Extremal Queries using BSP Trees

David Eberly
Geometric Tools, LLC
http://www.geometrictools.com/
Copyright © 1998-2008. All Rights Reserved.

Created: January 28, 2003
Last Modified: March 2, 2008

Contents

1 Introduction **2**

2 Extremal Query for a Convex Polygon **2**

3 Extremal Query for a Convex Polyhedron **4**

4 Obtaining $O(\log n)$ Queries **8**

5 An Implementation and Timing **10**

1 Introduction

Given a convex polygon (2D) or a convex polyhedron (3D), a problem that arises in some applications is to efficiently determine a point that is extreme in a specified direction. This is referred to as an *extremal query*. There must always be an extreme vertex, but it is not necessarily unique. A polygon can have an extreme edge and a polyhedron can have an extreme edge or extreme face.

A simple algorithm for computing an extreme vertex for a convex polygon or polyhedron with vertices $\mathbf{V}_i$ for $0 \le i < n$ in the direction $\mathbf{D}$ is to locate the vertex $\mathbf{V}_j$ for which $\mathbf{D} \cdot \mathbf{V}_j = \max_{0 \le i < n}\{\mathbf{D} \cdot \mathbf{V}_i\}$. This is clearly an $O(n)$ algorithm. The question is: Can we find an algorithm that is asymptotically more efficient, say $O(\log n)$? An $O(\log n)$ algorithm will output perform an $O(n)$ algorithm **in the limit as n approaches infinity**, but it is possible for that an $O(n)$ algorithm outperforms an $O(\log n)$ algorithm for small- or medium-sized n. In practice, it is important to have some measurements of the constant in the asymptotic order. Moreover, given implementations of the competing $O(n)$ and $O(\log n)$ algorithms, it is worthwhile to determine the *break-even* value of n–the value at which the $O(\log n)$ algorithm outperforms the $O(n)$ algorithm.

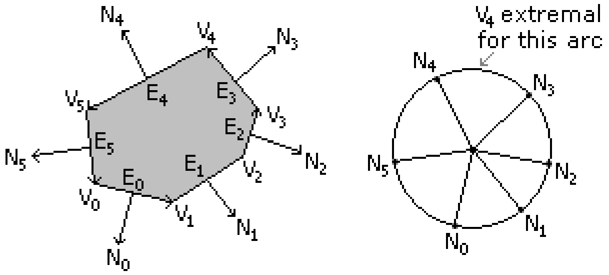
An extremal query for a convex polygon can be performed in $O(\log n)$ with no preprocessing of the polygon other than guaranteeing that its vertices are ordered. The algorithm is effectively a bisection of the dot products $\mathbf{D} \cdot \mathbf{V}_i$. This method does not have a counterpart for convex polyhedra. As it turns out, an extremal query algorithm of $O(\log n)$ does exist for convex polyhedra, but it requires a data structure that takes $O(n)$ time to build. For applications that have repetitive queries, the preprocessing time is not important. The idea for the data structure is due to [3] and [4], and it is referred to as the *Dobkin-Kirkpatrick hierarchy*. The algorithm itself, both the construction of the data structure and the extremal query, are discussed in detail in a well-written chapter in [5]. The construction relies on finding maximum independent sets in graphs, a problem known to be NP-complete. However, [1, 2] provide an approximation that gives sufficiently large independent sets that leads to an $O(n)$ construction while maintaining $O(\log n)$ for the query.

The construction is quite elegant and the details provided in [5] are enough to get you started on implementing the algorithm. Even so, the algorithm is intricate, requires some high-powered machinery to implement, including convex hull construction, and makes an implementation a formidable challenge. This document provides an alternative to the Dobkin-Kirkpatrick hierarchy. It is based on constructing a BSP tree for the spherical dual of a convex polyhedron. The BSP tree construction is $O(n)$ and the extremal query is $O(\log n)$ as long as you have a reasonably balanced tree. A heuristic for creating balanced trees is provided here. The implementation of the query is trivial and requires only a few lines of code. The construction of the tree is more complicated and assumes that a specific graph data structure exists for representing adjacency information for the vertices, edges, and triangles of the convex polyhedron.

