AHD: The Alternate Hierarchical Decomposition of Nonconvex Polytopes (Generalization of a Convex Polytope Based Spatial Data Model)

Rizwan Bulbul
Institute of Geoinformation and Cartography
Technical University of Vienna
Gusshausstrasse 27-29 E127
A-1040 Vienna, Austria
bulbul@geoinfo.tuwien.ac.at

Andrew U. Frank
Institute of Geoinformation and Cartography
Technical University of Vienna
Vienna, Austria
frank@geoinfo.tuwien.ac.at

***Abstract*—Robust convex decomposition, RCD, of polytopes is the convex decomposition of nonconvex polytopes using algorithms whose implementation is based on arbitrary precision arithmetic. Decomposing nonconvex polytopes using RCD can make the data representation model consistent enabling generalization with level of detail. Our approach, alternate hierarchical decomposition, AHD, for the decomposition of nonconvex polytopes with arbitrary genus, is a recursive approach whose implementation is robust, efficient and scalable to any dimension. Our approach decomposes the given nonconvex polytope with arbitrary genus into a set of component convex hulls, which are represented hierarchically in a tree structure, convex hull tree, CHT.**

Keywords-convex decomposition; hierarchial; polygon; convex hull tree

I. INTRODUCTION (REPRESENTATION OF GEOMETRY OF REGIONS)

An irregular subdivision of space contains nonconvex regions. A region is defined as a set of polygons [1], most of them may be nonconvex. The representation of geometry of (convex) regions as a collection of inequalities as $y \geq ax + b$ has been proposed in [2, 3]. Our approach for the representation of geometry of nonconvex regions is based on the representation of (convex) polytopes, which are defined by the intersection of finite set of half planes (half spaces in case of 3D). We use the representation of polytopes in dual space, which are represented by set of points and lines, depicting lines and the connection points of half planes in the Euclidean space respectively. In dual space, a convex polytope is defined as the convex hull of the dual points [4].

Our model for the representation of region is starting with the convex hull and removing convex parts to describe concavities (e.g. union of convex polytopes). The object representations based on simpler geometric structures are more easily handled [5] than complex structures. The algorithms for convex objects are simple and run faster than for arbitrary objects [6-9], for example, the decomposition of complex spatial objects into simple components considerably increases the query processing efficiency [6]. Therefore, the need arises to decompose nonconvex polytopes into simpler convex components.

The problem of decomposing a nonconvex object into its convex components is known as "convex decomposition" and has applications in diverse domains ranging from pattern recognition, motion planning and robotics, computer graphics and image processing, etc. [10-14]. The application of decomposition techniques in the spatial data modeling domain is sparse in literature, only few address the issue in the spatial database domain [6, 15]. There is a need of researching the decomposition techniques in the spatial data modeling domain, thus developing robust, efficient, and practical algorithms for the decomposition of complex spatial objects into simple components.

Our convex decomposition approach is simple in which the original input, nonconvex polytope with or without holes, is decomposed into its convex components that are convex hulls arranged hierarchically in a tree structure (details in section 3). Most of the decomposition solutions are based on the real RAM (Random Access Machine) model of computation. Our approach is robust as the underlying algorithms for recursively computing convex decompositions have been implemented using big numbers, which provide arbitrary precision. RCD of polytopes is the convex decomposition of nonconvex polytopes using algorithms whose implementation is based on arbitrary precision arithmetic. Decomposing nonconvex polytopes using RCD can make the data representation model consistent enabling generalization with level of detail. Our basic strategy is similar to the approach in [16] and the sum-difference approach [9, 17]. We use the QuickHull algorithm [18] for computation of convex hulls and all the procedures have been implemented as Haskell [19] modules. The solution is compact, efficient and robust because of the Haskell's built in list comprehensions, lazy evaluation, and support for big integers.

II. CLASSIFICATION OF APPROACHES TO POLYGON DECOMPOSITION

Polygon decomposition approaches are classified based on the following criteria [6, 10, 17];

A. Type of subpolygons (components)

Depending on application, different decompositions can result in variety of components, e.g, convex, star-shaped, or fixed orthogonal shapes.

B. Type of polygon being decomposed

Decompositions can also be classified on the basis of input polygons, e.g., closed or open, simple or connected, with or without holes, etc.

C. Relation of subpolygons

Decompositions that result in components, which do not overlap except at boundaries, are classified as partitioning. Other decompositions allowing component overlaps are classified as covering.

