3D Convex Hull: A Report (draft)

- Suhas Javagal

The objective of this project is to design and implement the Randomized Incremental Algorithm using Conflict graph data structure. The document is structured as follows. Section 1 provides an overview of the project. Section 2 will provide the user interface description. Section 3 provides a descriptions of the data structure used in this project.

1. Project Requirements

In this project, we design and implement application for visualizing 3D-convex hull. The goal of the application is to enable the user to view the execution of the algorithm in a step-by-step fashion. As the hull is created in a 3D, user should have the option to rotate and zoom into the hull in order to understand the structure better. To further aid in the viewer, it is necessary to have the facets of the hull to be viewed as opaque faces or a wireframe object.

- Input and Output: Although, a 3D point is not possible via the mouse, input to the application should be using files, where the each co-ordinates are between -1 and +1. Additionally, once the output is calculated successfully, the points on the hull should be written into an output file similarly.
- Teaching mode: The application should be designed in a manner where the user can control the steps of the

algorithm. And also the user should be able to configure the application in a manner the steps of the algorithm proceed after a certain configurable timeout. With this feature, the viewer can watch the various steps and understand the algorithm thoroughly.

- Other useful features: Creating an input file of 100 points to demonstrate this algorithm is a tedious task. Hence the application should have feature using which the a configurable set of random points are generated and 3D hull created using the random points generated. Thus relieving the user the responsibility of creating new input files. As the algorithm proceeds, stepping back is not possible. Thus in order to accommodate this feature the application should save an image after every step of the algorithm allowing user to refer to these images to understand previous steps of the algorithm.
- Development environment: The application is developed using Freeglut, GLEW libraries with Visual studio IDE. For the user interface windows GLUI library is used.
- For better visualization: The facets should be colored as depending on its state. As the algorithm proceeds various facets that get deleted or created should be identified by distinct colors for a clarity of the viewer. In addition, translucency of the facet is recommended to observe the hollow 3D hull and get the perspective on points that lie inside the hull. The points on hull should be differentiated (by color) from all other points in the input set.

2. Description of the User Interface

To enable the aspects of the application discussed in the Sec. 1, the application is provided with a user interface window and right click menu. The features are discussed below.

2.1 User Interface Window

- Application Modes (Radio buttons)
  - Step mode – Step by step execution on a button click
  - Movie mode – Step by step execution with a time interval (time interval is configurable)
  - Batch mode – Present the 3D convex hull immediately

- File operations
  - Text box - File input along with windows file picker
  - Text box - File output along with windows file picker
  - Text box - Snapshot file name (each snapshot file will be name with a prefix 1, 2, 3, etc.)

- N - Random point generation button
  - A spinner - to specify number of random points to generate
  - Button - generates random points on a sphere

- Auxiliary Buttons
  - Exit Application - Exit the application
  - Clear all - Clear all the points

2.2 Right Click Menu

- 3D Hull – generate the 3D hull immediately
- Rotate – Rotate the hull based on the position of the mouse pointer on the glut window. And stop rotation when next click is performed.
- Zoom in – A fixed step zoom-in (translate the object closer from viewer/camera)
- Zoom out – A fixed step zoom-out (translate the object farther from viewer/camera)
- Clear – Clear all the points and start fresh

3. Data structures overview

The application for 3D convex hull uses the Randomized incremental algorithm for convex hull generation. The algorithm used in this described in the text book. For the functioning of the algorithm, there are two other data structures that are implemented. These two data structures are described in the context of the project below.

3.1 **Conflict graph**: A data structure that is used to maintain all the facets that are in conflict with a point that belongs to the input set for 3D convex hull algorithms. The data structure consists of two sets. One for the input set of points and another for set of facets of the hull. Each point that randomly picked from the set is evaluated against the set of facets of the convex hull and checked if the point lies below/above the plane of the facet. If a point is evaluated to be above the facet then the points is *in conflict* and hence there exists an edge between the facet and the point if below then there exists no edge. Thus after every iteration where a random point is chosen, the point will have no conflict with any of the facets of the hull. In other words, the random point will either be part of the hull or inside the hull and hence the edges disappear.

After considering all points from the input set of points and continuously updating the conflict graph throughout the algorithm the conflict graph reaches a state where there exists no edge between the point set and facets, that is, the there exists no conflicts between points and facets. At this point the 3D convex hull for the given set of points is complete.

3.2 **Doubly-Connected edge list (DCEL):** It is not easy to achieve the conflict graph data structure without an intelligent way of representing the facets, vertices and edges. Hence the algorithm makes use of a data structure DCEL. This data structure considers each edge as two half edges both directed and going in opposite directions. Each entity will maintain certain pointers to other entities to ensure faster access. Each entity is described below.

- Vertex: Each input point is a vertex and this is the simplest object which contains the co-ordinates specified by the user.

- ➢ Halfedge: Each edge is represented as two halfedges and each halfedges are directed in an opposite direction. Each halfedge is the twin of the other. This object maintains the following information/operations.
  1) Pointer to the twin halfedge
  2) Pointer to next halfedge
  3) Pointer to the previous halfedge
  4) Pointer to the origin vertex
  5) Pointer to the destination vertex (not needed actually)
  6) Pointer to the facet whose boundary it is a part of.
  7) Get(), set() operations on twin, next, previous halfedges
  8) Get() operations on origin and destination vertex
  9) Get(), set() operation on facet it is pointing to
- ➢ Facet: A facet is the object that contains information about the facet of the hull. Each facet is bounded by half edges and as the halfedges are directed maintaining a pointer to a halfedge is sufficient. Facet supports only Get () and Set() on boundary edge pointers.

3.3 How to maintain the conflict graph information?

As the number of points to construct the given hull remains constant until the hull is generated. Each Face needs to maintain an information about what are the points for which there is a conflict. This information can be maintained as an array of 1 and 0 with semantics of *below* and *above*. Using this semantics and storing an array in each facet the conflict graph can be stored. However this becomes a problem as the number of points in the input set grows to a very big number. In that case it is suggested to have another object that maintains the conflict graph and maintains only the edges that indicate conflict and thus saving memory.

Note: This project uses no third party source code. Ideas for achieving step-by-step mode of execution was borrowed from classmates of this course CS531.