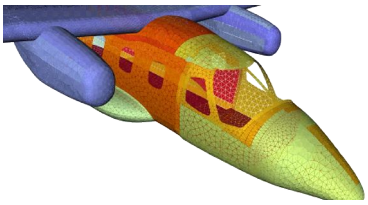
Scalable Cross-Platform Software Design: *Data Structures, Algorithms and Design Coursework#2*

25% of the module mark: Marking Scheme on Moodle

In this task the assessment allows you to demonstrate an ability to understand, analyse and design and then implement an engineering software task, comprising data structures and algorithms to the required professional standard. A key skill is to demonstrate is the ability to understand and analyse the task from a typical engineering client's style of problem description. Specifically, building upon an object oriented decomposition (a pre-requisite skill of the module), to enhance it with generic programming approaches and use of the STL. You are also provided with an opportunity to show skills in succinctly and clearly reporting this type of task.

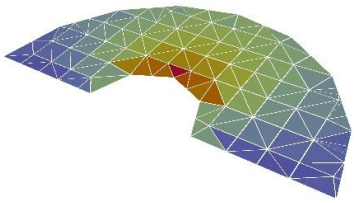
Summary: A client approaches us to design and implement a code to help them in their business. As is often the case, they will be describing technical material that we are unfamiliar with, however we do have an in-house expert (PS) who can advise us where necessary. Assuming we have a good familiarity with coding techniques, they offer us a project with ***a strict budget of 25 hours of person-hours.***

Scenario: (Your client's field of business)



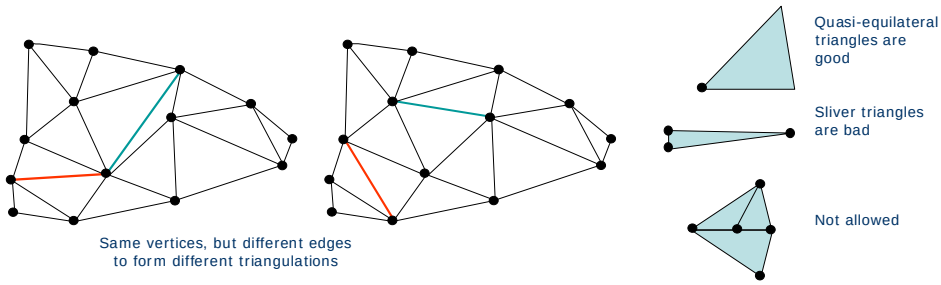
Many engineering disciplines use CAD/CAE packages which require models of geometrical structures.

Often, structures are modelled by a triangulated approximation to their surfaces. Triangles offer the ability to represent small and fine features with different sized triangles.



Similarly, we might discretise a flat 2D problem space in order to numerically solve some equations representing a physical process of interest - maybe current flow or temperature. We might visualise this as shown here using a colour map or using a "height" surface. In this work we are concerned with the second type of ***triangulation*** (also referred to as a *mesh).*

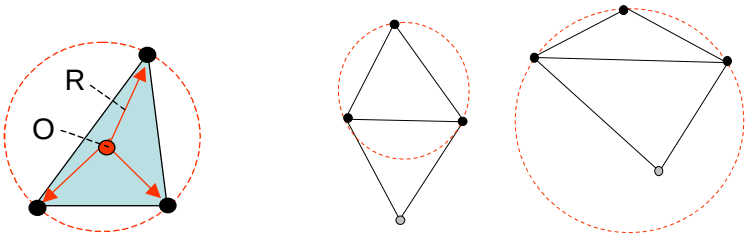
We can "triangulate" a particular problem with given fixed vertices in many different ways. However, we will inevitably ask, or be asked: What is the "best quality" triangulation? Note that a triangulation of given vertices is not unique: "any edge can be flipped".



The definition of a "good" triangulation obviously depends upon its intended use. For example, people solving a set of engineering equations are likely to prioritise different criteria than the computer games/animation community.

It can be shown (but not here) that there is a particular (unique-ish) choice of triangulation that minimises the interpolation error, the so-called Delaunay Triangulation. This is what we will consider.