D. Objective function

Most of the decompositions are classified based on the objective function. The two broad categories are decompositions that minimize the number of convex components and decompositions that minimize computational complexity in terms of execution time.

E. Dimension specific or dimension independent

Decomposition approaches can also be classified depending on the input polygon dimension. The majority of the decomposition techniques in literature are dimension dependent and mostly applicable in 2-D and 3-D [12] cases only.

The spatial data representation model in [15], uses hierarchical convex based polygon decomposition approach like our approach for multi scale organization of vector data on spatial data servers. Their approach has a run time of $O\ (n \log^2 n)^a$ and is complex separately dealing holes from the rest of the polygon and no implementation details provided. The work of [17] focuses on decomposition of orthogonal polygons. Another hierarchical strategy in [10] decomposes a polygon with or without holes into its approximate convex components. Their algorithm results smaller set of approximately convex components efficiently in $O\ (nr)^b$. In [12] a triangulation based algorithm for the problem of partitioning a polytope in R^3 has been proposed. The algorithm requires $O\ (n+r^2)$ space and runs in $O\ ((n+r^2)\ log\ r)$ time. A similar work in the pattern recognition domain has been done in [20], presenting a convex hull based algorithm for concavity tree extraction from pixel grid of 2-D shapes. Most of the algorithms for decomposition are optimized for one of the criteria [6], depending on application needs. Table 1 shows a characterization of our approach based on the aforementioned criteria.

[a] n is the number of vertices

[b] r is the number of notches/delta regions/concavities

TABLE I. CHARACTERIZATION OF OUR DECOMPOSITION TECHNIQUE

Criteria	Our Approach
Type of components	Convex
Type of input	Polytope with or without holes
Relation of components	Covering- as the component convex hulls at level *l+1* are contained within the convex hull at level *l*
Objective function	Does not fall to any category, but our objective is to achieve generalization of data modeling framework using convex polytopes.
Input dimension	Dimension independent

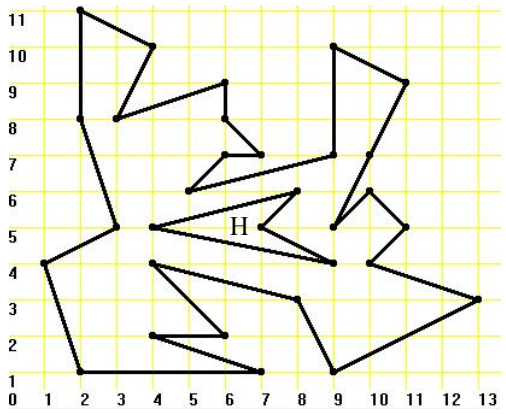
III. OUR APPROCH (ALTERNATE HIERACHIAL DECOMPOSITION)

Our approach, AHD, for the decomposition of nonconvex polytopes is a recursive approach whose implementation is robust, efficient and scalable to any dimensions. Our approach decomposes the given nonconvex polytope into a set of component convex hulls that are represented hierarchically in a CHT (related concepts are *convex deficiency tree* [18] and *concavity tree* [20]). CHT is a tree of convex hulls in which the root node contains the convex hull of the input polytope, non-leaf nodes contain convex hulls of the nonconvex delta regions (notches) of the nonconvex structure whose convex hull is represented by parent node. Leaf nodes contain the convex hulls of the convex delta regions. The convex hulls in CHT at different levels are assigned positive (+) and negative (-) signs alternately as shown in Fig. 2. The convex hulls at even and odd levels have positive and negative signs receptively. The positively signed convex hulls represent the convex regions to be included while the negatively signed convex hulls represent convex regions to be excluded.

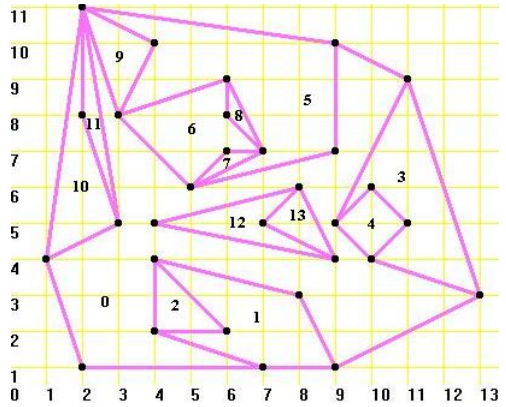
The AHD consists of two steps: (1) convex hull computation, and (2) delta region/s extraction (using symmetric difference). The process is then recursively applied to the delta regions until the input region is convex. The necessary invariants for our approach are;

- The input is a (nonconvex) polytope with or without holes represented in the dual space, which is a closed region of edges.
- An edge is a pair of points and each point has homogeneous big integer coordinates.
- The boundary edges are oriented in counter clockwise direction while the hole edges are oriented clockwise.
- In CHT, the convex hulls at level *l+1* are inside their respective parent at level *l*.

