Final Project

In the final project of this course you have to develop an algorithm to calculate the Convex Hull of a given 3D shape.

A bonus point is given if it is implemented the feature that allow the user to see every step of the algorithm (see next section).

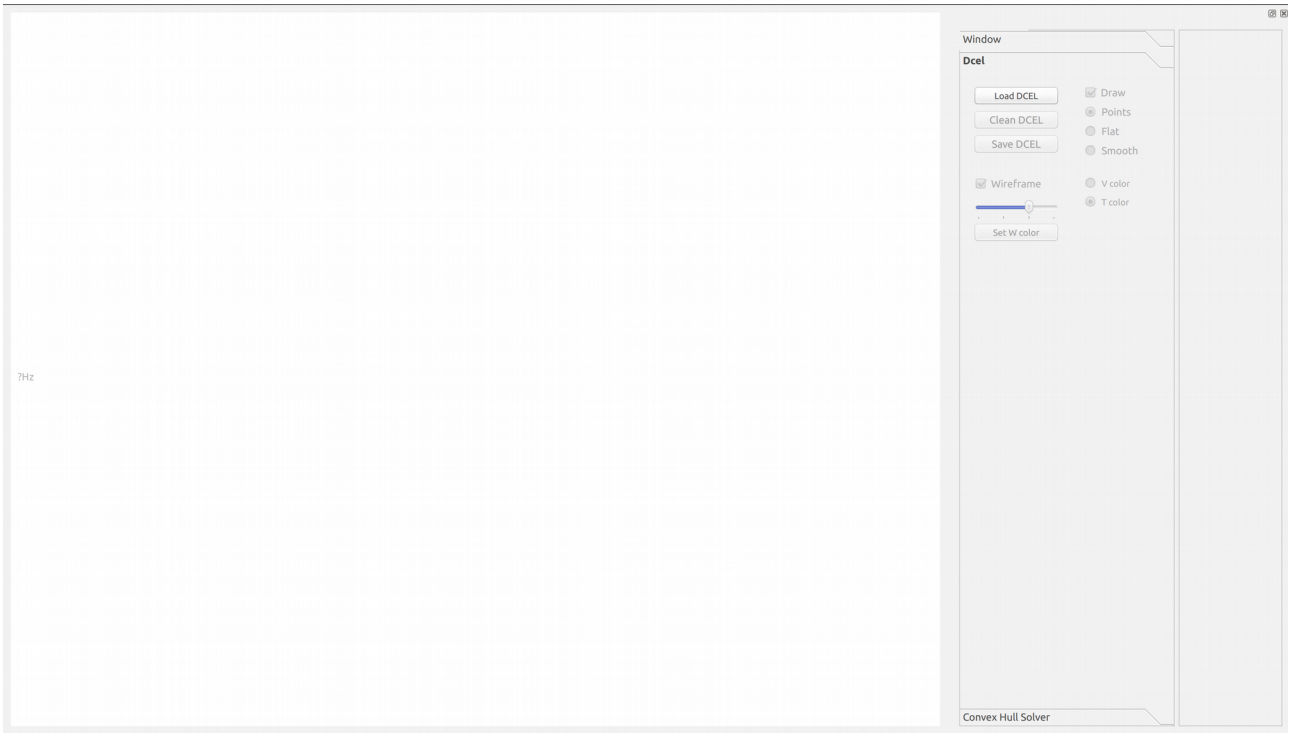
Therefore, the maximum grade will be 31/30.

The algorithm must be implemented in C++, and a base project is given.

