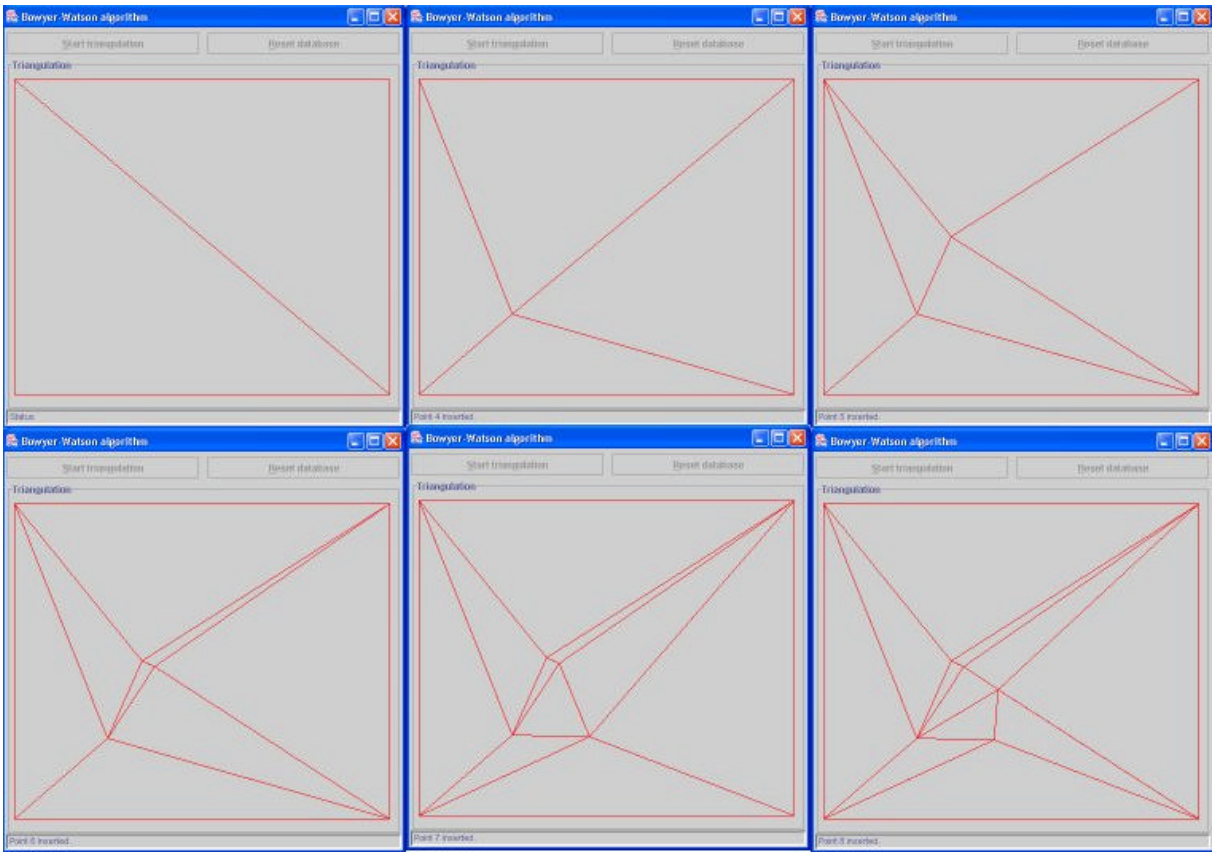
The Bowyer-Watson algorithm

An efficient implementation in a database environment



C.A.Arens
Department of Geodesy
Faculty of Civil Engineering and Geosciences
Delft University of Technology

Preface

This report describes a research towards how a TIN needs to be structured and indexed in a DBMS (Oracle) to efficiently implement the Bowyer-Watson algorithm. This provides a case for the research that determines if the present databases are able to store and use geographical data without slowing down the system and with the advantages that a DBMS offers.

The research in this report is performed during the course Geo-DBMS case study (Ge4631) as part of my study in Geodesy at the faculty of Civil Engineering and Geosciences at Delft University of Technology. The course goal is to gain experience in using databases to handle spatial data.

I would like to thank ir. E.Verbree for all his help as my supervisor. Furthermore, I would like to thank drs. C.W.Quak for getting me started with Java and drs. T.P.M.Tijssen for his help as database administrator.

Delft, July 2002,

C.A.Arens

Abstract

This report describes a research towards how a TIN needs to be structured and indexed in a DBMS (Oracle) to efficiently implement the Bowyer-Watson algorithm. This provides a case for the research that determines if the present databases are able to store and use geographical data without slowing down the system and with the advantages that a DBMS offers. There has to be a balance between decreasing the computational complexity and the increasing use of storage space.

The Bowyer-Watson algorithm primarily works with triangles. A logical choice for the data structure thus is a triangle based data structure. If Oracle Spatial is not used, this data structure contains two tables, one with the coordinates of the points, so that they are not stored multiple times, and one containing the vertices and neighbours of the triangles. The neighbours are stored to help decreasing the computational complexity. If Oracle Spatial is used, the point table is not necessary and the 3 vertices are stored as one geometry object (polygon).

The implementation of the Bowyer-Watson algorithm only queries one record at a time, while all fields contain integers. The search tree is built in the implementation. It uses the neighbour information in the triangle table to search for the triangle that contains the new point. This neighbour information is also used to find the rest of the triangles that have the new point in their circumscribing circles. When Oracle Spatial is used, it is necessary to create a spatial index to use Oracle Spatial functions to find the triangle that contains the new point. The R-tree index is used here, because it works the fastest.

Using Oracle Spatial is faster than without using it. Oracle Spatial allows a more efficient implementation of the Bowyer-Watson algorithm. It is possible to add all kinds of tricks to make the application faster. Practise confirms this, because the final version of the application is about 3 times faster than the very first version of the application.

Table of contents

1 Introduction

Geographical data is usually stored using the file system that is part of the operating system that is running. This situation distinguishes itself by the lack of possibilities to protect the stored data. For example, there are limited possibilities to provide, if necessary limited, access to different users at the same time. The possibility to recover data that has been deleted by accident is for example also limited. These problems can be solved by storing the data in a database. In this way it is possible to use one or more programs that protect the data. The collection of these programs is called the Database Management System (DBMS).
Solving the above problem causes a new problem. The storage of data in a database is namely more complicated than the storage of data in a regular file system. The data needs to be stored in a way that the DBMS can carry out its function. If this storage does not happen carefully, the system might be slower or the security of the data cannot be guaranteed.
Hence, it is important to protect the data by storing it in a database, so that the DBMS can guard these data. But it is also important to store the data in such a way that the working of the DBMS and the speed of the system can be guaranteed.

The goal of this research is to determine if the present databases are able to store and use geographical data without slowing down the system and with the advantages that a DBMS offers. To explore the possibilities of this kind of use of databases, a specific case is covered in this report. This case gives an answer to the following question: *"How does a TIN need to be structured and indexed in the Oracle DBMS to efficiently implement the Bowyer-Watson algorithm?"*
This question will be solved by developing an application that stores and visualises a TIN (a type of geographical data) by using a database. The focus will be on how to keep the computational complexity to a minimum by using an efficient data structure and index.
The application will be made in Java with Oracle 9i as database. The implementation issues of the algorithm are the most important, so the user interface and the error catching possibilities of the application are not perfect.

Section 2 contains general comments about creating a database application in Oracle. In section 3 there is a short introduction about TIN's. Section 4 continues with an algorithm that constructs a TIN. Section 5 has the possible data structures for storing a TIN and section 6 is about using spatial indices. Section 7 describes the implementation of the Bowyer-Watson algorithm in Java. Finally, section 8 has the conclusions.

2 Creating a database application in Oracle

Databases use SQL (Structured Query Language) for defining and manipulating data, e.g. creating tables and querying them. It is not possible to build an extensive application in SQL, because it does not support procedures. In Oracle there are basically two ways to create a database application, by using:

1. PL/SQL
2. Java

The following table briefly describes the difference between these two language environments [Oracle]:

PL/SQL:

- Data centric and tightly integrated into the database.
- Proprietary to Oracle and difficult to port to other database systems.
- Data manipulation is slightly faster in PL/SQL than in Java.
- Easier to use than Java (depending on your background).

Java:

- Open standard, not proprietary to Oracle.
- Incurs some data conversion overhead between the Database and Java type systems.
- Java is more difficult to use (depending on your background).

The application in this report is written in Java, because the visualisation part is easier to implement in this language environment. The application was written with the help of Oracle JDeveloper [JDeveloper], which is an Integrated Development Environment (IDE) for making applications using Java.

The advantage of running an application that uses a DBMS is that the database is managed. It is for example possible for different users to use the database, and the DBMS takes care of locking the database for all other users when one user is editing it. Another big advantage is that there are all kinds of constraints possible on the database, for example, who can use it for editing, viewing or a combination of those, but also on the fields itself, like not letting fields stay empty when inserting new data.
For protecting the database the DBMS has a transaction system. After giving commands to define and manipulate the database, it is possible to either confirm (commit) these commands or undo (rollback) them.

