Manual of the Space Colonization Algorithm (SCA) software for the study of branching patterns

A Ruiz Sobrino1#, C Martin Blanco1#, T Navarro 1#, I Almudí1, D Buchwalter3, D Funck4, JL Gattolliat5, MC Lemos2, F Jiménez2 & F Casares1**

March 22 2019

Contents

List of Figures

1. General description of algorithm

Our implementation of the SCA algorithm starts with a seed or an initial root, made of points from a branch (is it a line or segment?), with the capability to grow and branch in any direction, that can colonize an area filled with a user-determined number of attraction points (M) that determine the directions of growth and branching. The attraction points move following a random walk with velocity "v" (they can also be still), inducing a growth in length "CREC" in the branches located within the area of influence of M, defined as a radius "MAX". When a branch is at a distance equal or shorter than KILL (radius of "killing") from an attractor point, the point is erased, so that it no longer influences the branching process. A branching pattern is generated, with its origin at the seed, that stops when all M points have been erased. The SCA algorithm we have implemented aims at being a computational model for the study of 2D biological branching patterns, and specifically the gills of the Ephemeropteran *Cloeon dipterum*. Therefore, the 2D space containing M is user defined and can be an ellipse or any other shape defined as a polygon.

We have developed three programs in C++: "puntos.cpp", "SCA.cpp" and "images.cpp".

1. puntos.cpp: (Spanish for "points") it generates the coordinates for the attraction points (Ms) that will be used in the initialization of SCA.cpp, and the shape of the space within which the M points are distributed (ellipse or polygonal).
2. SCA.cpp: it generates "synthetic gills" using the Spatial Colonization Algorithm.
3. images.cpp: it processes the images (in .pnm format) from the graphical simulations obtained from the SCA.cpp as well as from the biological processed images. It extracts and exports the numeric data homogeneously.

See Figure 1 to see the workflow.

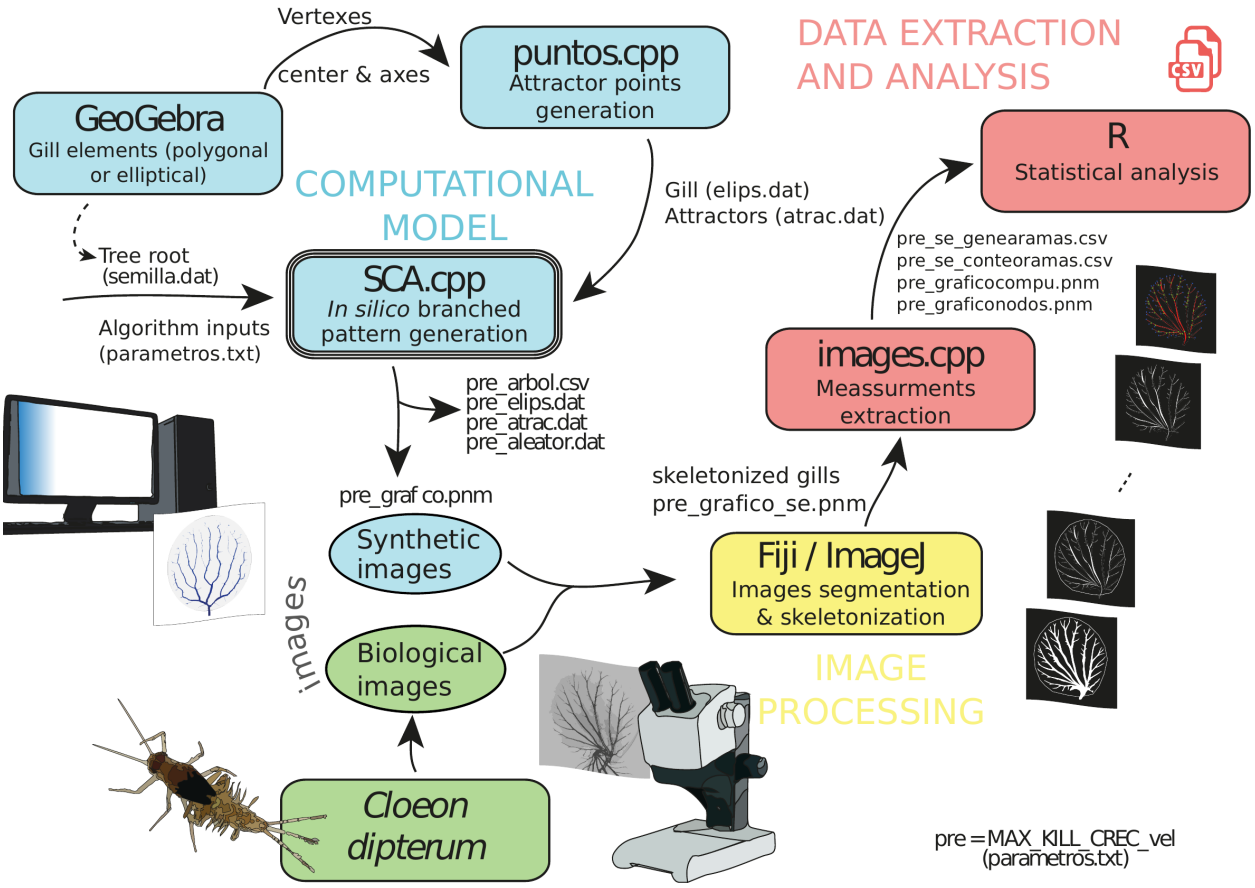


Figure 1: Workflow

2. C++ implementation

To carry out this project, three programs in C++ have been written. Their purpose and characteristics are explained in detail in their corresponded chapters in this manual.

We will describe one of the many possible options to compile and execute those programs in Windows OS using the integrated development environment (IDE) Eclipse. The election of this IDE as well as the operative system is not based on any objective efficiency criteria. Any other combination of OS and software that allows to work in C++ is equally valid.

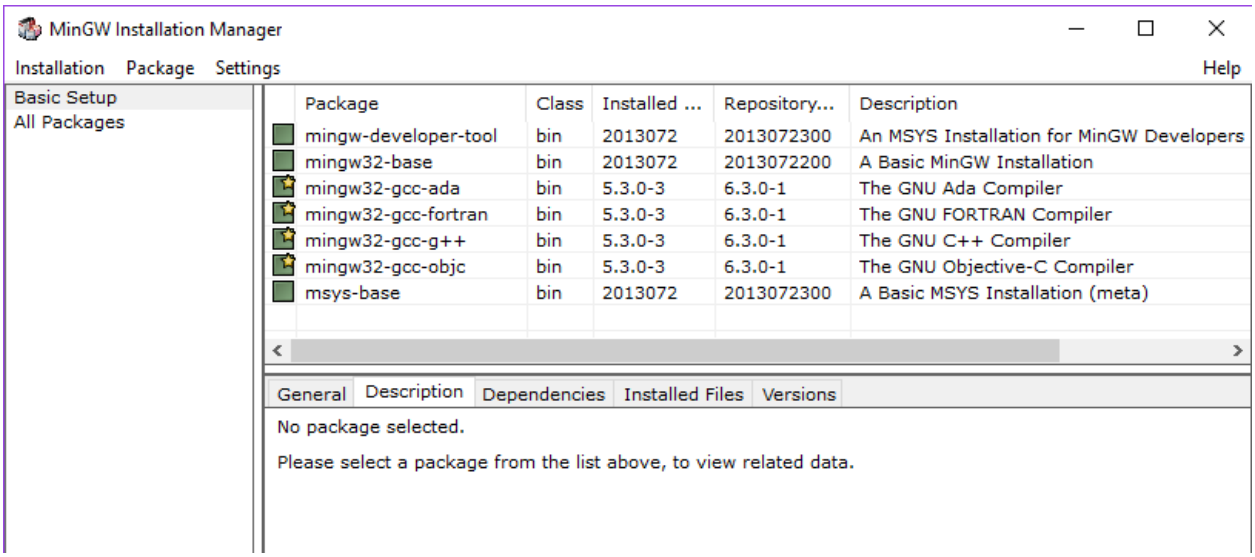


Figure 2: MinGW Installation

2.1 MinGW and Eclipse installation.

MinGW

In addition to the IDE, Eclipse, installation of a C/C++ compiler for Windows is needed. Cygwin or MinGW can be used, being both compatible with each other.

> Once MinGW installer is downloaded from its source page, execute it by selecting the option Basic Setup from the Installation menu, see Figure 2.
>
> For CygWin, once the installer has been executed after selecting "Install" from Internet and Direct connection (if a proxy is not available) at Select Packages, it is needed to select at least the following packages to compile a project in C++: binutils, gcc (gcc, gcc-core, gcc-g++, gcc-mingw-core, gcc-mingw-g++), g++, gdb and make. They can be searched for in "Search" and then install all the found references.

Now the PATH system variable must be added to MinGW directory bin. In order to do that, depending on the Windows version, the procedure should be similar to the following: right click on "My Computer > Properties > Advanced system settings > Environment Variables" find the PATH environment variable and select it. Click "Edit" then "New" and add "C:\MinGW\bin".