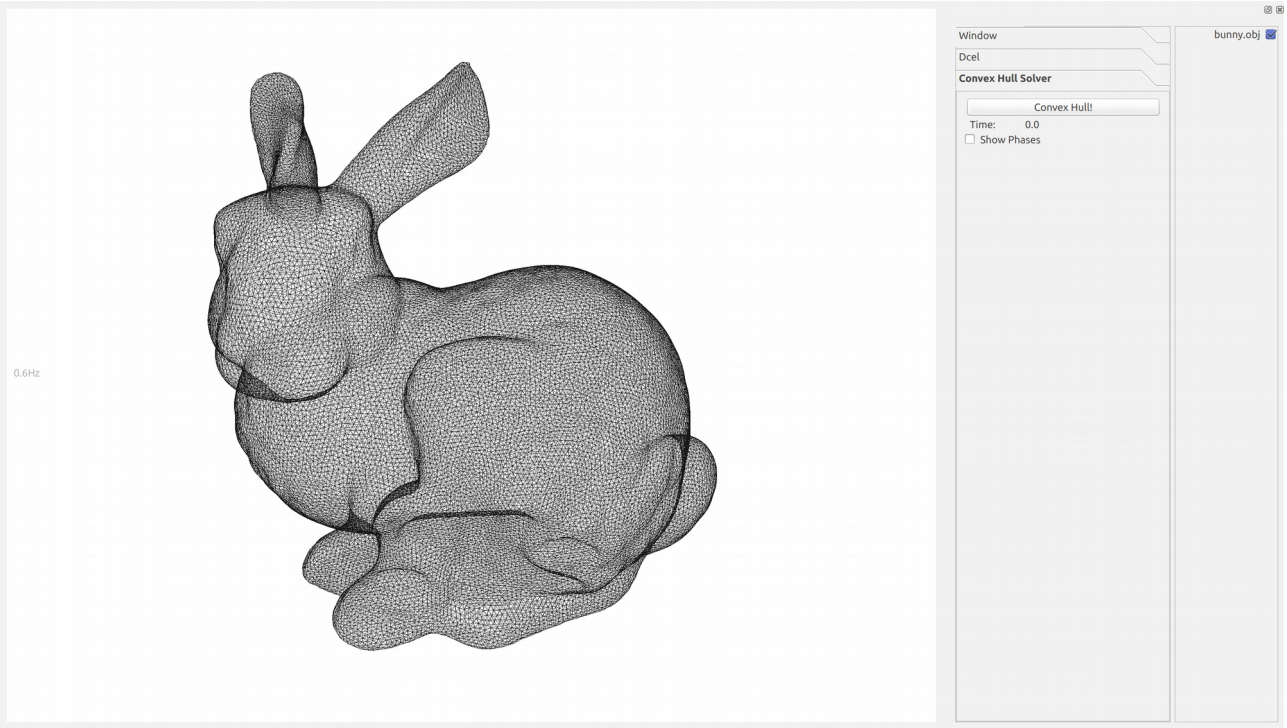
1. Base Project

In this base project there is a viewer that allows to visualize a 3D shape in a canvas, and a DCEL data structure that allows to store a 3D shape.

The MainWindow is composed by a single canvas (left) and some managers (right).



A manager allows you to “manage” something. In the image, the Dcel manager is focused (right). This manager is provided, and it allows you to choose all the rendering options for a Dcel.



All the managers are situated in the "GUI/managers" folder.

The Convex Hull Solver Manager is also given. Its task is to take the Dcel loaded on the Dcel manager and to create another Dcel manager that manages the Convex Hull. Therefore, the "Convex Hull!" button became pushable only when a mesh is loaded on the Dcel Manager.

The portion of the code executed when the button is pushed is in "GUI/managers/convexhullmanager.cpp", on the "on_convexHullPushButton_clicked()" method. In this method there is a portion dedicated where you can put your convex hull algorithm. The algorithm should modify the variable named "dcel", that is a DrawableDcel (a DrawableDcel is a Dcel that is also drawable on the canvas).

You can also modify "GUI/managers/convexhullmanager.h", where you can include your files for the convex hull algorithm.

In the Convex Hull manager there is a check box named "Show Phases", and it is correlated with the **bonus point**. If you want to implement this functionality, when this check box is checked and the ConvexHull button is pushed, on the canvas should be viewed every step of the algorithm (in other words, the canvas should be shown the updated Convex Hull every time you add a point on it).

Please DO NOT modify other portions of code of this base project if not authorized.

If you find a bug on the code, or if you have questions, please write an email at muntoni.alessandro@gmail.com and cordafab@gmail.com .

2. Virtual Machine vs Physical Machine

We are giving to you the base project and a Virtual Machine which is ready to use.

I suggest you to avoid to use the Virtual Machine and, if it is possible, to compile the base project on your physical machine. However, it is almost impossible to compile it on Windows (you can try to link manually all the required libraries: good luck!). Therefore, if you have a PC and if you can, install a linux based (better an ubuntu-based) OS and follow the guidelines on the README.TXT file to install all the required libraries and compile the project. If you have a Mac, please send an email to Fabrizio (cordafab@gmail.com) which will help you with the installation and linking of all the required libraries.

