Design Approach & Tests

The following sections provide the information on the design of the three classes provided alongside some tests conducted mainly to prove that it works. The results of all tests have been printed out to the screen when executing the main.cpp file.

Vertex Class

The Vertex class holds information for a point in space identified by an ID. The class only supports three-dimensional points since the application only demands this. It can be made such that a pointer is used rather than the hardwired length which can be done if the client requires it.

Alongside the basic elements such as the constructors, destructors, and getters/setters, the class also consists of two overloaded operators. First of which is the array access operator (i.e. []) which is used as getters and setters for the coordinate array. The second is the less than operator (i.e. <) which was designed to implement the STL sort algorithm to sort the points based on their IDs.

The main tests performed for this class are for the sort algorithm which is discussed within the Triangulation class section.

Triangle Class

Within this class, the information for a triangle is stored. This includes an array of vertices or the vertex IDs belonging to this triangle, the circumcentre and the radius of its circumcircle, the area of the triangle, its attributes, and an ID.

As for the methods provided here, a Triangle class object can identify whether a point lies inside it or its circumcircle through two different methods named *isPointInTriangle()* and *isPointInCircumcircle()*, respectively. Below are some tests performed on the former method.

Test 1

Objective:

To test whether the *isPointInTriangle()* method is able to detect a point which lies inside the triangle.

Methodology:

A point which lies within a particular triangle was chosen and the method was invoked using that as a parameter through a Triangulation class object. MATLAB was then used to visualise the point and the triangle to ensure it was within the triangle.

Results & Observations:

The method was able to detect that the point marked in Figure 1 lies inside of it.

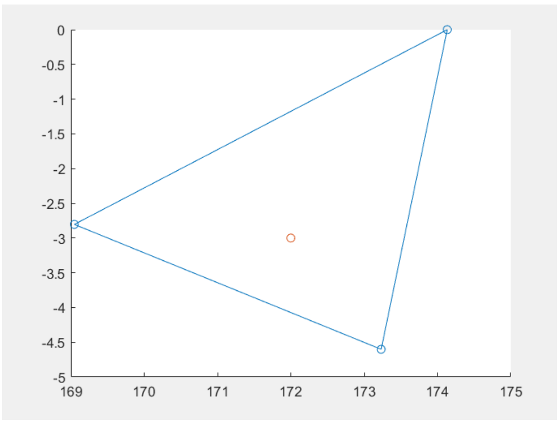

Figure 1 - The function was able to detect that this triangle contains this point.

End of Test 1

Test 2

Objective:

To test the performance of the *isPointInTriangle()* method once again but with an extreme case of the point being a vertex of multiple triangles.

Methodology:

The method is invoked using another point which is a vertex of multiple triangles, including the triangle it was invoked using.

Results & Observations:

Six out of the seven points were identified successfully however, the reason why one point went under the radar is due to the fact that one of the weights were calculated as -0.00000010152 and this was not rounded as a zero. Hence, precision issues begin to appear within the program.

End of Test 2

Moving on, as for the *isPointInCircumcircle()* method, an assumption is made that prior to the call of this method the circumcentre and radius are already calculated using the *calculateCircumcentre()* which is another method within this class. Usually, this function would be called through the Triangulation class method *calculateCircumcentreOf()*. Below are tests performed on these methods.

Test 3

Objective:

To verify whether the radius and circumcentre point provided by the *calculateCircumcentre()* method is accurate.

Methodology:

At first, since it is the prerequisite, the *calculateCircumcentre()* method is invoked for a particular triangle chosen at random. We can use MATLAB to visualise whether all vertices of the triangle fall on the circumference of the triangle or not.

Results & Observations:

Figure 2 shows the results from MATLAB. The point marked is the value calculated by the method. The circumcircle was drawn using the radius and centre points which indeed show accurate results as all three vertices of the triangle are on its circumference.