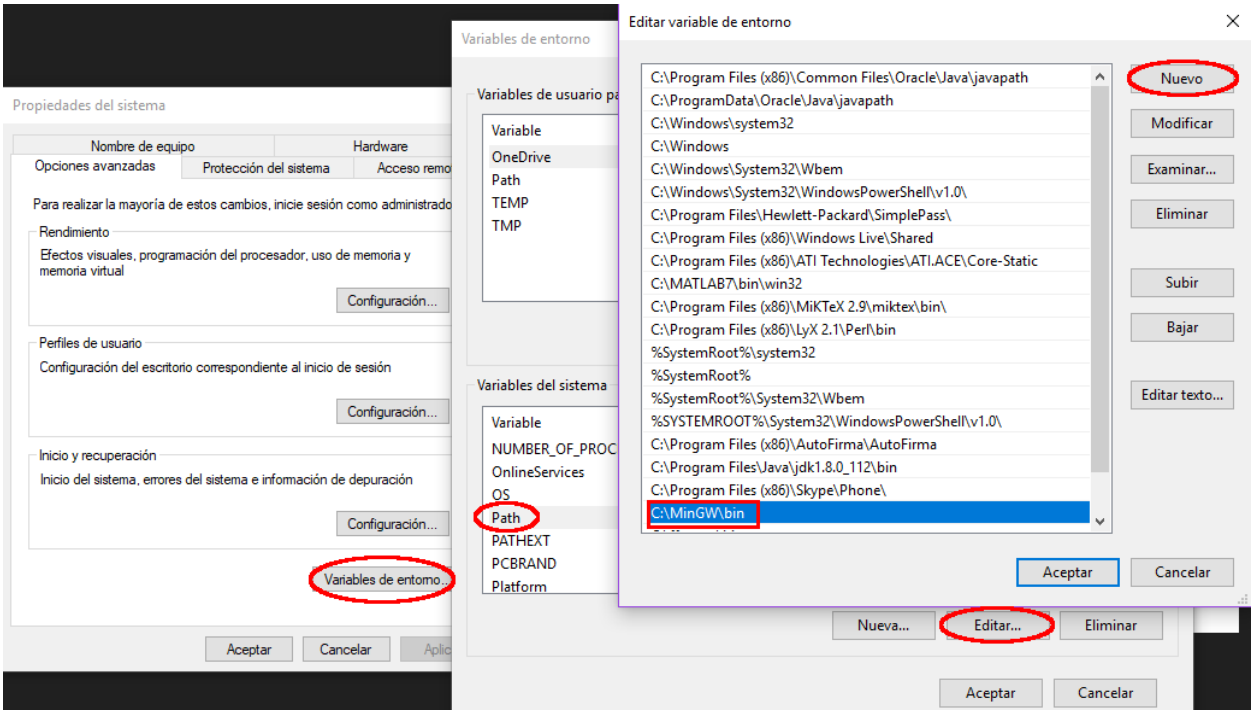


Figure 3: New value to Path

Analogously this has to be done with "C:\cygwin\bin".

Eclipse

Visit Eclipse then click "Download" to install it, the Photon version can be obtained which will allow to install the desired packages for *Eclipse*. In this case, once the installer is executed, select "Eclipse IDE for C/C++ Developers".

It is important to remark that *Eclipse* requires Java 1.7.0 or higher.

Once *Eclipse* is installed, the first time the program is initiated the path for the working directory will have to be indicated.

The programs that will be described below have to be imported into *Eclipse*. From *Eclipse* do the following steps:

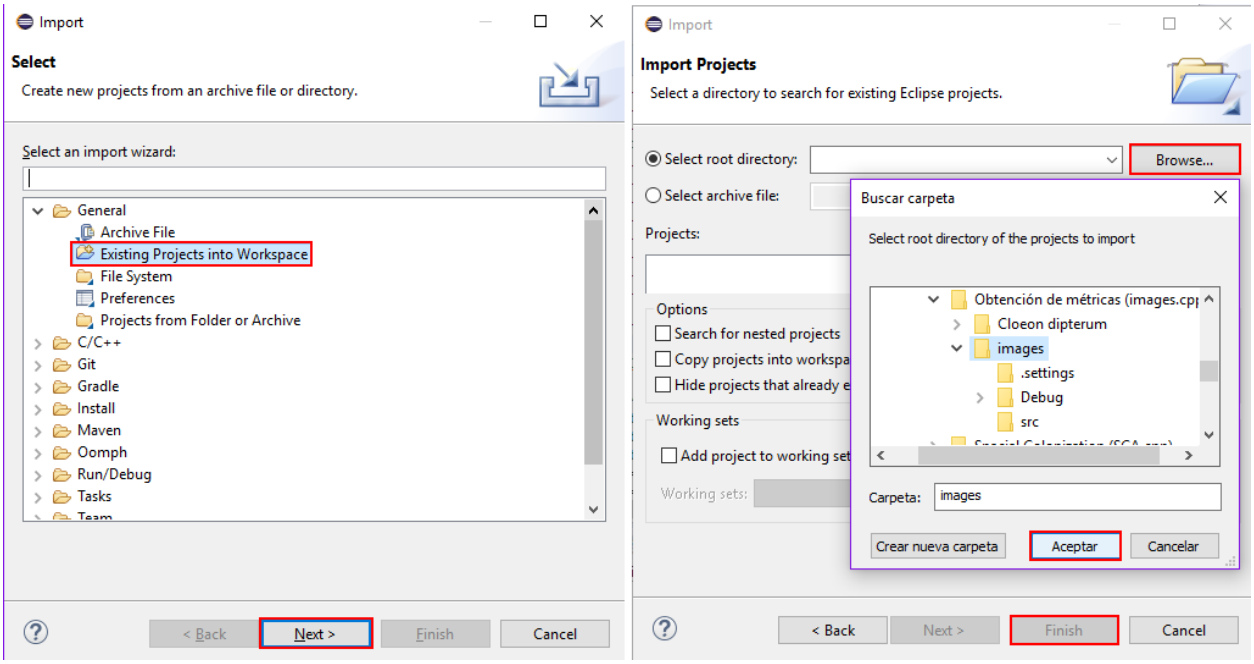


Figure 4: Importing existing project to Eclipse

Menu File > Import... > General > Existing Projects into Workspace > Next, and into Select root directory, with Browse... select the directories for "Generación de atractores (puntos.cpp)\puntos"; "Spacial Colonization (SCA.cpp)\SCA" and "Obtención de métricas (images.cpp)\images".

Click "Finish" to finish the import.

These folders must be included one by one, so these previous steps must be repeated three times, one for each folder.

It will be necessary to install the components of the graphical library freeGLUT, which is indispensable for the SCA.cpp and images.cpp programs to function. freeglut 3.0.0 MinGW Package can be downloaded from here. Once uncompressed, the content of the folders "bin", "include" and "lib" must copied into the folders with the same name in the C:\MinGW\ directory.

All the C++ programs described here are refined and optimized in such a way that no error or notification must appear once compiled. Nevertheless if an error occurred, it may be solved by doing one of the following steps:

- Checking that the file C:\Windows\System32\glu32.dll exists.
- Reviewing the settings for *Eclipse*, specially the Properties that appears by right clicking on the folder for each project, see Figure 5:
  - C/C++ Build > Settings > GCC C++ Compiler, Command : g++, All options: -O0 -g3 -Wall -c -fmessage-length=0 -std=c++11

– C/C++ Build > Settings > GCC C Compiler, Command : gcc, All options: -O0 -g3 -Wall -c -fmessage-length=0

– C/C++ Build > Settings > MinGW C++ Linker > Libraries, Libraries (-l): freeglut; opengl32; glu32

– C/C++ Build > Tool Chain Editor, Current toolchain: MinGW GCC, Current builder : CDT Internal Builder