Java can connect to the database by using JDBC (Java DataBase Connectivity). JDBC is a set of classes and interfaces written in Java to allow Java applications to send SQL statements to the DBMS. This will typically look like this:

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
Connection conn = DriverManager.getConnection("jdbc:oracle:thin:
@url:1521:dbname","user","pass");
Statement stmt = conn.createStatement();
stmt.executeUpdate("CREATE TABLE points (id INTEGER NOT NULL)");
stmt.close();
conn.close();
```

First a connection is made by specifying the driver, the database URL, the database name and the user. Next a statement is made. The embedded SQL is shown in bold. Finally, the statement and the connection need to be closed.

When using Java to send SQL commands to the DBMS, there is a commit automatically after each command. This causes some overhead on the system, so it is better to disable this feature and send a commit command at the end of the application. The implementation (see also above code):

```
conn.setAutoCommit(false);
stmt.executeQuery(“COMMIT”);
```

It is also possible to store some parts of the application as a Java stored procedure. These Java procedures are stored in the database and can be accessed through the JDBC or SQL. The main advantage is that a stored procedure improves the application performance by eliminating some network use and making use of the usually powerful database server.

In this case, the Oracle Spatial add-in is also used. This means that the database can also hold fields of the type geometry, next to normal data types such as numbers and strings. The geometry type can hold data types such as (a collection of) points, lines and polygons. Furthermore, Oracle Spatial has a number of functions that operate on the geometry field. The main function is SDO_RELATE that can check how two geometries relate to each other, e.g. if they intersect, touch or are disjoint. Using this function has the same advantages as using a stored procedure, because the work is done by and on the database server.

3 Triangular Irregular Network (TIN)

A spatial dataset can be represented as a tessellated structure to get more information about topological relationships within this dataset. There can be regular and irregular tessellations. If the constituent objects that form the plane of the dataset are all the same the tessellations is called regular, otherwise it is called irregular [Worboys]. The most common regular tessellation is the grid, which divides the spatial dataset in squares of the same size. The advantage of a grid is that neighbouring grid cells are easily found, which means it is easy to perform operations on the dataset. The major disadvantage is that, e.g. in a point dataset, the original points are lost because of the conversion to a grid. To overcome this disadvantage, an irregular tessellation between the original points can be used. The most commonly used irregular tessellation is the triangulated irregular network (TIN). This divides the plane into a set of irregular shaped triangles.

The tessellation of a plane into triangles is not unique. In spatial applications it is necessary to keep the triangles as equilateral as possible, otherwise long and thin triangles may cut surface variations crosswise. A triangulation that has this property is the Delaunay triangulation. A Delaunay network in two dimensions consists of non-overlapping triangles where no points in the network are enclosed by the circumscribing circles of any triangle; this is called the circle criterion [Midtbø].

To reach a Delaunay triangulation the points in a spatial dataset need to be connected so that they form triangles. There are several algorithms to do this. There are basically two types of algorithms. The first is static triangulation where the triangulation is valid (Delaunay) after every single point is processed. Some examples are [Midtbø]:

- the recursive split algorithm
- the divide-and-conquer algorithm
- the step-by-step algorithm
- the modified hierarchical algorithm

The second type of algorithms that reach a Delaunay triangulation is the dynamic triangulation where the triangulation is valid during processing. This makes it possible to view the contribution of a point to the TIN. The algorithms that do this are called incremental, because they process one point at the time.

In this research an incremental algorithm by Bowyer and Watson [Lattuada] is used. In [Midtbø] this is called the incremental delete-and-build algorithm.

4 Bowyer-Watson algorithm

The Bowyer-Watson algorithm adds points sequentially into an existing Delaunay triangulation, usually starting from a very simple triangulation (e.g. one large triangle) that encloses all the points to be triangulated [Filipiak]. Then the algorithm proceeds as follows for each point that is going to be added to the TIN:

1. Add a point to the triangulation.
2. Find all existing triangles whose circumscribing circle contains the new point. This can be done to find the triangle which contains the new point first. Then the neighbours of this triangle are searched and then their neighbours, etc., until no more neighbours have the new point in their circumscribing circle (figure 4.1 left).
3. Delete these triangles; this creates a convex cavity.
4. Join the new point to all the vertices on the boundary of the cavity (figure 4.1 right).

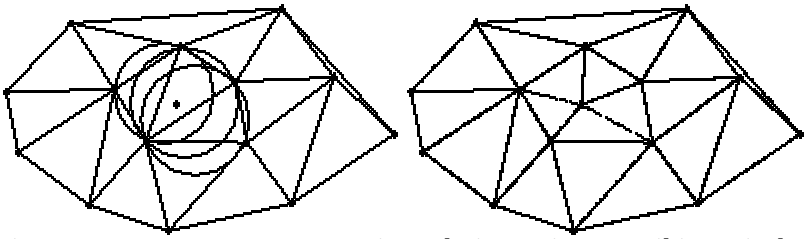


Figure 4.1: Bowyer-Watson triangulation: circumscribing circles that contain the new point (left), and the resulting triangulation (right).

The computational complexity of this algorithm is very important for determining the speed of storing TIN's in a database. This influences the data structure that is used for storing the TIN's (section XX). The computational complexity normally increases with the increase of the number of points in the TIN. Step number 2 in the algorithm above is the step where the complexity depends on the number of elements in the TIN. The time T to generate a triangulation for N points is [Lattuada]:

$$T = \sum_{k-1}^{N} (T_k + S_k)$$

where T_k is the time to find out in which triangle the new point lies and S_k the time to find all other elements in the cavity.

If we store neighbour relations for the triangles in the TIN data structure, S_k will get a complexity of $O(1)$, because it will be proportional to the number of elements in the cavity and independent from the number of points. The reason for this is that a search for the neighbouring triangle is obsolete in this situation. This means that T_k is the dominant factor in the computational complexity of this algorithm. In the worst case the complexity is $O(k)$ which yields in $T = O(N^2)$. However, the theoretical optimum is $T = O(N * \log N)$ [Lattuada]. The computational complexity can be decreased by choosing the data structure in

an efficient way. Another way to decrease the computational complexity is to implement a method that finds the triangle that contains the new point quickly.

5 Data structure

By choosing a certain data structure, the following points have to be taken into consideration:

- the maximum number of points to be triangulated;
- the available memory of the computer;
- if the TIN could be updated or not;
- the TIN construction algorithm;
- the type of operations performed upon the TIN.

In this research any number of points has to be able to be stored in the data structure. The usage of memory (storage space) should be limited to a minimum, but there has to be a balance with the computational complexity of the TIN construction algorithm, e.g. storing neighbour relationships in the data structure will cost more memory, but will also allow the algorithm to find neighbouring triangles quickly.

There are three main data structures for storing a TIN:

- Point based data structure: the points are stored together with their Delaunay neighbours.
- Triangle based data structure: the triangles themselves are stored in a table.
- Edge based data structure: the edges of the triangles are stored and the triangles can be implicitly reconstructed from the edge information.

The triangle based data structure is used in this research. There have been three different ideas to store the geometry of the triangles:

1. For each triangle a set of its three corners is stored. This can either be done by storing the coordinates directly into the data structure or by storing pointers to a table containing the coordinates of all the points. Some storage space is wasted if the coordinates are directly stored in the triangle table, because each point is in at least a couple of triangles. Storing pointers will take up less space.
   The SQL statements to create the point and triangle tables look like this:

```
CREATE TABLE points (
      id INTEGER NOT NULL PRIMARY KEY,
      x NUMBER NOT NULL,
      y NUMBER NOT NULL,
      z NUMBER NOT NULL);
```

   This will create a table to store the identifier and coordinates for each point.

```
CREATE TABLE triangles (
      id INTEGER NOT NULL PRIMARY KEY,
      vertex0 INTEGER NOT NULL,
      vertex1 INTEGER NOT NULL,
      vertex2 INTEGER NOT NULL,
      neighbour0 INTEGER NOT NULL,
      neighbour1 INTEGER NOT NULL,
      neighbour2 INTEGER NOT NULL);
```

2. When using Oracle Spatial, it is possible to store a geometry object. To store a triangle, one can make use of the polygon data type. For each triangle, the geometry has a polygon with three vertices and straight lines between these vertices.
   The SQL statement to create this triangle table looks like this:

```
CREATE TABLE triangles (
      id INTEGER NOT NULL PRIMARY KEY,
      geometry MDSYS.SDO_GEOMETRY NOT NULL,
      neighbour0 INTEGER NOT NULL,
      neighbour1 INTEGER NOT NULL,
      neighbour2 INTEGER NOT NULL);
```

   In this case it is not necessary to specify what kind of geometry the geometry column contains. This is taken care of by the way the triangles are exported to the database in Java.

3. It is also possible to store the circumscribing circles of the triangles instead of storing the triangles themselves. This allows finding all the triangles containing the new point in one pass. To determine the construction of the new triangles there should also be information about the edges of the triangle. The solution is to create a table with two geometries. There is no neighbour information necessary, because all the triangles will be found in one pass. The SQL implementation could look like this:

```
CREATE TABLE triangles (
      id INTEGER NOT NULL PRIMARY KEY,
      circle_geometry MDSYS.SDO_GEOMETRY NOT NULL,
      polygon_geometry MDSYS.SDO_GEOMETRY NOT NULL);
```

   The table contains the triangle id, the geometry in the form of a circumscribing circle and the second geometry in the form of a polygon containing three edges.

For each of these possibilities the three neighbours of the triangle are stored too, because this way it is less complex for the algorithm to find these neighbours. This will cost more memory, but there is a significant increase in the speed of the TIN construction.

By choosing option 2 and 3 there also needs to be a metadata table containing some information about the geometry object. A reference to the triangle table has to be created in the metadata table (option 2):

```
INSERT INTO user_sdo_geom_metadata VALUES ('TRIANGLES','GEOMETRY',
      MDSYS.SDO_DIM_ARRAY(MDSYS.SDO_DIM_ELEMENT('X', 0, 350000, 0.0000005),
      MDSYS.SDO_DIM_ELEMENT('Y', 350000, 700000, 0.0000005),
      MDSYS.SDO_DIM_ELEMENT('Z', -100, 400, 0.0000005)), NULL);
```

The reference contains the table name, the name of the geometry column and information about the minimum and maximum values of the coordinates that are used. In this case these are the Dutch RD-coordinates.

6 Spatial index

An index enables a DBMS to find records more quickly by using a search tree instead of checking every record sequentially. Indices can be created on each table and can improve the performance of queries dramatically. The downside is that it takes time to build indices on the records in a table and, more important, these indices need to be updated whenever the content of the table chances.

In this research querying the database consists of finding a specified record. When choosing option 1 in section 5 all the fields in the triangle table contain integers, so there is no need for a spatial index.

It was stated in section 4 that finding the triangle that contains the new point is the most complex operation. In section 7 a few ways to do this operation are described. When using option 2 or 3 in section 5 (using the geometry object) it is possible to use the Oracle function SDO_RELATE to find the triangle that contains the point. This function requires a spatial index.

There are basically two possible spatial indices in Oracle:

1. R-tree: Using the minimum bounding rectangles (MBR, figure 6.1) as index for the triangles.
2. Quad-tree: Tiling up the whole area in different levels as index for the triangles.

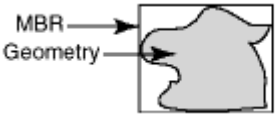


Figure 6.1: Minimum bounding rectangle of a geometry.

The SQL statement to create a default R-tree index:

```
CREATE INDEX triangles_idx ON triangles(geometry) INDEXTYPE IS
      MDSYS.SPATIAL_INDEX;
```

Oracle states that the Quad-tree option is usually faster than the R-tree when there is heavy updating activity. I tried both indices and it is almost 4 times faster to use the R-tree index to find the triangle that contains the new point.

7 Implementation

Initialisation
The triangulation will be stored as described in option 1 or 2 in section 5. Then a list of points is made from the input file that is given as argument when running the application. The Bowyer-Watson algorithm adds these points one by one into an existing Delaunay triangulation, so this triangulation needs to be constructed first. In this application a large rectangle divided in two large triangles is used as starting point. This rectangle is created by enlarging the extents from the list of points:

```
xMin = 2*xMin - xMax;
xMax = 2*xMax - xMin;
yMin = 2*yMin - yMax;
yMax = 2*yMax - yMin;
```

This ensures that the extra space around the extents of the points is evenly distributed, even if one of the values equals zero. The four points of this large rectangle are added to the points table and two triangles are formed (figure 7.1):

id	vertex0	vertex1	vertex2	neighbour0	neighbour1	neighbour2
0	0	1	2	-1	-1	1
1	2	3	0	-1	-1	0

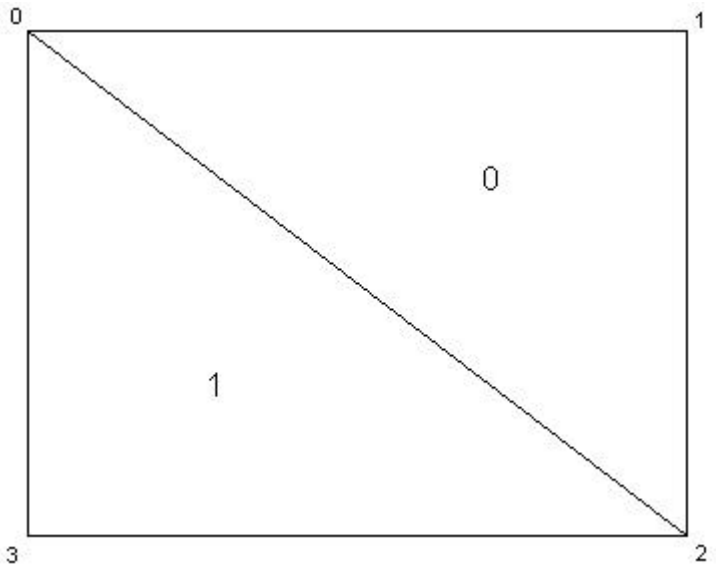


Figure 7.1: Initial triangulation consisting of 4 vertices (0,1,2,3) and two triangles (0,1).

Finally, the screen is redrawn and shows the two triangles.

Processing points
The points are processed one by one, after each point is added the Delaunay triangulation is valid. The first step is to find the triangles that have the point in their circumscribing circles (figure 7.2). The following method to evaluate if a point is in the circumscribing circle of a triangle is used:

1. Get the 3 vertices of the triangle from the database.
2. Translate these 3 vertices so that the new point is in the centre of the coordinate system. This is done, because it is easier to compute the determinant of 4 points when one of these points is in the centre of the coordinate system, because it resembles computing a determinant from 3 points.
3. Use the 3 translated vertices to compute the determinant.
4. If the determinant is smaller than or equals zero, the point is within or on the circumscribing circle of the triangle. Because of the floating point inaccuracy the implementation has an error margin of 0.00000001. This value is chosen, because there are several multiplications with 3 decimal digits (millimetres) followed by addition. There could be an error in the 9^{th} digit. However, this is only correct when using RD-coordinates. Another way to cope with the inaccuracy is the use of integer arithmetic. Here, the coordinates will be transformed to integers first and then all the calculations happen with integers. It is not necessary to have an error margin with this implementation.

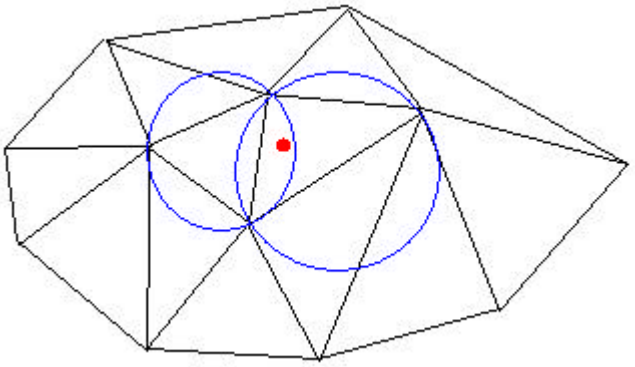


Figure 7.2: The new point (red) and the circumscribing circles (blue) that contain the new point.

The most complex task is to find the first triangle that has the point in its circumscribing circle. There are several methods to do this:

- Starting with the first triangle in the database and do the above computation for each triangle until it returns true.

- The same as above, but starting with the last triangle. It is more likely that the point that is going to be added is close to the last point that was added, so also close to the last triangle that was added.

- Using Oracle Spatial function SDO_RELATE, to relate a point geometry (the new point) with the triangle geometry (polygon). The SQL statement is:

```
SELECT * FROM triangles a WHERE SDO_RELATE(a.Geometry,
mdsys.sdo_geometry(3001,NULL,NULL,mdsys.sdo_elem_info_array(1,1,1),mdsys
.sdo_ordinate_array(x,y,z)),'mask=ANYINTERACT querytype=WINDOW')='TRUE'
```

This returns all the triangles that have any interaction with the new point. Interactions are e.g. touching, being completely within, being partly within, etc. When handling a point, this will always return the triangle that contains the point.

- If the number of triangles increases it will commonly take more and more triangles to check before the triangle that contains the point is found. Therefore another method is developed (see Appendix B). This method starts out with a triangle (in this case the last one added to accommodate for the probability that the new point is close to the last point added). For each edge of the triangle is determined if the point lies on the left side, the right side or right on top of the line that extents this edge (red/blue in figure 7.3). It is easy to see that if the point is on the right side of all the edges it is inside the triangle (green in figure 7.3). There are six other possibilities. Three of them will result in a unique direction to continue the search for the first triangle that contains the point (yellow in figure 7.3). For the other 3 areas there are two possibilities to continue the search (white in figure 7.3). If a point is on top of one or more edges there are some rules too. Table 7.1 shows a full overview. There are 3*3*3 = 27 possibilities of which 8 are not possible.

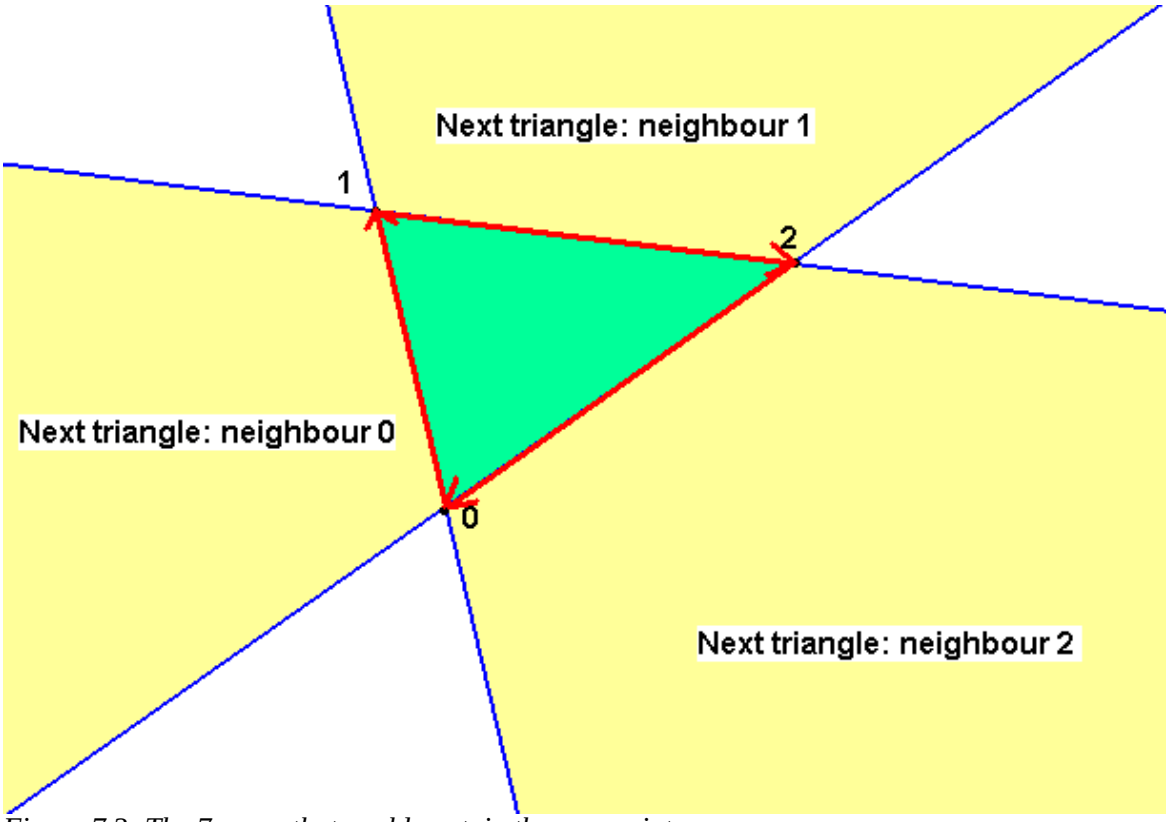


Figure 7.3: The 7 areas that could contain the new point.

To determine if a point is to the left, to the right or on top of a line a method described in [De Vries] can be used. This method takes three points (the new point and the vertices of a triangle) and returns:

- 1 if the point is left of the line.
- 0 if the point is on top of the line.
- -1 if the point is right of the line.

The Java-code looks like this:

```
private int whichSideEdge(Point p1, Point p2)
{
      double twiceTheArea = (this.x*p1.y-this.y*p1.x)+(p1.x*p2.y-
p1.y*p2.x)+(p2.x*this.y-p2.y*this.x);
      if (twiceTheArea > 0)
         return 1; // AddPoint is to the left of p1-->2
          else if (twiceTheArea == 0)
         return 0; // AddPoint is on p1-->2
          else
         return -1; // AddPoint is to the right of p1-->2
}
```

The area of the triangle through the 3 points is computed. This area is positive if all points are in an anti-clockwise sequence, negative if all points are in a clockwise sequence and zero if the new point is on top of the line. Because it is known that the two points from the triangle are in clockwise sequence, it is easy to determine on which side of this edge the new point will be.

Table 7.1: The decision structure for the edge checking method.

Edge 0 → 1	**Edge 1 → 2**	**Edge 2 → 0**	**Action**
right	right	right	first triangle found
		on line	first triangle found
		left	continue with neighbour 2
	on line	right	first triangle found
		on line	first triangle found, but point already exists
		left	continue with neighbour 2
	left	right	continue with neighbour 1
		on line	continue with neighbour 1
		left	continue with neighbour 2 or 1
on line	right	right	first triangle found
		on line	first triangle found, but point already exists
		left	continue with neighbour 2
	on line	right	first triangle found, but point already exists
		on line	combination impossible
		left	combination impossible
	left	right	continue with neighbour 1
		on line	combination impossible
		left	combination impossible
left	right	right	continue with neighbour 0
		on line	continue with neighbour 0
		left	continue with neighbour 0 or 2
	on line	right	continue with neighbour 0
		on line	combination impossible
		left	combination impossible
	left	right	continue with neighbour 0 or 1
		on line	combination impossible
		left	combination impossible

If the rules from table 7.1 are followed, there will form a trail of triangles leading to the triangle that contains the point (figure 7.4). Note that the trail can go in different

directions to find the triangle that contains the point. It is possible to compute which neighbouring triangle to follow if there are two possibilities, but this takes more computation while there is no guarantee in increasing speed.

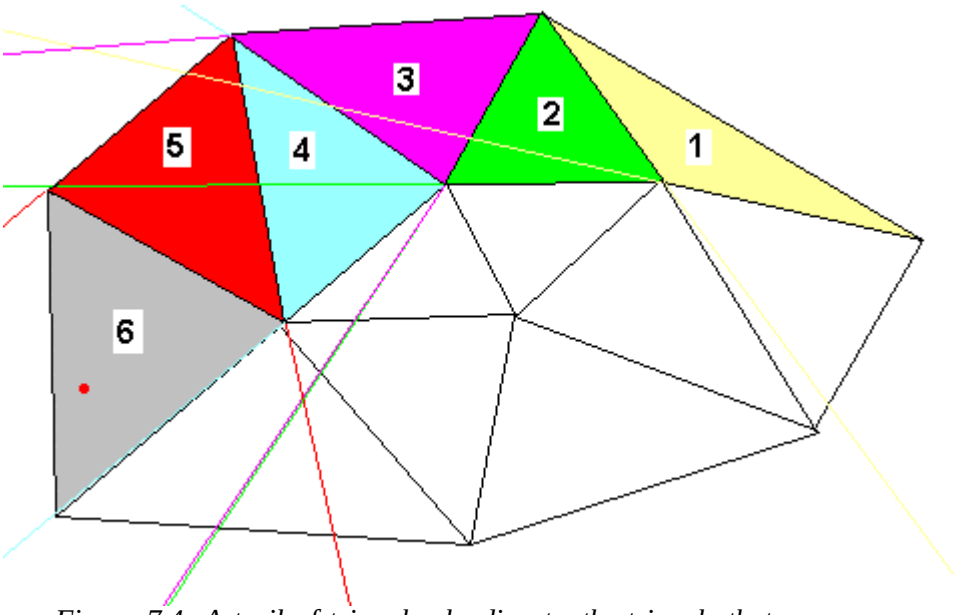


Figure 7.4: A trail of triangles leading to the triangle that contains the new point.

When the first triangle is found, this triangle is used as a starting point to find the other triangles that have the point in their circumscribing circle. This is less complex, because we stored the neighbouring triangles for each triangle. Before the point is inserted we want to know which triangles are no longer valid and have to be deleted and we want to know which combination of triangles and edges (green in figure 7.5) get a new neighbour. To make the insertion easier it is necessary to get these combinations in the right order. The right order is the edges sorted in the way that they connect to each other. It does not matter which edge is first in the list.

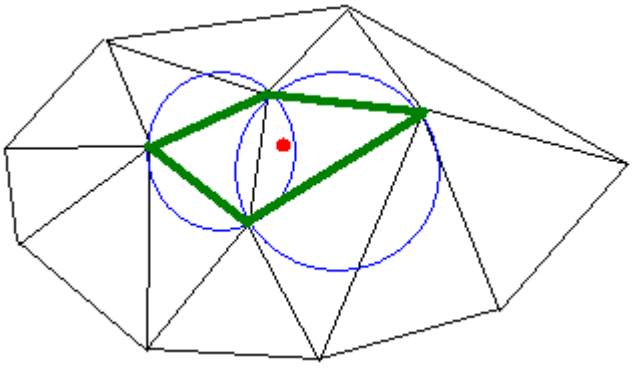


Figure 7.5: The edges that form the cavity around the deleted triangles.

This is the method that is used:

- Add the triangle (starting with the first triangle) to a list containing the triangles that are no longer valid and keep track of a list of triangles that is already checked.
- Get the sequence of neighbours that are going to be checked. Check out the edge that neighbours the last triangle (for the first triangle, any sequence is fine) and start with the following edge. This ensures that the edges are found in the right order. So, there are three possible sequences: {0,1,2}, {1,2,0} and (2,0,1}.
- Iterate over the sequence:
    - If the neighbour lies outside the triangulation (id = -1) store the current triangle and edge in the cavity lists and proceed with next neighbour.
    - Check if the triangle is already processed. Continue with the next neighbour if this is true.
    - Check if the new point is in the circumscribing circle of the neighbouring triangle. If this is true a recursion will follow, starting at the beginning of this method. If this is false this triangle is on the edge of the cavity, so the triangle and edge are stored in the cavity lists and the next neighbour in sequence is checked.
- End iteration.

Now the information needed to insert the point is available. First there is a check if the point is within the large rectangle extent and if the point is not already known. Then the triangles surrounding the cavity are updated with new neighbours. The new point will be connected to all the vertices on the cavity edge (orange lines in figure 7.6).

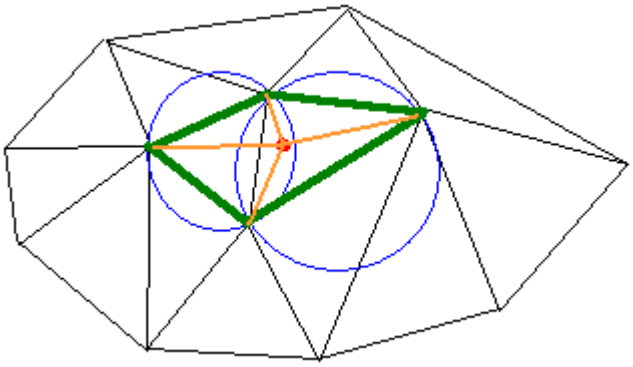


Figure 7.6: The lines (orange) that form the new triangles that contain the new point (red).

Finally, the screen is redrawn to reflect the changes before the next point is going to be processed. Figure 7.7 shows a sequence with possible application output.

Figure 7.7: A sequence of possible application output. After each step the triangulation is a valid Delaunay triangulation.

It is also possible to get all the circumscribing circles that contain the new point in one go. Option 3 from section 5 has to be used in that case. The SQL-query to get all the circumscribing circles that contain the point is:

```
SELECT id FROM circles c WHERE SDO_RELATE(c.Geometry,
mdsys.sdo_geometry(3001,NULL,NULL,mdsys.sdo_elem_info_array(1,1,1),mdsys.sd
o_ordinate_array(x,y,z)),'mask=ANYINTERACT querytype=WINDOW')='TRUE'
```

This will result in a list of id's that contain the new point in their circumscribing circles. With these id's you can select the edges from the triangles out of the other table (triangles, see option 3, section 5). The edges that are found only one time can be connected with the new point. The other edges are inside the cavity and can be thrown away.

The records of the found id's and two new records can simply be used to store the new triangles, because there is no need to store neighbour relationships.

8 Conclusions and recommendations

Conclusions

The question that is going to be answered in this section is: *"How does a TIN need to be structured and indexed in the Oracle DBMS to efficiently implement the Bowyer-Watson algorithm?"*. There has to be a balance between decreasing the computational complexity and the increasing use of storage space.

The Bowyer-Watson algorithm primarily works with triangles. A logical choice for the data structure thus is a triangle based data structure. If Oracle Spatial is not used, this data structure contains two tables, one with the coordinates of the points, so that they are not stored multiple times, and one containing the vertices and neighbours of the triangles. The neighbours are stored to help decreasing the computational complexity. If Oracle Spatial is used, the point table is not necessary and the 3 vertices are stored as one geometry object (polygon).

The implementation of the Bowyer-Watson algorithm only queries one record at a time, while all fields contain integers. The search tree is built in the implementation. It uses the neighbour information in the triangle table to search for the triangle that contains the new point. This neighbour information is also used to find the rest of the triangles that have the new point in their circumscribing circles. When Oracle Spatial is used, it is necessary to create a spatial index to use Oracle Spatial functions to find the triangle that contains the new point. The R-tree index is used here, because it works the fastest.

Using Oracle Spatial is faster than without using it. Oracle Spatial allows a more efficient implementation of the Bowyer-Watson algorithm. It is possible to add all kinds of tricks to make the application faster. Practise confirms this, because the final version of the application is about 3 times faster than the very first version of the application.

Recommendations

In this research the implementation is mostly on the client side. It is interesting to know what happens with the performance if more parts of the program are implemented on the server side, for example with the use of Java stored procedures for finding all the triangles that contain the new point.

Further more, it might be useful to do a benchmark with many points in the triangulation and test which implementation is the quickest. A closer look on spatial indexing is then necessary. In this research, the ideas for implementation were the main focus.

References

Midtbø, T., *Spatial Modelling by Delaunay Networks of Two and Three Dimensions*, Trondheim: University of Trondheim, 1993.

Lattuada, R., *A triangulation based approach to three dimensional geoscientific modelling*, London: University of London, 1998.

De Vries, J., *Ruimtelijke gegevenstoegang d.m.v de Point-Tritree*, Delft: Delft University of Technology, 1998.

Worboys, M.F., *GIS: A Computing Perspective*, London: Taylor & Francis, 1995.

Websites visited in June 2002:

Filipiak, M., *http://www.epcc.ed.ac.uk/overview/publications/training_material/tech_watch/96_tw/tw-meshgen/MeshGeneration.book_1.html*

JDeveloper, *http://www.oracle.com/ip/develop/ids/index.html?java.html*

Oracle, *http://www.oracle.com*

Appendix A: Java source code

The application is written in Java. In Java each class has its own file with source code. In this case there are 5 files:

- BowyerWatson.java: The application starts in this file. It calls the other class-files and it contains the Graphical User Interface and a class to redraw the triangulation.
- DatabaseOperation.java: This class contains all functions that operate on the database, such as creating the tables and doing the initialisation. It also reads in the points from a file.
- Point.java: This is the super class of AddPoint and contains a specific function to subtract two points from each other. It also allows instantiating points that are not going to be added to the triangulation.
- AddPoint.java: This is the class that has all the operations on a new point that is going to be added to the triangulation.
- Triangle.java: This is the class that can instantiate triangles and add them to the database.

Note that the following source code is from option 2 in section 5. The source code from the other options is slightly different. In Appendix B is the Java code for finding the first triangle with option 1 (section 5).

BowyerWatson.java