If you can't, use the Virtual Machine.

3. Guidelines for a Good Final Project

1. Please, before submitting, rename the .pro file with a name of this format: <matr>_<surname>_<name>.pro.
2. The base project compiles with zero warnings (on MacOS, all warnings are generated by external libraries because MacOS uses a different compiler than GCC). Please make sure to submit a project with zero warnings (unless libraries warnings on MacOS). Warnings are indicative of bad and error-prone coding. A zero-warnings code doesn't mean that is good code, but good code always produces zero warnings.
3. Please separate the definition of a class (files .h) from its implementation (files .cpp), and please do not use "using namespace ..." on headers files (why? http://stackoverflow.com/questions/5849457/using-namespace-in-c-headers).
4. Please try to avoid the usage of global variables and other shortcuts. Try to follow the Object Oriented paradigm as best as you can (why? http://c2.com/cgi/wiki?GlobalVariablesAreBad).
5. Please organize your code following the Object Oriented Paradigm. Keep separated algorithms from data structures, write short (when is possible) methods which solve standalone problems. Take a look at the code written on the base project!
6. Please use const keyword. A const method in a class is a method which guarantees that the state of the object is preserved. Therefore, a const method cannot modify attributes and it can only execute const methods on them. If you don't implement const methods on your class, you are forcing other programmers, that need your class, to do not create const methods! Use also const keyword to passing input parameters by reference: it guarantees that a parameter is only an input and it will not be modified by the function (and passing parameters by reference is definitely more efficient than passing them by value!). If you don't know how to use const keyword, take a look at this tutorial: (http://www.cprogramming.com/tutorial/const_correctness.html).
7. Use nullptr instead of NULL: we are coding in C++11, not ANSI C89!

4. TIPS

4.1. CHECK THE VISIBILITY OF A TRIANGLE FROM A POINT

At some different points of the convex hull algorithm, it is required to check if a point "sees" a triangle. As for 2D, in a 3D space there is a 4x4 matrix, called *M*, that, given 3 points *p1*, *p2* and *p3* and another point *p*, allows you to find out if *p* is in the left or right semi-space delineated by *p1*, *p2* and *p3* by simply checking the sign of the determinant of M (I'm not showing you this matrix, you have to find it by yourself!).

However, numeric errors are always around the corner. Therefore:

1. Always use double data type for real numbers (no floats);
2. If you have to check if the determinant is positive, then do not write
   ```
   det > 0
   ```
   but instead write
   ```
   det > std::numeric_limits<double>::epsilon()
   ```
3. If you have to check if the determinant is negative, then do not write
   ```
   det < 0
   ```
   but instead write
   ```
   det < -std::numeric_limits<double>::epsilon()
   ```

This tip should save you a lot of debugging time!

4.2. USE STL CLASSES

Stl classes are more efficient than QT classes (example: std::vector vs QVector). When is possible, use stl classes and, if you don't know how to do something, remember that Google is your friend and a lot of other programmers solved the same problems that you are trying to solve.

Furthermore, check std::list, std::set and std::map data structures. Sometimes, vectors are not the optimal data structure. For example, if the most common operation you are going to do on a collection of objects is to check if an object exists or not:

- std::vector: *O(n)*
- std::set: *O(log n)*

However, to walk on all the elements of lists, sets and maps you need to use iterators, which are very important in OOP and C++. If you never used iterators, this is a good time to start:

http://www.cprogramming.com/tutorial/stl/iterators.html

4.3. DCEL DATA STRUCTURE

The Dcel that you have to use for the convex hull uses pointers for everything. It is a very complex data structure, and it is young. It is designed in this way to guarantee efficiency on the operations of add, modify and remove vertices, edges and faces (which are very common operations on a convex hull algorithm).

However, pointers are difficult to manage, and if you are doing something wrong (like putting the wrong next half edge to an half edge) then 99% of times it crashes. To find the error in these cases is very difficult (it is difficult regardless of which data structure you use), therefore, I invite you to read the next section!

Also in the Dcel there are a lot of useful iterators! Check how to use them!