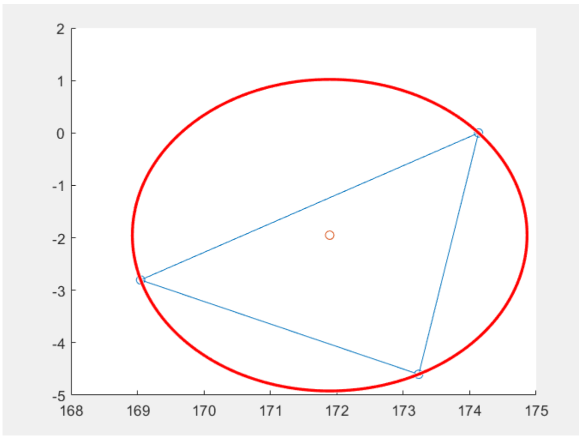

Figure 2 - Visualizing the results procured from the circumcenter method.

End of Test 3

Test 4 & 5

Objective:

To verify whether the implemented method can detect if a point is within a triangle’s circumcircle.

Methodology:

Two points are chosen such that they fall within and outside the circumcircle. The *isPointInCircumcircle()* method is invoked using these.

Results & Observations:

Results showcased that the method is able to identify each correctly. A point to note here is that a vertex is considered inside the circumcircle when it lies on the circumference itself. This was configured in such a manner since when performing Delaunay Triangulation (DT), adding a point which potentially lies on the circumference of a triangle's circumcircle might require the triangle to readjust.

End of Test 4 & 5

Next, to calculate the area of the triangle, the *calculateArea()* method is provided. This is once again invoked by the Triangulation class (*calculateAreaOf()*). The area is calculated using the vertices of the triangle.

Test 6

Objective:

To be able to calculate the area of a triangle using the *calculateArea()* method successfully.

Methodology:

The calculations are verified by squaring the 11th attribute of the associated triangle. As the properties listed at the end of the .tri files, this represents the sqrt(area).

Results & Observations:

The result provided by the method was the same as the one taken from the attributes. Note that the algorithm used by the method was also tested with a dummy triangle during development which also ensured that the functionality is as expected.

End of Test 6

Alongside the above, the Triangle class is also packed with overloaded operators, namely the less than operator (i.e. <) and the array access operator (i.e. []). These follow the same patterns as for the Vertex class with the exception that the array access operator is used for the vertex IDs of the triangle.

Triangulation Class

This class is the main interface binding together the above two classes by having containers for each type alongside some advanced features to aid in solving the triangulation problem. Methods such as *isPointInAnyTriangle()*, *calculateCircumcentreOf()*, *isPointInCircumcircle(),* and *calculateAreaOf()* simply call the respective Triangle class methods to perform the tasks and were used for tests discussed above providing accurate results in all but one case (due to precision). On the other hand, there are other methods which are yet to be verified and are discussed below.

The *isOldPointInCircumcircle()* method was programmed to provide aid for the *isDelaunay()* method. The overall functionality of the former was to check if any existing point in the mesh falls inside the circumcircle of any triangle other than itself, which in turn is able to answer if the mesh is DT or not. The reason why the *isPointInCircumcircle()* method was not used here was because that method will be comparing floating-point numbers to check if the point is the same as one of the triangle's vertexes. However, it might just be easier to compare old points with associated IDs instead. In addition to this difference, the *isOldPointInCircumcircle()* does not have the triangle container as an input (used to store the output) since, as mentioned above, it is used to aid the check if the mesh is DT or not.

Test 7

Objective:

To be able to verify the functionality of the *isOldPointInCircumcircle()* method by invoking the *isDelaunay()* method and analysing the result.

Methodology:

In order to test the functionality of the two methods, the *triangulation#1.tri* was checked to see if it was DT or not. Since no information has been provided whether this mesh follows DT, all points which violate the DT rules are printed to the screen and visualised to test correctness.

Results & Observations:

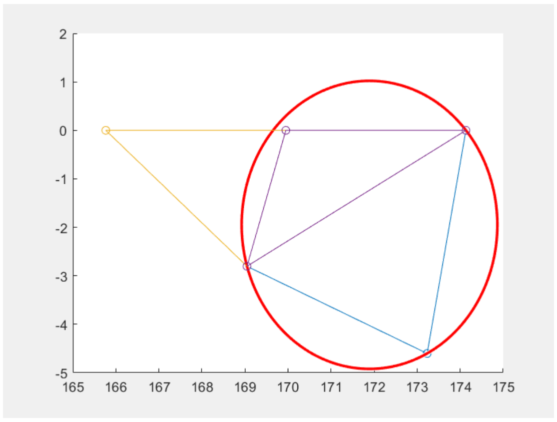

Figure 3 - It is clearly seen that one of the purple triangle's vertex lies inside the circumcircle of the blue triangle.