2 Extremal Query for a Convex Polygon

Consider a convex polygon with counterclockwise ordered vertices $\mathbf{V}_i$ for $0 \le i < n$. The edge directions are $\mathbf{E}_i = \mathbf{V}_{i+1} - \mathbf{V}_i$ where it is assumed we are using modular arithmetic on the indices for wrap-around, $\mathbf{V}_n = \mathbf{V}_0$ and $\mathbf{V}_{-1} = \mathbf{V}_{n-1}$. Outward pointing unit-length normals $\mathbf{N}_i$ may be constructed for the edges. The normal vectors may be drawn as points on a unit circle. The arcs connecting the points correspond to the edges of the polygon. This view of the circle is called the *polar dual* of the polygon. Figure 2.1 illustrates for a six-sided polygon.

Figure 2.1 Left: A convex polygon. Right: The polar dual of the the polygon.



If $\mathbf{D} = \mathbf{N}_i$, then all points on the edge $\mathbf{E}_i$ are extremal. If $\mathbf{D}$ is strictly between $\mathbf{N}_0$ and $\mathbf{N}_1$, then $\mathbf{V}_1$ is the unique extremal point in that direction. Similar arguments apply for $\mathbf{D}$ strictly between any pair of consecutive normals. The normal points on the circle decompose the circle into arcs, each arc corresponding to an extremal vertex of the polygon. An end point of an arc corresponds to an entire edge being extremal. The testing of $\mathbf{D}$ to determine the full set of extremals point is listed below, where $(x, y)^\perp = (y, -x)$:

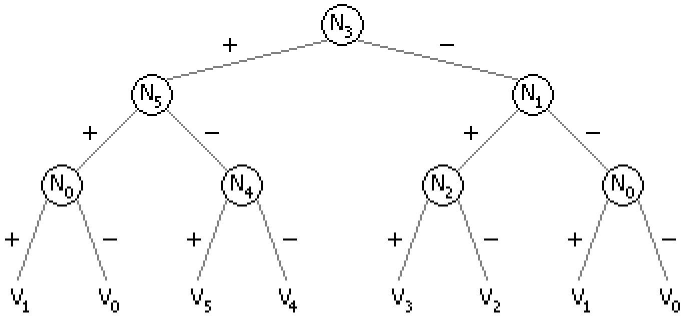
- Vertex $\mathbf{V}_i$ is optimal when $\mathbf{N}_{i-1} \cdot \mathbf{D}^\perp > 0$ ($\mathbf{D}$ is *left of* $\mathbf{N}_{i-1}$) and $\mathbf{N}_i \cdot \mathbf{D}^\perp < 0$ ($\mathbf{D}$ is *right of* $\mathbf{N}_i$).
- Edge $\mathbf{E}_i$ is optimal when $\mathbf{N}_{i-1} \cdot \mathbf{D}^\perp = 0$ and $\mathbf{N}_i \cdot \mathbf{D}^\perp < 0$.

The indexing is computed in the modular sense, where $\mathbf{N}_n = \mathbf{N}_0$ and $\mathbf{N}_{-1} = \mathbf{N}_{n-1}$. Assuming we will be projecting the extremal set onto the query axis, we can collapse the two tests into a single test and just use one vertex of an extremal edge as the to-be-projected point:

- Vertex $\mathbf{V}_i$ is optimal when $\mathbf{N}_{i-1} \cdot \mathbf{D}^\perp \geq 0$ and $\mathbf{N}_i \cdot \mathbf{D}^\perp < 0$.

Generally there are n arcs for an n-sided polygon. We could search the arcs one at a time and test if $\mathbf{D}$ is on that arc, but then we are back to an $O(n)$ search. Instead we can create a BSP tree for the polar dual that supports an $O(\log n)$ search. A simple illustration using the polygon of Figure 2.1 suffices. Figure 2.2 illustrates the construction of the BSP tree.

Figure 2.2 A BSP tree constructed by recursive subdivision of the unit disk. The left child of node $\mathbf{N}_j$ is marked with a '+' to indicate $\mathbf{N}_j \cdot \mathbf{D}^{\perp} \geq 0$. All normal vectors of the nodes in the left subtree are left of $\mathbf{N}_j$. The right child is marked with a '−' to indicate $\mathbf{N}_j \cdot \mathbf{D}^{\perp} < 0$. All normal vectors of the nodes in the right subtree are right of $\mathbf{N}_j$.