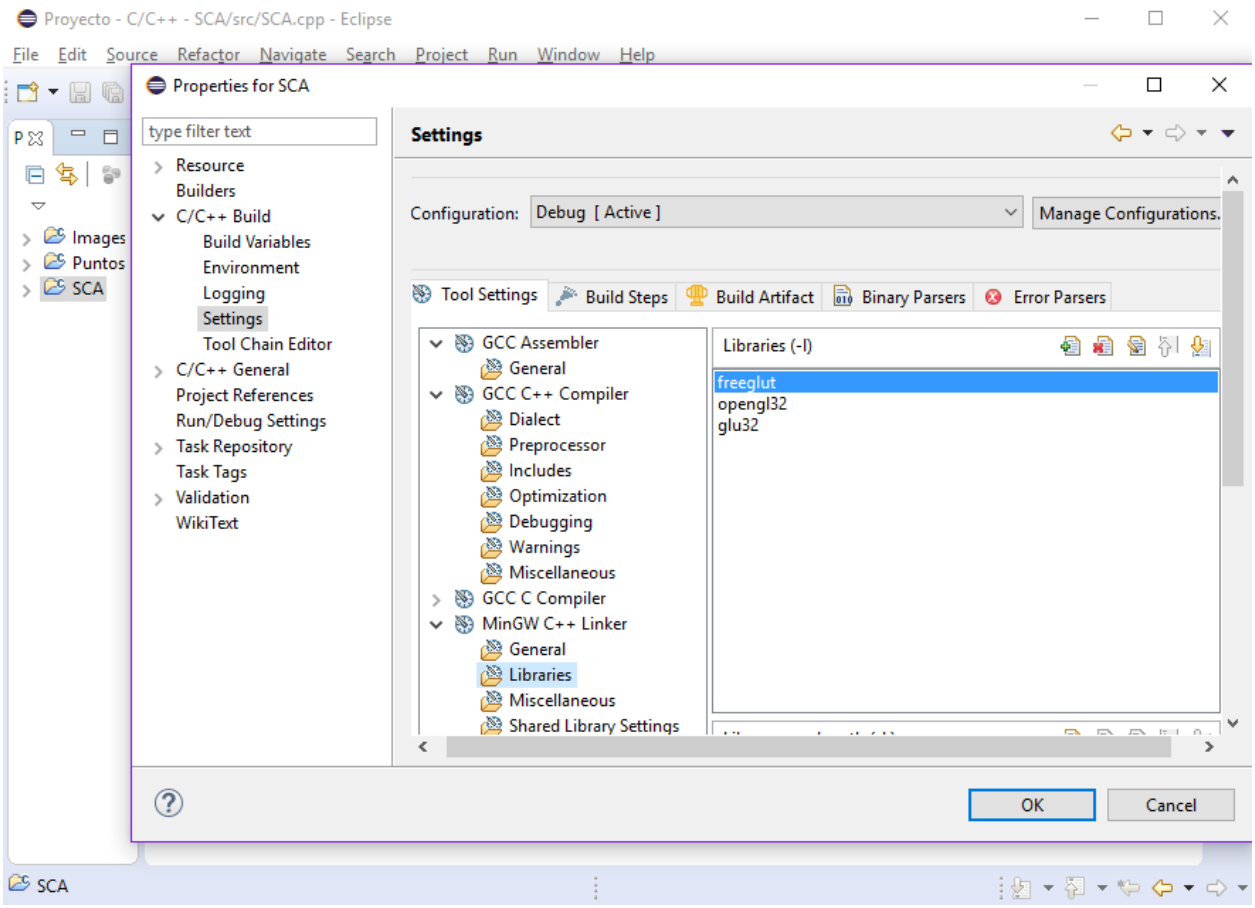


Figure 5: Setting the project properties in Eclipse

- Lastly, if the errors continue, look it up on the Internet with the description that appears on the console from Eclipse. It will probably be an error in the IDE settings or some missing file in the compiler.

2.2. Compiling and executing a program

It is convenient to compile the programs in Eclipse after each modification made in its code and to check that no error is produced before executing it.

To compile or build the executable file from a program it is just needed to press the "hammer" icon, which will apply this action to the program selected as active. This can also be achieved by the combination of the keys "ctrl + B", but in this case all the programs opened in the editor are compiled.

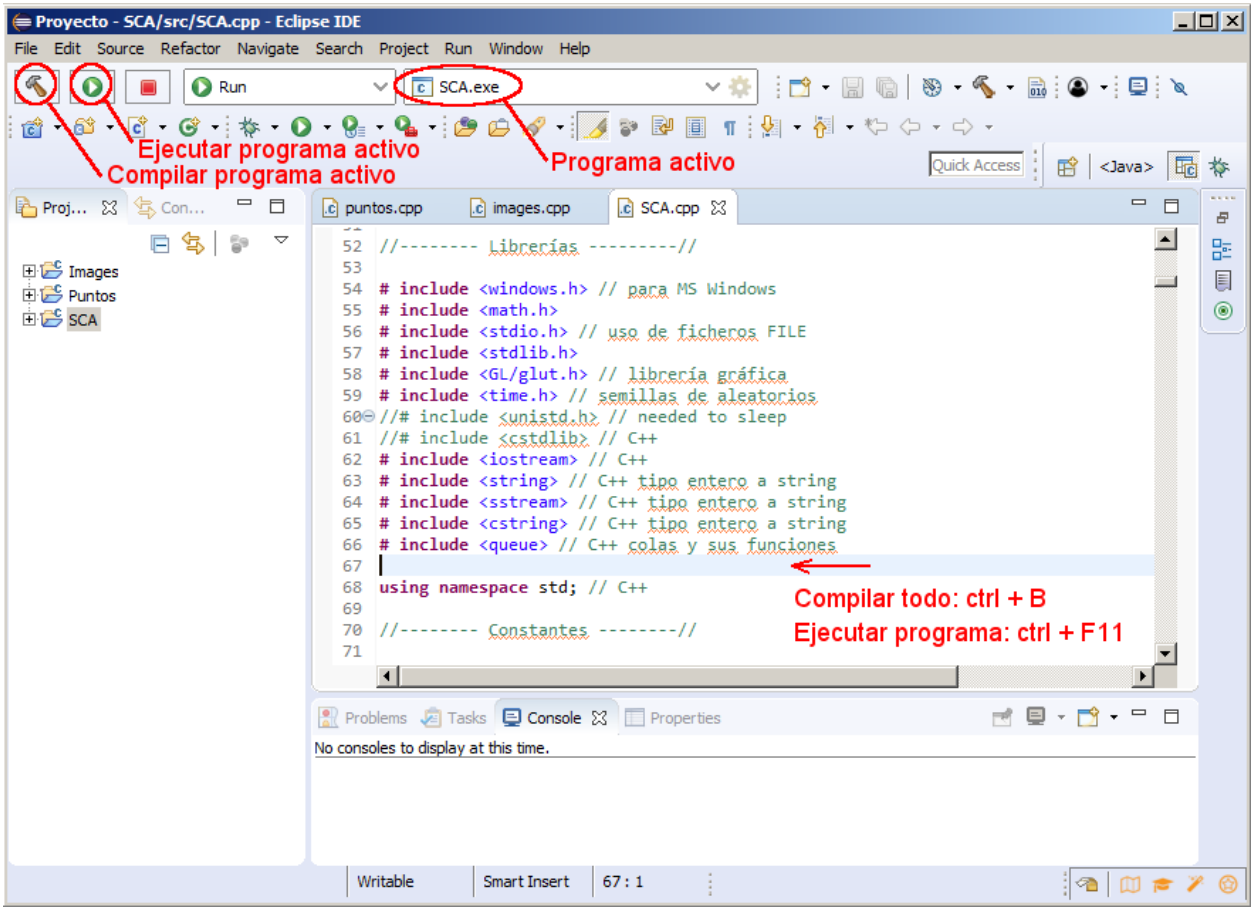


Figure 6: Compiling and executing a program in Eclipse

Lastly, the key button that is displayed as a white triangular shape over a green circle ("play") next to the compile button is used to execute the program that is active in the tab. Pressing "ctrl + F11" will also launch the execution of the program open in the editor.

To compile as well as to execute a program using the cited commands, the mouse pointer must be in the edition tab with the code from that program, see Figure 6.

3. Computational model

In this chapter the process to generate an *in silico* branched pattern similar to the one present in the gills of *Cloeon dipterum* is described in detail, using the SCA computational model.

We next describe the consecutive steps of the model execution framework.

3.1 Gill shape. GeoGebra

The first step is the election of the outline that will give shape to the "virtual gill", in which the attraction points will be uniformly randomly distributed. Two typologies will be considered: elliptical and polygonal. To define the foci and semi-axes of an ellipse or the coordinates of a polygonal shapes any program may be used. One such program is "Geogebra".

3.2. Attraction points: "puntos.cpp" program

Knowing all the elements that define the perimeter that will act as outline of the gill, elliptical or polygonal, the next step is to generate the coordinates for the attraction points. These points will be distributed randomly within the gill's perimeter, and will be generated with the program "puntos.cpp". The parameters for defining the gill's perimeter and the attraction points coordinates will be exported in two separate file simultaneously and will be used as input for the SCA.cpp program.

3.2.1. Program options.

In the section options for the user, inside the code of the program puntos.cpp, see Figure 7, it has been introduced the following parameters:

1. M ($\in \mathbb{N}$): size of the subset of the plane (square MxM) that will contain the gill. We select M=512
2. NPA ($\in \mathbb{N}$): number of attraction points wished to generate. In case of starting with a specific concentration of points, NPA it is calculated multiplying this with the gill area, which will be obtain easily from the entering line in GeoGebra (command Área(c), being c the name of the ellipse or the polygon used as gill).
3. F ($\in 0, 1$): shape of the outline of the gill in which interior will be generated the attraction points. Depending on the value of F, it will have to be indicated the elements defining the gill.
   3.1 F = 0 for the elliptical shape: $c1, c2$ determine the coordinates for the center of the ellipse, being a and b their horizontal and vertical axes respectively.
   3.2 F = 1 for the polygonal shape: polygon[] [2] will include the coordinates of all the vertexes in the polygon, in the way a_i, b_j.

