Verification of Partitions of 2d and 3d Objects

Leonidas Palios

Department of Computer Science, University of Ioannina, 45110 Ioannina, Greece

Abstract

We consider the problems of deciding whether a given collection of polygons (polyhedra resp.) forms (i) a partition or (ii) a cell complex decomposition of a given polygon (polyhedron resp.). We describe simple $O(n \log n)$-time and $O(n)$-space algorithms for these problems, where n is the total description size of the input. If, in the input, vertices are referenced by means of indices to an array of distinct vertices, then our cell complex decomposition verification algorithms run in $O(n)$ time.

1. Introduction

In recent years, there has been growing interest in algorithms that enable us to check the output of a program. This issue has been considered from different viewpoints, and the research yielded results ranging from characterizations of problems that are checkable [2] to special-purpose checkers. In particular, in computational geometry, this issue proves to be crucial as the newer and more efficient algorithms are more and more complicated. Usually, the authors of an algorithm for computing some geometric object describe properties of the object which can be used to verify the correctness of the computation (see e.g. [1] and [4] for 2d and 3d triangulations of point sets). Often, however, more machinery is needed.

The first verification algorithms ("checkers") were provided by Mehlhorn *et al.* who defined the properties that a checker should have (correctness, simplicity, efficiency) and described checkers for convex polyhedra and convex hulls [5]; they have also studied checkers for trapezoidal decompositions, planar point location structures for trapezoidal decompositions, and Voronoi and Delaunay diagrams. Devillers *et al.* extended the notion of checkers and provided checkers for convex polytopes in two and higher dimensions, and for various types of planar subdivisions [3]. Their algorithms run in linear time, assuming that the input is a 2d or a 3d ordered geometric graph.

Email address: palios@cs.uoi.gr (Leonidas Palios).

In this paper, we present simple and efficient verification algorithms for partitions and cell complex decompositions of simple polygons and polyhedra. The algorithms rely on appropriately matching the edges of the 2d decompositions and the facets of the 3d decompositions, and run in $O(n \log n)$ time using $O(n)$ space, where n is the total size of the input. If, in the input, vertices are referenced by means of indices to an array of distinct vertices, then our cell complex decomposition verification algorithms run in optimal $O(n)$ time.

Note: In the following, a polygon or a polyhedron is understood to be a *simple* one.

2. The Two-dimensional Case

We assume that each polygon in the input is described by the sequence of its vertices along its boundary, where each vertex is given by its coordinates.

2.1. *Verification of a partition of a polygon*

Lemma 1 *Let P and $\mathcal{C}$ be a polygon and a collection of polygons respectively. Additionally, let $E_I = \{e - \partial P \mid e \text{ is an edge of a polygon in } \mathcal{C}\}$ and $E_B = \{e \cap \partial P \mid e \text{ is an edge of a polygon in } \mathcal{C}\}$. Then, the collection $\mathcal{C}$ forms a partition of P if and only if*

(i) the set E_I of (parts of) edges of the polygons in $\mathcal{C}$ can be partitioned into sets S and S' such that
- $\bigcup_{s \in S} closure(s) = \bigcup_{s \in S'} closure(s)$,

- *the closures of the segments in S and S' define a partition of $\bigcup_{s \in S}$ closure(s), and*
- *for each (part of) edge e in S (resp. S'), if $S'[e]$ (resp. $S[e]$) is the set of segments of S' (resp. S) that are collinear with and intersect e, then, locally around e, the interior of the polygon in $\mathcal{C}$ with edge e and the interiors of the polygons in $\mathcal{C}$ whose edges contributed the elements of $S'[e]$ (resp. $S[e]$) lie on opposite sides of the line supporting e;*

(ii) the closures of the segments in E_B form a partition of the boundary ∂P of P, and for each (part of) edge e in E_B, locally around e, the interior of the polygon in $\mathcal{C}$ with e as an edge and the interior of P lie on the same side with respect to the line supporting e.

The partition verification algorithm applies Lemma 1. It uses an (initially empty) array A of size equal to the total number of edges of the polygons in $\mathcal{C}$ and of the polygon P.

