# Delaunay triangulation

## Abstract

Delaunay triangulation is a form of triangulation that tends to maximize the minimum angle of all the triangles in the triangulation. In other words, Delaunay triangulation avoids skinny triangles. The triangulation was invented by Boris Delaunay.

For more information, see http://en.wikipedia.org/wiki/Delaunay_triangulation

## Classes Included

The triangulation code consists of 4 classes:

- `Delaunay_Triangulation` – The main object used for Triangulation.
- `Point_dt` – represents a point in 3D space.
- `Triangle_dt` – represents a triangle, consists of 3 points.
- `Circle_dt` – represents a circle, consists of a point & radius.

We have also included a few classes used for testing and explaining the usage of the code:

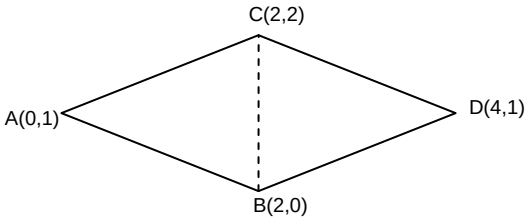
- Test Package
  - `MemoryUsage` – used to check how many points can be stored.
  - `ReadWriteTest` – used to test the integrity of file IO.
  - `TimeMeasurement` – used to test triangulation times.
  - `SimpleTriangulationUsage` – example for triangulation.
  - `SimpleFindUsage` – example for finding containing triangles.
- GUI Package
  - `MyFrame` – a User interface, demonstrates Triangulation.
  - `Visibility` – used by 'MyFrame'.

## Usage / Code Examples

### Simple Triangulation

The following example (from "`SimpleTriangulationUsage.java`") will demonstrate how to perform a triangulation.

The Triangulation objects is assigned with 4 points: A(0,1), B(2,0), C(2,2) and D(4,1), as such:



The triangulation will create 2 triangles ($\Delta ABC$ and $\Delta DCB$), that are accessible via the triangles Iterator, as demonstrated in this code:

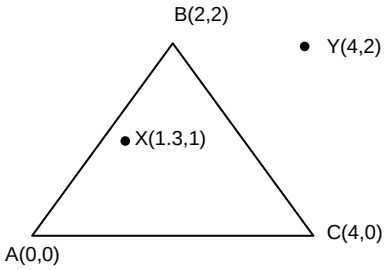
```
Delaunay_Triangulation dt = new Delaunay_Triangulation();
Point_dt pointA = new Point_dt(0, 1);
Point_dt pointB = new Point_dt(2, 0);
Point_dt pointC = new Point_dt(2, 2);
Point_dt pointD = new Point_dt(4, 1);
dt.insertPoint(pointA);
dt.insertPoint(pointB);
dt.insertPoint(pointC);
dt.insertPoint(pointD);

Iterator<Triangle_dt> iterator = dt.trianglesIterator();
while (iterator.hasNext()) {
    Triangle_dt curr = iterator.next();
    if (!curr.isHalfplane()) {
        System.out.println(curr.p1() + ", " + curr.p2() + ", " + curr.p3());
    }
}
```

Note that the code finds 2 triangles, and 4 half-planes (which are not printed, to keep this example really simple). For more information about Half-planes, see http://mathworld.wolfram.com/Half-Plane.html.

### Simple Find Triangle

Given a point P, the Delaunay Triangulation object also has the ability to find the triangle that contains P. In the following example, we will demonstrate this ability. The code can be found in "`SimpleFindUsage.java`".

For this example, we assign 3 points into the Delaunay Triangulation object, and query it to find the triangles of 2 points, as such:




The code:

```
Delaunay_Triangulation dt = new Delaunay_Triangulation();
Point_dt pointA = new Point_dt(0, 0);
Point_dt pointB = new Point_dt(2, 2);
Point_dt pointC = new Point_dt(4, 0);

dt.insertPoint(pointA);
dt.insertPoint(pointB);
dt.insertPoint(pointC);

Point_dt pointX = new Point_dt(1.3, 1);
Point_dt pointY = new Point_dt(4, 2);

FindAndPrint(dt, pointX);
FindAndPrint(dt, pointY);
```

The code for *FindAndPrint*:

```
Triangle_dt triangle = triangulation.find(point);
if (triangle.isHalfplane()){
    System.out.println("Point " + point + " is not in the triangle.");
} else {
    System.out.println("Point " + point + " is in the triangle.");
}
```