Volume Queries in Polyhedra

John Iacono* and Stefan Langerman**

Department of Computer Science
Rutgers University
New Brunswick, NJ, USA

Abstract. We present a simple and practical data structure for storing a (not necessarily convex) polyhedron P which can, given a query surface S cutting the polyhedron, determine the volume and the area of the portion of the polyhedron above S. The queries are answered in a time linear in the size (complexity) of S. The space and preprocessing time for this data structure are linear in the size of P. We also present an intermediary data structure for planar graphs which is of independent interest.

1 Introduction: Polygons

This paper studies a very natural generalization of the following problem in R^2: Given a simple polygon P, construct a data structure which, given a query chord c (a line segment cutting P into exactly two pieces, whose boundaries are on edges of P), returns the area of the polygon above c.

In [2] (also [3]), a solution is proposed for convex polygons: consider the polygon P with edges $e_1, ..., e_n$ oriented in clockwise order starting at the lowest vertex of P, and let $a_i = a(e_i)$ be the signed area of the trapezoid defined by the edge e_i and its projection on the x axis. The area $a(e_i)$ is negated if the edge e_i is a bottom edge, that is, directed towards the left. We call $a(e_i)$ the projective area of e_i. The area of P is simply $\sum_{i=1}^{n} a_i$. Suppose that the chord c has its endpoints on the edges e_j and e_k. Let a'_j and a'_k be the projective areas of the lower portions of e_j and e_k defined by c. The area of P above c is $\sum_{i=j}^{k} a_i - a'_j - a'_k - a(c)$. By storing $s_t = \sum_{i=1}^{t} a_i$ for $t = 1, ..., n$, we can compute $\sum_{i=j}^{k} a_i = s_k - s_{j-1}$ in constant time, and so we get:

Theorem 1 *Given a convex polygon P with n vertices, there is a data structure that after $O(n)$ preprocessing time can return the area of P above a query chord c in constant time.*

For non convex polygons, [3] describes a data structure that requires $O(n \log n)$ preprocessing time and that give the area of the polygon above a query chord in $O(\log n)$ time. Recently, in [1], the authors notice that due to the fact that the query chord doesn't cross any edge of the polygon, the algorithm used for Theorem 1 works as is. They also show other applications of the structure.

* `iacono@cs.rutgers.edu`, Research supported by NSF grant CCR-9732689
** `lfalse@cs.rutgers.edu`

2 Polyhedra

We now discuss a three-dimensional generalization to the polygon problem. It is interesting to note that the ideas presented here could be used to extend the data structure to higher dimensions.

In order to generalize the problem to 3 dimensions, we first need to generalize the concept of a chord. In our setting, we have a polyhedron P with n vertices. A *query surface* S is a polyhedral surface separating P into exactly two pieces, and the boundary of S lies on the surface of P. Note that this implies that the description of S will be at least as large as the number of edges of P crossed by the boundary of S. On the other side, this is usually likely to be much smaller than n. A volume query on a surface s will return the volume of the portion of P lying above s. We will show the following:

Theorem 2 *Given a polyhedron of size n, there is a structure that with $O(n)$ preprocessing time will answer volume queries in time linear in the size of the query surface.*

The idea will be similar to the 2-dimensional case. First we define the (signed) *projective volume* $v(f)$ of a polygon f to be the volume of the polyhedron defined by f and its projection on the plane $z = 0$. For a facet f of P, $v(f)$ will be negated if the inside of P is above f. The volume of P can then be expressed as the sum for all facets f of P of $v(f)$.

So, in order to answer a volume query for a surface s, we need to

(i) compute the sum of $v(f)$ for all facets f strictly above s,
(ii) add the projective volume of all the upper portions of the facets that are cut by s and
(iii) subtract the projective volume of all the polygons forming s.

Let k be the size of the description of s. We can easily compute the value for (iii) in $O(k)$ time.

By storing a variant of the 2-dimensional data structure of Theorem 1 in each facet of P, it is possible to find the projective volume of a cut facet in $O(1)$ time. Since the number of cut facets is less than k, we can compute (ii) in $O(k)$ time. So the only task remaining is to compute (i). In order to do that, we need some preprocessing: compute the projective volume of each face. Label each face with this value. Deform the surface of the polyhedron into an embedded planar graph, labeling each face with the corresponding volume that was previously computed. We can now reduce our problem to a problem for embedded planar graphs.

3 Planar Graphs

We define a query cut of a planar graph to be a set of directed edges that when removed from a planar graph separate a connected component, the query component, from the rest of the graph. The edges of a query cut are to be presented in clockwise order about the query component.

Theorem 3 *Given an embedding of a planar graph G, where weights are assigned to every face, there is a structure that with $O(n)$ preprocessing time will return the sum of the weights of the faces in a query component given a query cut in time linear in the number of edges in the cut.*

The boundary of a query surface for the polyhedron clearly defines a query cut for the planar graph, and the answer returned by the planar graph structure will answer to the point (i) of the previous section.

Note that this data structure is of independent interest and could be useful in areas such as cartography and computer networks.

We will first solve a slightly simpler problem: define a query circuit to be a circular vertex-disjoint alternating sequence of vertices and edges in an embedding of a planar graph.

Theorem 4 *Given an embedding of a planar graph G, where weights are assigned to every face, there is a structure that with $O(n)$ preprocessing time will return the sum of the weights of the faces inside a query circuit in time linear in the size of the circuit.*

Preprocessing: First, construct a directed path Q on the plane that visits every face at least once, and starts and ends in the outside face. This can be done quickly by constructing the Eulerian path of a spanning tree for the dual graph of G. For each face f, assign the weight of that face to some point on Q in that face. For any point p on Q, define $t(p)$ to be the sum of the weight points from the beginning of the path to p.

Consider every cell as a clockwise oriented cycle (i.e. every edge of G becomes two directed edges). Start with all edges weights at 0, and follow the path Q. For each edge Q crosses (with, say, intersection p), add $t(p)$ to its weight if it is a right turn from Q, and $-t(p)$ if it is a left turn.

Query: Traverse the edges of the query circuit in clockwise order. Sum the weights of the directed edges traversed. This will actually compute the sum of all weight points on Q (and weights of the corresponding faces) that are inside the query circuit. This can be done in time linear in the size of the query circuit.

We now describe the modification for obtaining Theorem 3:

Preprocessing: The same preprocessing is done as for the query cycle based structure. In addition for each face, defined by edges $e_1 \dots e_m$ in clockwise order, the structure used in Theorem 1 is used to store $w(e_1) \dots w(e_m)$.

Query: The edge cut set defines a query circuit. However, as there may be many edges on the query circuit between each pair of cut edges, it is undesirable to directly sum the weights of each edge on the query circuit as is done in the previous structure. However, by querying the structure of Theorem 1 for every face between adjacent query edges, one can compute the sum of the weights of all edges on a face that are on the query circuit in constant time. Thus the entire query may be carried out in time linear in the size of the edge cut set.

References

1. R. Boland and J. Urrutia. Polygon Area Problems. *Proc. of the 12th Canadian Conf. on Computational Geometry*, Fredericton, NB, Canada, 2000.
2. F. Contreras-Alcalá. Cutting polygons and a problem on illumination of stages. *Masters thesis, Dept. Comp. Sci. University of Ottawa*, Ottawa, ON, Canada, 1998. `http://www.csi.uottawa.ca/~fhca/thesis/`
3. J. Czyzowicz, F. Contreras-Alcalá and J. Urrutia. On measuring areas of polygons. *Proc. of the 10th Canadian Conf. on Computational Geometry*, Montréal, QC, Canada, 1998.