2d Partition Verification Algorithm

```
1. Orient the boundary of polygon $P$ and the boundaries of the polygons in $\mathcal{C}$ in a compatible fashion (e.g., the boundary is traversed in a ccw fashion).
2. For each polygon $Q$ in $\mathcal{C}$ do
     for each edge $\overrightarrow{uv}$ of $Q$ do
       if $u$ is lex-smaller than $v$
       then add the entry $(u, v, +1)$ in $A$;
       else add the entry $(v, u, -1)$ in $A$;
3. For each edge $\overrightarrow{uv}$ of $P$ do
     if $u$ is lex-smaller than $v$
     then add the entry $(u, v, -1)$ in $A$;
     else add the entry $(v, u, +1)$ in $A$;
4. Sort the entries $(u_i, v_i, \pm 1)$ of the array $A$ by slope of the line $u_i v_i$ and, in case of ties, lexicographically taking into account the coordinates of the vertices $u_i$ and $v_i$.
5. For each slope separately, traverse the related entries of $A$ verifying that those with "+1" and those with "−1" partition the same set of segments. If this terminates successfully, then the collection $\mathcal{C}$ of polygons defines a partition of the polygon $P$, otherwise it does not.
```

Time complexity. Steps 1, 2, and 3 can be completed in time linear in the total description size n of the input, while Step 4 takes $O(n \log n)$ time. Step 5 takes $O(n)$ time: for each slope, the sorting of the entries implies that each new entry with "+1" either is the extension of the latest entry with "+1" or starts a new segment in the union of the "+1"-entries (if not, $\mathcal{C}$ does not form a partition of P), and similarly for the "−1"-entries. The algorithm uses $O(n)$ space.

2.2. *Verification of a cell complex decomposition of a polygon*

Lemma 2 *Let P and $\mathcal{C}$ be a polygon and a collection of polygons respectively. Then, the collection $\mathcal{C}$ forms a cell complex decomposition of P if and only if the set of edges of the polygons in $\mathcal{C}$ can be partitioned into two sets E_I and E_B such that*

(i) for each edge e in E_I there exists exactly one other edge, say d, in E_I such that $e = d$ and, locally around e, the interiors of the polygons in $\mathcal{C}$ with edges e and d lie on opposite sides of the line supporting e;

(ii) the edges in E_B form a partition of the boundary ∂P of P and for each edge e in E_B, locally around e, the interior of the polygon in $\mathcal{C}$ with edge e and the interior of P lie on the same side with respect to the line supporting e.

The cell complex verification algorithm applies Lemma 2; it too uses an (initially empty) array A of size equal to the total number of edges of the polygons in $\mathcal{C}$.

2d Cell Complex Verification Algorithm

```
1. Collect all the vertices and assign to each one of them a distinct positive integer $id(\ )$.
2. Orient the boundary of polygon $P$ and the boundaries of the polygons in $\mathcal{C}$ in a compatible fashion.
3. For each polygon $Q$ in $\mathcal{C}$ do
     for each edge $\overrightarrow{uv}$ of $Q$ do
       if $u$ is lex-smaller than $v$
       then add $(id(u), id(v), +1)$ in $A$;
       else add $(id(v), id(u), -1)$ in $A$;
4. Sort the array $A$ lexicographically.
5. Traverse the sorted array $A$ deleting matching entries (i.e., $(k, k', -1)$ and $(k, k', +1)$), which now appear next to each other.
6. Collect the unmatched entries and by applying a depth-first traversal construct the graph that these entries form. If this graph is a closed path identical to the boundary of the polygon $P$, then the collection $\mathcal{C}$ of poly-
```

gons defines a cell complex decomposition of P, otherwise it does not.

Time complexity. If n is the total description size of the input, Step 1 can be executed in $O(n \log n)$ time by means of a balanced binary search tree to determine identical vertices. Steps 2 and 3 take $O(n)$ time; so does Step 4 (by using radix sorting), and Steps 5 and 6. Thus, the algorithm takes a total of $O(n \log n)$ time. The space required by the algorithm is $O(n)$.

Observe that all the steps of the above algorithm but Step 1 take $O(n)$ time. Moreover, if the input consists of the list V of distinct vertices of the polygon P and of the polygons in the collection $\mathcal{C}$, followed by the vertices of each polygon, where each vertex is referenced by its index to the list V, then we do not need to execute Step 1. Such a representation is commonly used, and is easily produced by partitioning programs. Then, it can be determined whether the collection $\mathcal{C}$ forms a cell complex decomposition of P in $O(n)$ time using $O(n)$ space.

