Delynoi

An object-oriented C++ library for the generation of polygonal meshes

Delynoi Primer

Version 1.0

Rev. 0
August, 2017

Copyright and License

Delynoi 1.0, Copyright © 2017
by Catalina Álvarez, Nancy Hitschfeld-Kahler, Alejandro Ortiz-Bernardin

Department of Computer Science
Department of Mechanical Engineering
Facultad de Ciencias Físicas y Matemáticas
Universidad de Chile
Av. Beauchef 851, Santiago, Chile

TABLE OF CONTENTS

1 Features of Delynoi

- The meshes are generated on arbitrary domains, created from user points. Domains have no restrictions on convexity.
- It allows the inclusion of completely contained or intersecting holes, which are processed if required.
- The meshes are generated from seed points, which can be either read directly from a text file, included one by one, or created from a number of generation rules included in the library. New generation rules can be included easily.
- Meshes can be stored in OFF-style text files, or used directly on another program.
- To generate the meshes, the library first obtains the conforming Delaunay triangulation using Triangle; the triangulation is considered a mesh that is left available for use in case it is desired. Then, it computes the constrained Voronoi diagram.

2 Source code

The source code is available to be downloaded from Delynoi's repository:

https://github.com/capalvarez/Delynoi

Download the code before proceeding with the rest of this tutorial.

3 Up and running with Delynoi

Delynoi has been tested on Unix-like machines only. Delynoi is dependant on CMake, so make sure it is available in your machine. If it is not, install it before proceeding with the rest of this tutorial. To install CMake on Debian-based operating systems (such as Ubuntu), on a terminal type and execute:

```
sudo apt-get install cmake
```

Unpack the code to a folder of your choice. Fig. 1 shows the content of Delynoi that was unpacked to "/home/Software/"

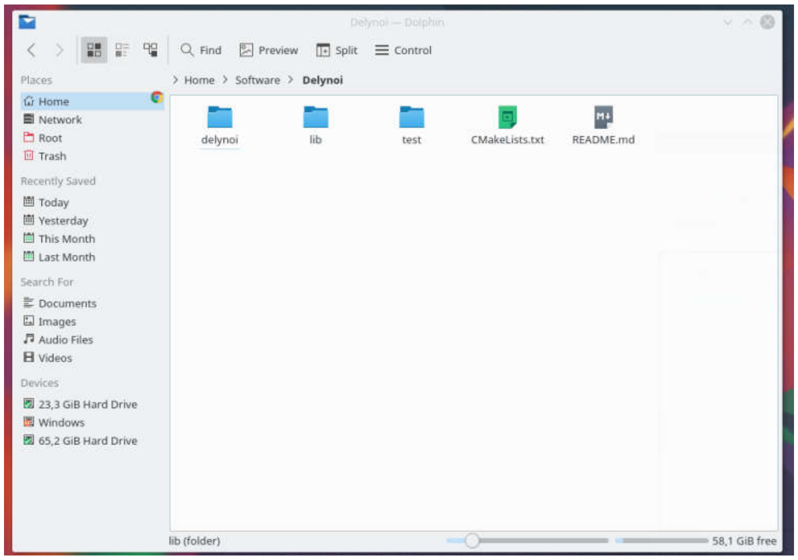


Fig. 1: Delynoi source code.

Inside the “test” folder of Delynoi’s root directory (see Fig. 2) it can be found a simple example showing Delynoi’s usage. In this folder there is also a CMakeLists.txt file to help running the examples. In this part of the tutorial we assume no previous knowledge on CMake, and explain all steps for compiling and running a custom Delynoi program.

We include the file “LShapeTestMain.cpp”, which generates a polygonal mesh from random points in an L-shaped domain.