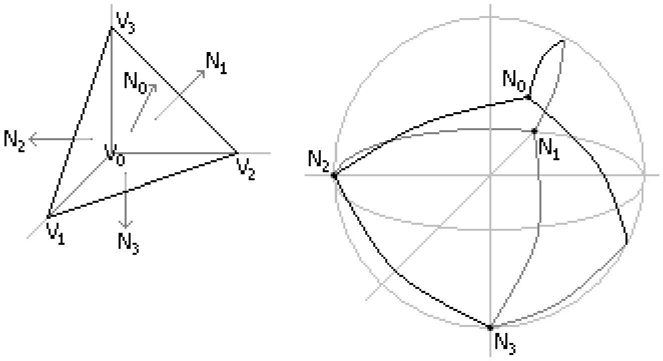
Each node $\mathbf{N}_j$ represents the normal for which j is the median value of theindices represented by all nodes in the subtree rooted at $\mathbf{N}_j$.

Given a direction vector $\mathbf{D}$, suppose the sign tests take you down the path from $\mathbf{N}_3$ to $\mathbf{N}_5$ to $\mathbf{N}_4$ and then to $\mathbf{V}_5$. This indicates that $\mathbf{D}$ is left of $\mathbf{N}_3$, right of $\mathbf{N}_5$, and left of $\mathbf{N}_4$. This places $\mathbf{D}$ on the arc from $\mathbf{N}_4$ to $\mathbf{N}_5$, in which case $\mathbf{V}_5$ is the extremal vertex in the specified direction.

3 Extremal Query for a Convex Polyhedron

A convex polyhedron has vertices $\mathbf{V}_i$ for $0 \leq i < n$ and a set of edges and a set of faces with outer pointing normals $\mathbf{N}_j$. The set of extremal points for a specified direction is either a polyhedron vertex, edge, or face. To illustrate, Figure 3.1 shows a tetrahedron and a unit sphere with vertices that correspond to the face normals of the tetrahedron, whose great circle arcs connecting the vertices correspond to the edges of the tetrahedron, and whose spherical polygons correspond to the vertices of the tetrahedron. This view of the sphere is called the *spherical dual* of the polyhedron.

Figure 3.1 Left: A tetrahedron. Right: The spherical dual of the tetrahedron.

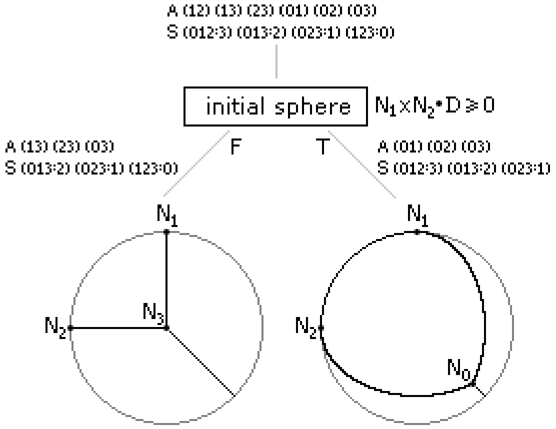


The tetrahedron has vertices $\mathbf{V}_0 = (0,0,0)$, $\mathbf{V}_1 = (1,0,0)$, $\mathbf{V}_2 = (0,1,0)$, and $\mathbf{V}_3 = (0,0,1)$. The face normals are $\mathbf{N}_0 = (1,1,1)/\sqrt{3}$, $\mathbf{N}_1 = (-1,0,0)$, $\mathbf{N}_2 = (0,-1,0)$, and $\mathbf{N}_3 = (0,0,-1)$. The sphere is partitioned into four spherical triangles. The interior of the spherical triangle with $\langle \mathbf{N}_0, \mathbf{N}_1, \mathbf{N}_2 \rangle$ corresponds to those directions for which $\mathbf{V}_3$ is the unique extreme point. Observe that the three normals forming the spherical triangle are the normals for the faces that share vertex $\mathbf{V}_3$.

