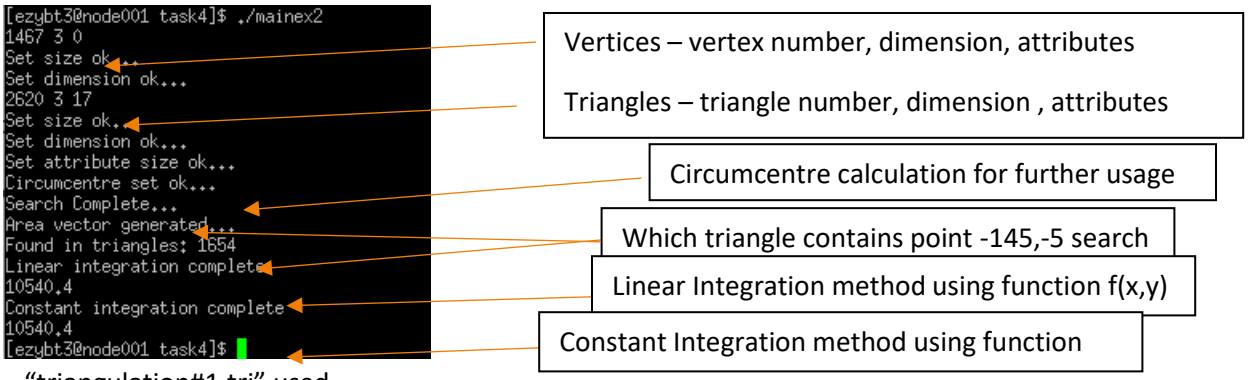
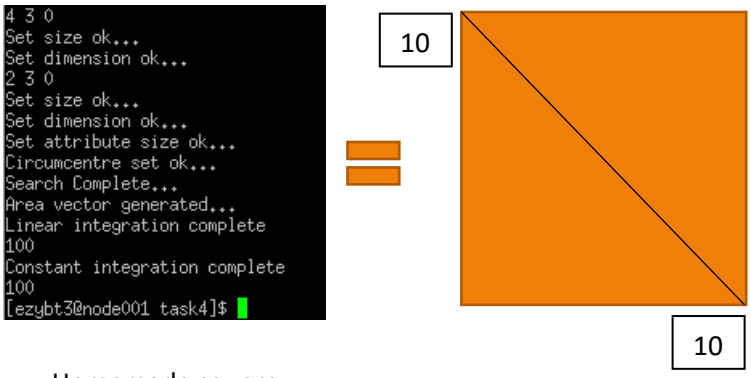
1 PART 1



"triangulation#1.tri" used

"Triangulation#2.tri" used



Homemade square

The required interface is tested with the sample triangulation file that were given. To make things simpler a square was designed in the same format of the triangulation files and tested with the interface. The square given has dimensions 10x10 therefore its area is 100 units. Integration functions returns the area because function f(x,y) returns 1 hence 3rd dimension is not involved. On the other hand, by looking at the triangulation#1 image output, its area was estimated to be 364x30 = 10920 units (low precision) and the interface found its area as 10540.4.

Point search function is also implemented and a test value was used for "triangulation#1.tri". Points used were (-145,-5) and this returned the triangle 1654. When this triangle's vertices were inspected it was found that vertices lied on points v1(-144.125,-6.39464) , v2(-144.537 ,-3.97344) , v3(-145.859 ,-5.06022) as expected.

File output of the triangulation is very similar to the original file, the difference being the string attributes at the end of the file were not implemented hence not shown at the output.

For faster further use of the individual triangle's circumcenters and areas, these values are saved in the class.

All classes are templated therefore could be used with different types. Triangulation class is built from the previously built vertex and triangle classes.

With the current data provided it was seen that the precision does not exceed beyond the compiler capabilities. Demonstration was done using floats but since the classes are templated they could be implemented in doubles to increase precision.