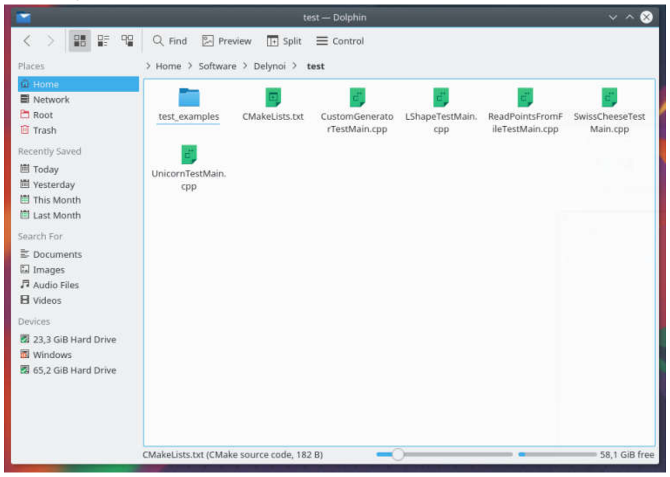
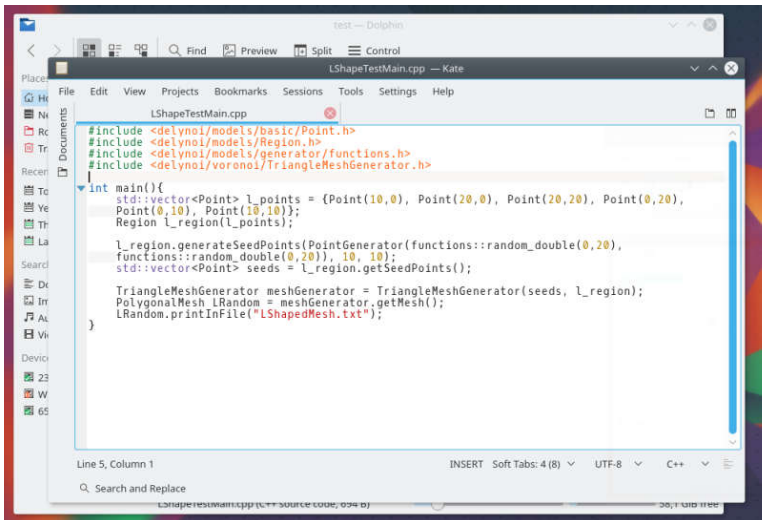


Fig. 2: Delynoi’s test folder. The main C++ setup file implementing a problem of interest must be placed in this folder. Several main setup C++ files are shown. In this part of the tutorial, the C++ file “LShapeTestMain.cpp” will be used.

Open “LShapeTestMain.cpp” file. If you are interested, browse the code in this file to realize how a mesh is generated using Delynoi. In case you require the mesh to be written to a text file, check the instructions that are provided as comments in “LShapeTestMain.cpp” (see Fig. 3). Modify accordingly, save and close the setup file.



```cpp
#include <delynoi/models/basic/Point.h>
#include <delynoi/models/Region.h>
#include <delynoi/models/generator/functions.h>
#include <delynoi/voronoi/TriangleMeshGenerator.h>

int main(){
    std::vector<Point> l_points = {Point(10,0), Point(20,0), Point(20,20), Point(0,20),
    Point(0,10), Point(10,10)};
    Region l_region(l_points);

    l_region.generateSeedPoints(PointGenerator(functions::random_double(0,20),
    functions::random_double(0,20)), 10, 10);
    std::vector<Point> seeds = l_region.getSeedPoints();

    TriangleMeshGenerator meshGenerator = TriangleMeshGenerator(seeds, l_region);
    PolygonalMesh LRandom = meshGenerator.getMesh();
    LRandom.printInFile("LShapedMesh.txt");
}
```

Fig. 3: Main C++ setup file for generation of a polygonal mesh from random points in a L-shaped domain.

As mentioned before, the test folder contains a file named “CMakeLists.txt”. The file inside “test” folder is shown in Fig. 4.