In order to explain our approach, we present here an example that demonstrates the decomposition steps and shows the output CHT. Suppose that we are given a 2-D nonconvex polytope with a hole (H) in dual space (as shown in Fig. 1a) as input. The application of AHD results in the decomposition of the input into its component convex hulls (Fig. 1b). The complete AHD process is illustrated in Fig. 6.



(a)



(b)

Figure 1. Input polytope (a) and its decomposition (b)

The output is a hierarchical representation of convex hulls as CHT as shown in Fig. 2. The data structure used for CHT representation is an arbitrary tree, every node of which contains a convex hull. We have implemented CHT in Haskell using *Rose tree* [21], which recursively defines an arbitrary tree as: Tree = **Node** CHull [Tree]. Thus every node contains a convex hull and may contain one or more children trees represented as a list of Tree with the associated node. The tree that has no children is leaf node defined as leafNode = **Node** CHull [ ].

IV. THE ALGORITHM IN PSEUDOCODE

Our AHD process has the function "build", which decomposes the given input polytope into its convex hull components and the component hulls are hierarchically arranged into the CHT. The pseudocode for "build" function is as given in Fig. 3. The pseudocode of the algorithm for computing the QuickHull [18] is given in Fig. 4.

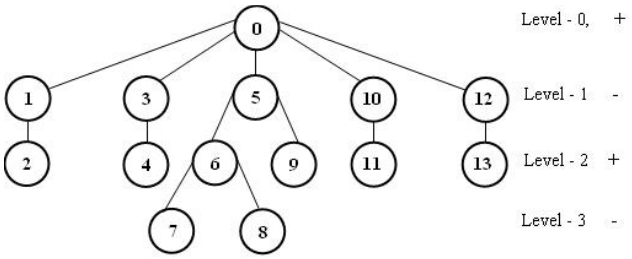


Figure 2. CHT for the demonstration example of Fig. 1

Input: A list *pe* of polytope edges having counterclockwise direction
1: $t \leftarrow$ Node [] [] // Initialize t as an empty convex hull tree
2: **if** *pe* is empty
3: **then return** t
4: **else if** *pe* contains only a single edge
5: **then**
6: *ch* ← list of points containing unique endpoints of the single edge
7: $t \leftarrow$ Node *ch* [] *// a tree containing single node*
8: **return** t
9: **else**
10: *ch* ← convex hull of *pe*
11: *che* ← list of convex hull edges of *pe*
12: *dre* ← list of delta region edges which are not convex hull edges
13: *dhp* ← list of delta region points which are on hull
14: *dr* ← list of closed delta regions *// region is a list of edges*
15: **for** all delta regions in *dr*
16: *ct* ← go to 1 *//each region in dr results in a children tree recursively*
17: $t \leftarrow$ Node *ch* [*ct*]
18: **return** t
Output: A convex hull tree t

Figure 3. Pseudocode of the algorithm 'build' that populates the CHT

Input: A list p of polytope points
1: Initialize *ch* as an empty list of convex hull points
2: $x \leftarrow$ point in p with max x-coordinate value
3: $y \leftarrow$ point in p with max y-coordinate value
4: *s1* ← list of points in p strictly above line xy
5: *s2* ← list of points in p strictly above line yx
6: *ch* ← ([*x*] + **qHull** (*x*, *y*, *s1*)) + [*y*] + **qHull** (*y*, *x*, *s2*))
7: **return** *ch*
Output: A list *ch* of convex hull points

(a)

Input: Two end points of a line a and b and a list S of points above that line ab
1: Initialize *sh* as an empty list of sub hull points
2: **if** S is empty
3: **then return** []
4: **else**
5: c ← farthest point of S from line ab
6: *partA* ← points in S strictly above line ac
7: *partB* ← points in S strictly above line cb
8: *chA* ← qHull (*a*, *c*, *partA*)
9: *chB* ← qHull (*c*, *b*, *partB*)
10: *sh* ← *chA* + [*c*] + *chB*
11: return *sh*
Output A list of sub hull points *sh*

(b)

Figure 4. Pseudocode of the algorithm quickhull (a) and its helping function qHull, which computes the sub hulls of partitions recursivley