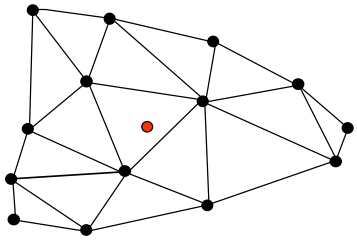
A Delaunay Triangulation, DT, is defined to be the triangulation where the circumcircle of each triangle is empty, that is it contains no vertices except its own. The circumcircle of a triangle is that circle that passes through its vertices: there are known formulae to get the circumcentre, O, and circumradius, R, from the vertex coordinates. See appendix #2.



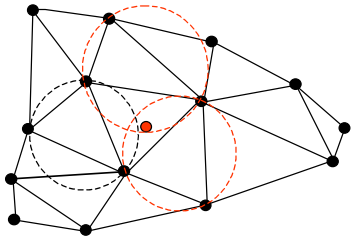
Left Delaunay, right non-Delaunay

Generating Delaunay Triangulations: There are a number of algorithms for connecting a given set of vertices to produce a DT. For example, the *incremental Bowyer-Watson* algorithm.

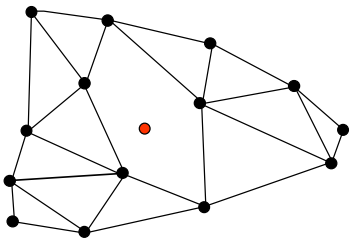
Consider an existing Delaunay triangulation and we want to insert a new vertex and re-triangulate: it turns out that this is a local operation.



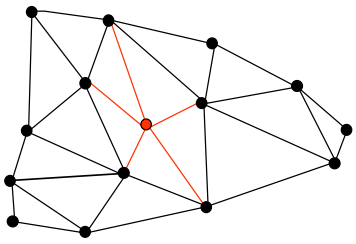
Initial mesh: we want to add the red vertex



Identify those triangles whose circumcircles enclose the new vertex: just the red ones here



Delete them



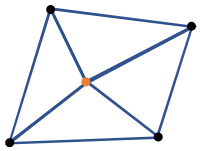
Each exposed edge seeds a new triangle connected to the new vertex

The algorithm inserts the vertices one at a time. We usually start with one "large" triangle big enough to enclose the whole problem and when all our points have been inserted, delete the large triangle's vertices leaving just what we want.

A naive implementation of the Bowyer Watson Algorithm usually fails. This is not the fault of the algorithm, but rather its implementation on a computer with finite numerical precision: This is not in the scope of our work here.

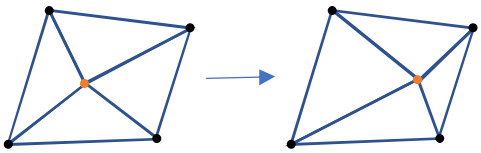
Fortunately, we are not being asked to implement the full Bowyer Watson algorithm, rather prepare a C++ object oriented data structure with algorithms that later might form the basis for generating Delaunay triangulations – we have a *context* and an understanding of possible future *upgrade* requirements.

Hints on the basic data structure: There are a number of design issues to consider:



What are the basic *objects*? Maybe vertex, triangle and triangulation? – relate to the picture on the right? What are the basic data elements defining each? What functionality should each have? How do these objects relate to each other? More importantly, if we are managing collections of objects, such as triangles, where is the "collection of" entity stored – "who owns it"?

Note to students: Please recall that this is the advanced software course and the fundamental C and intermediate Object-oriented modules are the required pre-requisites of this module and so students are required to follow all the good practice discussed in them. (E.g. constructors, copy constructors, use of private data …) You will lose credit for failing to do so.



One may like to consider whether it is advisable that each triangle somehow stores its own "copy" of each of its vertices. Would this be memory efficient? Would it be robust? Remember, the same (e.g. orange) vertex is used in more than one triangle. Maybe we pre-empt the client saying that that if a user moves a particular vertex, all triangles should follow it. So, should each triangle store its own copy of its vertex or somehow *refer/point to* which vertices it uses that are held somewhere else in the data structure?

Comments on required algorithms for generating a Delaunay Triangulation:

1) In which existing triangle does the new vertex fall: ***fast search algorithm needed?***

In this assessment, a brute force "check all triangles" approach to this requirement is acceptable, but I seek to give extra credit to those who at least discuss better alternatives and/or implement them – watch the time budget through!