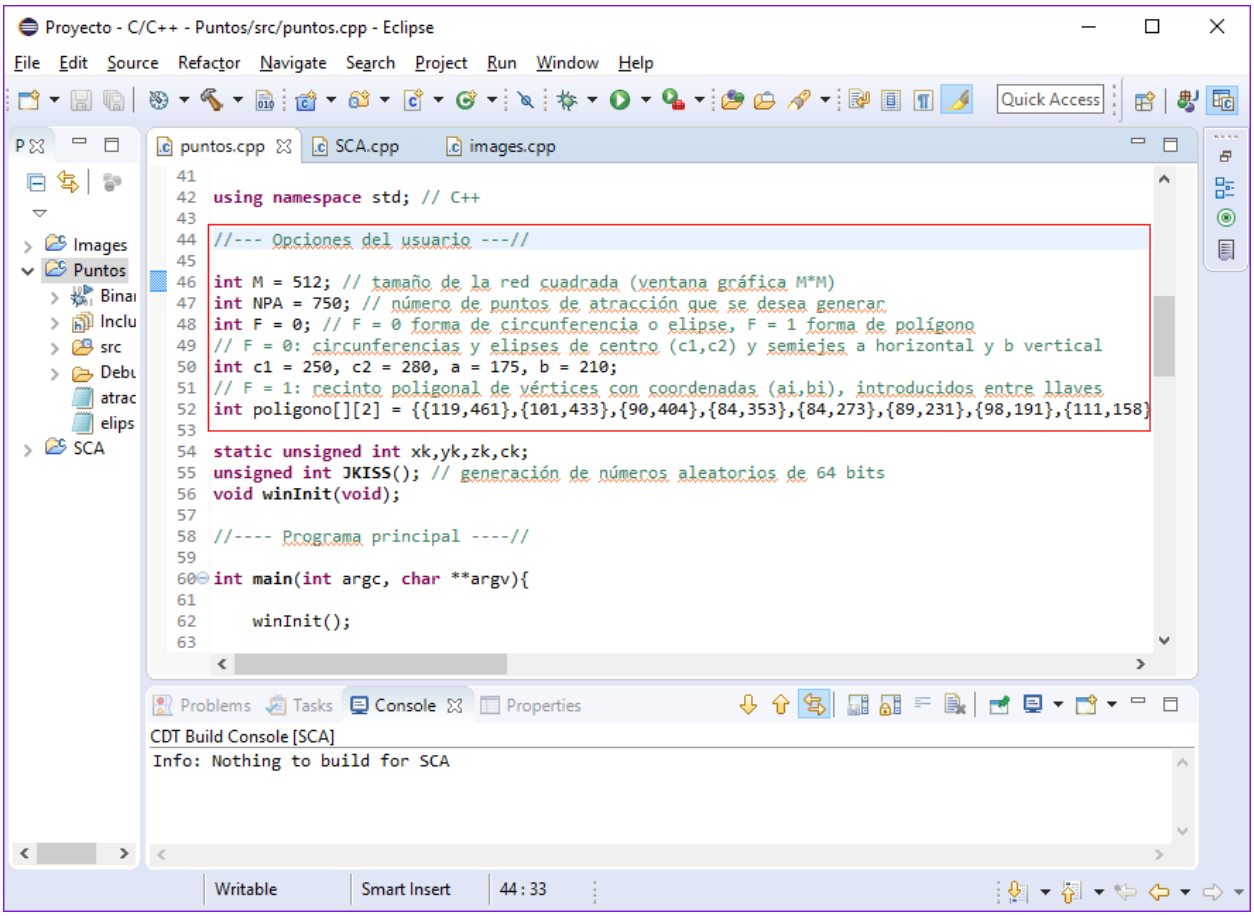


Figure 7: User program options inside puntos.cpp code

3.2.2. Output data

The execution of "puntos.cpp" will yield two files inside the directory "Generación de atractores (puntos)\" which will be required as input for the program "SCA.cpp":

1. "atrac.dat": stores the coordinates (x_i, y_i) from the attraction points generated.
2. "elips.dat": stores the defining elements of the gill, $(c1, c_2, a, b)$ if it is an ellipse or the coordinates (a_i, b) if it is a polygon.

3.3. SCA. SCA.cpp program

The implementation of the space colonization algorithm in its bi-dimensional version has been carried out in the program SCA.cpp

Using of the SCA.cpp program:

The program will allow: The execution of the space colonization algorithm in the interior of the selected gill perimeter. Visualization of the graphical evolution of the algorithmic process in each iteration The generation of a “synthetic gill”.

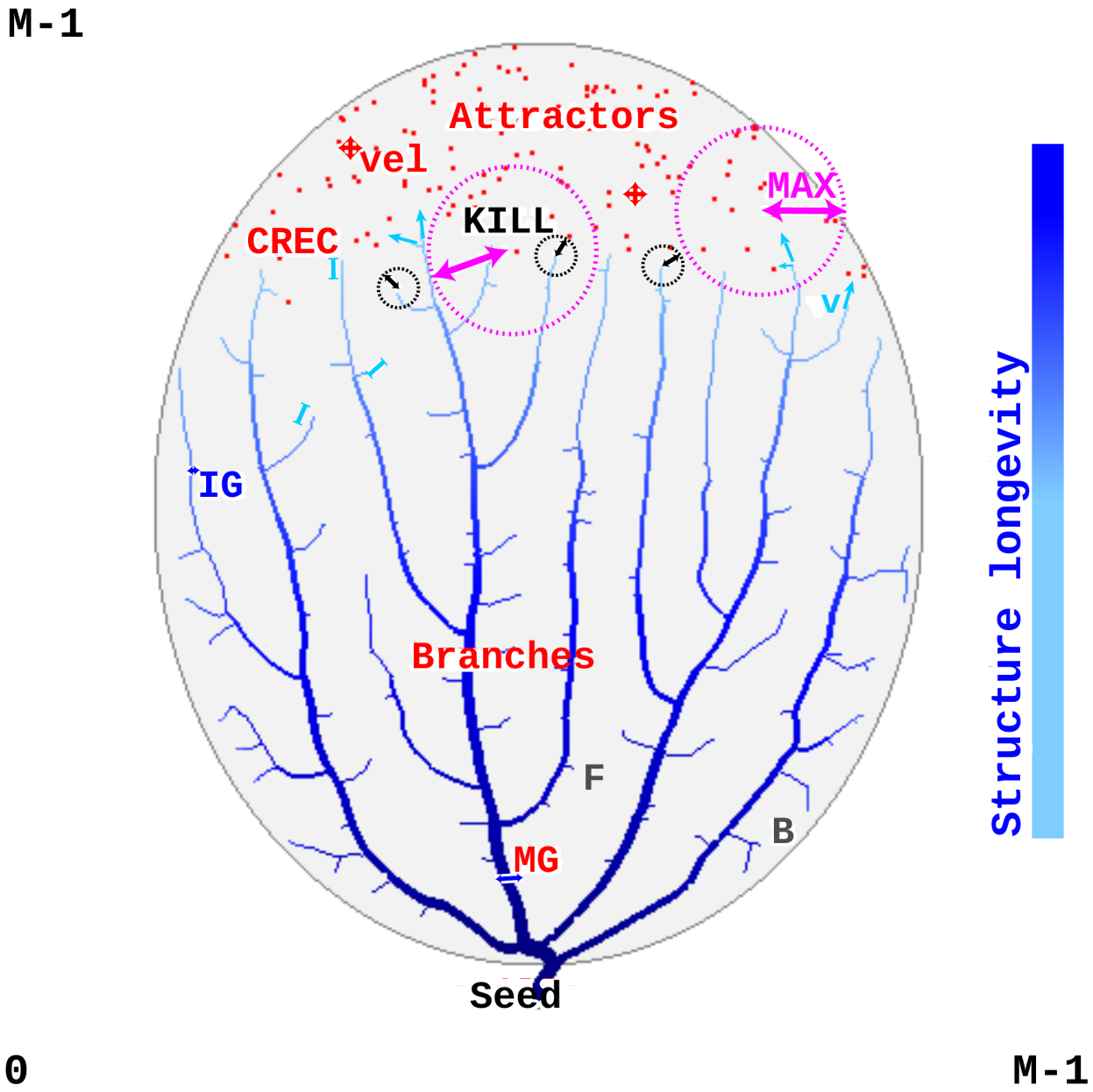


Figure 8: SCA.cpp execution (MAX = 60, KILL = 9, CREC = 2, vel = 1)

3.3.1. Program options

The SCA.cpp code includes, accessible in the section User options & Flags, see Figure 9, a series of optional variables to be specified by the user, which will determine the behavior of

the program. Those are:

1. M ($\in \mathbb{N}$): size of the subset of the plane (square MxM) that will contain the gill. We use M=512.

2. V ($\in 0, 1$): graphical output window for the evolution of the structure in each iteration.
   2.1. V = 0: will not show the graphical output
   2.2. V = 1: will show the graphical output.

3. F ($\in 0, 1$): background from the enclosure of the distribution of the attraction points.
   3.1. F = 0: will not draw the gill background.
   3.2. F = 1: will draw the gill background.

4. B ($\in 0, 1$): outline from the enclosure of the distribution of the attraction points.
   4.1. B = 0: will not draw the gill outline.
   4.2. B = 1: will draw the gill outline.

5. T ($\in \mathbb{N}$): actualization period for the graphical window output (actualize when the number of iterations is multiple of T).

6. IG ($\in \mathbb{N}$): initial thickness, in pixels, of a new branch.

7. MG ($\in \mathbb{N}$): maximum thickness, in pixels, that a branch can reach.

8. C ($\in \mathbb{N} \cup 0$): gill growth in each algorithm iteration.
   8.1. $C = 0$: no growth, the gill has its maximum size from the beginning.
   8.2. $C > 0$: there is growth, C indicates the initial length in pixels of the vertical semi axis b of the ellipse (C<b) or the size percentage from the polygon used as gill ($C < 100$), depending on the case.

9. CE ($\in 0, 1$): branch expansion when the gill growth is activated ($C > 0$).
   9.1. CE = 0: the branches do not expand through the gill growth.
   9.2. CE = 1: the branches expand through the gill growth.

