**CS348a: Computer Graphics** Handout #36
**Geometric Modeling** Original Handout #29
Stanford University Thursday, 10 December 1992

| | |
|---|---|
| Original Lecture #14: | December 1, 1992 |
| Topics: | Hierarchical Structures for Convex Polygons and Polyhedra |
| Scribe: | Radu Tutos [*] |

In the previous lecture we discussed the method of approximating a convex polygon with a series of incrementally simpler polygons. We'd now like to extend the method to convex polyhedra. Our goal is to build a series of incrementally "coarsened" polyhedra, where each member in the series has a fewer number of vertices than the preceding one. To achieve that, we take the original polyhedron $P_0$ and remove a large set of low degree vertices which are independent (non-adjacent), and then remove their incident edges. Each removed vertex leaves a small well defined *hole*, which we retriangulate by computing the three dimensional convex hull of its boundary. Since the number of edges and faces in a polyhedron decreases linearly with its number of vertices, the resulting polyhedron will be less complex than the original one. We then repeat this coarsening process until we build a polyhedron $P_k$ with at most $p$ number of vertices. This gives us a sequence of convex polyhedra on which we can build a hierarchical representation of the original one:

$$P_0 \rightarrow P_2 \rightarrow \ldots \rightarrow \ldots \rightarrow P_k$$

We now need to determine the degree of the vertices that we remove at each coarsening iteration. If the degree of the these vertices is too high, we end up replacing a large number of existing edges with new ones needed to retriangulate holes with complex shapes. This is an undesirable result, since we want to capture in the coarsened polyhedron as many features as we can as from the original one. On the other hand, if we set the degree of the vertices we want to remove too low, we may discover that no vertices meet our criteria. The following lemma helps us find the optimum upper bound on the degree of these vertices, and it shows the way to build the hierarchical set of convex polyhedra:

**Lemma 1.** *Let $P$ be a convex polytope with $m$ vertices; then there exists an independent set $I$ of vertices, such that*

1. $|I| \geqslant m/18$,
2. $deg(v) \leqslant 8$ *for all* $v \in I$,
3. *$I$ can be found in $O(m)$ time.*

**Proof:** We construct $I$ using a greedy algorithm.

[*]The first section is partly based on the 1991 notes of Brian Rogoff

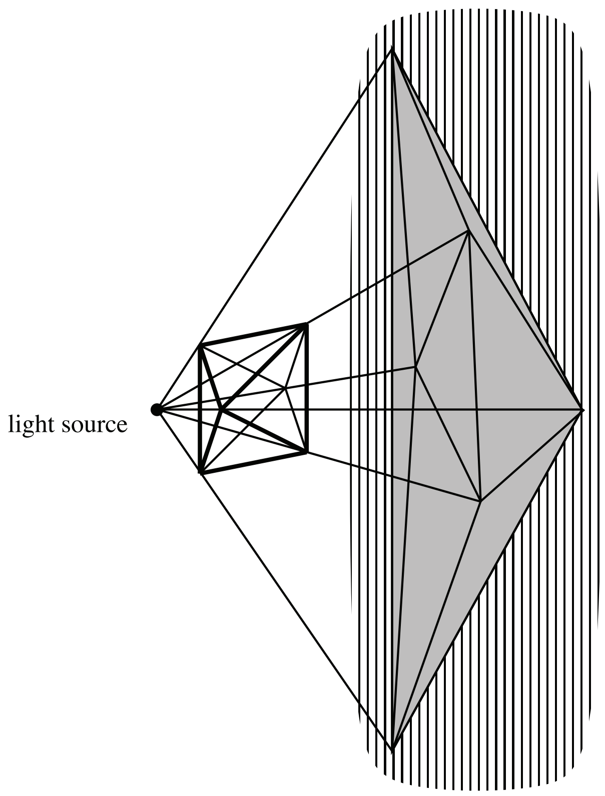

Figure 1: A projection mapping a polytope to a straight-line planar arrangement.

```
Begin
   Set $I = \phi$
   For each vertex $v$ in $P$ do
      If $v$ has not been marked and degree($v$) $\leqslant 8$
         Then add $v$ to $I$ and mark the neighbors of $v$
End
```

The algorithm produces a maximal independent set $I$ of vertices each of which has degree at most eight. We first prove that at least half of the vertices in $P$ will have degree less equal than 8.

It is easily shown that a convex polytope is topologically equivalent to a planar graph. If we put a light source just outside one face $f$ of a convex polytope, and then project the polytope onto a plane parallel to $f$ on the opposite side of the polytope (see figure 1), we get a straight-line planar subdivision which we have already studied quite a bit. The face $f$ corresponds to the exterior face in this projection, and every other face of the polytope corresponds to an interior face of the subdivision.