3. The Three-dimensional Case

We assume that each polyhedron in the input is described by a list of its facets, each represented by the sequence of its vertices along its boundary, where each vertex is given by its coordinates.

Our verification algorithms rely on two lemmata similar to Lemma 1 and Lemma 2.

3.1. *Verification of a partition of a polyhedron*

Lemma 3 *Let P and $\mathcal{C}$ be a polyhedron and a collection of polyhedra respectively. Additionally, let $F_I = \{f - \partial P \mid f$ is a facet of a polyhedron in $\mathcal{C}\}$ and $F_B = \{f \cap \partial P \mid f$ is a facet of a polyhedron in $\mathcal{C}\}$. Then, the collection $\mathcal{C}$ forms a partition of P if and only if*

(i) the set F_I of (parts of) facets of the polyhedra in $\mathcal{C}$ can be partitioned into two sets S and S' such that the closures of the polygons in each of these sets defines a partition of $\bigcup_{s \in S}$ closure(s), and for each (part of) facet f in S (S' resp.), the interior of the polyhedron of $\mathcal{C}$ with facet f and the interiors of the polyhedra of $\mathcal{C}$ with facets the polygons of S' (S resp.) which intersect with f lie (locally around f) on opposite sides of the plane supporting f;

(ii) the closures of the polygons in F_B form a partition of the boundary ∂P of P, and for each (part of) facet f in F_B, locally around f, the interior of the polyhedron of $\mathcal{C}$ with facet f and the interior of P lie on the same side with respect to the plane supporting f.

In light of Lemma 3, we have:

3d Partition Verification Algorithm

```
1. Orient the facets of the polyhedron $P$ and of
   the polyhedra in $\mathcal{C}$ in a compatible fashion
   (i.e., in counterclockwise order as seen from
   outside the polyhedron).
2. For each polyhedron $Q$ in $\mathcal{C}$ do
     for each facet $f$ of $Q$ do
       $u \leftarrow$ the lex-smallest vertex of $f$;
       if $f$'s boundary forms a left turn at $u$
       then add the entry $(f, +1)$ in $A$;
       else add the entry $(f, -1)$ in $A$;
3. For each facet $f$ of $P$ do
     $u \leftarrow$ the lex-smallest vertex of $f$;
     if $f$'s boundary forms a left turn at $u$
     then add the entry $(f, -1)$ in $A$;
     else add the entry $(f, +1)$ in $A$;
4. Sort the entries $(f_i, \pm 1)$ of the array $A$ by
   slope of the plane supporting $f_i$ and, in case
   of ties, lexicographically on the second field
   of the entry.
5. For each slope separately, verify that the re-
   lated facet records with "+1" and those with
   "−1" partition the same polygonal regions.
   If this matching terminates successfully, then
   the collection $\mathcal{C}$ defines a partition of the
   polyhedron $P$, otherwise it does not.
```

Time complexity. Steps 1, 2, and 3 can be completed in time linear in the total description size n of the input. Step 4 takes $O(n \log n)$ time. Step 5 can also be completed in $O(n \log n)$ time: for each different plane slope, collect the entries with "+1" as second field, and apply on the associated facets Steps 2 and 4 of the 2d partition verification algorithm (see Section 2.1); Step 5 of that algorithm is applied next, except that the difference of the union of the "+1" and the "−1" entries is computed and used to form a graph; the same procedure is applied to the entries of the array A with "−1" as second field; finally, the two resulting graphs are checked to see whether they are identical. Thus, the algorithm takes a total of $O(n \log n)$ time. The algorithm uses $O(n)$ space.

3.2. *Verification of a cell complex decomposition of a polyhedron*

Lemma 4 *Let P and $\mathcal{C}$ be a polyhedron and a collection of polyhedra respectively. Then, the collection $\mathcal{C}$ forms a cell complex decomposition of P if and only if the set of facets of the polyhedra in $\mathcal{C}$ can be partitioned into two sets F_I and F_B such that*

(i) for each facet f in F_I there exists exactly one other facet, say f', in F_I such that $f = f'$ and, locally around f, the interiors of the polyhedra in $\mathcal{C}$ with edges f and f' lie on opposite sides of the plane supporting f;

(ii) the edges in F_B form a partition of the boundary ∂P of P and for each facet f in F_B, locally around f, the interior of the polyhedron of $\mathcal{C}$ with facet f and the interior of P lie on the same side with respect to the plane supporting f.