```
package bowyerwatson;

import java.sql.*;
import java.io.*;
import java.awt.*;
import javax.swing.*;
import java.awt.Color;
import java.awt.event.*;

public class BowyerWatson
{
  // ----------------------------------------------------------------------
  // Class variables.
  // ----------------------------------------------------------------------
  static Connection conn;
  JPanel menuPanel, triangulationPanel, statusPanel, guiPanel;
  JLabel statusLabel;


  // ----------------------------------------------------------------------
  // Constructor.
  // ----------------------------------------------------------------------
  public BowyerWatson() throws SQLException
  {
    menuPanel = new JPanel();
    buildMenu();
    triangulationPanel = new JPanel();
    buildTriangulationSpace();
    statusPanel = new JPanel();
    buildStatus();

    guiPanel = new JPanel();
    guiPanel.setLayout(new BorderLayout());
    guiPanel.setBorder(BorderFactory.createEmptyBorder(2,2,2,2));

    guiPanel.setPreferredSize(new Dimension(500,500));

    guiPanel.add(menuPanel, BorderLayout.NORTH);
    guiPanel.add(triangulationPanel, BorderLayout.CENTER);
    guiPanel.add(statusPanel, BorderLayout.SOUTH);
  }


  // ----------------------------------------------------------------------
  // Build menu.
  // ----------------------------------------------------------------------
  private void buildMenu() throws SQLException
  {
    JButton startButton = new JButton("Start triangulation");
    startButton.setMnemonic(KeyEvent.VK_S);
    startButton.setEnabled(false);
    startButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e)
      {/*
        statusLabel.setText("Status: Triangulation started...");
        try
        {
          // Initialize TIN.
          String dir = "c:";
```

```
        String fileName = "input.txt";
        DatabaseOperation.initTIN(conn, dir, fileName);

        triangulationPanel.repaint();
        statusLabel.setText("Status: Initialisation complete...");

        // Process point(s)
        DatabaseOperation.processPoints(conn);
      }
      catch(Exception x) {}
    */}
  });

  JButton resetButton = new JButton("Reset database");
  resetButton.setMnemonic(KeyEvent.VK_R);
  resetButton.setEnabled(false);
  resetButton.addActionListener(new ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {/*
      statusLabel.setText("Status: Resetting...");
      try
      {
        DatabaseOperation.resetDatabase(conn);
      }
      catch(Exception x)
      {
        statusLabel.setText("Status: Error in resetting database!");
      }
      statusLabel.setText("Status: Resetting...");
    */}
   });

  menuPanel.setBorder(BorderFactory.createEmptyBorder(5, 1, 5, 1));
  menuPanel.setLayout(new GridLayout(1, 2, 10, 0));
  menuPanel.add(startButton);
  menuPanel.add(resetButton);
}


// ------------------------------------------------------------------------
// Build triangulation space.
// ------------------------------------------------------------------------
private void buildTriangulationSpace()
{
    triangulationPanel.setBorder(BorderFactory.createCompoundBorder(Borde
    rFactory.createTitledBorder("Triangulation"),
    BorderFactory.createEmptyBorder(5,5,5,5)));
    triangulationPanel.setLayout(new BorderLayout());
    DrawTriangulation dt = new DrawTriangulation();
    triangulationPanel.add(dt, BorderLayout.CENTER);
}


// ------------------------------------------------------------------------
// Build status line.
// ------------------------------------------------------------------------
private void buildStatus()
{
  statusLabel = new JLabel(" Status");
```

```
    Font ft = new Font("Default", 0, 11);
    statusLabel.setFont(ft);

    statusPanel.setBorder(BorderFactory.createLoweredBevelBorder());
    statusPanel.setLayout(new GridLayout(0, 1));
    statusPanel.add(statusLabel);
  }


  // ---------------------------------------------------------------------
  // Application starts here.
  // ---------------------------------------------------------------------
  public static void main(String[] args) throws SQLException, IOException
  {
    // Timer start
    long startTime = System.currentTimeMillis();

    // Call constructor
    BowyerWatson app = new BowyerWatson();

    // Connect to database.
      conn = DatabaseOperation.connectToDatabase();

    // Build GUI
    JFrame frame = new JFrame("Bowyer-Watson algorithm");

    frame.setContentPane(app.guiPanel);

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setVisible(true);

     // Initialize TIN.
    String pathName = "c:\\input.txt";
    DatabaseOperation.initTIN(app, conn, pathName);

    // Process point(s)
    DatabaseOperation.processPoints(app, conn);

    // Timer end
    long endTime = System.currentTimeMillis();
    long duration = endTime - startTime;
    String durationMessage = " Processing time: "+duration+"
milliseconds.";
    app.statusLabel.setText(durationMessage);
    System.out.println(durationMessage);

    // Close connection to database.
   conn.close();
  }

  class DrawTriangulation extends JPanel
  {
    public void paintComponent(Graphics g)
    {
      // Paint background
      super.paintComponent(g);

      // Paint triangles.
      g.setColor(Color.red);
```

```
      for (int i=0;i<DatabaseOperation.polygonList.size();i++)
      {
        Polygon p = (Polygon)DatabaseOperation.polygonList.get(i);
        g.drawPolygon(p);
      }
    }
  }
}
```

DatabaseOperation.java

```
package bowyerwatson;

import oracle.sql.STRUCT;
import oracle.sdoapi.OraSpatialManager;
import oracle.sdoapi.adapter.*;
import oracle.sdoapi.geom.*;
import java.sql.*;
import java.io.*;
import java.util.*;
import oracle.jdbc.driver.*;
import java.awt.*;

public class DatabaseOperation
{
  // ---------------------------------------------------------------------
  // Class variables.
  // ---------------------------------------------------------------------
  private static Vector pointList = new Vector();
  private static Vector oraclePolygonList = new Vector();
  public static Vector polygonList = new Vector();
  private static double x0, y0, ax, ay;


  // ---------------------------------------------------------------------
  // Constructor.
  // ---------------------------------------------------------------------
  public DatabaseOperation()
  {
  }


  // ---------------------------------------------------------------------
  // Connects to database.
  // ---------------------------------------------------------------------
  public static Connection connectToDatabase() throws SQLException
  {
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@www.gdmc.nl:1521:geobase","a
rens","calin");
    return conn;
  }


  // ---------------------------------------------------------------------
  // Resets database.
  // ---------------------------------------------------------------------
  public static void resetDatabase(Connection conn) //throws SQLException
  {
    System.out.println("Deleting old triangle table...");
    try
    {
      Statement stmt = conn.createStatement();
      stmt.executeUpdate("DROP TABLE TRIANGLES");
      stmt.close();
    }
    catch (SQLException e)
    {
      System.out.println(e);
```

```
    }

    System.out.println("Deleting metadata reference...");
    try
    {
      Statement stmt1 = conn.createStatement();
      boolean i = stmt1.execute("DELETE FROM user_sdo_geom_metadata WHERE
table_name = 'TRIANGLES'");
      System.out.println(i);
      stmt1.close();
    }
    catch (Exception e2)
    {
      System.out.println(e2);
    }
  }


  // ----------------------------------------------------------------------
  // Empties triangle table.
  // ----------------------------------------------------------------------
  public static void emptyTriangleTable(Connection conn) throws
SQLException
  {
    System.out.println("Emptying old triangle table...");
    try
    {
      Statement stmt = conn.createStatement();
      stmt.executeUpdate("DELETE FROM TRIANGLES");
      stmt.close();
    }
    catch (SQLException e)
    {
      System.out.println(e);
    }
  }


  // ----------------------------------------------------------------------
  // Creates index on triangle table.
  // ----------------------------------------------------------------------
  public static void createIndex(Connection conn) throws SQLException
  {
    System.out.println("Creating index...");
    try
    {
      Statement stmt = conn.createStatement();
      stmt.executeUpdate("CREATE INDEX triangles_idx ON triangles(geometry)
INDEXTYPE IS MDSYS.SPATIAL_INDEX");
      stmt.close();
    }
    catch (SQLException e)
    {
      System.out.println(e);
    }
  }
```

```
  // ---------------------------------------------------------------------
  // Drops index on triangle table.
  // ---------------------------------------------------------------------
  public static void dropIndex(Connection conn) throws SQLException
  {
    System.out.println("Dropping index...");
    try
    {
      Statement stmt = conn.createStatement();
      stmt.executeUpdate("DROP INDEX triangles_idx");
      stmt.close();
    }
    catch (SQLException e)
    {
      System.out.println(e);
    }
  }

  // ---------------------------------------------------------------------
  // Creates empty table for the triangles.
  // ---------------------------------------------------------------------
  private static void createTriangleTable(Connection conn) throws
SQLException
  {
    System.out.println("Creating new table...");
    try
    {
      Statement stmt = conn.createStatement();
      stmt.executeUpdate("CREATE TABLE TRIANGLES (id INTEGER NOT NULL
PRIMARY KEY, GEOMETRY MDSYS.SDO_GEOMETRY NOT NULL, neighbour0 INTEGER NOT
NULL, neighbour1 INTEGER NOT NULL, neighbour2 INTEGER NOT NULL)");
      stmt.executeUpdate("INSERT INTO user_sdo_geom_metadata VALUES
('TRIANGLES','GEOMETRY', MDSYS.SDO_DIM_ARRAY(MDSYS.SDO_DIM_ELEMENT('X', 0,
350000, 0.0000005), MDSYS.SDO_DIM_ELEMENT('Y', 350000, 700000, 0.0000005),
MDSYS.SDO_DIM_ELEMENT('Z', -100, 400, 0.0000005)), NULL)");
      stmt.close();

    }
    catch (SQLException e)
    {
      System.out.println(e);
    }
  }


  // ---------------------------------------------------------------------
  // Create extent.
  // ---------------------------------------------------------------------
  private static void createExtent(BowyerWatson app, Connection conn)
throws SQLException, InvalidGeometryException,
GeometryOutputTypeNotSupportedException
  {
    System.out.println("Calculating extent...");

    // Return minimum and maximum values.
    Point initPoint = (Point)pointList.get(0);
    double xMin = initPoint.x, xMax = initPoint.x, yMin = initPoint.y, yMax
= initPoint.y;

    for (int i=1;i<pointList.size();i++)
    {
```

```
      Point p = (Point)pointList.get(i);
      if (p.x < xMin) xMin = p.x;
      if (p.x > xMax) xMax = p.x;
      if (p.y < yMin) yMin = p.y;
      if (p.y > yMax) yMax = p.y;
    }