10. FC ($\in \mathbb{N}$): gill growth period (it grows when the number of iterations is multiple of FC).

11. S ($\in 0, 1$): random appearance of new attraction points when the gill expansion is activated ($CE = 1$).
    11.1. S = 0: new attraction points do not appear on the gill.
    11.2. S = 1: new attraction points appear on the gill.

12. R ($\in 0, 1$): cut and regeneration of part of the branched region.
    12.1. R = 0: no regeneration is produced in any part of the gill.
    12.2. R = 1: regeneration happens in the region that obeys $y > mx + n$.
    12.2.1. m ($\in \mathbb{R}$): slope of the cutting line $y = mx + n$.
    12.2.2. n ($\in \mathbb{R}$): ordinate on the line's origin $y = mx + n$.

13. E ($\in 0, 1$): capture of the graphical view for each iteration in order to conserve the evolution.

13.1. E = 0: does not capture the graphical output in each iteration.
13.2. E = 1: does capture the graphical output in each iteration.

14. MO ($\in 0, 1$): color mode for the graphical output, depending on the chromatic combination branch-background. 14.1. $MO = 0$: represents black background and the rest of the elements (branch and outline) in white.
14.2. $MO = 1$: represents the background in white and the rest of elements in blue hues.

15. TF ($\in \mathbb{N}$): maximum time limit for the program to execute, in number of iterations.

16. P ($\in [0, 1]$): diffusion coefficient of the attraction points (probability of they not moving).

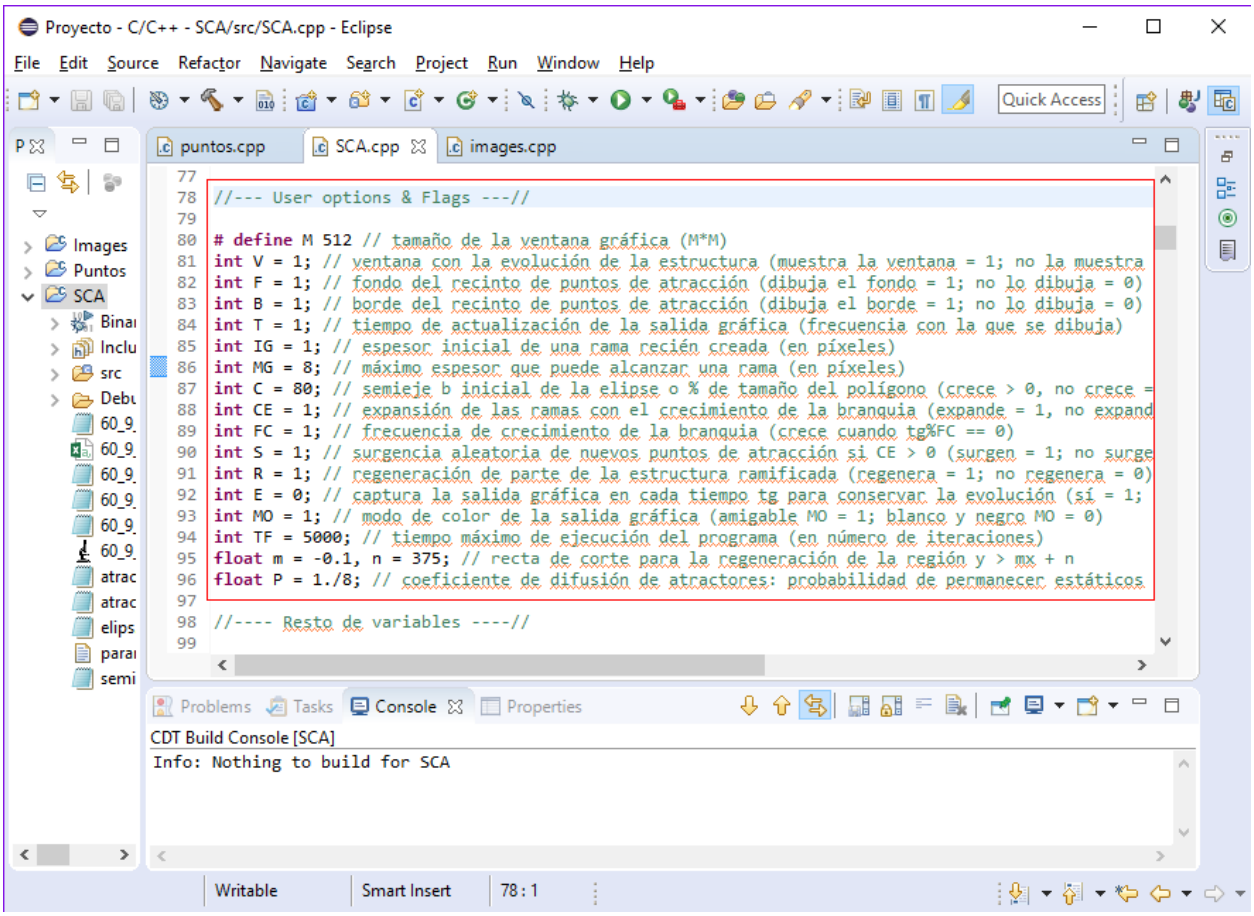


Figure 9: User program options and flags inside SCA.cpp code

3.3.2. Input data

The four main "SCA.cpp" program inputs are read from an external file, named "parametros.txt". In this file there are included the following terms, separated by a new line break, in

this same order:

1. MAX ($\in \mathbb{N}$): influence radius for the attraction points on the branches (detection distance of the attraction points by the branches, in pixels).
2. KILL ($\in \mathbb{N}$): radius influence to inhibit an attractor point by a branch (minimum distance between a branch and an attractor point to this to still be active, in pixels).
3. CREC ($\in \mathbb{N}$): growth length of branches in each iteration (size that a branch can increase on each iteration, in pixels).
4. vel ($\in \mathbb{N}$): velocity of the attraction points in their random movement (number of movements that an attractor point can make in each iteration).

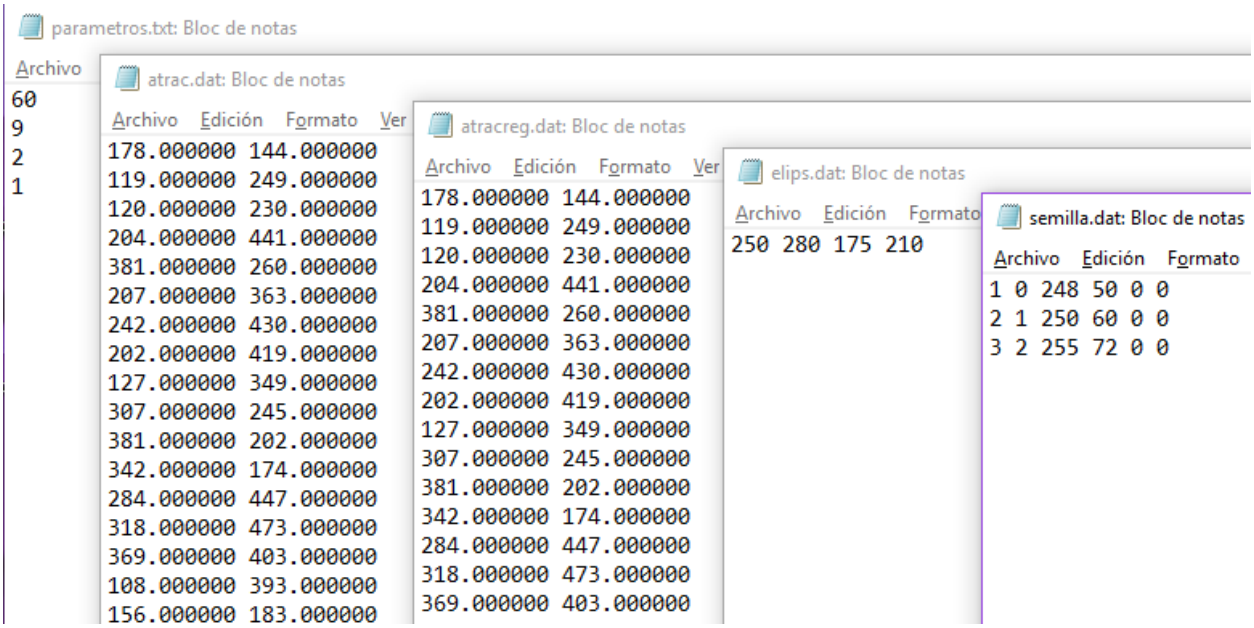


Figure 10: SCA.cpp program inputs examples

Three external additional files, "atrac.dat", "elips.dat" and "semilla.dat", give the rest of the parameters needed for the SCA.cpp program to function. Respectively they contain:

5. cx,cy ($\in [0, M-1]$): coordinates for each attractor point, each pair separated by a newline break ("atrac.dat" is generated by the program "puntos.cpp").
   5.1. cx,cy ($\in [0, M-1]$): in case that gill regeneration option is activated (R = 1), the file "atracreg.dat" with the attraction points must be included, in the same format as "atrac.dat" (it can be a copy of this or not).
6. c1,c2; a,b ($\in [0, M-1]$): centers and semi-axes of the perimeter if it is elliptical, or pair of coordinates of the vertexes separated by a newline break if it is polygonal (elips.dat it is generated by the program puntos.cpp).