Once the decomposition is done, the resulting CHT can be processed to retrieve the decomposed object from its convex hull components at any level of detail depending on application. The pseudocode of the algorithm for complete processing of a CHT, which retrieves the original object, is given in Fig. 5.

```
Input:  A convex hull tree t, Node x xts : x is convex hull at root node and xts is list of
        children trees
1:      Initialize pe , cr as empty list of edges
2:      if t is empty  // Node [] []
3:        then return pe
4:      else if xts is empty  // a single node tree
5:        then
6:             pe ← list containing edges of convex hull x
7:             return pe
8:      else
9:             pe ← list containing edges of convex hull x
10:            for all child trees in xts
11:                cr ← cr + go to 1 // each children tree in xts results in a component reg
        recursively
12:            mr ← merge regions in cr into the region pe
13:            pe ← mr
14:            return pe
Output: A list pe of polytope edges having counterclockwise direction
```

Figure 5. Pseudocode of the algorithm 'process' that processes the CHT to extract represented region

V. IMPLEMENTATION IN HASKELL

The algorithms have been implemented as Haskell modules. The selection of Haskell provides ease of implementation giving simple and compact code. This is possible because of Haskell's built in "list" data structure and its associated higher order functions, lazy evaluation and support for big numbers.

Most computational algorithms are designed with the assumption that all the numerical computations involved give exact results. In reality, the results of numerical computations in current systems are not exact. Results are rounded to the nearest approximates, and inconsistencies can occur because of the round-off errors, the problem called nonrobustness. Thus the accuracy of numerical computations in current systems is limited by the finite precision arithmetic. The problem is more severe in case of geometric algorithms [22], as these involve both metric and topological processing. The rounding errors in metric computations because of the fixed precision arithmetic can cause inconsistencies in topological processing, leading to inaccurate geometric computation models.

In the polygon decomposition domain, the issue of inaccuracies in numerical computations due to fixed precision arithmetic is considered in [9, 20].

To achieve a robust implementation of AHD, we are using the homogeneous big integers for coordinate representation of half planes in dual space. The viability of using big integers for metric processing has been experimented in [23]. It has been found that contrary to the widespread belief, programming with big integers is straightforward and easier than using conventional integer or floating point data types. Metric computations with big numbers are conceptually simpler and need no testing for approximation problems.

VI. SOLUTION CHARACTERISTICS

A. Robustness

The implementation of our approach avoids the problem of numerical nonrobustness [24-27] by using exact algebra. This has been achieved through use of big integers allowing arbitrary precision arithmetic. Thus, topological inconsistencies are avoided, as the underlying algorithms have been implemented keeping in mind the arbitrary precision arithmetic for numerical computations involved.

B. Level of Detail

Depending on the applications, the complete decomposition of nonconvex polygons to convex polygons may not be needed [10]. In such cases approximations are used by halting the decomposition process using some threshold. Our approach is flexible in that the decomposition process can be stopped at any level depending on application thus allowing level of detail approximation.

C. Dimension independence

To the best of our knowledge, the algorithms for decomposition problem are not dimension independent as most work for 2-D and some deal with 3-D objects. Although for demonstration purposes we are using 2-D polytopes, our solution is easily extendable to n-dimensional polytopes. This is achievable if the procedure for computing convex hull is dimension independent. For this any of the existing dimension independent convex hull algorithms [28, 29] can be used.

D. Number of Components

The solutions to the problem of decomposing a nonconvex polygon into its optimal (minimum) number of convex components vary depending on whether Steiner points allowed [10]. Although our solution was not aiming to optimize the number of resultant components, however we found that our solution is not as bad as the mostly used decompositions for near optimal number of components. The number of components is the number of CHT nodes, which is 13 and partitioning the example in Fig. 1 results in 14 convex components.

E. Complexity Analysis

Given a nonconvex polytope p with n points, the convex hull and delta regions of p can be computed in $O(n \log n)$ and in liner time, respectively. If r is the number of delta regions of p then complexity becomes $O(r \times n \log n)$. However, size of r is negligibly smaller than n for higher values of n. Thus, the worst case scenario can not be $O(n^2 \log n)$ and our approach has an average time complexity of $O(n \log n)$.

VII. SUMMARY AND FUTURE WORK

The implementation of our decomposition approach is efficient, robust, and simple. A dimension independent implementation of our approach has already been implemented using the incremental convex hull algorithm [28] and the work is currently in writing phase. In addition, the CHT has been tested to support basic Boolean operations of intersection, union and symmetric difference. The results are in draft to be presented soon. In future, we will test the support for operations for dimension independent implementation of CHT and experiment with real data sets.