    // Enlarge extent.
    xMin = 2*xMin - xMax;
    xMax = 2*xMax - xMin;
    yMin = 2*yMin - yMax;
    yMax = 2*yMax - yMin;

    // Add 4 envelope points to points table.
    Point p1 = new Point(xMin,yMax,0);
    Point p2 = new Point(xMax,yMax,0);
    Point p3 = new Point(xMax,yMin,0);
    Point p4 = new Point(xMin,yMin,0);

    // Add triangles to database.
    Triangle t1 = new Triangle(0,p1,p2,p3,-1,-1,1);
    t1.addToDatabase(conn);
    Triangle t2 = new Triangle(1,p3,p4,p1,-1,-1,0);
    t2.addToDatabase(conn);

    // Get translation
    Point[] pointList = t1.getVertices();

    double x1, y2;
    x0 = (pointList[0]).x;
    y0 = (pointList[0]).y;
    x1 = (pointList[1]).x;
    y2 = (pointList[2]).y;
    Rectangle rect = app.triangulationPanel.getBounds();
    ax = ((rect.width-25)/(x1-x0));
    ay = ((rect.height-45)/(y0-y2));
  }

  // ---------------------------------------------------------------------
  // Initializes TIN (make two big triangles around the points in the file.
  // ---------------------------------------------------------------------
  public static void initTIN(BowyerWatson app, Connection conn, String
pathName) throws SQLException, IOException,
GeometryOutputTypeNotSupportedException, InvalidGeometryException,
GeometryInputTypeNotSupportedException
  {
    // Create empty point and triangle tables.
    //resetDatabase(conn);
    //createTriangleTable(conn);
    emptyTriangleTable(conn);

    // Add points to pointList.
    addPointsToList(pathName);

    // Get extents from points and enlarge this envelope.
    createExtent(app, conn);

    // Redraw triangulation.
    getPolygons(app, conn);
  }
```

```
  // ---------------------------------------------------------------------
  // Triangulate the points in the input-file.
  // ---------------------------------------------------------------------
  public static void processPoints(BowyerWatson app, Connection conn)
throws SQLException, GeometryOutputTypeNotSupportedException,
InvalidGeometryException, GeometryInputTypeNotSupportedException
  {
    System.out.println("Begin processing points...");

    // For each point do:
    for (int i=0;i<pointList.size();i++)
    {
      // Get the point.
      AddPoint pointToAdd = new AddPoint((Point)pointList.get(i));

      // Process the point.
      pointToAdd.processPoint(app, conn);
    }
    app.statusLabel.setText(" Points processed.");
  }


  // ---------------------------------------------------------------------
  // Add the points in the input-file to the Vector pointList.
  // ---------------------------------------------------------------------
  private static void addPointsToList(String pathName) throws IOException
  {
    System.out.println("Reading points...");

    // Open input-file.
    File inFile = new File(pathName);
    BufferedReader in = new BufferedReader(new FileReader(inFile));
    String s = new String();

    // Read every line.
    while ((s = in.readLine())!=null)
    {
      // Split up the lines in point features.
      StringTokenizer st = new StringTokenizer(s);
      double x = Double.parseDouble(st.nextToken());
      double y = Double.parseDouble(st.nextToken());
      double z = Double.parseDouble(st.nextToken());
      Point p = new Point(x,y,z);

      // Add point to pointList.
      pointList.addElement(p);
    }
    in.close();
    System.out.println("Number of points: "+pointList.size());
  }


  // ---------------------------------------------------------------------
  // Draw the triangulation.
  // ---------------------------------------------------------------------
  public static void getPolygons(BowyerWatson app, Connection conn) throws
SQLException, GeometryInputTypeNotSupportedException,
InvalidGeometryException
  {
    // Empty polygonList.
    oraclePolygonList.removeAllElements();
```

```
    polygonList.removeAllElements();

    // Get all the triangles from the database.
    Statement stmt = conn.createStatement();
    GeometryAdapter sdoAdapter =
OraSpatialManager.getGeometryAdapter("SDO", "8.1.6", null, STRUCT.class,
null, conn);
    String queryString = "SELECT geometry FROM triangles";
    OracleResultSet rset = (OracleResultSet)stmt.executeQuery(queryString);

    // For each triangle do:
    while (rset.next())
    {
      // Get the triangle and draw it.
      oracle.sdoapi.geom.Polygon p =
(oracle.sdoapi.geom.Polygon)sdoAdapter.importGeometry(rset.getObject(1));
      oraclePolygonList.addElement(p);
    }
    rset.close();
    stmt.close();

    for(int i=0;i<oraclePolygonList.size();i++)
    {
      oracle.sdoapi.geom.Polygon pol =
(oracle.sdoapi.geom.Polygon)oraclePolygonList.get(i);
      Triangle tri = new Triangle(-1,pol,-1,-1,-1);
      Point[] pointList2 = tri.getVertices();

      Point firstPoint = pointList2[0];
      Point secondPoint = pointList2[1];
      Point thirdPoint = pointList2[2];

      int[] xPoints = {(int)(ax*(firstPoint.x-x0)),(int)(ax*(secondPoint.x-
x0)),(int)(ax*(thirdPoint.x-x0))};
      int[] yPoints = {(int)(-ay*(firstPoint.y-y0)),(int)(-
ay*(secondPoint.y-y0)),(int)(-ay*(thirdPoint.y-y0))};
      java.awt.Polygon paintpol = new java.awt.Polygon(xPoints,yPoints,3);
      polygonList.addElement(paintpol);
    }

    app.triangulationPanel.repaint();
  }
}
```

Point.java

```
package bowyerwatson;

public class Point
{
  // ------------------------------------------------------------------------
  // Class variables.
  // ------------------------------------------------------------------------
  double x;
  double y;
  double z;


  // ------------------------------------------------------------------------
  // Constructor #1.
  // ------------------------------------------------------------------------
  public Point(double x, double y, double z)
  {
    this.x = x;
    this.y = y;
    this.z = z;
  }


  // ------------------------------------------------------------------------
  // Constructor #2.
  // ------------------------------------------------------------------------
  public Point(Point p)
  {
    this.x = p.x;
    this.y = p.y;
    this.z = p.z;
  }


  // ------------------------------------------------------------------------
  // Substract Point from substractPoint.
  // ------------------------------------------------------------------------
  public Point substractPoint(Point substractPoint)
  {
    double theX, theY, theZ;
    theX = substractPoint.x - this.x;
    theY = substractPoint.y - this.y;
    theZ = substractPoint.z - this.z;

    Point resultPoint = new Point(theX,theY,theZ);
    return resultPoint;
  }
}
```

AddPoint.java

```
package bowyerwatson;

import java.sql.*;
import java.util.*;
import oracle.jdbc.driver.*;
import oracle.sql.STRUCT;
import oracle.sdoapi.OraSpatialManager;
import oracle.sdoapi.adapter.*;
import oracle.sdoapi.geom.*;

public class AddPoint extends bowyerwatson.Point
{
  // ---------------------------------------------------------------------
  // Class variables.
  // ---------------------------------------------------------------------
  // Vector is a resizable array; they need to be emptied after each point
that is processed.
  static Vector processedTriangles = new Vector();
  static Vector deleteTriangles = new Vector();
  static Vector cavityTriangles = new Vector();
  static Vector cavityEdges = new Vector();
  static int counter = 0;


  // ---------------------------------------------------------------------
  // Constructor #1.
  // ---------------------------------------------------------------------
  public AddPoint(double x, double y, double z)
  {
    super(x,y,z);
  }


  // ---------------------------------------------------------------------
  // Constructor #2.
  // ---------------------------------------------------------------------
  public AddPoint(Point p)
  {
    super(p);
    counter++;
  }


  // ---------------------------------------------------------------------
  // Clean up class variables.
  // ---------------------------------------------------------------------
  public static void cleanUp()
  {
    processedTriangles.removeAllElements();
    deleteTriangles.removeAllElements();
    cavityTriangles.removeAllElements();
    cavityEdges.removeAllElements();
  }
```

```
  // ------------------------------------------------------------------------
  // Process an AddPoint.
  // ------------------------------------------------------------------------
  public void processPoint(BowyerWatson app, Connection conn) throws
SQLException, InvalidGeometryException,
GeometryOutputTypeNotSupportedException,
GeometryInputTypeNotSupportedException
  {
    // Get the Delaunay cavity.
    this.locateTriangles(conn);

    // Run Insert AddPoint to TIN/Triangle.
    this.insert(app, conn);

    // Clean up class variables.
    this.cleanUp();

    // Redraw the triangulation.
    DatabaseOperation.getPolygons(app, conn);
  }


  // ------------------------------------------------------------------------
  // Locate triangles.
  // ------------------------------------------------------------------------
  private void locateTriangles(Connection conn) throws SQLException,
InvalidGeometryException, GeometryInputTypeNotSupportedException
  {
    // Locate first triangle that contains AddPoint (returns -1, if outside
triangulation).
    Triangle firstTriangle = this.locateFirstTriangle(conn);

    // If point is not outside the triangulation:
    if (firstTriangle.id != -1)
    {
      int[] nSequence = {1,2,0};
      this.neighbouringTriangle(firstTriangle, conn, nSequence);
/*

System.out.println("processedTriangles:"+processedTriangles.toString());

      System.out.println("deleteTriangles:");
      for (int i=0;i<deleteTriangles.size();i++)
        System.out.println(((Triangle)deleteTriangles.get(i)).id);

      System.out.println("cavityTriangles/cavityEdges:");
      for (int i=0;i<cavityTriangles.size();i++)

System.out.println(((Triangle)cavityTriangles.get(i)).id+"/"+cavityEdges.ge
t(i));
*/
    }
    else
    {
      deleteTriangles.addElement(firstTriangle);
    }
  }
```

```
  // ---------------------------------------------------------------------
  // Locate the first triangle that contains AddPoint.
  // ---------------------------------------------------------------------
  private Triangle locateFirstTriangle(Connection conn) throws
SQLException, InvalidGeometryException,
GeometryInputTypeNotSupportedException
  {
     // Get triangle containing point.
    Statement stmt = conn.createStatement();
    GeometryAdapter sdoAdapter =
OraSpatialManager.getGeometryAdapter("SDO", "8.1.6", null, STRUCT.class,
null, conn);
    String queryString = "Select * FROM triangles A WHERE
SDO_RELATE(A.Geometry,
mdsys.sdo_geometry(3001,NULL,NULL,mdsys.sdo_elem_info_array(1,1,1),mdsys.sd
o_ordinate_array("+this.x+","+this.y+","+this.z+")),'mask=ANYINTERACT
querytype=WINDOW') = 'TRUE'";