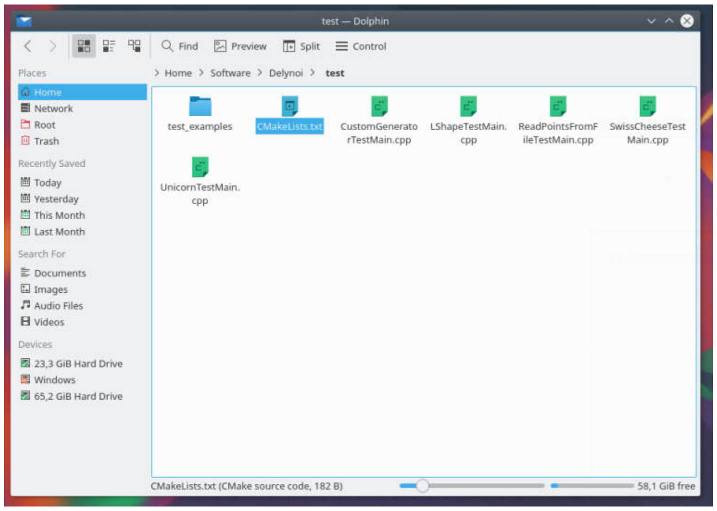
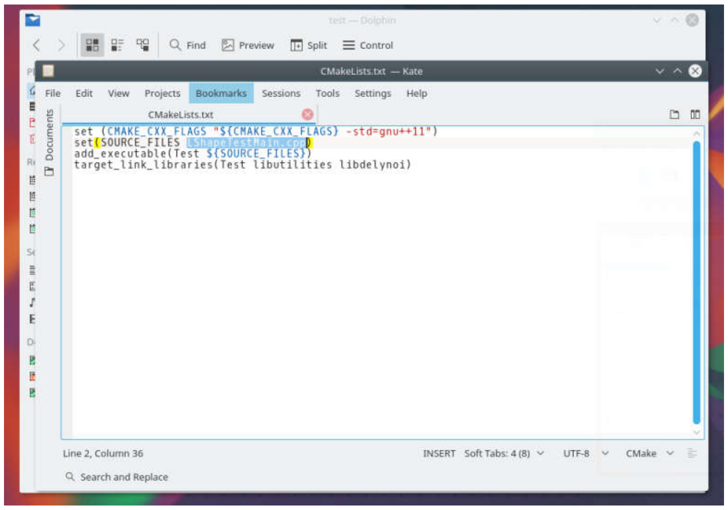


Fig. 4: CMakeLists.txt is located in test folder and controls which main C++ setup file is processed in Delynoi.

Open "CMakeLists.txt" and on the highlighted zone, write the name of the main C++ setup problem file, in this case, "LShapeTestMain.cpp," as shown in Fig. 5. Save and close the file.



```
set (CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=gnu++11")
set(SOURCE_FILES LShapeTestMain.cpp)
add_executable(Test ${SOURCE_FILES})
target_link_libraries(Test libutilities libdelynoi)
```

Fig. 5: Open "CMakeLists.txt" and on the highlighted zone, write the name of the main C++ setup problem file.

Go back to the Delynoi library root folder and there create a folder "build" (Fig. 6).

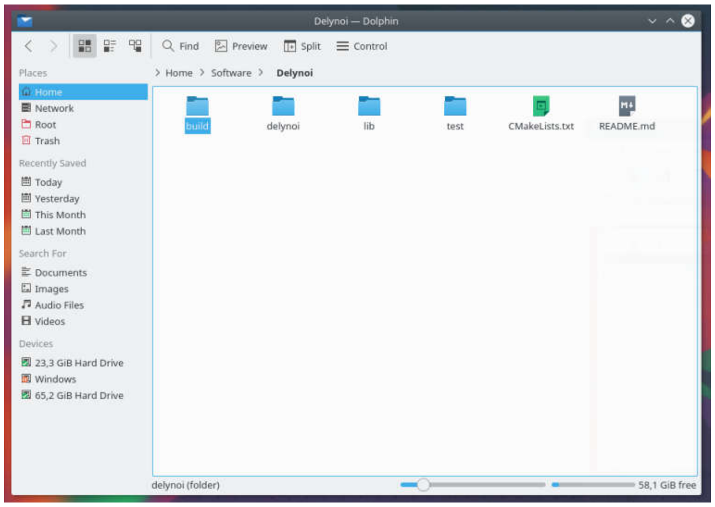


Fig. 6: In Delynoi's root folder create the folder "build".

