Documentation Project Gas **65170 Arriu Simone – GitHub Repository**

Organization

The classes implemented to achieve the goal have been organized in the following three new folders:

- Data_structure
- Algorithms
- Drawable

Data_structure

The first folder, "Data_structures," contains the classes implemented to keep and organize the data to compute the triangulation, and these are:

- NodeDag
- Triangle
- Dag
- Triangulation

NodeDag

This class allows managing the elements that composed the dag itself. Each node of the dag have four main attributes, and that are all pointers: one is used to "point" at the related triangle of the node, while the other three are used to "point" at the first, second and third child of the node. Naturally, there was implemented also the methods used to get and set all these informations.

Triangle

This class describes the triangles that composed the triangulation: each triangle has three pointers to the point (vertices of the triangle), a pointer to the related node of the triangle, and other three pointers used to manage the possible adjacent triangles. The adjacents of the triangle are managed as pointers because in this way is possible to set some changes efficiently, like the update of adjacents or the deletion operation (the adjacent methods are defined in another class). The choice to implement these two main data structures (NodeDag and Triangle) using the pointers is that they are related between them, because a node in the dag is always related to a particular triangle, and the same in the other case: given a triangle, I want to know the related node associated to it. The other main attribute is the boolean "removeTriangle", that was implemented because some triangles, (after the edge flip or after the creation of three new triangles) are no longer considered in the triangulation: this attribute is used when it's necessary to draw the triangulation and when are used the methods to validate the triangulation. There are also some methods used to compare two triangles and to get others informations about the triangle itself.

Dag

This class describes the data structures that handle the dag. The dag is managed and represented using a vector of pointers belonging to the class NodeDag, described previously. In this class, all the methods are necessary to update and manage the dag. One of these allows to create a new pointer to the node and add it to the vector; another is used to manage the case when it's necessary to set the children of a given node (or two given nodes in the case of an edge flip). For clear the data structure, it was implemented some "for" statement used to delete the pointers in the vector, because the constructor doesn't create the pointers, they are just attributes of the NodeDag class. One of the main methods of this class is "findNode," that returns the node containing a given point, method very useful at the beginning of the algorithm when it's necessary to know which is the node related to the triangle that contains the point in input.

Triangulation

This class is used to compute and update the triangulation: we store the points and the triangles in two vectors of pointers, like the Dag. There are methods used to create a new triangle or point and to add them in the triangulation. Most importantly, this class also contains the static methods used to manage the adjacency between the triangles that composed the triangulation. All these methods are necessary because the adjacents of a triangle can change in various cases: when a new triangle is built, or when the edge flip is computed. So, there are methods used to remove old adjacency, to update some of them already existed but that not are no longer related to a not exist triangle, or simply methods used to find the adjacent of a triangle given two points for example. These are the main functionalities of this class, but there are also the methods implemented to clear the vectors of triangles and points when they are no longer useful (clear triangulation). Even in this class, like in the Dag, two methods clean the memory allocated by the pointer using two "for" statement that scan the vectors and delete the pointers.

Algorithms

The folder "Algorithms" contains the two main algorithms used to compute the Delaunay triangulation, that is the following two classes:

- DelaunayTriangulationAlgorithm
- LegalizeEdge

DelaunayTriangulationAlgorithm

This class is the "core" of the project because it contains the methods used to manage the insertion of a point into the triangulation, to compute the Delaunay triangulation itself correctly. This class has two main important attributes: the triangulation and the dag, two objects that are updated to obtain the Delaunay triangulation. One of the most important methods is the one that allows setting the bounding triangle, the triangle that contains all the other points: the data structures are updated, and the dag always contains this node as the root. Given a point in input, the method that inserts the point inside the triangulation is "fillTriangulation", from here, this point is used to find the triangle where it lies (there are also a control to check if a point is already existent in the triangulation), and from this point are created the three new triangles. This class update the data structures triangulation and dag with the respective methods and include the classes that use static methods, like the Triangulation to manage the adjacency and the LegalizeEdge (described later) to check if it's necessary to compute the edge flip or not. This class also contains the methods used to check if the triangulation is correct or not: about this; this class has another important attribute, a map "index_point" composed by <Point2Dd, unsigned int>. This map, for each point in the triangulation, store the pair formed by the point itself and its index in the vector of points belonging to the triangulation. The "getTrianglesValidation" method use the map managed before, when it's necessary to fill the matrix composed by "unsigned int": for each triangle in the triangulation (check only useful triangle), we use the map to find the indexes of the points related to the three vertices that composed the current triangle. Finally, this class contains also the methods used to clear the triangulation (remove the data from the data structures used, include the pointers inside the vectors in the Triangulation and the Dag).

LegalizeEdge

This class contains the static method used to compute the legalization of the edges created by new triangles. I manage this method like another algorithm because it's related to the Delaunay triangulation, but it is also another algorithm, that executes another task, that is the edge flip. In this method, the parameters are the current triangle, the point in input, the points pi, pj, and other parameters like the triangulation and the dag. First, we find the adjacent of the given triangle on the two points pi and pj: if there aren't adjacents, the edge is legal, but if we found an adjacent, we check using "isPointLyingInCircle" if the edge is illegal, and in this case, we compute the edge flip. So, we "remove" the current triangle and its adjacent, delete the oldest adjacency, create the new triangles (updating the triangulation and the dag) and update the new adjacency about the triangles just created. Then, we call the legalizeEdge method on the new edges again.

Drawable

Now we describe the classes inside the "Drawable" folder:

- DelaunayTriangulationDrawable
- VoronoiDiagramDrawable

DelaunayTriangulationDrawable

The Delaunay triangulation drawable has some attributes used to draw the triangulation (like the size of the edges, the color, etc.) and one attribute that is a vector of pointers belonging to the class Triangle: this vector contains the triangles to draw, imported by the Delaunay triangulation algorithm. There is also a boolean "visible" useful to draw the triangulation in two cases: in the first one (visible = true) the method drawBoundingVisible allow to draw the whole triangulation (include the bounding triangle and the incident edges) with the methods drawPoint2D and drawTriangle2D. In the second case (visible = false) is draw only the triangulation inside the bounding box inside the drawBoundingInvisible method, without the bounding triangle, so displays only the vertices that are not belonging to the bounding triangle and these triangles that are different to the bounding triangle itself. There is also the method used to clear the drawable, in particular, the vector that stores the pointer to the triangles.

VoronoiDiagramDrawable

This class allows drawing the Voronoi diagram, dual of the Delaunay triangulation. To draw it, it was necessary to compute the circumcircle of the triangles that composed the triangulation and use an attribute to store the triangles of the triangulation, like in the previous class. Each circumcircle computes the Voronoi vertex, so, inside the method used to draw the diagram, for each triangle belonging to the triangulation, we compute the circumcircle, and we draw it. But it's necessary to link the other vertices between them, so if the current triangle has some adjacents, we also compute the circumcircles of its adjacents triangles, and we link them by the drawLine2D method. The boolean "state" is used to make the diagram visible or not according to the clicked button.