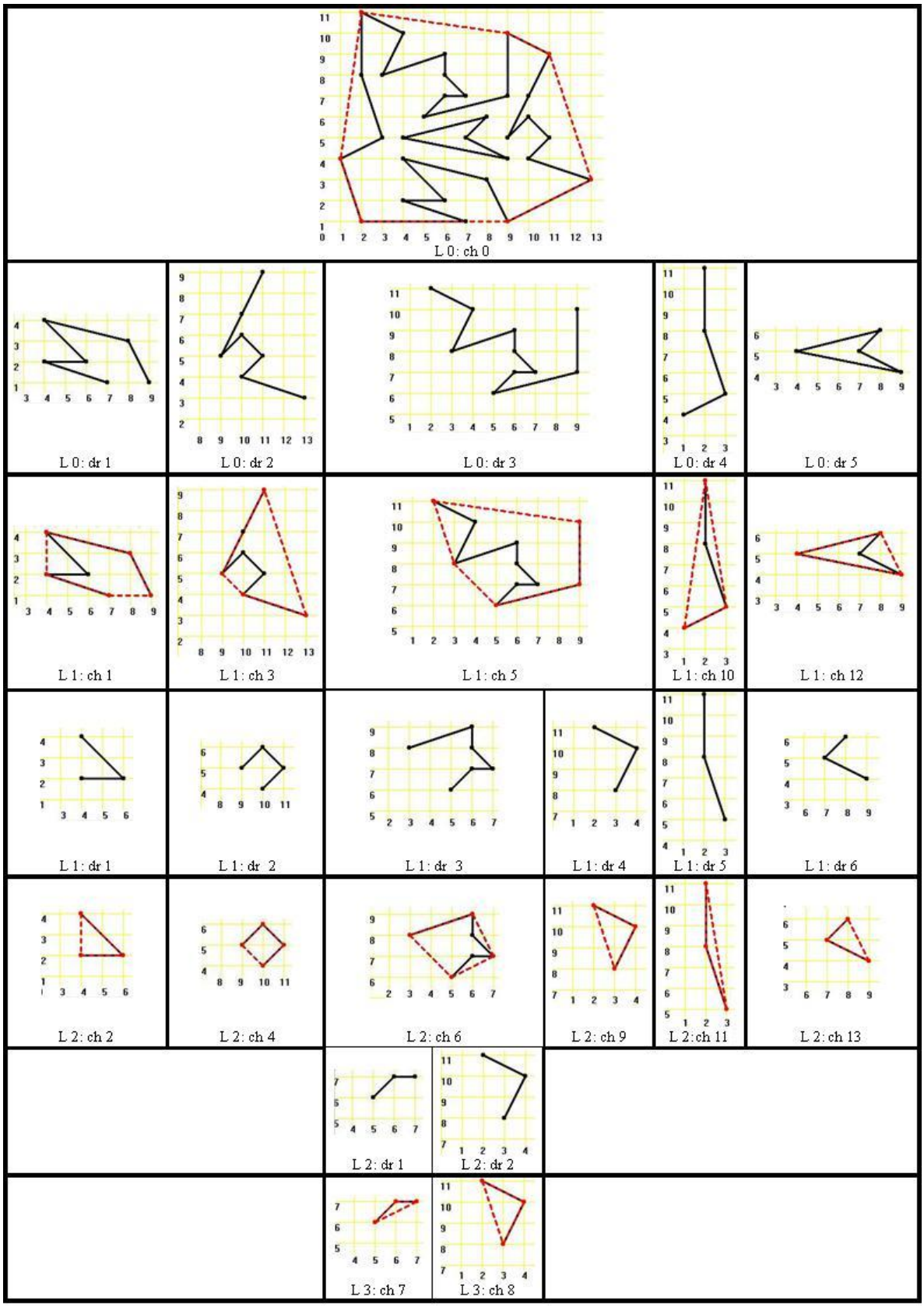


Figure 6. The AHD process steps. L represents the level, dr represents the delta region and ch represents the convex hull

REFERENCES

[1] P. Rigaux, M. Scholl, and A. Voisard, *Spatial databases with application to GIS*: Morgan Kaufmann Publishers Inc., 2002.

[2] O. Gunther, *Efficient structures for geometric data management*: Springer-Verlag New York, Inc., 1988.