Generally, the normal and edge directions of a polytope lead to a partitioning of the sphere into spherical convex polygons. The interior of a single spherical convex polygon corresponds to the set of directions for which a vertex of the polytope is the unique extreme point. The number of edges of the spherical convex polygon is the number of polytope faces sharing that vertex. Just as for convex polygons in 2D, we can construct a BSP tree of the spherical polygons and use it for fast determination of extreme vertices. The method used for 2D extends to 3D with each node of the BSP tree representing a hemisphere determined by $\mathbf{N}_i \times \mathbf{N}_j \cdot \mathbf{D} \geq 0$, where $\mathbf{N}_i$ and $\mathbf{N}_j$ are unit-length normal vectors for two adjacent triangles.

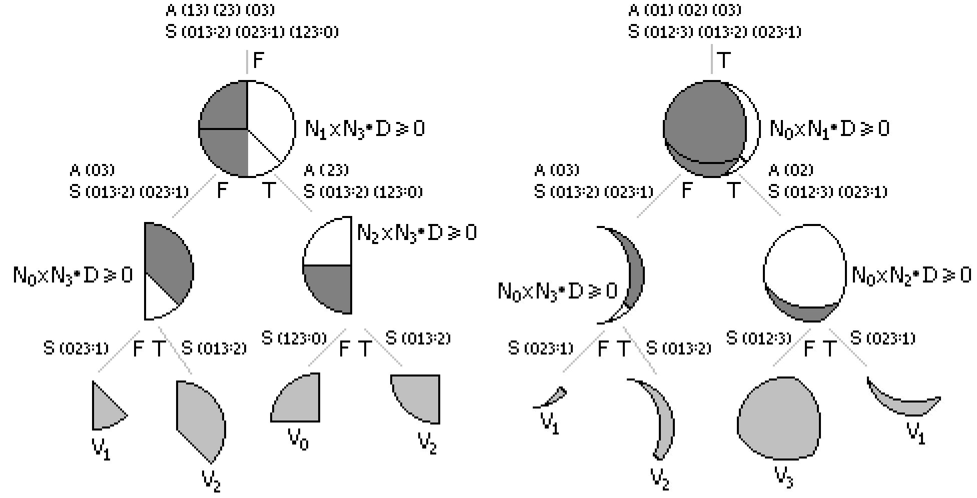
The vector $\mathbf{H}_{ij} = \mathbf{N}_i \times \mathbf{N}_j$ is perpendicular to the plane containing the two normals. The hemispheres corresponding to this vector are $\mathbf{H}_{ij} \cdot \mathbf{D} \geq 0$ and $\mathbf{H}_{ij} \cdot \mathbf{D} < 0$. The tetrahedron of Figure 3.1 has six such vectors, listed as $\{\mathbf{H}_{12}, \mathbf{H}_{13}, \mathbf{H}_{23}, \mathbf{H}_{01}, \mathbf{H}_{02}, \mathbf{H}_{03}\}$. Please note that the subscripts correspond to normal vector indices, not to vertex indices. Each arc of the sphere connecting two normal vectors corresponds to an edge of the tetrahedron, label the arcs A_{ij}. The root node of the tree claims arc A_{12} for splitting. The condition $\mathbf{N}_1 \times \mathbf{N}_2 \cdot \mathbf{D} \geq 0$ splits the sphere into two hemispheres. Figure 3.2 shows those hemispheres with viewing direction $(0,0,-1)$.

Figure 3.2 The root of the BSP tree and the two hemispheres obtained by splitting. Both children are displayed with a viewing direction $(0, 0, -1)$. The right child is the top of the sphere viewed from the outside and the left child is the bottom of the sphere viewed from the inside.


A (12) (13) (23) (01) (02) (03)
S (012:3) (013:2) (023:1) (123:0)
initial sphere
N1xN2•D ≥ 0
F
T
A (13) (23) (03)
S (013:2) (023:1) (123:0)
A (01) (02) (03)
S (012:3) (013:2) (023:1)
N1
N2
N3
N1
N2
N0

The set of arcs and the set of spherical polygons bounded by the arcs are the inputs to the BSP tree construction. These sets are shown at the top of the figure. An arc is specified by A_{ij} and connects $\mathbf{N}_i$ and $\mathbf{N}_j$. A spherical polygon is $S_{i_1,\ldots,i_n:\ell}$ and has vertices $\mathbf{N}_{i_1}$ through $\mathbf{N}_{i_n}$. The vertex $\mathbf{V}_\ell$ of the original polyhedron is the extreme vertex corresponding to the spherical polygon. In our example the spherical polygons all have three vertices. Figure 3.3 shows the BSP trees for the children of the root.