CPP file found in . /main.cpp

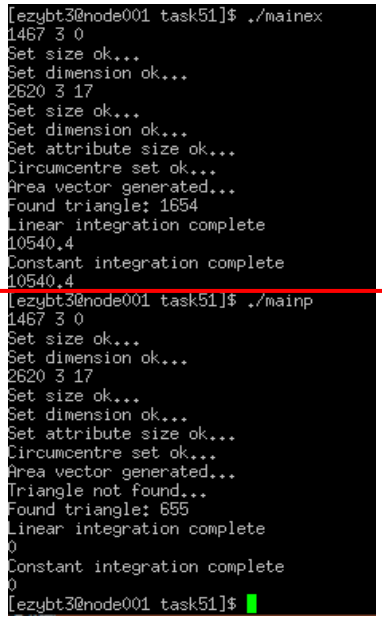
Header files ./

Triangle.h,vertex.h,triangulation.h,f.h

2 PART 2

When the structure of TASK 4 is considered, there are very limited opportunities to parallelize the execution. An attempt was made to parallelize file input since they will be stored in two different objects. The original triangulation constructor was changed to adapt the parallelization and these adaptations are file specific therefore this was only done to attempt to increase speed and can't be regarded as good engineering practice. When the program runs, the execution time is actually slower than the serial code. Serial code finished execution for triangulation#1.tri file in 0.025seconds where parallelized code took 0.054 seconds. This is due to the assignment allocations for the cores. Also when the code was parallelized data structure was almost unpredictable therefore the functions returned unexpected values. This is due to the structure of the std::vector where data is pushed in as it comes.

The constructor of the triangulation class was changed to adapt the program to parallelization. The attempt was successful and data held in the file put in to vectors in parallel.



Without -fopenmp compile

"triangulation#1.tri" used

With -fopenmp compile

"triangulation#1.tri" used

CPP file found in ./main.cpp

Header files ./

Triangle.h,vertex.h,triangulation.h,f.h

Changes can be seen in triangulation.h file.

3 PART 3

All files that were designed in the coursework were to be used in this part. Necessary operations and functions are embedded in to the previous tasks so they could be used when the implementation of part 2 is done.

Initial implementation of Delaunay triangulation was done using the material that was developed in previous tasks. If more time budget and more background material were given this task could have been finished.

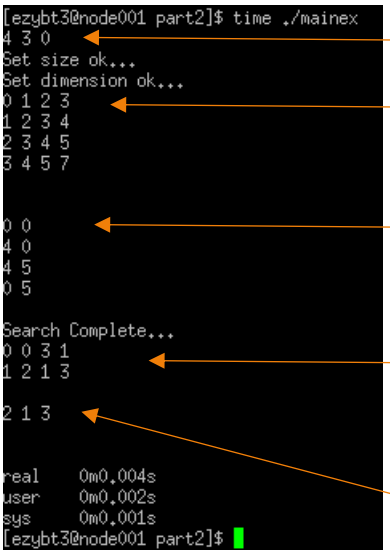
Current implementation Pseudo code:

```
File input -> new triangulation
get verticies of triangulation
for each vertices in triangulation
        find boundaries of the vertices
create vertices using found boundaries
create super triangle in triangulation using the boundaries as vertices
new triangle= badtriangles
for each point in vertices
        if point is in triangle
                add triangle to badtriangles
        erase badtriangle in triangulation
// TODO CODE
For each triangles in badtriangles
        Find edges
For each edge
        Add new triangle to triangulation
```

Code can be found at: /part3/main.cpp

Header files ./part3*

Triangle.h,vertex.h,triangulation.h,f.h,Interval.h



Vertices – vertex number, dimension, attributes

Vertexid, x, y, z

New generated Vertices - x, y (2D)

Current Badtriangles = supertriangle enclosing the space that set to be removed and replaced with another

Triangles in triangulation after the first bad triangle being removed.

“vertices#1.node used”