From Euler's theorem on planar graphs, we have

$$\sum_{v \in P} degree(v) \leqslant 6m - 12$$

If half of the vertices had degree $\geqslant 9$, then the total degree would be at least

$$9\left\lfloor\frac{m}{2}\right\rfloor+3\left\lceil\frac{m}{2}\right\rceil \geqslant 6m-3$$

which already exceeds Euler's bound. Thus less than half the vertices have degree $\geqslant 9$; in other words, more than half the vertices have degree $\leqslant 8$.

During the marking process, one vertex being included in $I$ marks at most eight other vertices. Even if those 8 vertices all have low degree, we still can include one out of every 9 low degree vertices. Since the total number of low degree vertices is at least $\frac{m}{2}$ we can find an independent set of size at least $\frac{m}{18}$. During the algorithm each edge is checked twice: once from each end, thus the running time is linear. $\square$

The size of the polyhedra in the hierarchy decreases geometrically, so the sequence will stop at $P_k$ with $k = O(\log m)$, and the total time and storage to compute the sequence is linear.

In order to walk up the hierarchy from $P_k$ to $P_0$, we need a a hierarchy pointer $hp$ from every new edge $e$ created in $P_{i+1}$ to the vertex $v$ of $P_i$ whose removal generated $e$. The value of the hierarchy pointer for each edge $e$ in $P_{i+1}$ is obtained as follows:

```
If $e$ is not in $P_i$ then
   $hp(e)$ = v
else if $e$ is in $P_i$ but its left face is different than in $P_i$ then
   $hp(e)$ = v
else if $e$ is in $P_i$ and it has the same left face as in $P_i$ then
   $hp(e)$ = null
```

The resulting hierarchy of polyhedra allows us to construct efficient algorithms to answer queries on the original polyhedron $P_0$. We will go over some of these algorithms shortly.

## Computing the Convex Hull

When we coarsen a convex polyhedron with the method described above, we have retriangulate each hole that is created when a vertex is removed. To do that, we compute the convex hull of the adjoint vertices. We define the convex hull of a set of points in space as the smallest convex polyhedron which embodies all the points. We will now describe two algorithms to do this computation. As a note, there are other more efficient algorithms to compute the convex hull, but they are more complex as well.

### The Incremental Algorithm

Let $p_1 \ldots p_n$ be the set of points for which we want to compute the convex hull, where the points are sorted in the X direction. We now incrementally compute the convex hull for the sets of points $p_1 \ldots p_3$, $p_1 \ldots p_4$, up to $p_1 \ldots p_n$. For each set $p_1 \ldots p_{i+1}$ in this sequence, we

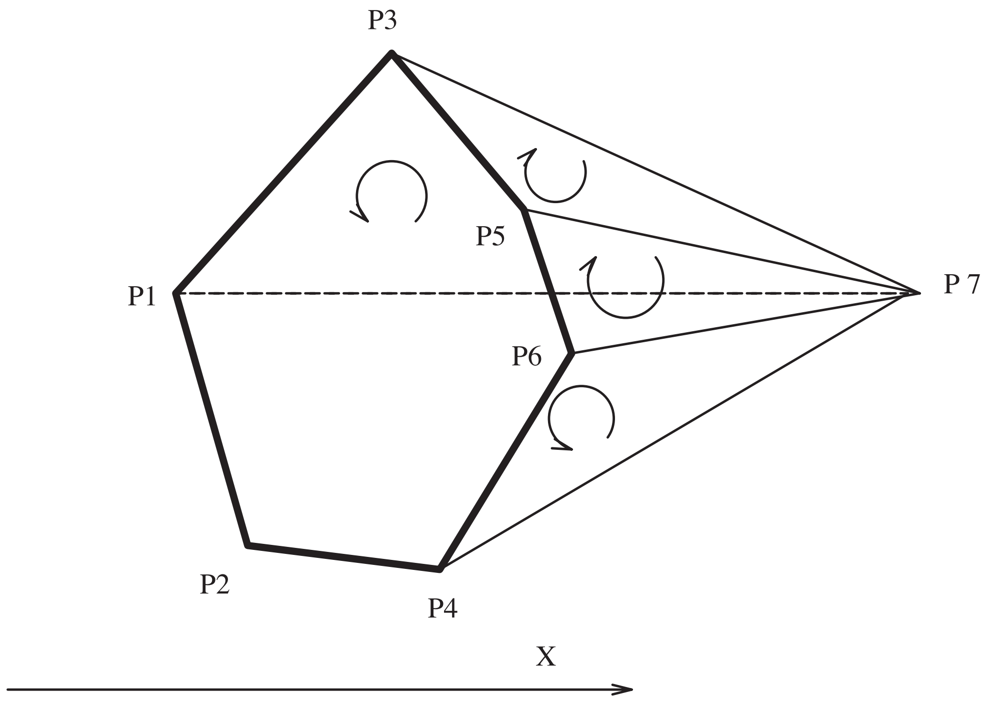

Figure 2: Computing the 2-D convex hull using the incremental algorithm.