You can find the documentation of the Dcel at this link: http://muntonialessandro.bitbucket.org/dcel/. There also is a English documentation, but it is actually incomplete (it's a work in progress): http://muntonialessandro.bitbucket.org/dcel/en/.

If you have questions about the Dcel, please feel free to email me or to write a post on the moodle forum.

4.4. *Debugging*

Debugger is your friend and it is a very powerful instrument. 90% of the situations putting a break point is faster (and a smarter choice) than putting std::cout (or std::cerr) everywhere.

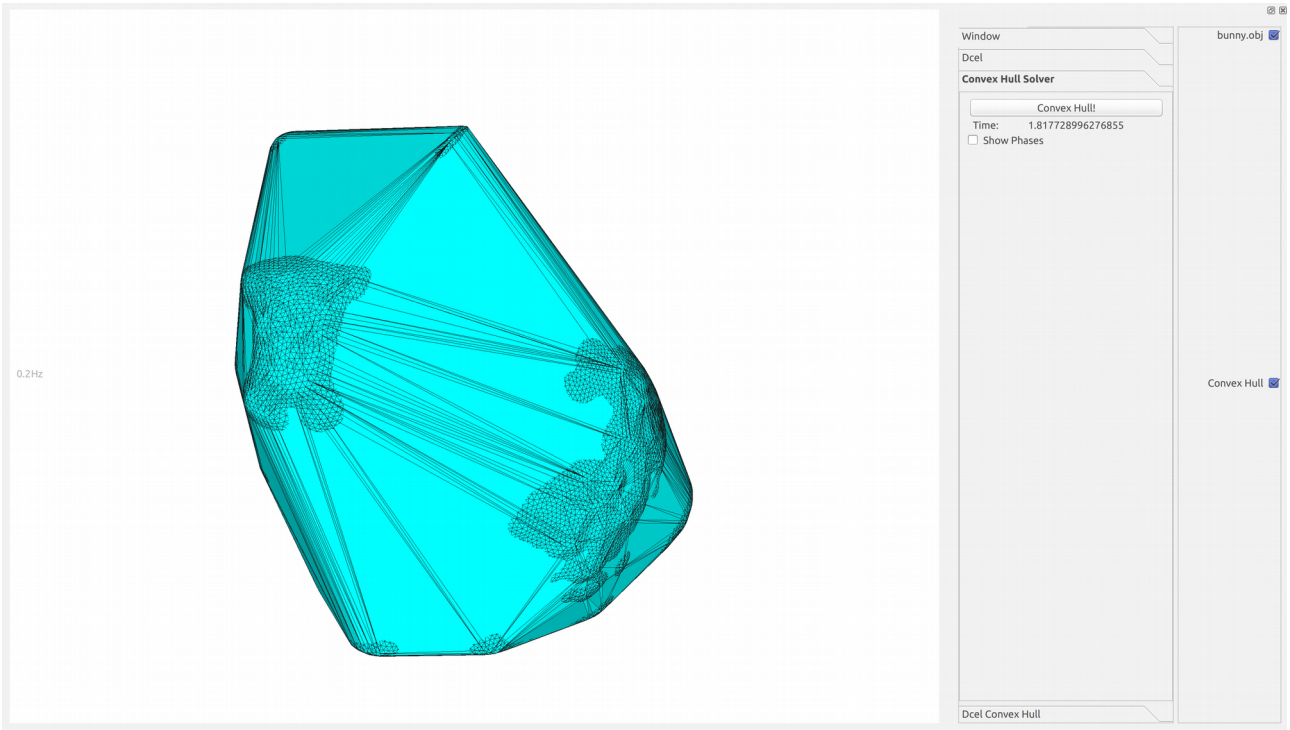
The debugger allows you to find the exact point where your application is crashing and why, and to see the state of all the variables in every scope during the execution of your application. If you never used a debugger, this is the best time to start and learn.