    OracleResultSet rset = (OracleResultSet)stmt.executeQuery(queryString);

    if (rset.next())
    {
      int id = rset.getInt(1);
      Polygon polygon =
(Polygon)sdoAdapter.importGeometry(rset.getObject(2));
      int neighbour0 = rset.getInt(3);
      int neighbour1 = rset.getInt(4);
      int neighbour2 = rset.getInt(5);
      rset.close();
      stmt.close();
      Triangle tri = new
Triangle(id,polygon,neighbour0,neighbour1,neighbour2);
      return tri;
    }
    else
    {
      Point fakePoint = new Point(-1,-1,-1);
      Triangle tri = new Triangle(-1,fakePoint,fakePoint,fakePoint,-1,-1,-
1);
      return tri;
    }
  }


  // ---------------------------------------------------------------------
  // Locates on which side of the line p1-->p2 AddPoint is.
  // ---------------------------------------------------------------------
  private int whichSideEdge(Point p1, Point p2)
  {
    double twiceTheArea = (this.x*p1.y-this.y*p1.x)+(p1.x*p2.y-
p1.y*p2.x)+(p2.x*this.y-p2.y*this.x);
    if (twiceTheArea > 0)
      return 1; // AddPoint is to the left of p1-->2
    else if (twiceTheArea == 0)
      return 0; // AddPoint is on p1-->2
    else
      return -1; // AddPoint is to the right of p1-->2
  }
```

```
  // ------------------------------------------------------------------------
  // Locate neighbouring triangles that contain AddPoint.
  // ------------------------------------------------------------------------
  public void neighbouringTriangle(Triangle tri, Connection conn, int[]
neighbourSequence) throws SQLException, InvalidGeometryException,
GeometryInputTypeNotSupportedException
  {
    deleteTriangles.addElement(tri);
    processedTriangles.addElement(new Integer(tri.id));

    int[] nbSequence = neighbourSequence;
    int[] neighbours = {tri.neighbour0, tri.neighbour1, tri.neighbour2};

    // Check all the neighbouring triangles of a triangle that contains the
AddPoint in the circle.
    for(int i=0;i<=2;i++)
    {
      int sequence = nbSequence[i];
      int id = neighbours[sequence];

      // Skip neighbours with id = -1 (outside triangulation).
      if (id != -1)
      {
        // Skip triangles that are already processed.
        if (!processedTriangles.contains(new Integer(id)))
        {
          // Get the neighbouring triangle.
          Statement stmt = conn.createStatement();
          GeometryAdapter sdoAdapter =
OraSpatialManager.getGeometryAdapter("SDO", "8.1.6", null, STRUCT.class,
null, conn);
          String queryString = "SELECT * FROM triangles WHERE id = "+id;
          OracleResultSet rset =
(OracleResultSet)stmt.executeQuery(queryString);
          rset.next();

          int triangleId = rset.getInt(1);
          Polygon polygon =
(Polygon)sdoAdapter.importGeometry(rset.getObject(2));
          int neighbour0 = rset.getInt(3);
          int neighbour1 = rset.getInt(4);
          int neighbour2 = rset.getInt(5);
          int[] theNeighbours = {neighbour0, neighbour1, neighbour2};

          rset.close();
          stmt.close();

          Triangle neighbouringTriangle = new
Triangle(triangleId,polygon,neighbour0,neighbour1,neighbour2);
          // Check if the triangle contains the point inside the circle.
          boolean pointInCircle = this.inCircle(neighbouringTriangle,
conn);

          if (pointInCircle)
          {
            // Check the neighbours of this triangle (it will be stored
with calling the function).
            int k=0;
            while (tri.id != theNeighbours[k])
            {
              k++;
```

```
            }
            int[] nSequence = {(k+1)%3,(k+2)%3,(k+3)%3};
            neighbouringTriangle(neighbouringTriangle, conn, nSequence);
          }
          else
          {
            // Add the triangle as a triangle that stays intact, but gets a
new neighbour.
            cavityTriangles.addElement(neighbouringTriangle);
            for(int j=0;j<3;j++)
              if (theNeighbours[j] == tri.id)
                cavityEdges.addElement(new Integer(j));
            processedTriangles.addElement(new Integer(id));
          }
        }
      }
      else
      {
        // If a triangle breaks up at the triangulation edge, this edge
needs to stay intact.
        cavityTriangles.addElement(tri);
        cavityEdges.addElement(new Integer(sequence));
      }
    }
  }


  // ------------------------------------------------------------------------
  // Return true if AddPoint is in the circumscribing circle of a Triangle
tri.
  // ------------------------------------------------------------------------
  public boolean inCircle(Triangle tri, Connection conn) throws
SQLException
  {
    // Triangle as 3 vertices.
    Point[] pointList = tri.getVertices();
    Point uPoint = pointList[0];
    Point vPoint = pointList[1];
    Point wPoint = pointList[2];

    // Translate triangle with AddPoint in the center.
    Point luPoint, lvPoint, lwPoint;
    luPoint = this.substractPoint(uPoint);
    lvPoint = this.substractPoint(vPoint);
    lwPoint = this.substractPoint(wPoint);

    // Get determinant.
    double determinant, det1, det2, det3;
    det1 = ((luPoint.x * luPoint.x) + (luPoint.y * luPoint.y)) *
((lvPoint.x * lwPoint.y) - (lvPoint.y * lwPoint.x));
    det2 = ((lvPoint.x * lvPoint.x) + (lvPoint.y * lvPoint.y)) *
((luPoint.x * lwPoint.y) - (luPoint.y * lwPoint.x));
    det3 = ((lwPoint.x * lwPoint.x) + (lwPoint.y * lwPoint.y)) *
((luPoint.x * lvPoint.y) - (luPoint.y * lvPoint.x));
    determinant = det1 - det2 + det3;

    // If determinant is <=0 then return true, otherwise false.
    boolean circleCriterion = false;
    if (determinant <= 0.00000001)
      circleCriterion = true;
```

```
    // Return true if Delaunay circle criterion goes.
    return circleCriterion;
  }


  // ------------------------------------------------------------------------
  // Inserts and updates triangles in database to accommodate for AddPoint.
  // ------------------------------------------------------------------------
  public void insert(BowyerWatson app, Connection conn) throws
SQLException, GeometryOutputTypeNotSupportedException,
InvalidGeometryException
  {
    // Check for point outside envelope, if so: skip insertion.
    Triangle tri = (Triangle)deleteTriangles.get(0);
    if (tri.id != -1)
    {
      // Check for known point, if so: skip insertion.
      if ((deleteTriangles.size() + 2) == cavityTriangles.size())
      {
        // Get the id for the newly formed triangles and the new point id.
        Statement stmt = conn.createStatement();
        String queryString = "SELECT id FROM triangles WHERE id = (SELECT
MAX(id) FROM triangles)";
        OracleResultSet rset =
(OracleResultSet)stmt.executeQuery(queryString);
        rset.next();
        int newTriangleId = rset.getInt(1) + 1;
        rset.close();
        stmt.close();

        // Make some lists for the new triangles part.
        Vector vertexList = new Vector();
        Vector neighbouringTriangleList = new Vector();

        int lengthCavity = cavityTriangles.size();

        // Add two new triangles to deleteTriangles (initialized with lots
of -1's).
        Point fakePoint = new Point(-1,-1,-1);
        Triangle firstNewTriangle = new
Triangle(newTriangleId,fakePoint,fakePoint,fakePoint,-1,-1,-1);
        deleteTriangles.addElement(firstNewTriangle);
        Triangle secondNewTriangle = new
Triangle(newTriangleId+1,fakePoint,fakePoint,fakePoint,-1,-1,-1);
        deleteTriangles.addElement(secondNewTriangle);

        // For every triangle around the cavity, save the new neighbours.
        for(int i=0;i<lengthCavity;i++)
        {
          Triangle cavityTriangle = (Triangle)cavityTriangles.get(i);
          int triangleId = cavityTriangle.id;
          Point[] vertices = cavityTriangle.getVertices();
          int[] neighbours = {cavityTriangle.neighbour0,
cavityTriangle.neighbour1, cavityTriangle.neighbour2};
          int rand = ((Integer)cavityEdges.get(i)).intValue();

          String[] neighbourFields = {"neighbour0", "neighbour1",
"neighbour2"};
          String neighbourField = neighbourFields[rand];

          int neighbourTriangleId = neighbours[rand];
```

```
          if (neighbourTriangleId != -1)
          {
            // Update triangles in the database.
            Statement stmt2 = conn.createStatement();
            String updateString = "UPDATE triangles SET
"+neighbourFields[rand]+" = "+((Triangle)deleteTriangles.get(i)).id+" WHERE
id = "+triangleId;
            stmt2.executeUpdate(updateString);
            stmt2.close();

            // Add a vertex to the vertex list and a neighbouring triangle
to that list.
            vertexList.addElement(vertices[((rand+1)%3)]);
            neighbouringTriangleList.addElement(new Integer(triangleId));
          }
          else
          {
            // Add a vertex to the vertex list and a neighbouring triangle
to that list.
            vertexList.addElement(vertices[rand]);
            neighbouringTriangleList.addElement(new
Integer(neighbourTriangleId));
          }
        }

        // New triangles part.
        for(int i=0;i<lengthCavity;i++)
        {
          //  Two existing vertices form the triangle together with the new
point.
          int nextVertex = (i+1) % lengthCavity;
          int nextNextVertex = (i+lengthCavity-1) % lengthCavity;

          int id = ((Triangle)deleteTriangles.get(i)).id;
          Point vertex0 = (Point)(vertexList.get(i));
          Point vertex1 = (Point)(vertexList.get(nextVertex));
          Point vertex2 = this;
          int neighbour0 =
((Integer)neighbouringTriangleList.get(i)).intValue();
          int neighbour1 = ((Triangle)deleteTriangles.get(nextVertex)).id;
          int neighbour2 =
((Triangle)deleteTriangles.get(nextNextVertex)).id;

          Triangle triangleToAdd = new
Triangle(id,vertex0,vertex1,vertex2,neighbour0,neighbour1,neighbour2);

          // Use two new triangles, plus the rest existing rows in the
database (insert/update).
          if (i < (lengthCavity-2))
            triangleToAdd.updateInDatabase(conn);
          else
            triangleToAdd.addToDatabase(conn);
        }
        app.statusLabel.setText(" Point "+counter+" inserted.");
        System.out.println("Point "+counter+" inserted.");
      }
      else
      {
        app.statusLabel.setText(" Point "+counter+" known.");
        System.out.println("Point "+counter+" known.");
      }
```