calculate the convex hull based on the $p_1 \ldots p_i$ convex hull that we just computed, and the point $p_{i+1}$, which is on the right side of all previous points on the X direction. takev

We will first describe the algorithm in the 2-D case. To calculate the the convex hull for $p_1 \ldots p_{i+1}$, we take the following steps:

```
Begin
   1.  draw an edge from $p_{i+1}$ to the closest point

   2.  Go up, add edges to each consecutive point on the hull
   while the triangles that we construct have a positive area
   in the clockwise direction

   3.  Go down, add edges to each consecutive point on the
   hull while the triangles that we construct have a positive
   area in the counterclockwise direction

   4.  Keep the last two edges added in steps 2 and 3, and
   remove all edges that left inside the newly created convex hull
```

The sign test for the area of triangle with vertices at $p$, $q$, and $r$ is done by calculating the determinant

$$\begin{vmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{vmatrix}$$

The algorithm is illustrated in Fig. 2 for $i = 6$. Going up from from the edge $p_7p_6$ which we have drawn at step 1, we add the edges $p_7p_5$ and $p_7p_3$. Since the signed area of the triangle $p_7p_3p_1$ is negative, we do not add the edge $p_7p_1$, and we stop. Similarly, going down we add the edge $p_7p_4$. At step 4, we remove the edges $p_4p_6$, $p_6p_5$, and $p_5p_3$ >from the $p_1 \ldots p_6$ convex hull, which we will replace with the newly created edges $p_7p_3$ and $p_7p_4$.

The algorithm can be extended to calculate the convex hull for a set of points in 3-D. To do the calculation for $p_1 \ldots p_{i+1}$, we draw an edge from $p_{i+1}$ to the closest vertex on the $p_1 \ldots p_i$ convex hull, and then we add new edges by doing sign tests on the volumes of the tetrahedras defined by these edges. The sign test for a tetrahedron with vertices at $p$, $q$, and $r$ is done by calculating the determinant

$$\begin{vmatrix} p_x & p_y & p_z & 1 \\ q_x & q_y & q_z & 1 \\ r_x & r_y & r_z & 1 \\ s_x & s_y & s_z & 1 \end{vmatrix}$$

The incremental algorithm is relatively simple, but it is not efficient and and is not easy to implement in 3-D.

## The Gift Wrapping Algorithm

Again, we will first describe the algorithm for a set of points in 2-D. Intuitively, the basic idea is to proceed from a vertex and wrap the set of points with a rope.

Let $p_1 \ldots p_n$ be the set of points for which we want to compute the convex hull, where the points are sorted in the X direction. Proceeding from the first point in the set $p_1$, we rotate a line on the point in a counterclockwise direction until we hit another point $p_a$. The edge $p_1p_a$ is then added to the convex hull. We then repeat the rotation of the line on $p_a$ until we made a complete circle. Note that for every edge that we add to the convex hull, all the points in the set are on the left side of the half space defined by that edge. This process is illustrated in Fig. 3.

The method can be extended of compute the convex hull for a set of points in 3-D. The basic idea is to proceed from a face on the convex hull to the adjacent faces, in the manner in which one where wiwraps a sheet around a plane-bounded object. The adjacent faces are determined by rotating a plane on the line defined by the edge of an existing face of the convex hull. The first point that the plane hits will be part of the convex hull as well. The method is illustrated in Fig. 4, where a new face $p_dp_bp_c$ is determined from an existing face $p_ap_bp_c$.

The first step is to find a face defined by three points on the convex hull. As in the planar case, the first of these points can be picked as the one with the least $x$ coordinate. The second point is found by rotating a plane on the line parallel to the $y$ or $z$ axis, and the third point is found by rotating a plane on the line determined by the first two points.

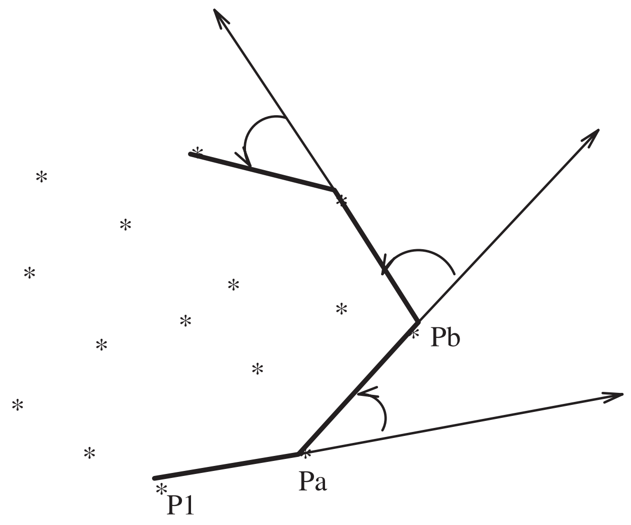

Figure 3: Computing the 2-D convex hull using the gift wrapping algorithm.