7. i,ipad; cx,cy; ddx,ddy ($\in \mathbb{N} \cup \{0\}$): index, predecessor, coordinates and direction of the initial branch points, separated each sixth by a newline break ("semilla.dat" is not generated by "points.cpp").

Every file required to initialize "SCA.cpp" ("parametros.txt", "atrac.dat", "atracreg.dat", "elips.dat", "semilla.dat") must be placed on the Spacial Colonization (SCA)\ directory for the program to function.

3.3.3. Output data

Below we list and describe the outputs resulting from the execution of "SCA.cpp", generated in the directory Spacial Colonization (SCA)\, see Figure 11 for an example of the output data:

1. Graphical output of the window "Modelo SCA", recorded on the file "MAX_KILL_CREC_vel grafico.pnm".
2. The file "MAX_KILL_CREC_vel_aleator.dat" stores the random seeds used (xk,yk,zk,ck) and other relevant information for execution of "SCA.cpp" (IG, MG, C, CE, FC, S, P, time in which the cut is produced for regeneration, tr, m y n).
3. The file "MAX_KILL_CREC_vel_atrac.dat" stores the pair of coordinates from the attraction points used (cx,cy).
   3.1. The file "MAX_KILL_CREC_vel_atracreg.dat" stores the attraction points used for the gill regeneration if (R= 1), in the same format as "MAX_KILL_CREC_vel_atrac.dat".
4. The file "MAX_KILL_CREC_vel_elips.dat" stores the defining elements of the gill perimeter, (c1,c2,a,b) if it is an ellipse; or the coordinates of all the vertexes if it is a polygon (pairs ai ,bi).
5. The file "MAX_KILL_CREC_vel_arbol.csv" stores all the characteristic elements of all the points in a branch, those are: index, coordinates, generation time, and index of the predecessor branch point (i,cx,cy,tg,ipad).

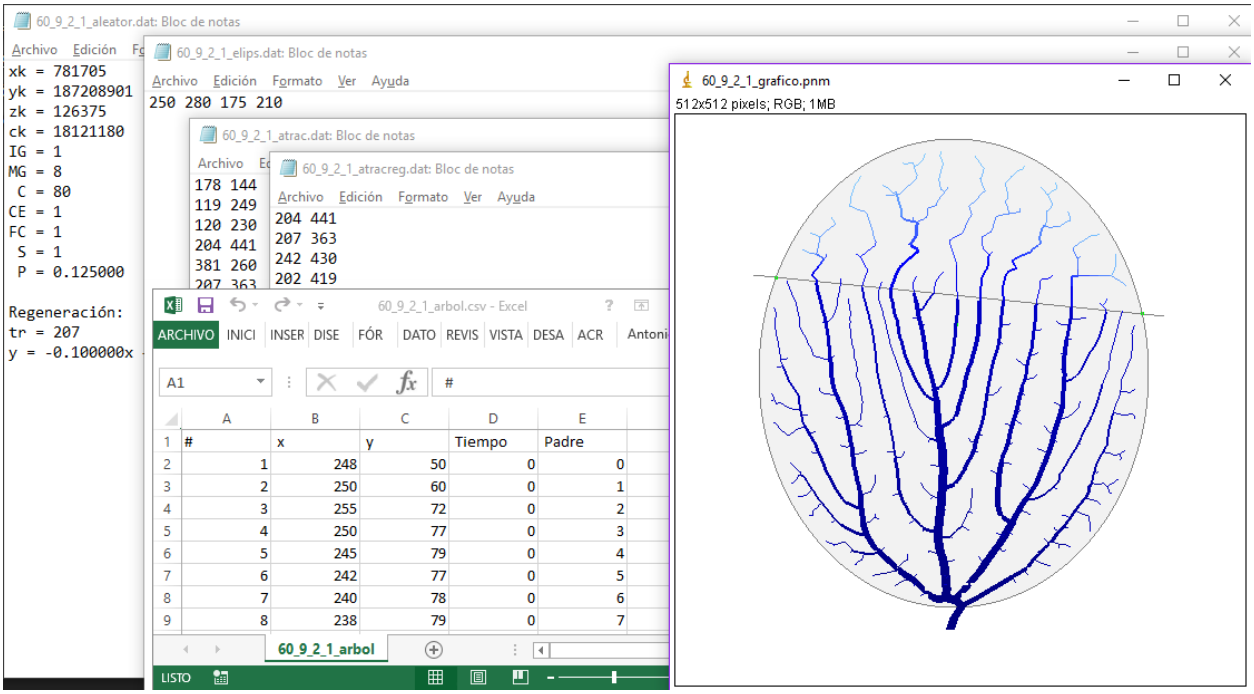


Figure 11: SCA.cpp program outputs examples

3.3.4. Pseudo-code

// MAX -> maximum radius of influence of M
// KILL -> maximum reach of a branch
// CREC -> growth in length of each segment per iteration
// vel -> velocity of Ms in their random motion
// coordinates of attractors (M) and of seed branch

```
Program SCA {
    load starting parameters;
    initialize variables (time, M number);
    enter the coordinates of seed (initial branch);

    while(number of M > 0){
        graphic output M and branches;
        time++;

        for(all M){
            associate M to closest branch;
            if(distance(M,closest branch) < KILL{
                remove M;
                M number--;
            }
```

```
            if(distance(M,closes branch) < MAX{
                M attracts the branch
                update the growth vector of this branch;
            }
        }
        for (all branches){
            if (there are attracting M){
                branch or grow:
                extend a new branch of length CREC along direction v;
            }
        }
        while(vel) for(all M) move M randomly;
        if(number of M == 0){
            save data;
            stop and finish program;
            }
        }
    }
```

4. Image processing

This sections deals with the processes aimed at extracting features from the image/SCA output to be further analyzed (i.e. metric characterization). The image must have the following characteristics:

- Size: 512px x 512 px; ".pnm" format, RGB color.
- The skeletonized tracheas and gill perimeter must be 1 px thickness.
- The pattern must have continuity (only one connected component), and no crossings of branches (no loops allowed in the structure).
- The gill perimeter must be convex and must be connected only by the root of the branched structure, it cannot touch any branch, see Figure 12.
- Root of the branched structure must be placed at the inferior-most part of the gill (that is to say, there must be no branch fragment at lower height than the start of the root in the gill's perimeter).
- It must have only two colors: white for the perimeter and the branched structure over black background.

There are two types of images, depending on their origin: synthetic images resulting from the "SCA.cpp" program execution, and biological images obtained under the microscope from real gills. In the first case, the options for "SCA.cpp" will be configured so the graphical output obtained will have 1 px throughout. In the second case, digital images must be pre-processed to get a skeletonized image.