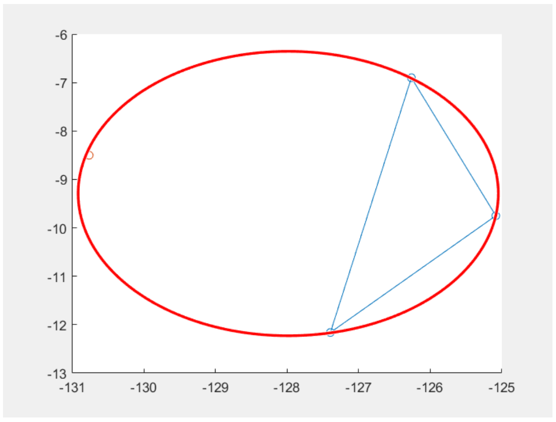

Figure 4 - Another point which just about sits on the circumference of the circumcircle.

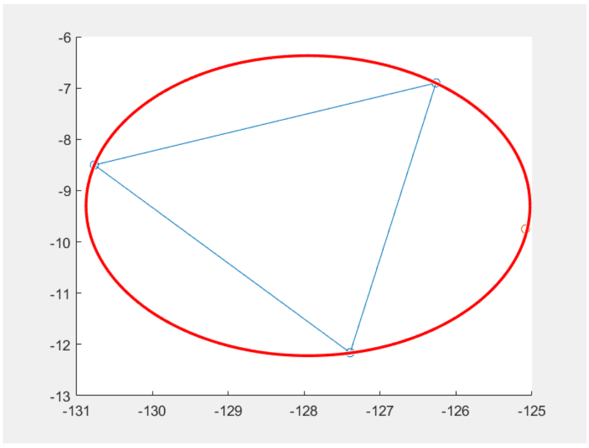

Figure 5 - Another point which is very close to the circumference.

As seen from the figures, there is clear evidence that this triangulation is not DT compliant. Although it may be also be due to the precision of the results which might cause some inaccuracy as some points lie very close to the circumference of the circumcircle. Through this test, it is clear that the two functions work correctly but might face some precision issues which is uncontrollable.

End of Test 7

The next test performed is for the integration method within the class. This, along with the specific approximation methods, utilize generic programming to provide any function to be integrated over the mesh. The method has another input which is used to select the type of approximation being used to calculate the answer.

Test 8

Objective:

To be able to verify the functionality of the integration method through its calculated output.

Methodology:
The test uses a mesh with a single triangle (inside *Test8.tri* file) and the function to integrate is set to one (made as a functor, *one*). In theory, this should return the area of the triangle using both methods.

Results & Observation:
The calculated value is indeed the area of the sole triangle. Further testing is recommended but due to the given budget, this test will suffice proving the functionality.

End of Test 8
In order to allow data to be streamed into the Triangulation-type objects naturally, the streaming operators have been overloaded. Each perform the read/write on a given wildcard T-type stream. The reason why this has been templated is to not limit the possibilities of the stream type to files only. Testing of the input stream operator have been conducted throughout the above tests clearly showing the appropriate functionality. A point to bear in mind here is that when the data is imported into the object, it will sort the points and triangles according to their IDs using the STL sort algorithm. Now, this does slow down the program however, it will allow a straightforward access of the objects in the container throughout the program and will make it more efficient as there would not be a search each time an access is required. Note that this will only be the case if no ID is skipped.

Test 9
Objective:
To be able to write out the mesh information in the same manner they were read in using the output stream operator.

Methodology:
The mesh with a single triangle used in the previous test will be written out to a new file to see if the data is placed correctly.