2) From the "start" triangle found above, we need to search over its adjacent triangles and then their neighbours etc until sure *all* triangles whose circumcircles enclose the vertex are found.

We are not required to implement this algorithm in our assignment, but we note it as a future need that may impact upon our data structure design

3) We need a fast and accurate implementation of the "in circumcircle" test. See the appendix.

4) Is this parallelisable?

In this assessment, I do not ask for a parallel code, just serial. However, I seek to give extra credit to those who at least discuss, or even partially implement, the scope for parallelisation – which aspects? – watch the time budget through

Specific Assessment Requirements:

Assume that you are provided with a triangulation stored in files as described in Appendix 3: *Note to students: Please be aware that in recent years understanding the file format has caused some students difficulty. Do not leave it until near the deadline to clarify this in your mind – talk to me earlier.*

- Design and implement a data structure and an interface for one or more C++ classes to encapsulate the triangulation. *See hints above.*

- Use of the STL is likely to be appropriate and all code must run on Jenna.

Specifically, your interface must permit the following:

- File streaming input and output of triangulations in the format given in the appendix. It is expected that the professional and conventional use of stream operators covered in the pre-requisite modules is used.

- Queries of the form: Given a point x,y at run time which triangle contains it. *Note to students: Do not waste time writing a human being screen-keyboard interface for this. The task requires that an "object" in your data structure design has a member function that accepts suitable arguments and returns the required result.*

- Queries of the form: given a function (known at compile time) f(x,y), return the integral of f(x,y) over the domain of the triangulation. For this we will approximate the integration in two possible ways - see Appendix #1. *Hint to students: Surely this is seeking to assess your ability to implement a generic programming code similar to the example we consider quite early on in the module?*

- A check whether a triangulated mesh provided by the user is Delaunay. *Note to students – same comment as for the point contained in triangle queries above*

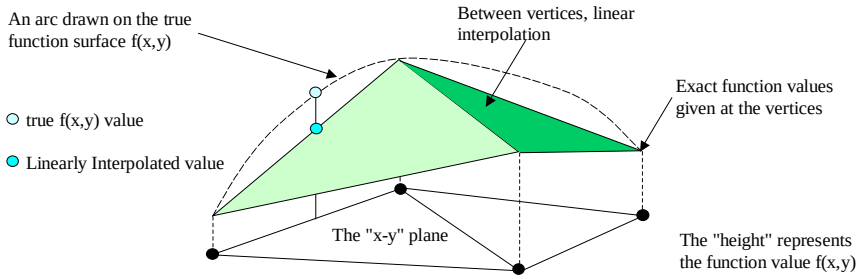
Required submission: Your codes and a brief report to convince the client that it works correctly. As in any professional client paying for work scenario, evidence to support the assertion *it works* is critical. As guidance, excluding listings, I would predict that 5-10 well-spaced pages will be sufficient for the report. A high standard of in-code documentation is essential for the award of a good mark.

Note to students: I am looking to credit your selection of sensible tests to run and the associated discussion of what each test proves and whether there is scope or need to test further. Again, be aware of the time budget; the client can only expect to get what they pay for!

All student Work for assessment is to be uploaded via the Moodle submission link by the deadline given. Submission by email is not accepted

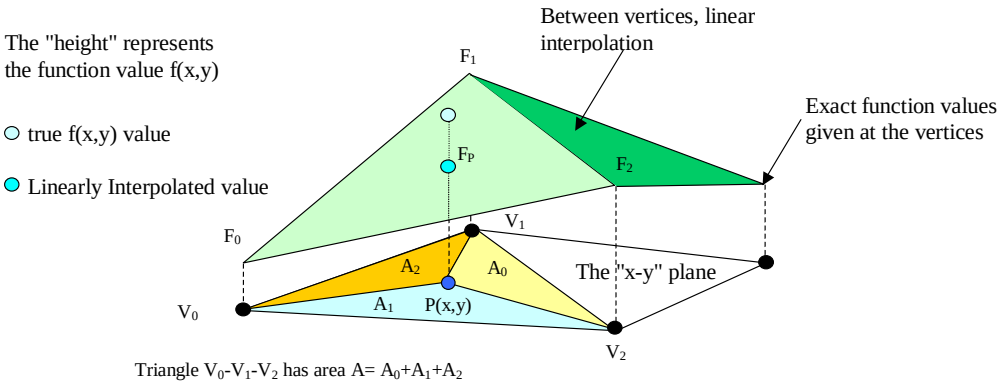
Marks and feedback will be provided 15 working days after the submission deadline; each student will receive an individual copy of the marking rubric posted on Moodle by email. If required, further individual feedback is then available by requesting a one-to-one meeting.