```
    }
    else
    {
      app.statusLabel.setText(" Point "+counter+" outside triangulation.");
      System.out.println("Point "+counter+" outside triangulation.");
    }
  }
}
```

Triangle.java

```
package bowyerwatson;

import oracle.sql.STRUCT;
import oracle.sdoapi.OraSpatialManager;
import oracle.sdoapi.adapter.*;
import oracle.sdoapi.geom.*;
import java.sql.*;
import oracle.jdbc.driver.*;

public class Triangle
{
  // ---------------------------------------------------------------------
  // Class variables.
  // ---------------------------------------------------------------------
  int id;
  Polygon polygon;
  int neighbour0;
  int neighbour1;
  int neighbour2;


  // ---------------------------------------------------------------------
  // Constructor #1.
  // ---------------------------------------------------------------------
  public Triangle(int id, Point vertex0, Point vertex1, Point vertex2, int
neighbour0, int neighbour1, int neighbour2) throws InvalidGeometryException
  {
    this.id = id;

    GeometryFactory gF = OraSpatialManager.getGeometryFactory();
    double coordArray[] = {vertex0.x, vertex0.y, vertex0.z, vertex1.x,
vertex1.y, vertex1.z, vertex2.x, vertex2.y, vertex2.z, vertex0.x,
vertex0.y, vertex0.z};
    LineString[] ls = {gF.createLineString(3,coordArray)};
    this.polygon = gF.createPolygon(ls);

    this.neighbour0 = neighbour0;
    this.neighbour1 = neighbour1;
    this.neighbour2 = neighbour2;
  }


  // ---------------------------------------------------------------------
  // Constructor #2.
  // ---------------------------------------------------------------------
  public Triangle(int id, Polygon polygon, int neighbour0, int neighbour1,
int neighbour2) throws InvalidGeometryException
  {
    this.id = id;
    this.polygon = polygon;
    this.neighbour0 = neighbour0;
    this.neighbour1 = neighbour1;
    this.neighbour2 = neighbour2;
  }
```

```
  // ---------------------------------------------------------------------
  // Constructor #3.
  // ---------------------------------------------------------------------
  public Triangle(Triangle tri)
  {
    this.id = tri.id;
    this.polygon = tri.polygon;
    this.neighbour0 = tri.neighbour0;
    this.neighbour1 = tri.neighbour1;
    this.neighbour2 = tri.neighbour2;
  }


  // ---------------------------------------------------------------------
  // Return vertices of a triangle.
  // ---------------------------------------------------------------------
  public Point[] getVertices()
  {
    Polygon p = this.polygon;
    CurveString cs = p.getRingAt(0);
    CoordPoint cp0 = cs.getPointAt(0);
    CoordPoint cp1 = cs.getPointAt(1);
    CoordPoint cp2 = cs.getPointAt(2);
    Point wPoint = new Point(cp0.getX(),cp0.getY(),cp0.getZ());
    Point vPoint = new Point(cp1.getX(),cp1.getY(),cp1.getZ());
    Point uPoint = new Point(cp2.getX(),cp2.getY(),cp2.getZ());
    Point[] pointList = {wPoint, vPoint, uPoint};
    return pointList;
  }


  // ---------------------------------------------------------------------
  // Add the triangle to the database.
  // ---------------------------------------------------------------------
  public void addToDatabase(Connection conn) throws SQLException,
GeometryOutputTypeNotSupportedException, InvalidGeometryException
  {
    Statement stmt = conn.createStatement();
    GeometryAdapter sdoAdapter =
OraSpatialManager.getGeometryAdapter("SDO", "8.1.6", null, STRUCT.class,
null, conn);
    Geometry geom = (Geometry)this.polygon;

    String queryString = "INSERT INTO triangles VALUES(?, ?, ?, ?, ?)";
    PreparedStatement ps = conn.prepareStatement(queryString);
    ps.setInt(1, this.id);
    ps.setObject(2, sdoAdapter.exportGeometry(STRUCT.class, geom));
    ps.setInt(3, this.neighbour0);
    ps.setInt(4, this.neighbour1);
    ps.setInt(5, this.neighbour2);
    ps.executeUpdate();
    stmt.close();
  }


  // ---------------------------------------------------------------------
  // Updates the triangle to the database.
  // ---------------------------------------------------------------------
  public void updateInDatabase(Connection conn) throws SQLException,
GeometryOutputTypeNotSupportedException, InvalidGeometryException
  {
```

```
      Statement stmt = conn.createStatement();
      GeometryAdapter sdoAdapter =
OraSpatialManager.getGeometryAdapter("SDO", "8.1.6", null, STRUCT.class,
null, conn);
      Geometry geom = (Geometry)this.polygon;

      String queryString = "UPDATE triangles SET geometry = ?, neighbour0 =
?, neighbour1 = ?, neighbour2 = ? WHERE id = ?";

      PreparedStatement ps = conn.prepareStatement(queryString);

      ps.setObject(1, sdoAdapter.exportGeometry(STRUCT.class, geom));
      ps.setInt(2, this.neighbour0);
      ps.setInt(3, this.neighbour1);
      ps.setInt(4, this.neighbour2);
      ps.setInt(5, this.id);
      ps.executeUpdate();
      stmt.close();
  }
}
```

Appendix B: Java source code finding first triangle option 1 (section 5)

```
  // ---------------------------------------------------------------------
  // Locate the first triangle that contains AddPoint.
  // ---------------------------------------------------------------------
  private Triangle locateFirstTriangle(Connection conn) throws SQLException
  {
    // Initialise counter
    int counter = 0;

    // Get last triangle from the database.
    Statement stmt2 = conn.createStatement();
    String queryString2 = "SELECT MAX(id) FROM triangles";
      OracleResultSet rset2 =
(OracleResultSet)stmt2.executeQuery(queryString2);
    rset2.next();
    int theId = rset2.getInt(1);
    rset2.close();
    stmt2.close();

    Triangle tri = new Triangle(-2,-1,-1,-1,-1,-1,-1);
    boolean foundFirstTriangle = false;

    while (!foundFirstTriangle)
    {
      counter++;

      if (tri.id == -1)
        break;

      // Get triangle: theId from the database.
      Statement stmt = conn.createStatement();
      String queryString = "SELECT * FROM triangles  WHERE id = "+theId;
      OracleResultSet rset =
(OracleResultSet)stmt.executeQuery(queryString);

      rset.next();
      tri.id = rset.getInt(1);
      tri.vertex0 = rset.getInt(2);
      tri.vertex1 = rset.getInt(3);
      tri.vertex2 = rset.getInt(4);
      tri.neighbour0 = rset.getInt(5);
      tri.neighbour1 = rset.getInt(6);
      tri.neighbour2 = rset.getInt(7);

      rset.close();
      stmt.close();

      // Get the coordinated of the vertices from the database.
      Statement stmt1 = conn.createStatement();
      String queryString1 = "SELECT * FROM points WHERE id =
"+tri.vertex0+" or id = "+tri.vertex1+" or id = "+tri.vertex2;
      OracleResultSet rset1 =
(OracleResultSet)stmt1.executeQuery(queryString1);

      rset1.next();
      int idu = rset1.getInt(1);
      double xu = rset1.getDouble(2);
      double yu = rset1.getDouble(3);
      double zu = rset1.getDouble(4);
      Point point2 = new Point(idu,xu,yu,zu);
```

```
rset1.next();
int idv = rset1.getInt(1);
double xv = rset1.getDouble(2);
double yv = rset1.getDouble(3);
double zv = rset1.getDouble(4);
Point point1 = new Point(idv,xv,yv,zv);

rset1.next();
int idw = rset1.getInt(1);
double xw = rset1.getDouble(2);
double yw = rset1.getDouble(3);
double zw = rset1.getDouble(4);
Point point0 = new Point(idw,xw,yw,zw);
rset1.close();
stmt1.close();

int wS01 = this.whichSideEdge(point0, point1);
int wS12 = this.whichSideEdge(point1, point2);
int wS20 = this.whichSideEdge(point2, point0);
System.out.println(wS01+" "+wS12+" "+wS20);

int result = -1;

switch (wS01)
{
  case -1:
    switch (wS12)
    {
      case -1:
        switch (wS20)
        {
          case -1:
            result = 4;
            break;
          case 0:
            result = 4;
            break;
          case 1:
            result = 2;
            break;
        }
        break;
      case 0:
        switch (wS20)
        {
          case -1:
            result = 4;
            break;
          case 0:
            result = 4;
            break;
          case 1:
            result = 2;
            break;
        }
        break;
      case 1:
        switch (wS20)
        {
          case -1:
```

```
          result = 1;
          break;
        case 0:
          result = 1;
          break;
        case 1:
          result = 2;
          break;
      }
      break;
  }
  break;
case 0:
  switch (wS12)
  {
    case -1:
      switch (wS20)
      {
        case -1:
          result = 4;
          break;
        case 0:
          result = 4;
          break;
        case 1:
          result = 2;
          break;
      }
      break;
    case 0:
      switch (wS20)
      {
        case -1:
          result = 4;
          break;
        case 0:
          result = -1;
          break;
        case 1:
          result = -1;
          break;
      }
      break;
    case 1:
      switch (wS20)
      {
        case -1:
          result = 1;
          break;
        case 0:
          result = -1;
          break;
        case 1:
          result = -1;
          break;
      }
      break;
  }
  break;
case 1:
  switch (wS12)
```

```
          {
            case -1:
              switch (wS20)
              {
                case -1:
                  result = 0;
                  break;
                case 0:
                  result = 0;
                  break;
                case 1:
                  result = 0;
                  break;
              }
              break;
            case 0:
              switch (wS20)
              {
                case -1:
                  result = 0;
                  break;
                case 0:
                  result = -1;
                  break;
                case 1:
                  result = -1;
                  break;
              }
              break;
            case 1:
              switch (wS20)
              {
                case -1:
                  result = 0;
                  break;
                case 0:
                  result = -1;
                  break;
                case 1:
                  result = -1;
                  break;
              }
              break;
          }
          break;
      }
      if (result == -1)
        tri.id = -1;
      else if (result == 0)
        theId = tri.neighbour0;
      else if (result == 1)
        theId = tri.neighbour1;
      else if (result == 2)
        theId = tri.neighbour2;
      else
        foundFirstTriangle = true;
    }
    System.out.println("Number of triangles processed to find first
containing point: "+counter);
    return tri;
  }
```