., Tai W.-K.: A simple gpu-based approach for 3d voronoi diagram construction and visualization. *Simulation modelling practice and theory 13*, 8 (2005), 681–692. 2

[JBS06] Jones M. W., Baerentzen J. A., Sramek M.: 3d distance fields: A survey of techniques and applications. *Visualization and Computer Graphics, IEEE Transactions on 12*, 4 (2006), 581–599. 2

[Kar04] Karavelas M. I.: A robust and efficient implementation for the segment voronoi diagram. In *International symposium on Voronoi diagrams in science and engineering* (2004), Citeseer, pp. 51–62. 2

[Kar12] Karras T.: Maximizing parallelism in the construction of bvhs, octrees, and k-d trees. In *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics* (2012), Eurographics Association, pp. 33–37. 2

[KL14] Kim Y. J., Liu F.: Exact and adaptive signed distance fieldscomputation for rigid and deformablemodels on gpus. *IEEE Transactions on Visualization and Computer Graphics 20*, 5 (2014), 714–725. 2

[LBD*92] Lavender D., Bowyer A., Davenport J., Wallis A., Woodwark J.: Voronoi diagrams of set-theoretic solid models. *Computer Graphics and Applications, IEEE 12*, 5 (1992), 69–77. 2

[Lee82] Lee D.-T.: Medial axis transformation of a planar shape. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* (1982), 363–369. 2

[LH07] Lefebvre S., Hoppe H.: Compressed random-access trees for spatially coherent data. In *Proceedings of the 18th Eurographics conference on Rendering Techniques* (2007), Eurographics Association, pp. 339–349. 2

[LK11] Laine S., Karras T.: Efficient sparse voxel octrees. *Visualization and Computer Graphics, IEEE Transactions on 17*, 8 (2011), 1048–1059. 2

[PLKK10] Park T., Lee S.-H., Kim J.-H., Kim C.-H.: Cuda-based signed distance field calculation for adaptive grids. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on* (2010), IEEE, pp. 1202–1206. 2

[QZS*04] Qu H., Zhang N., Shao R., Kaufman A., Mueller K.: Feature preserving distance fields. In *Volume Visualization and Graphics, 2004 IEEE Symposium on* (2004), IEEE, pp. 39–46. 2

[RT07] RONG G., TAN T.-S.: Variants of jump flooding algorithm for computing discrete voronoi diagrams. In *Voronoi Diagrams in Science and Engineering, 2007. ISVD'07. 4th International Symposium on* (2007), IEEE, pp. 176–181. 2

[SGG*06] SUD A., GOVINDARAJU N., GAYLE R., KABUL I., MANOCHA D.: Fast proximity computation among deformable models using discrete voronoi diagrams. *ACM Transactions on Graphics (TOG) 25*, 3 (2006), 1144–1153. 2

[SGGM06] SUD A., GOVINDARAJU N., GAYLE R., MANOCHA D.: Interactive 3d distance field computation using linear factorization. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games* (2006), ACM, pp. 117–124. 2

[Str99] STRAIN J.: Fast tree-based redistancing for level set computations. *Journal of Computational Physics 152*, 2 (1999), 664–686. 2

[TT97] TEICHMANN M., TELLER S.: Polygonal approximation of voronoi diagrams of a set of triangles in three dimensions. In *Tech Rep 766, Lab of Comp. Sci., MIT* (1997). 2

[VO98] VLEUGELS J., OVERMARS M.: Approximating voronoi diagrams of convex sites in any dimension. *International Journal of Computational Geometry & Applications 8*, 02 (1998), 201–221. 2

[WLXZ08] WU X., LIANG X., XU Q., ZHAO Q.: Gpu-based feature-preserving distance field computation. In *Cyberworlds, 2008 International Conference on* (2008), IEEE, pp. 203–208. 2

[YLW11] YIN K., LIU Y., WU E.: Fast computing adaptively sampled distance field on gpu. In *Pacific Graphics Short Papers* (2011), The Eurographics Association, pp. 25–30. 2

[ZGHG11] ZHOU K., GONG M., HUANG X., GUO B.: Data-parallel octrees for surface reconstruction. *Visualization and Computer Graphics, IEEE Transactions on 17*, 5 (2011), 669–681. 2