However, sometimes the debugger is not enough. Sometimes you will need to see what is happening on the mesh. A few tips:

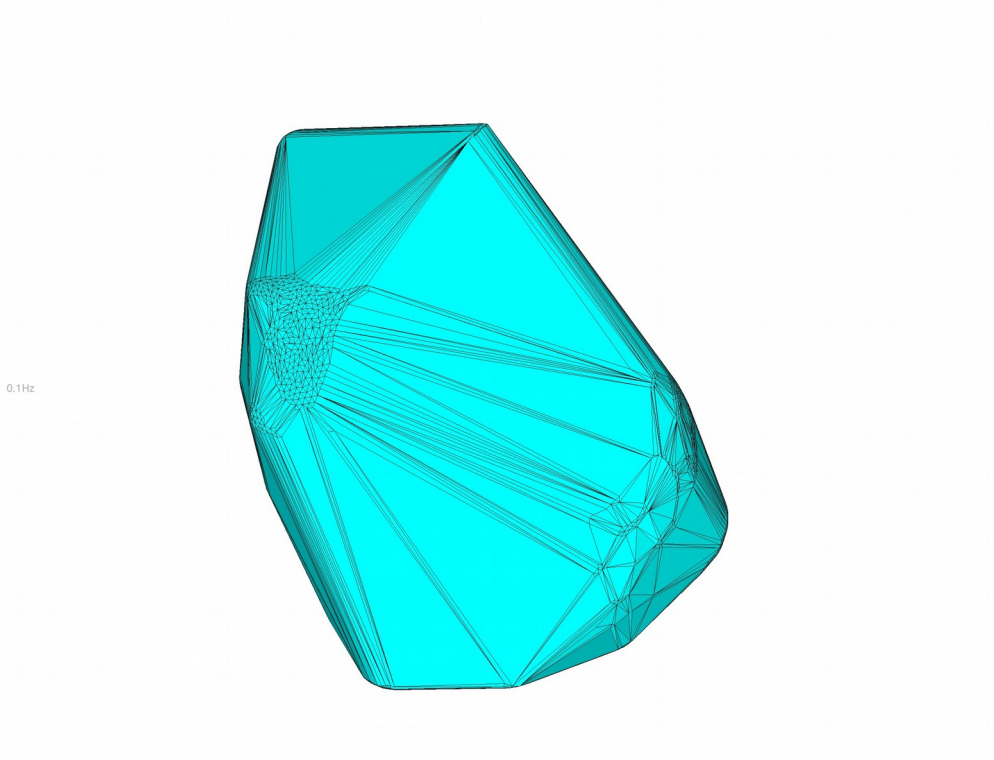
- Use colors: every face has a color field, and you can use it when you want. Useful when you need to find which face is creating problems!
- DrawableDcel allows you to add debugging points and cylinders that don't modify the Dcel fields. Check how to do that on the DrawableDcel class!
- The viewer of the base project is not the best instrument if you need to compare different meshes (for example, the meshes produced by your algorithm at every step). Then you can save a dcel in a ply file (and maybe save multiple ply files) and open it in a more powerful viewer like MeshLab: sometimes could be very helpful! Remember that you can save a mesh on a ply file only if references of every component of the dcel are correct.

4.5. *Results*

When you will have a Convex Hull, don't worry if you see some triangles of the input mesh on the CH. It is a problem of OpenGL and the rendering, that is done in an old way I didn't have time to fix it.



This is the convex hull of the bunny overlapped with the input bunny model. Without overlapping, the convex hull becomes this:



5. Grade and Deadlines

The grade will be composed by:

- 7/30: Correctness of the project;
- 5/30: Documentation;
- 10/30: Structure of the project, code modularity and style;
- 8/30: Efficiency;
- 1/30: Bonus point.

The first deadline is for all the students that need to have the grade registered before August 20th for the ERSU scholarship. If you want your grade before August 20th, you must submit your project by August 4th at 23:59.

The deadline for the project is August 31st. If you submit your project after August 31st at 23:59, your grade will be penalized by a point every month. This is a summary table:

Submit by	Maximum Grade
23:59 August 4th	31/30 by August 20th
23:59 August 31st	31/30
23:59 September 30th	30/30
23:59 October 31st	29/30
23:59 November 30th	28/30
23:59 December 31st	27/30
23:59 January 31st	26/30
23:59 February 28th	25/30

After March 1st, you will have to submit the project with the specifications of 2017.

Remember that this is an *individual* project: you can collaborate for the resolution of high level problems, but projects with similar pieces of code will be not tolerated (*both* projects will be rejected).