Go inside the "build" folder and on a terminal, type and execute:

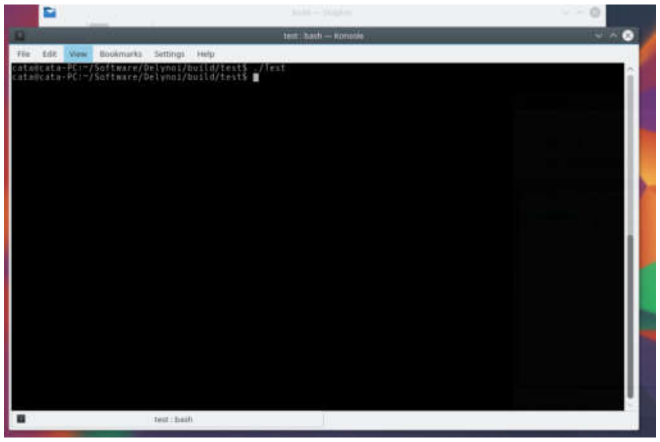
```
cmake ..
```

to create the makefiles. Then, to compile the program, type and execute:

```
make
```

Several files are created. Also, another folder called "test" is created inside "build". The executable of the test problem is stored in this "test" folder and is called "Test". Go inside "build/test/" folder (Fig. 7) and, on a terminal, type and execute:

```
./Test
```



```
cata@cata-PC:~/Software/Delynoi/build/test$ ./Test
cata@cata-PC:~/Software/Delynoi/build/test$ 
```

Fig. 7: Go inside "build/test/" folder and on a terminal type and execute ./Test

4 Creating a mesh

The most common expected use case for Delynoi is generating a mesh from scratch, first defining the domain, and then procedurally creating the seed points from the list of predefined generation rules. Such case is presented in the "UnicornTest-Main.cpp" and "LShapeTestMain.cpp" text files found in the "test" folder. Open the files to see the details of each implementation.

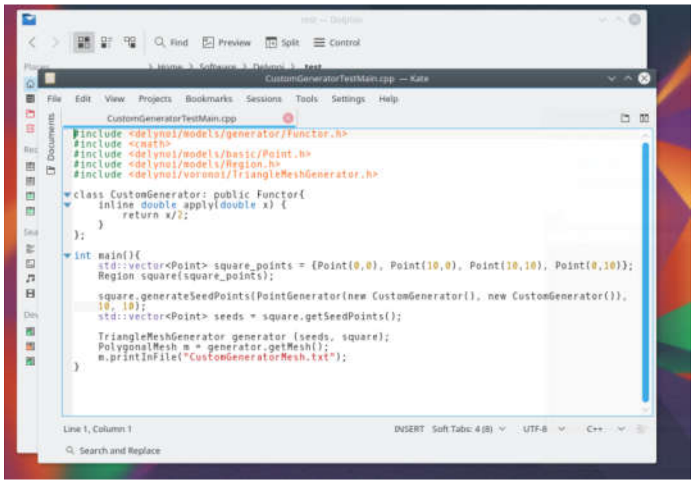
To run each example, the instructions are exactly the same as seen in section 3. Modify the “CMakeLists.txt” file inside the “test” folder, changing “example.cpp” for the name of the example.

5 Including a custom generation function

The predefined generating functions can be found inside the Delynoi library, in a file named “functions.cpp”, located in “delynoi/src/models/generation”.