Results & Observations:
From the created file, it can be seen that the long decimal values are chopped off when stored as a float but other than this, the format is such that it can be used within the client's viewer application. One might argue that the read/write do not provide error checking however, this is done intentionally to allow the wildcard stream to not be forced to include the specific methods for error checking but also for performance.

End of Test 9
The class is also designed to allow the user to add a new vertex or triangle.

Test 10
Objective:
To be able to add a new vertex to the mesh.

Methodology:
The method to add new vertex is invoked to see if the data values and ID are correctly placed in the container. Values from the container are then printed in the format: ID X Y Z

Results & Observations:
The printed results are as expected, where the ID is incremented from the last one in the container and the data is present.

End of Test 10

Test 11

Objective:

To be able to add a new triangle into the mesh.

Methodology:

A similar test to the previous is performed when adding a new triangle.

Results & Observations:

Results are once again as expected (printed out in the format: ID v0 v1 v2). Note that calls to both methods (adding vertex/triangle) allow writing out the correct data to an output stream. Moreover, the user is also expected to add the attributes of the triangle using the setters later, if needed.

End of Test 11

Note that the other .tri files provided have also been tested with the program but with different test triangles. The program works seamlessly with these as well. As for whether the Triangulation is DT or not, the *triangulation#3* is the only DT. Tests have also successfully been executed on Jenna.

Design Strengths & Weaknesses

A list of the perceived strengths:

- Containers hold pointers rather than objects for efficient push_back (no copying of the whole object).
- Key assumption on the IDs being incremental and not skipped will allow quick access instead of having to perform find_if all the time.
- The number of attributes has not been hardwired to a value for flexibility.
- Use of operators in classes to access data easily.
- Generic programming to allow codes to be flexible and robust.
- Optimization techniques have been applied where applicable.
- STL algorithms being used provide the most optimised code for doing the job.
- Triangle objects only store Vertex IDs which is very similar to storing pointers to the Vertex. Far more efficient than having a copy of each Vertex.
- Since sort is only applied once when the data is brought into the object, vectors are suitable containers here although a map container with IDs being paired to particular objects would also be interesting to investigate.

A list of the perceived weaknesses:

- Program uses prints to allow the client to see program status however, this heavily effects program execution and should be removed once the client is happy.
- To check if the mesh is Delaunay Triangulation, as long as one violation to the rule is found, a decision can be made. But note that in the implementation, to show correct functionality, all violations are shown.
- Adding a new vertex or triangle may be inefficient to do with vectors since if the capacity is reached, it will reallocate which involves copying lots of data. It is recommended that new points are added to the .tri file itself.
- Another assumption is that points do not have attributes.
- The read/write method for points and cells could have been templated such that there is only the need of one of each (two instead of four) however, the data structure would have to be changed for which there is no more budget.

- Having an if-statement within the for-loop in the *isPointInCircumcircle()* and *isOldPointInCircumcircle()* method is inefficient however we must filter the point if it belongs to the triangle itself. Moreover, the print to the screen can also be removed for better performance.
- Numerical precision is vital for such applications however, using C++ alone will not provide this. Instead, the GNU Multiple Precision Arithmetic Library (GMP) should be included to overcome this limitation.
- When checking if the point is in any triangle, an alternative mechanism which would be far more efficient is where the triangles are first sorted according to which are near each other and then using the x-y coordinates. Once the triangles are sorted in such a manner, the vertex can be check exactly at the centre of the container. From there it would go either left or right based on which side the point lies. The algorithm would then check in the midpoint of that half, and so on.

Parallelization of Code

One of the mechanisms to improve performance of a code is to parallelize it. Parallelizing was tested briefly on loops found throughout the code. Few points are discussed here on the key findings. Firstly, loops using iterators need to have a >, >=, <, or <= check in order for it to be parallelized, which is not the case as != is used. Moreover, other loops such as for File I/O are usually not parallelized since I/O and parallel execution do not get along very well. Generally, having to access member variables of objects in parallel regions is not straight-forward. It is recommended that a function is called instead which has the parallel region within and the needed parameters passed as function parameters. This was not tested fully due to the budget constraints however, there is a potential to parallelise the code in such a manner.