[3] O. Gunther and E. Wong, "Convex polyhedral chains: a representation for geometric data." vol. 21: Butterworth-Heinemann, 1989, pp. 157-164.

[4] M. d. Berg, M. v. Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, First ed.: Springer-Verlag, 1997.

[5] J. Fernández, L. Cánovas, and B. Pelegrín, "Algorithms for the decomposition of a polygon into convex polygons," *European Journal of Operational Research,* vol. 121, pp. 330-342, 2000.

[6] H.-P. Kriegel, H. Horn, and M. Schiwietz, "The performance of object decomposition techniques for spatial query processing," in *Data structures and efficient algorithms*, 1992, pp. 104-123.

[7] B. Schachter, "Decomposition of Polygons into Convex Sets." vol. 27: IEEE Computer Society, 1978, pp. 1078-1082.

[8] C. L. Bajaj and T. K. Dey, "Convex decomposition of polyhedra and robustness," *SIAM Journal on Computing,* vol. 21, pp. 339 - 364 1992.

[9] P. Leonidas, "Decomposition problems in computational geometry," in *Department of Computer Sceinces*. vol. PhD Thesis: Princeton University, 1992.

[10] J.-M. Lien and N. M. Amato, "Approximate convex decomposition of polygons," *Computational Geometry,* vol. 35, pp. 100-123, 2006.

[11] J.-M. Lien and N. M. Amato, "Approximate convex decomposition," in *Proceedings of the twentieth annual symposium on Computational geometry* Brooklyn, New York, USA: ACM, 2004.

[12] B. Chazelle and L. Palios, "Triangulating a nonconvex polytope," *Discrete and Computational Geometry,* vol. 5, pp. 505-526, 1990.

[13] R. Liu, H. Zhang, and J. Busby, "Convex hull covering of polygonal scenes for accurate collision detection in games," in *Proceedings of graphics interface 2008* Windsor, Ontario, Canada: Canadian Information Processing Society, 2008.

[14] X. Jianning, "Morphological Decomposition of 2-D Binary Shapes Into Modestly Overlapped Octagonal and Disk Components," *Image Processing, IEEE Transactions on,* vol. 16, pp. 337-348, 2007.

[15] T. Ai, Z. Li, and Y. Liu, "Progressive Transmission of Vector Data Based on Changes Accumulation Model," in *Developments in Spatial Data Handling*, 2005, pp. 85-96.

[16] S. B. Tor and A. E. Middleditch, "Convex Decomposition of Simple Polygons." vol. 3: ACM, 1984, pp. 244-265.

[17] J. M. Keil, "Polygon Decomposition," in *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, Eds.: Elsevier, 2000, pp. 491-518.

[18] J. O'Rourke, *Computational Geometry in C (Cambridge Tracts in Theoretical Computer Science)*: Cambridge University Press, 1998.

[19] S. Jones, *Haskell 98 Language and Libraries: The Revised Report*: {Cambridge University Press}, 2003.

[20] O. E. Badawy and M. S. Kamel, "Hierarchical representation of 2-D shapes using convex polygons: a contour-based approach," *Pattern Recognition Letters,* vol. 26, pp. 865 - 877, 2005

[21] R. Bird, *Introduction to functional programming using Haskell*. Harlow [u.a.]: Prentice Hall Europe, 2000.

[22] K. Y. Chee, "Robust geometric computation," in *Handbook of discrete and computational geometry*: CRC Press, Inc., 1997, pp. 653-668.

[23] R. Bulbul and A. U. Frank, "Big Integers for GIS: Testing the Viability of Arbitrary Precision Arithmetic for GIS Geometry," in *12th AGILE International Conference on Geographic Information Science* Hannover, Germany, 2009.

[24] W. R. Franklin, "Cartographic errors symptomatic of underlying algebra problems," in *International Symposium on Spatial Data Handling*, Zurich, Switzerland, 1984, pp. 190-208.

[25] K. Mehlhorn and S. Näher, *The LEDA Platform of Combinatorial and Geometric Computing*: Cambridge University Press, 1999.

[26] C. Li, S. Pion, and C. K. Yap, "Recent progress in exact geometric computation," *Journal of Logic and Algebraic Programming,* vol. 64, pp. 85-111, 2005.

[27] "CGAL, Computational Geometry Algorithms Library."

[28] F. Karimipour, A. U. Frank, and M. R. Delavar, "An operation-independent approach to extend 2D spatial operations to 3D and moving objects," in *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems* Irvine, California: ACM, 2008.

[29] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software (TOMS),* vol. 22, pp. 469 - 483, 1996