Appendix #1: Linear Interpolation Over Individual Triangles



If we have a triangulation and we know the *true* values of the function f(x,y) at its vertices, how do we interpolate to estimate the function at *any* point P in the x, y plane?

Well we consult a friendly mathematician who provides the following:



If the values of the function f(x,y) are known to be F_0, F_1 and F_2 at the vertices V_0 , V_1 and V_2 then the linearly interpolated value at any point P(x,y) inside the triangle is,

$$F_P(x,y) = \frac{A_0(x,y)F_0 + A_1(x,y)F_1 + A_2(x,y)F_2}{A}$$

where the A_i are the areas of the sub-triangles shown in the figure - they are functions of where P is - hence x,y. A is the total area of the triangle.

For interest the quantities $\left\{\frac{A_0(x,y)}{A}, \frac{A_1(x,y)}{A}, \frac{A_2(x,y)}{A}\right\}$ are called the area, or barycentric coordinates of the point P with respect to the triangle. These have many convenient properties.

How to numerical approximate the integral of the function with respect to x and y?

Again, the same friendly mathematician provides:

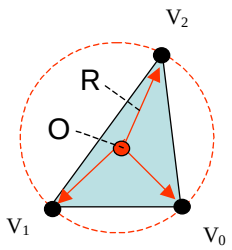
There are two ways we could evaluate an approximation to $\iint_T dx\,dy\ f(x,y)$ where T is a triangle:

1) ***Constant value approximation*** $\iint_T dx\,dy\ f(x,y) \approx A_T\ f(O_{xT}, O_{yT})$ i.e. evaluate the function at the circumcentre, O, of each triangle and weight by the triangle's area.

2) ***Linear interpolation approximation*** using the above and the useful, but non-obvious, fact that:

$$\iint_{\substack{Triangle\\ area}} dx\,dy\ A_i(x,y) = A/3$$ *Not sure about this - consult your client!*

Appendix #2: The Circumcentre of Individual Triangles



It can be shown that if vertex V_i has coordinates X_i, Y_i that the coordinates of the circumcentre O_x , O_y and the circumradius R are found from solutions of,

$$\begin{pmatrix} X_0^2+Y_0^2 \\ X_1^2+Y_1^2 \\ X_2^2+Y_2^2 \end{pmatrix} = \begin{pmatrix} X_0 & Y_0 & 1 \\ X_1 & Y_1 & 1 \\ X_2 & Y_2 & 1 \end{pmatrix} \begin{pmatrix} 2O_x \\ 2O_y \\ R^2-O_x^2-O_y^2 \end{pmatrix}$$

(See if you can prove this - just for fun, no marks.)

This order 3 matrix equation can be solved quite effectively using basic algebraic techniques which you are expected to identify yourselves from the wide range of resources available to an engineer.

Note to students: This is not a maths module! However, the ability to understand algorithms when presented in the language of the client is definitely a skill to value. I am very open to any approaches to explain the mathematics further and will discuss in class. However, reflect on what exactly do we need from these 2 appendices – just to identify the key "formulae" we need to implement.

Appendix #3: File Format

Ascii file format for reading/writing 2D triangulations. This is your client's legacy format - can't be changed. Triangulation files should have the extension ".tri" and test examples are on Moodle.

"triangulation.tri"

no_points dimensions (2 or 3) no_attributes_per_point

i x_i y_i $\{z_i\}$ **attribute**$_1$. . . . . // no_points off

no_cells no_vertices_per_cell no_attributes_per_cell

i v_1. . . . $v_{\text{no\_vertices\_per\_cell}}$ **attribute**$_1$. . . . . // no_cells off

At least maintaining the same format means you can use the client's "visualisation" software, (available on Moodle): its use will be explained in the lectures as well and it is rather helpful to debug your software to be able to *see* what is happening.