Figure 3.3 The BSP trees for the children of the root. The algebraic test is listed next to each root. The links to the children are labeled with T when the the test is true or labeled with F when the test is false.



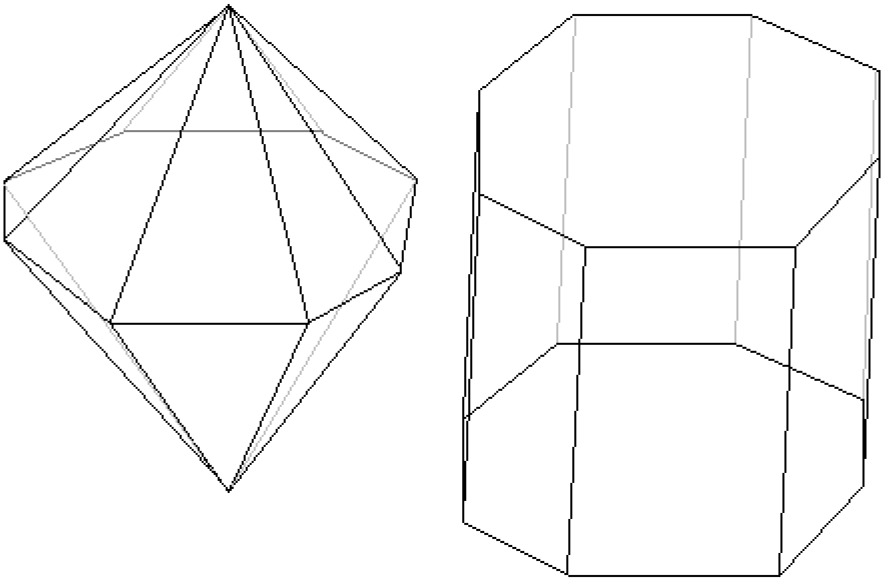
During BSP tree construction, the root node claims the first arc A_{12} and uses the vector $\mathbf{H} = \mathbf{N}_1 \times \mathbf{N}_2$ for testing other vectors corresponding to arcs A_{ij}. Let $d_i = \mathbf{H} \cdot \mathbf{N}_i$ and $d_j = \mathbf{H} \cdot \mathbf{N}_j$. If $d_i \geq 0$ and $d_j \geq 0$, then the arc is completely on one side of the hemisphere implied by $\mathbf{E}$. A_{ij} is placed in a set that will be used to generate the BSP tree for the right child of the root. If $d_i \leq 0$ and $d_j \leq 0$, then the arc is completely on the other side of the hemisphere and is placed in a set that will be used to generate the BSP tree for the left child of the root. If $d_i d_j < 0$, then the arc is partially in each hemisphere and is added to both sets. This is exactly the algorithm we used in 2D.

In 3D we have some additional work in that the spherical faces must be processed by the tree to propagate to the leaf nodes the indices of the extreme vertices represented by those nodes. In fact, the processing is similar to that for arcs. Let $S_{i,j,k:\ell}$ be a face to be processed at the root node. Let $d_i = \mathbf{E} \cdot \mathbf{N}_i$, $d_j = \mathbf{E} \cdot \mathbf{N}_j$, and $d_k = \mathbf{E} \cdot \mathbf{N}_k$. If $d_i \geq 0$ and $d_j \geq 0$ and $d_k \geq 0$, then the spherical face is completely on one side of the hemisphere implied by $\mathbf{E}$. $S_{i,j,k:\ell}$ is placed in a set that will be used to generate the BSP tree for the right child of the root. If $d_i \leq 0$, $d_j \leq 0$, and $d_k \leq 0$, then the face is completely on the other side of the hemisphere and is placed in a set that will be used to generate the BSP tree for the right child of the root. Otherwise the arc is partially in each hemisphere and is added to both sets. In general for a spherical face with n vertices, the face is used for construction of the right child if all dot products are nonnegative, for construction of the left child if all dot products are nonpositive, or for construction of both children if some dot products are positive and some are negative.