The cell complex verification algorithm applies Lemma 4. It uses two auxiliary arrays A and B, which are initially empty; it works as follows:

3d Cell Complex Verification Algorithm

```
1. Collect all the vertices and assign to each one
   of them a distinct positive integer $id(\ )$.
2. Orient the facets of the polyhedron $P$ and of
   the polyhedra in $\mathcal{C}$ in a compatible fashion.
3. For each polyhedron $Q$ in $\mathcal{C}$ do
     for each facet $f$ of $Q$ do
       $a \leftarrow$ the lex-smallest vertex of $f$;
       $b \leftarrow$ the lex-largest vertex of $f$;
       $c \leftarrow$ $f$'s vertex farthest away from the
           line $\overline{ab}$ (and lex-smallest, if ties);
       if $f$'s boundary is directed $a \rightsquigarrow c \rightsquigarrow b$
       then add $((id(a), id(b), id(c)), +1)$ in $A$;
       else add $((id(a), id(b), id(c)), -1)$ in $A$;
4. Sort the array $A$ lexicographically.
5. Traverse the sorted array $A$, verify that
   matched entries (which now appear next to
   each other) correspond to matching facets
   and delete them.
6. For each facet $f$ whose entry in $A$ has not
   been matched do
     for each edge $\overrightarrow{uv}$ of $f$ do
       if $u$ is lex-smaller than $v$
       then add $(id(u), id(v), +1, f)$ in $B$;
       else add $(id(v), id(u), -1, f)$ in $B$;
7. Sort the array $B$ lexicographically.
8. If the entries in $B$ do not appear in matching
   pairs (i.e., $(k, k', -1, f)$ and $(k, k', +1, f')$),
   then the collection $\mathcal{C}$ of polyhedra does not
   define a cell complex decomposition of the
   given polyhedron $P$.
9. For each matching pair $(id(u), id(v), -1, f)$
   and $(id(u), id(v), +1, f')$ in $B$ do
     if the two facets $f, f'$ are not coplanar
     then make $u$ and $v$ adjacent;
10. Apply a depth-first traversal and check that
    the graph formed matches the edge skeleton
    of the polyhedron $P$. If it does, then the col-
    lection $\mathcal{C}$ of polyhedra defines a cell complex
    decomposition of $P$, otherwise it does not.
```

Time complexity. Step 1 takes $O(n \log n)$ time, as described in Section 2.2. Steps 2, 3, 5, and 6 take $O(n)$ time. Steps 4 and 7 can be completed in linear time as well by using radix sorting. Finally, Steps 8, 9, and 10 also take linear time. Thus, the algorithm requires a total of $O(n \log n)$ time. The space required by the algorithm is $O(n)$.

Similarly to the two-dimensional case, if the polyhedron P and the polyhedra in the collection $\mathcal{C}$ are represented by a list of facets, each described by a sequence of indices to an array of distinct vertices which indicates the order of the vertices around the boundary of the facet, then it can be determined whether the collection $\mathcal{C}$ forms a cell complex decomposition of P in time and space linear in the total description size of P and $\mathcal{C}$.

References

[1] D. Avis and H. El Gindy, Triangulating point sets in space, *Discrete and Computational Geometry* **2**, 99–111, 1987.

[2] M. Blum and S. Kannan, Designing programs that check their work, *Journal ACM* **42(1)**, 269–291, 1995.

[3] O. Devillers, G. Liotta, F.P. Preparata, and R. Tamassia, Checking the convexity of polytopes and the planarity of subdivisions, *Computational Geometry: Theory and Applications* **11(3-4)**, 187–208, 1998.

[4] H. Edelsbrunner, F. Preparata, and D. West, Tetrahedralizing point sets in 3 dimensions, *Journal of Symbolic Computation*, **10**, 335–347, 1990.

[5] K. Mehlhorn, S. Näher, T. Schilz, S. Schirra, M. Seel, R. Seidel, and C. Uhrig, Checking geometric programs or Verification of geometric structures, *Proc. 12th Symp. on Computational Geometry*, 159–165, 1996.