A Java Implementation of an Algorithm to Find the Convex Hull of a Set of Three-Dimensional Points

---- C&G course project report

zhangmin 994838

1. INTRODUCTION

The implementation of an algorithm is different from the description of it. It requires the programmer to think over most of the details and extreme situations of the algorithm, and maybe some times there will be no need to deal with all bad situations, especially for some specific project. This program implements one of the algorithms on finding the convex hull of a set of 3-dimensional points, in Java.

2. ALGORITHM OUTLINE

The program is based on the Preparata-Hone algorithm.

Given a set of 3-dimensional points, the essential work to find the convex hull of the set is to find the edges linking the **hull points**, these edges also form the facets on the convex hull's surface. In this algorithm, The main idea is to find the facets that wrap the other points in, that's **"3-d wrapping"**.

Initially the points are sorted according to their y-coordinates. After that, we always divide the points into two parts, each part contains about half number of the points (number mod 4 = 0). Then we apply the recursive divide-and-conquer technique. For each part, we find the convex hull of the sub points set, and then merge the two parts into a new convex hull. As you can imagine, the hard work focuses on the **construction of "end situation" hull**, viz. the basic smallest convex hull, and the **merging** of the two sub hulls. In this algorithm, we choose the 4-point tetrahedron as the smallest convex hull unit, or the end situation of recursive process. When merging the two sub hulls, we take two steps. First we do 2-dimensional merge of the hulls, and then 3-d merge. The purpose of 2-d merge is to find a starting edge for 3-d merge. This edge will surely be one edge in the merged hull, and it can be thought of as the sidest edge of all hull-linking edges. Based on the already found edge by 2-d merge, the 3-d wrapping merge starts. The 3-d wrapping process always holds the latest created edge as its reference edge, and the latest passed facet as the reference facet. And then we individually from the two sub hulls select a best edge as the candidate edge to advance with. The selection is made by testing the angle between two facets. One facet is defined by every testing edge and the reference edge, the other is the reference facet. The biggest is the one we want. The same comparison is made between the two "best edges" of their own sub hulls and the winner is the next edge to advance with. A new edge is then created to link the vertexes of the selected edge and the reference edge, and it will be used as the new reference edge to make the next advance.

At last, we will get a **net** (or **graphics**) of edges, each edge on the net is an edge of the convex hull, and all edges of the convex hull are on the net . The points on the edges are certainly the convex hull points, as well as the edges.

3. THE DATA STRUCTURE AND THE OUTPUT OF THE CONVEX HULL

In Day's implementation, it outputs the convex hull by going through the edges and output every facet of the convex hull. In stead of output the facets, I think outputting the edges is enough and more efficient.

The data structure is important. Well-designed data structure can make computation simple and efficient. In this implementation, the data type edge has to maintain information both for 2-d edge chain and for 3-d edge net . So we have the class "**Edgeptr**" hold two pointers to its front and back edges for 2-d chain, **"fwd2d"** and **"bwd2d"**. And its two Vertexes are in charge of linking with other edges in 3-d net. The **"nedgerit"** and "**nedgelef"** links allow efficient scanning of edges

around a vertex in either a clockwise or anticlockwise direction, respectively. Once the process completed, we got a set of linked edges of the convex hull.

To output the convex hull, we make a recursive traverse of the set of edges and mark the visited edges(See RecordType.java, method `traversal, Line 268`). This traversal can start from any edge of the edge set. The current edge registers itself to a recorder first and then orderly visits it's four adjacent edges, every vertex two edges. Once the edge to visit has been visited, no further advance should be proceeded. After the recursive process completes, we have traversed the edges on the net, every edge once and only once. Then the output is done.

4. THE DISCUSSION OF EDGE GARBAGE

In the Pascal implementation of the algorithm, a memory problem is remained unsolved. During the finding process, a large number of edges are generated. Most of these edges will not appear in the final convex hull. When we do 3-d merge, we create many new edges that link the two sub hulls and simultaneously wrap the interior part. As these edges are linked to the edge chain (or net) , some old edges are inevitably released from the chain at most time. If these edges were not handled properly, a great number of memory garbage thus are accumulated. This can give rise to the situation of memory leak, especially when we deal with a point set that contains large amount of points or when we have to repeatedly do the work in one application. In my Java implementation, this problem can be solved by using the idea of non-reference object and Garbage Collection in Java. In Java, objects will be deleted by Java Runtime Environment when it determines that they are no longer being used. This process is called (automatic) garbage collection. An object is eligible for garbage collection when there are no more references to that object. With no need of programmer's intervening, these garbage could be collected and released by a special thread called Garbage Collector. The garbage collector periodically frees the memory used by objects that are no longer needed. It runs from the start of an application, and silently works during the whole lifetime. Thanks for this feature of Java, our whole work to avoid the memory leak described before is to make sure that the useless edges are not referenced by any object or variable.

In this program, during the 3-d merge process, the first step is to find the next edge to advance. This step may searches more than one perspective edges and stops at the edge that will be reserved. The other passed over edges will be unlinked by making the end edge point to a new edge. We should notice two points. One is that every edge has four pointers to other edges and equally it is referenced by four edges. That means once the edge is unlinked from the chain, one of it's two references related with its one vertex is released. The second point is that the deprecated edges will not be singly released, but released in a group. according to our algorithm's feature, we are guaranteed that (1)when we unlink references related to one vertex of the edge, the other side is still linked to an edge set that (2) all end references of the end edges of the set will be unlinked from the merged hull.

But in he process of 2-d merge, it is not the case. In other words, edges unlinked from the 3d net, may still linked to the 2d chain.

To make sure every useless edge is not referenced by other edges any longer, we first maintain all edges' references by adding them to a "container" or an array, at the time they are created. Then after the process of 3-d merge, we pick out the edges linked to the net, the rest edges are surely definitely useless and are to be destroyed)(See ConvexHull.java `memClean(), line 56`).

Until now, we can say that all deprecated edges are unlinked from the useful objects, and they now can be collected by the garbage collector.

Moreover, even if we didn't use Java, we still could accomplish this job. Now we can also use an array or "container" to maintain the references to all edges, and when the finding task completed, we use a traversal algorithm to traverse the final convex hull, and distinguish between the useless edges and reserved edges. This traversal algorithm is the same as the one provided in my program.

5. CONCLUSIONS

This is a simple implementation of Computational Geometry algorithm. I found it's very important carefully design the algorithm and find suitable data structure to solve a CG problem. Also the extreme situations have to be considered with carefully.

REFERENCES

1. A.M.Day *The Implementation of an Algorithm to Find the Convex Hull of a Set of Three-Dimensional Points. A.C.M Graphics 9,1(1990), 105-132*

2. Junhui Den Computational Geometry CourseNote Dec, 1999