A query for a specified direction $\mathbf{D}$ is structured the same as for convex polygons. The signs of the dot products of $\mathbf{D}$ with the $\mathbf{H}$ vectors in the BSP tree are computed, and the appropriate path is taken through the tree. A balanced tree will have depth $O(\log n)$ for a polyhedron of n vertices, so the query takes logarithmic time. However, there is a technical problem. The spherical arcs as described so far might not lead to a balanced tree. Consider a polyhedron formed by an $(n-2)$-sided convex polygon in the xy-plane with vertices $\mathbf{V}_i = (x_i, y_i, 0)$ for $1 \leq i \leq n-2$ and by two vertices $\mathbf{V}_0 = (0, 0, z_0)$, with $z_0 < 0$, and

$\mathbf{V}_{n-1} = (0, 0, z_{n-1})$, with $z_{n-1} > 0$. Figure 3.4 shows such a polyhedron.

Figure 3.4 Left: A convex polyhedron for which the point-in-spherical-polygon test, using only the original arcs, is $O(n)$ (the figure shows $n = 8$, but imagine a very large n). Right: The inscribed convex polyhedron whose edges generate the arcs of the spherical convex polygons.

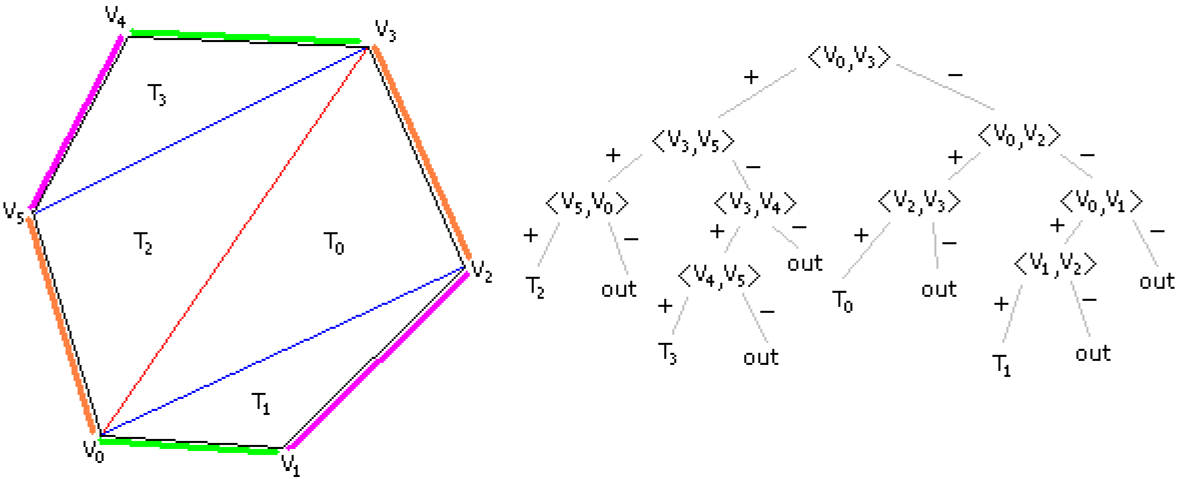


The spherical dual has 2 spherical convex polygons, each with $n-2$ arcs and $n-2$ spherical convex polygons, each with 4 arcs. If $\mathbf{D}$ is contained by one of the $(n-2)$-sided spherical polygons, the determination of this using only the given arcs requires $n-2$ point-on-which-side-of-arc queries. This is an $O(n)$ algorithm.

4 Obtaining $O(\log n)$ Queries

The pathological problem mentioned previously is avoided by appealing to an $O(\log n)$ query for point-in-convex-polygon determination. In fact, this problem uses what you may think of as the Dobkin-Kirkpatrick hierarchy restricted to two dimensions. However, it is phrased in terms of a binary search using binary separating lines. Consider the convex polygon of Figure 4.1.

Figure 4.1 A convex polygon with bisectors used for an $O(\log n)$ point-in-polygon query. The polygon is partitioned into four triangles labeled T_0 through T_3.



If we use only the polygon edges for containment testing, we would need 6 tests, each showing that the query point is to the left of the edges. Instead, we use *bisectors*. The first bisector is segment $\langle \mathbf{V}_0, \mathbf{V}_3 \rangle$ and is drawn in red in the figure. The query point $\mathbf{P}$ is either to the left of the bisector, where $(\mathbf{V}_3 - \mathbf{V}_0) \cdot (\mathbf{P} - \mathbf{V}_0)^{\perp} \cdot \geq 0$, or to the right of the bisector, where $(\mathbf{V}_3 - \mathbf{V}_0) \cdot (\mathbf{P} - \mathbf{V}_0)^{\perp} \cdot < 0$. The original polygon has 6 edges. The polygon to the left of the bisector has 4 edges, and the containment test is applied to that left polygon. The bisector edge has already been tested, so only the 3 remaining edges need to be tested for the containment.