To include a custom generator function, a new class inheriting from Functor must be created. Any Functor class must include a method named “apply”, which receives a double and returns a double, doing the required processing to the received value.

An example of a custom generating function can be seen in Fig. 8.

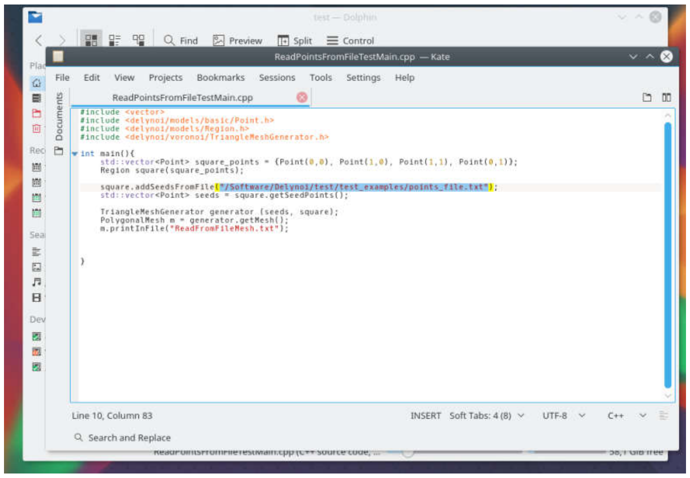


```cpp
#include <delynoi/models/generator/Functor.h>
#include <cmath>
#include <delynoi/models/basic/Point.h>
#include <delynoi/models/Region.h>
#include <delynoi/voronoi/TriangleMeshGenerator.h>

class CustomGenerator: public Functor{
    inline double apply(double x) {
        return x/2;
    }
};

int main(){
    std::vector<Point> square_points = {Point(0,0), Point(10,0), Point(10,10), Point(0,10)};
    Region square(square_points);

    square.generateSeedPoints(PointGenerator(new CustomGenerator(), new CustomGenerator()),
    10, 10);
    std::vector<Point> seeds = square.getSeedPoints();

    TriangleMeshGenerator generator (seeds, square);
    PolygonalMesh m = generator.getMesh();
    m.printInFile("CustomGeneratorMesh.txt");
}
```

Fig. 8: Creation of a custom generation function and its use for the mesh generation.

6 Creating a mesh from a predefined list of points

Most of the included examples compute the meshes inside the given domain from seed points procedurally generated. To use a list of predefined points, one can either create the points one by one, or read them from a text file (Fig. 9). We consider the second case as it is more general.



```cpp
#include <vector>
#include <delynoi/models/basic/Point.h>
#include <delynoi/models/Region.h>
#include <delynoi/voronoi/TriangleMeshGenerator.h>

int main(){
    std::vector<Point> square_points = {Point(0,0), Point(1,0), Point(1,1), Point(0,1)};
    Region square(square_points);

    square.addSeedsFromFile("/Software/Delynoi/test/test_examples/points_file.txt");
    std::vector<Point> seeds = square.getSeedPoints();

    TriangleMeshGenerator generator (seeds, square);
    PolygonalMesh m = generator.getMesh();
    m.printInFile("ReadFromFileMesh.txt");


}
```

Fig. 9: Reading the seed points from a text file.

The highlighted zone in Fig. 9 specifies the name of the text file containing the points. In this case, the text file is left as an example in the “delynoi/test/test_examples” folder. When using a custom file, one must be careful setting the complete path to the file, starting from “/home”.

7 Including holes in the domains

Delynoi can generate meshes in arbitrary shaped domains, including holes either inside the domain, or intersecting its boundary. In the first case, the domain is untouched, keeping the holes inside. In the second case, the library processes the holes and the domain, clipping as necessary. Both cases, however, are exactly equal from the users’ perspective.

The file named “SwissCheeseTestMain.cpp” shows how to include several holes in a domain.

--- THE END ---