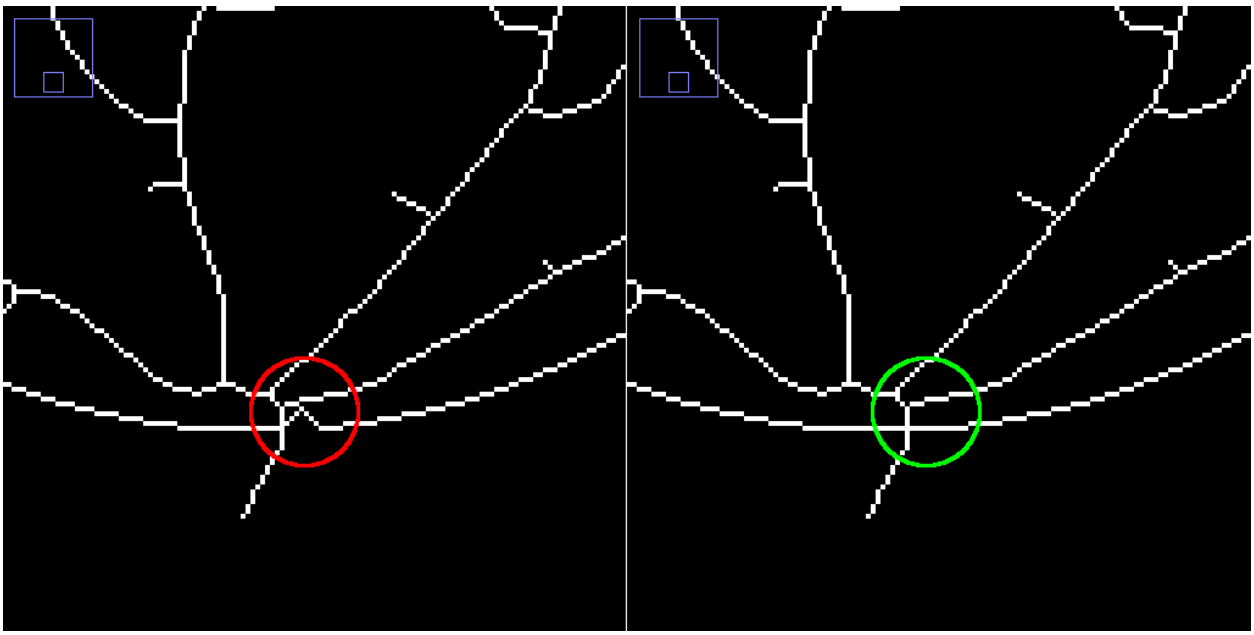


Figure 12: Correction of a skeletonized image

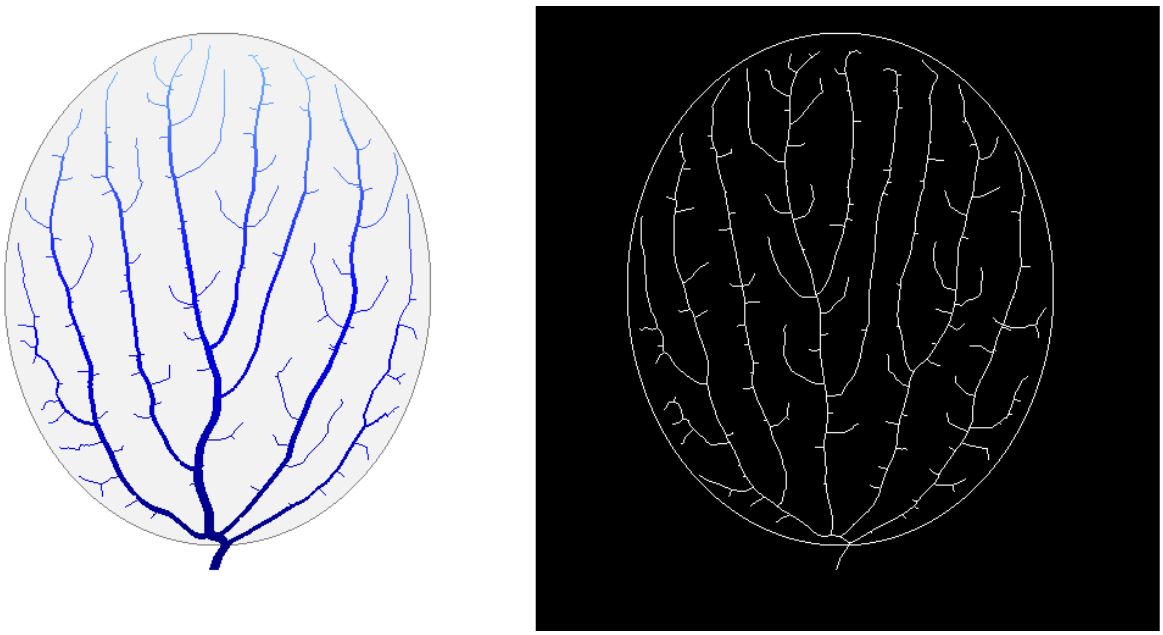


Figure 13: Synthetic image generated by the SCA.cpp program and its skeletonization

4.1. Synthetic images

They can be directly produced from "SCA.cpp" to be used by the "images.cpp program", which will yield the metrics for the given images. For that purpose, the following options need to be established in the "SCA.cpp" program:

1. F = 0: to not to draw the background of the gill
2. B = 1: to represent the gill outline
3. IG = 1: to fixate in 1 the initial thickness of the branches
4. MG = 1: to fixate in 1 the maximum thickness of the branches
5. MO = 0: to establish the white over black background mode.

In any case, it is possible to process an image that has not been obtained with this combination of options for its posterior analysis by "images.cpp".

4.2. Convert synthetic image to skeletonized image

A 512X512 px digital image of a natural gill, is opened with "ImageJ" or "FIJI". Next, "Drawing tools", in the last icon on the menu toolbar, must be activated. The next script can be followed:

1. Using "Flood Fill Tool", the background is painted in white (RGB code 255,255,255).
2. Changing the image to 8 bits, by clicking: Image > Type > 8-bit.
3. Clicking: Image > Adjust > Threshold... and then moving both sliding bars to maximum (255) with dark background option activated, then clicking Apply.
4. Transforming the image to binary color by clicking: Process > Binary > Make binary.
5. Obtaining the skeleton of the branched structure by clicking: Process > Binary > Skeletonize
6. If the image is black over white background (it usually happens in ImageJ, not in FIJI), invert the colors by clicking: Edit > Invert.
7. Changing the image type to RGB by clicking: Image > Type > RGB color.
8. Checking that the outline of the gill remains convex (specially in the part connected to the root of the branched structure)
9. Save the image in .pnm format by clicking: File > Save as > PGM... (it will actually save it as .pnm).

After those steps an image is ready to be used as input for images.cpp, see Figure 14 to see each step of converting a synthetic gill image into a skeletonized image.

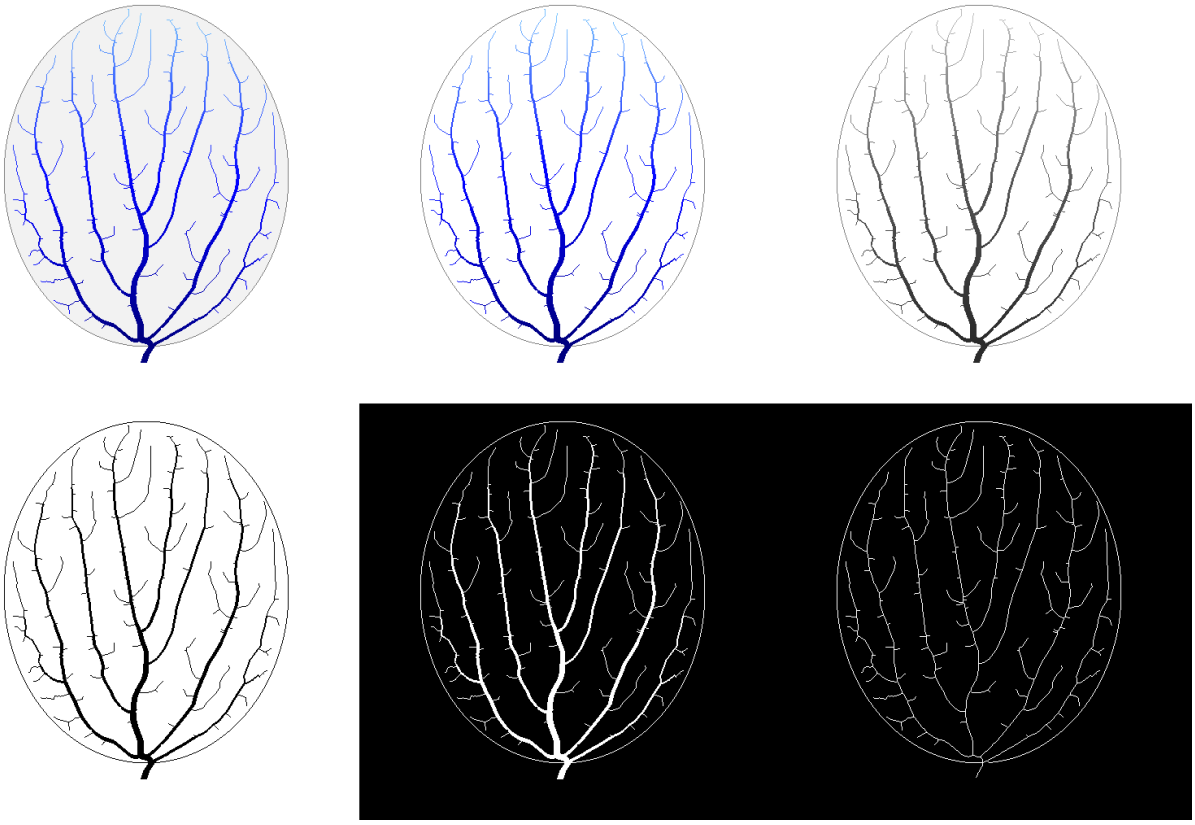


Figure 14: Edition process of a synthetic gill to obtain a skeletonized image

5. Data analysis and extraction

5.1. "images.cpp" program

In order to extract the relevant information that describes, as a whole, the geometry of the pattern, the program "images.cpp" has been implemented.

Using "images.cpp":

- Uses as input skeletonized branched patterns, obtained directly from synthetic gills or after processing of natural gill images, in ".pnm" format.
- It detects branching points and branches ending points after going through all the skeleton structure from the root.
- It extracts and exports the metrics and information from the input images.

5.1.1 Program options

In the code from "images.cpp" there are included a series of global variables, called user "Opciones del usuario" (Spanish for "User's Options"), see Figure 15, which have to be established prior to the execution of the program.

1. M ($\in \mathbb{N}$): size the window (square MxM) for the graphical output. Set to M=512
2. "fruta (chain of characters)": path of the directory where the image is placed (already processed) which will be analyzed.
3. "fprefijo (chain of characters)": name of the image file without extension format. For instance, "7 L M se" or "85 15 2 1".
4. "fnombre (chain of characters)": image format extension:".pnm".
5. N ($\in \mathbb{N}$): only branches in the lines multiple of N will be computed to extract from them the metrics (for convenience fix N = 1).
6. S ($\in 0, 1$): establishes where the program starts to store the metrics.
   6.1. S = 0: registers only those lines in which there is presence of a gill (recommended option).
   6.2. S = 1: registers from line 0 of the image to line 512.
7. T ($\in 0, 1$): mode of counting the branches found in each analyzed line.
   7.1. T = 0: counts all the pixels from branches found (recommended option).
   7.2. T = 1: counts only by changes in black-white and vice versa, omitting intermediate branch pixels.
8. "mili" ($\in \mathbb{N}$): waiting time, in millisecond, between each branch count and the next.