The next bisector to be used is one of the segments drawn in blue, either $\langle \mathbf{V}_3, \mathbf{V}_5 \rangle$ or $\langle \mathbf{V}_0, \mathbf{V}_2 \rangle$. Naturally, the choice depends on which side of the bisector $\langle \mathbf{V}_0, \mathbf{V}_3 \rangle$ the query point $\mathbf{P}$ occurs. If $\mathbf{P}$ is to the left of $\langle \mathbf{V}_0, \mathbf{V}_3 \rangle$ and to the left of $\langle \mathbf{V}_3, \mathbf{V}_5 \rangle$, then the only subpolygon that might contain $\mathbf{P}$ is a triangle. The remaining edge to test is $\langle \mathbf{V}_5, \mathbf{V}_0 \rangle$. If instead $\mathbf{P}$ is to the right of $\langle \mathbf{V}_3, \mathbf{V}_5 \rangle$, then $\mathbf{P}$ is potentially in triangle T_3. There are two remaining edges to test, so we can consider this yet another bisection step.

The binary tree in the right image of Figure 4.1 shows the query tree implied by the bisectors and polygon edges. A tree link marked with a '+' indicates "to the left of" and a link marked with a '−' indicates "to the right of". The query point is tested against each edge, leading to a path from the root of the tree to a leaf node. The leaves are labeled with the region defined by the path. Four of the leaf nodes represent the triangles that make up the convex polygon. Six of the leaf nodes represent the exterior of the polygon; that is, a query point can be "outside" one of the six edges of the polygon.

The choice of the bisectors is based on selecting the medians of the ranges of indices for the subpolygon of interest. This leads to a balanced tree, so a polygon of n edges. The point-in-polygon query requires computing vector differences and dot products for a linear path of nodes through the tree. Such a path has $O(\log n)$ nodes.

The same idea may be applied to determining whether a unit-length vector $\mathbf{D}$ is contained in a spherical convex polygon on the unit sphere. In the 2D problem, the dot product whose sign determines which side of the bisector the point is on was of the form

$$d = (\mathbf{V}_i - \mathbf{V}_j) \cdot (\mathbf{P} - \mathbf{V}_j)^{\perp}$$

We cared about $d \geq 0$ or $d < 0$. In the 3D problem, we use the normal vectors as the vertices and the direction vector as the query point. The dot product of interest is

$$d = \mathbf{N}_i \times \mathbf{N}_j \cdot \mathbf{D}$$

Imagine walking along the spherical arc from $\mathbf{N}_j$ to $\mathbf{N}_i$. The vector $\mathbf{N}_i \times \mathbf{N}_j$ points to your left as you walk along the arc. The spherical point $\mathbf{D}$ is to your left whenever $d \geq 0$ and is to your right whenever $d < 0$.

What this means is that the BSP tree we build for the spherical dual must use the bisectors of the spherical polygons as well as the arcs which are their boundary edges. Moreover, we do not just have one spherical polygon to test–we have n such polygons, one for each of the n vertices of the original convex polyhedron. Given the collection of all bisector arcs and boundary edge arcs, we can build the BSP tree one arc at a time. Each arc is tested at a node of the BSP tree to see on which side of the node arc it lies. If fully on the left side, we send the arc to the left subtree of the node for further classification. If fully on the right side, we send the arc to the right subtree. When the arc arrives at a leaf node, it is similarly tested for sideness and stored as the appropriate child of the leaf node (which now becomes an interior node). If an arc straddles the great circle containing the node arc–one arc end point is to the left of the node arc and one arc end point is to the right of the node arc–we send the arc to both subtrees of the node. That is, no actual splitting of the arc is performed. This avoids expensive arc-arc intersection finding.