## Query Problems

### Where Will a Plane Hit a Polyhedron ?

Given a polyhedron $P_0$ and a plane which rotates on a line, we want to find the vertex $v_0$ where the plane first touches $P_0$. We use an incremental algorithm on the hierarchy of polyhedra $P_0, P_1 \ldots P_k$ which we constructed in the first section. We first determine the vertex $v_k$ where the plane touches $P_k$, and based on this we determine the corresponding vertex for $P_{k-1}$, $P_{k-2} \ldots P_0$.

Suppose the plane first touches $P_{i+1}$ at vertex $v_{i+1}$. Then, the plane will first touch $P_i$ either at $v_{i+1}$ itself, or at an adjacent vertex that was removed in $P_{i+1}$ (which can be found using the hierarchy pointer). Since at each iteration $P_{i+1} \rightarrow P_i$ the computation is local around vertex $pi$, this is an $O(logn)$ algorithm.

### Is a Point inside a polyhedron ?

Given a polyhedron $P_0$, we want to determine if a point $q$ is inside it. As before, we use an incremental algorithm on the hierarchy of polyhedra $P_0 \ldots P_k$, starting at $P_k$. The search stops if $q$ is inside a polyhedron $P_i$, since obviously it will be inside $P_0$ as well. Assume we determined that $q$ is not inside $P_{i+1}$, and we want to determine if it is inside $P_i$. To do that, we “grow” a sphere centered at $q$, which will hit $P_{i+1}$ on some face $f$. If $f$ belongs to $P_i$ as well, then the sphere will also hit $P_i$ at $f$, and $q$ is not inside $P_i$. If $f$ is not in $P_i$, then $P_i$ has a vertex $v$ “outside” $f$ which is inside the sphere. Doing a local computation on the tetrahedron determined by $f$ and $v$, it can be determined if $q$ is inside $P_i$ or not. The algorithm is illustrated in Fig. 5. Since this algorithm also implies a local computation at each iteration, it is $O(logn)$.

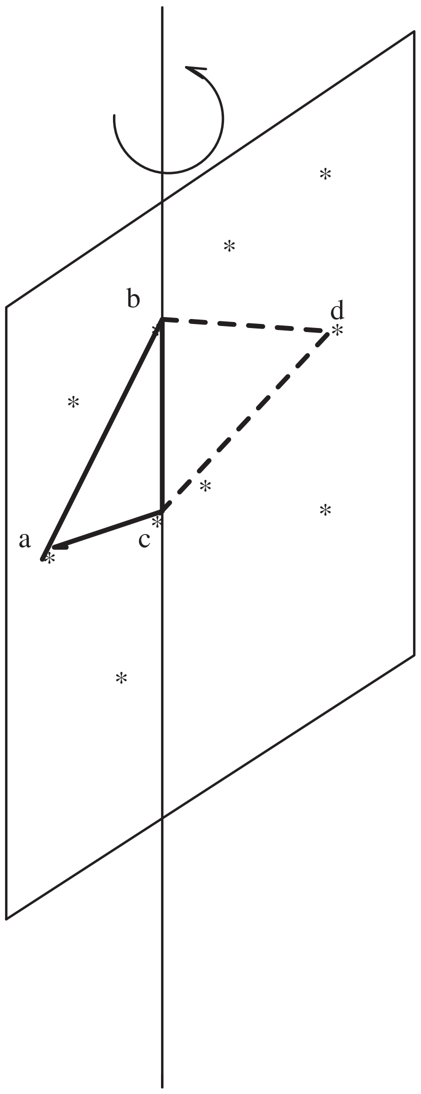

Figure 4: Determining a facet on the 3-D convex hull using the gift wrapping algorithm.

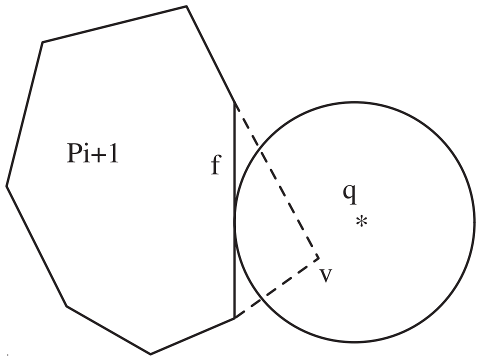

Figure 5: Determining if point q is inside a polyhedron (2-D view).