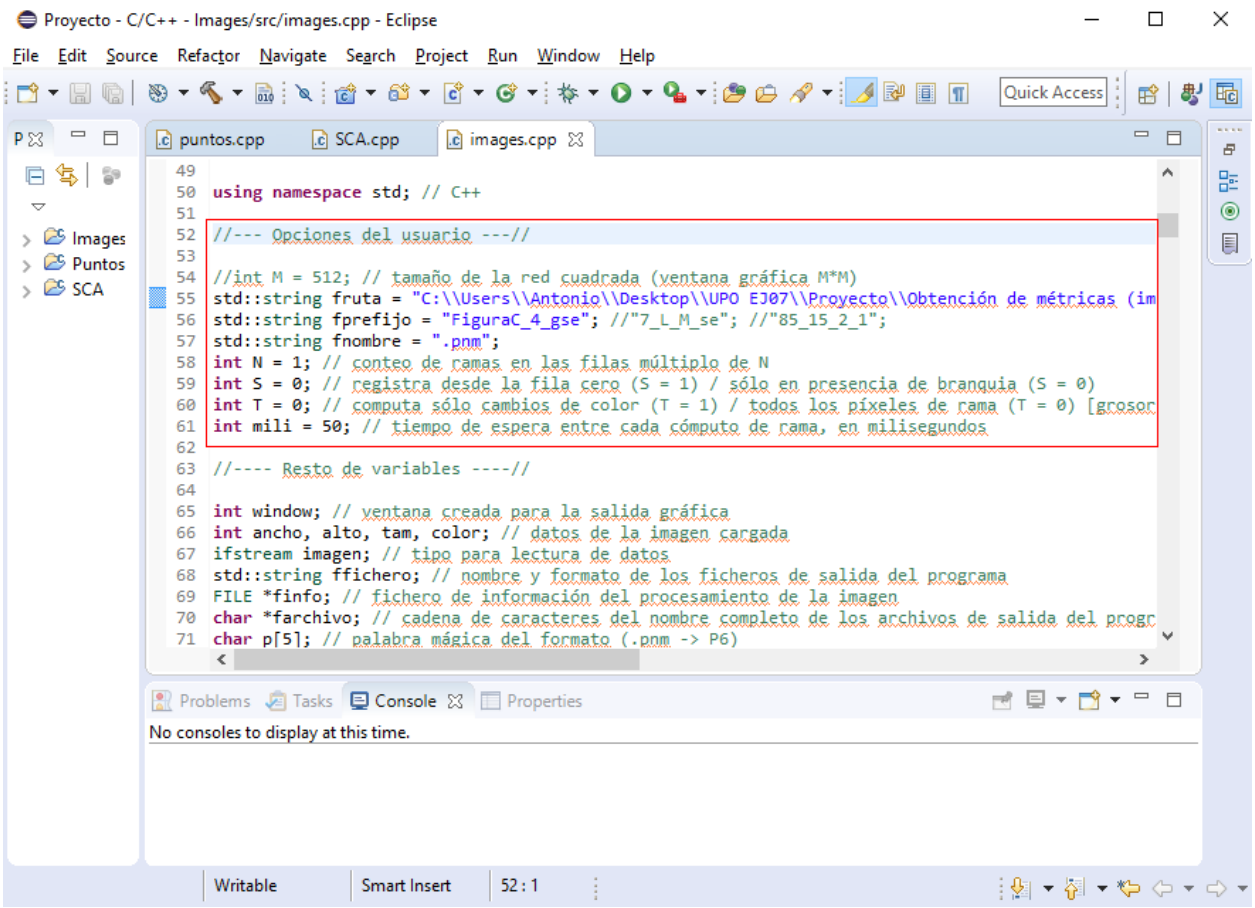


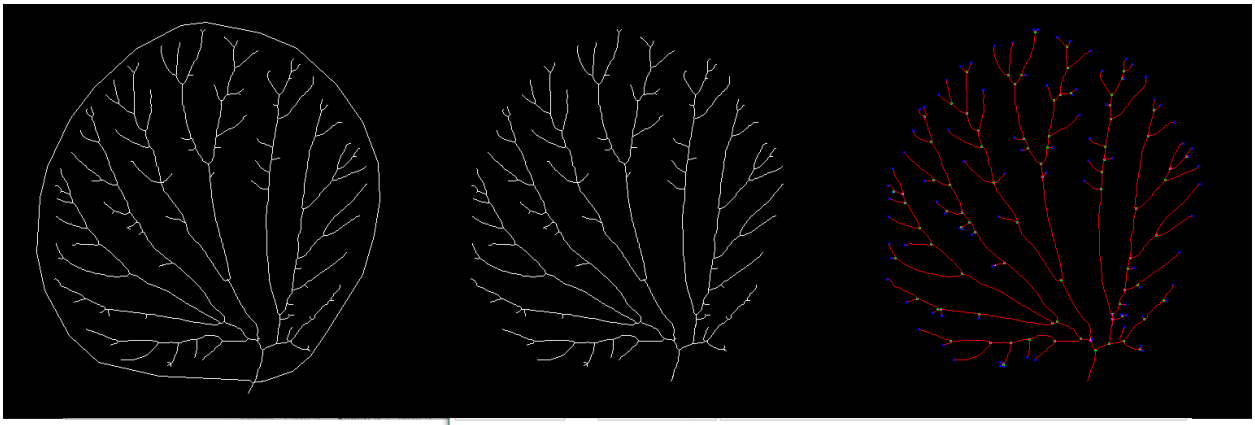
Figure 15: User program options inside images.cpp code

5.1.2 Input data

There is only one input for “images.cpp”, which is the graphical output from “SCA.cpp” or the processed image from the biological image, with the required characteristics described previously in section 4.1:

Image file in format “.pnm”, RGB color and skeletonized (without nodes >= 2x2 pixels without loops), with its root in the most inferior part of the gill, with perimeter elliptical or convex polygonal (the outline will be suppressed automatically to estimate the fractal dimension).

The image must be placed in the directory path introduced in the user options “fruta + fprefijo + fnombre”.



4_L_M_gse_info.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
Imagen: pm = P6; ancho = 512; alto = 512;
colores = 255
Impresión de la imagen realizada
correctamente.

Detectados 84 puntos de ramificación.
Detectadas 89 puntas de ramas.
Eficiencia de alcance: 89/89.

Dimensión fractal estimada: 1.257024.
Cantidad de rama: 4364.
Área de la branquia: 145634.
Densidad de rama: 0.029966.
Generación media: 7.966292, varianza:
6.189878.

Las métricas se han computado en las filas
indicadas.
```

	A	B	C	D	E	F	G	H
1	Núm. Rama	Generación	Longitud	Media Gen.	Varianza Gen.			
2	1	8	248					
3	2	9	41					
4	3	9	43					

4_L_M_gse_genearamas_sincro

LISTO 100 %

L12 0

	A	B	C	D	E	F	G	H	I
1	Fila	Branquia	Rama	Rama/Fila	iRama/iFila	Rama/Branquia	iRama/iBranquia	Nodos	Nodos/Fila
2	0	2	0	0	0	0	0	0	0
3	1	6	0	0	0	0	0	0	0
4	2	20	1	0,5	1	0,05	0,071429	0	0
5	3	54	2	0,666667	1	0,037037	0,029412	0	0
6	4	112	3	0,75	1	0,026786	0,017241	0	0

4_L_M_gse_conteoramas_sincro

Figure 16: Input and output examples from images.cpp program

5.1.3. Output data

All output files generated by “images.cpp” will be placed in the directory indicated in the program option “fruta + fprefijo + fnombre”. Those are:

1. Graphical output of the structure without the gill perimeter (to visually check that this has been correctly detected) stored in the file “fprefijo_graficocompu_sincro.pnm”.

2. Graphical output of the structure with the nodes and branches ending points highlighted (to visually check that they have been correctly detected), stored in the file “fprefijo_graficonodos_sincro.pnm” .

3. A data file called “fprefijo_conteoramas_sincro.csv” stores data and metrics extracted from the image, by files: width or accumulated area of the gill (“Branquia”), quantity of branch pixels accumulated (“Rama”), and its ratios (“Rama/Fila”, “ΔRama/ΔFila”, “Rama/Branquia”, “ΔRama/ΔBranquia”), sum of detected nodes (“Nodos”) and its ratios (“Nodos/Fila”, “ΔNodos/ΔFila”, “Nodos/Branquia”, “ΔNodos/ΔBranquia”), sum of branch ending points found (“Puntas”), and its ratios (“Puntas/Fila”, “ΔPuntas/ΔFila”, “Puntas/Branquia”, “ΔPuntas/ΔBranquia”), and calculations to

obtain the fractal dimension of the branched structure, which is the value that repeats until the end of the last column in this .csv file.

4. A data file called "fprefijo_genearamas_sincro.csv" stores the branch number of all the branches found (in order of detection by the algorithm in the "code images.cpp"), the branch generation (number of nodes between the root and the end of the branch), an estimation of the length in pixels of the branch, and the mean and variance from all the values of the column "Generación" ("Media Gen." and "Varianza Gen.", respectively).

5. A text file called "fprefijo_info.txt" which registers all the information for the processing of the image: its path, dimensions, colors and format, the branching points and branches ending points, the efficiency of the process (if all detected branches have been gone through), fractal dimension, quantity of branches, gill area and density (branch/area), and the generation mean and variance.