The order of the arcs is important in determining the structure of the BSP tree. We do want a balanced tree. A heuristic to obtain a balanced tree is to sort the arcs. The bisector arcs occur first, the boundary edge arcs last. The first bisector arc of an n-sided spherical polygon splits the polygon into two subpolygons, each with half the number of boundary edge arcs. If we were to do a point-in-spherical-polygon query using such a bisector edge first, we will eliminate half of that spherical polygon's bisectors and half of its boundary arcs from further processing. This suggests that we order the bisector arcs based on how many other arcs they reject during a sidedness test. In the implementation, I maintain an ordered set of arcs, using a *separation* measure. The first bisector arc for an n-sided spherical polygon has a separation of $n/2$, measuring how many boundary arcs of that polygon separate the end points. A boundary arc itself has a separation measure of 1. The BSP tree is built using the arcs in decreasing order of separation. My numerical experiments showed that indeed the BSP trees are balanced.

5 An Implementation and Timing

The Foundation library files `Wm3ExtremalQuery3.h` and `Wm3ExtremalQuery3.cpp` are the base class for extremal queries for convex polyhedra. The straightforward $O(n)$ method for computing the extreme points just involves projecting the vertices onto the specified direction vector and computing the extreme projection values. This algorithm is implemented in `Wm3ExtremalQuery3PRJ.h` and `Wm3ExtremalQuery3PRJ.cpp`.

The files `Wm3ExtremalQuery3BSP.h` and `Wm3ExtremalQuery3BSP.cpp` implement the BSP tree algorithm described in this document. The identification of adjacent polyhedron normal vectors $\mathbf{N}_i$ and $\mathbf{N}_j$ requires building a vertex-edge-face data structure. The class implemented in `Wm3BasicMesh.h` and `Wm3BasicMesh.cpp` suffices, but the edges adjacent to a vertex are not required for the extremal queries. You could modify `BasicMesh` to eliminate this adjacency information.

The next table shows the results of the experiment to compare the BSP tree approach to a simple project-all-vertices approach. The column with header n is the number of vertices of the convex polyhedron. Each polyhedron was used in 10^7 extremal queries. The execution times are listed in the second and third columns, and are in seconds. The target machine was an AMD Athlon XP 2800+ (2.08GHz). The next to last column is the BSP time divided by $\log n$. This ratio is expected to be a constant for large n; that is, we expect the query to be $O(\log n)$. The last column is the project-all-vertices time divided by n, since we expect this algorithm to be $O(n)$.

n	BSP time t_b	project-all time t_p	$t_b/\log n$	t_p/n
4	2.141	0.812	1.0705	0.2030
8	3.922	1.547	1.3073	0.1933
16	5.422	2.563	1.3555	0.1601
32	5.937	4.328	1.1874	0.1352
64	6.922	7.765	1.1536	0.1213
128	7.922	14.391	1.1317	0.1124
256	9.281	27.359	1.1601	0.1068
512	10.250	53.859	1.1388	0.1051
1024	11.532	104.125	1.1532	0.1016
2048	12.797	210.765	1.1633	0.1029

Of interest is the break-even n. It is somewhere between 32 and 64. If the convex polyhedra in your applications have a small number of vertices, the project-all-vertices approach is clearly the choice. For larger number of vertices, the BSP approach wins.

A sample application to illustrate the queries is in the SampleMiscellaneous folder, project ExtremalQuery. A convex polyhedron is displayed using an orthogonal camera. You may rotate it with the mouse. The extreme vertices in the x-direction are drawn as small spheres. The orthogonal camera is used to make it clear that the points are extreme.

References

[1] H. Edelsbrunner and H.A. Maurer, *Finding extreme points in three dimensions and solving the post-office problem in the plane*, Inform. Process. Lett., vol. 21, pp. 39-47, 1985.

[2] H. Edelsbrunner, *Algorithms in Computational Geometry*, Springer-Verlag, Heidelberg, 1987

[3] D.P. Dobkin and D.G. Kirkpatrick, *Determining the separation of preprocessed polyhedra–a unified approach*, Proc. 17th Internat. Colloq. Automata Lang. Program, Lecture Notes in Computer Science, vol. 43, Springer-Verlag, Heidelberg, pp. 400-413, 1990

[4] D.G. Kirkpatrick, *Optimal search in planar subdivisions*, SIAM J. Comp., vol. 12, pp. 28-35, 1983

[5] Joseph O'Rourke, *Computational Geometry in C, 2nd edition*, Cambridge University Press, Cambridge, England, 1998