AP2 PROJECT 2019

Documentation

The information provided is divided into seven sections:

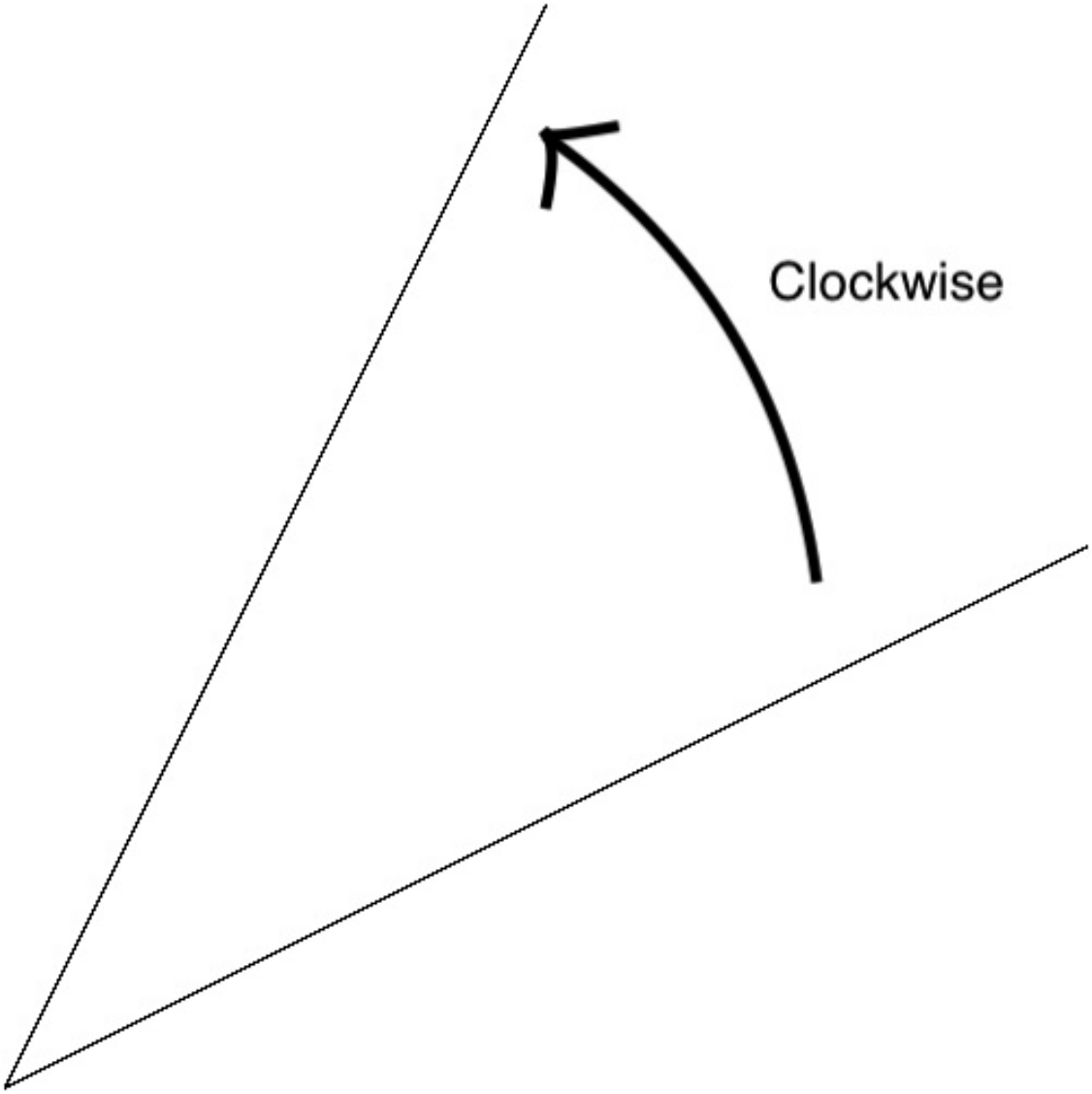
1. Class Point
2. Class ConvexPolygon
3. Polygon_Calculator
4. Tests
5. Compilation
6. Makefile
7. Additional notes and clarifications

---

1. Class Point

Useful information

- Quadrance: $x^2 + y^2$, see we are not taking square roots here.
- Slope
- Quadrant: Tells in which quadrant you are (1,2,3,4) or 5 if you are in the axis.
- Distance: Returns the distance to some point.
- Clockwise: Tells if the angle of rotation from one point to another is positive. *Observe I'm calling clockwise what caucasian people call counterclockwise*, try not to be confused. From now on, the terminology will be mathematical:
    - Positive sense of rotation
    - Negative sense of rotation



- isInside: Tells if the point is inside two given points. With an uncertainty of 0.01%.

Operations

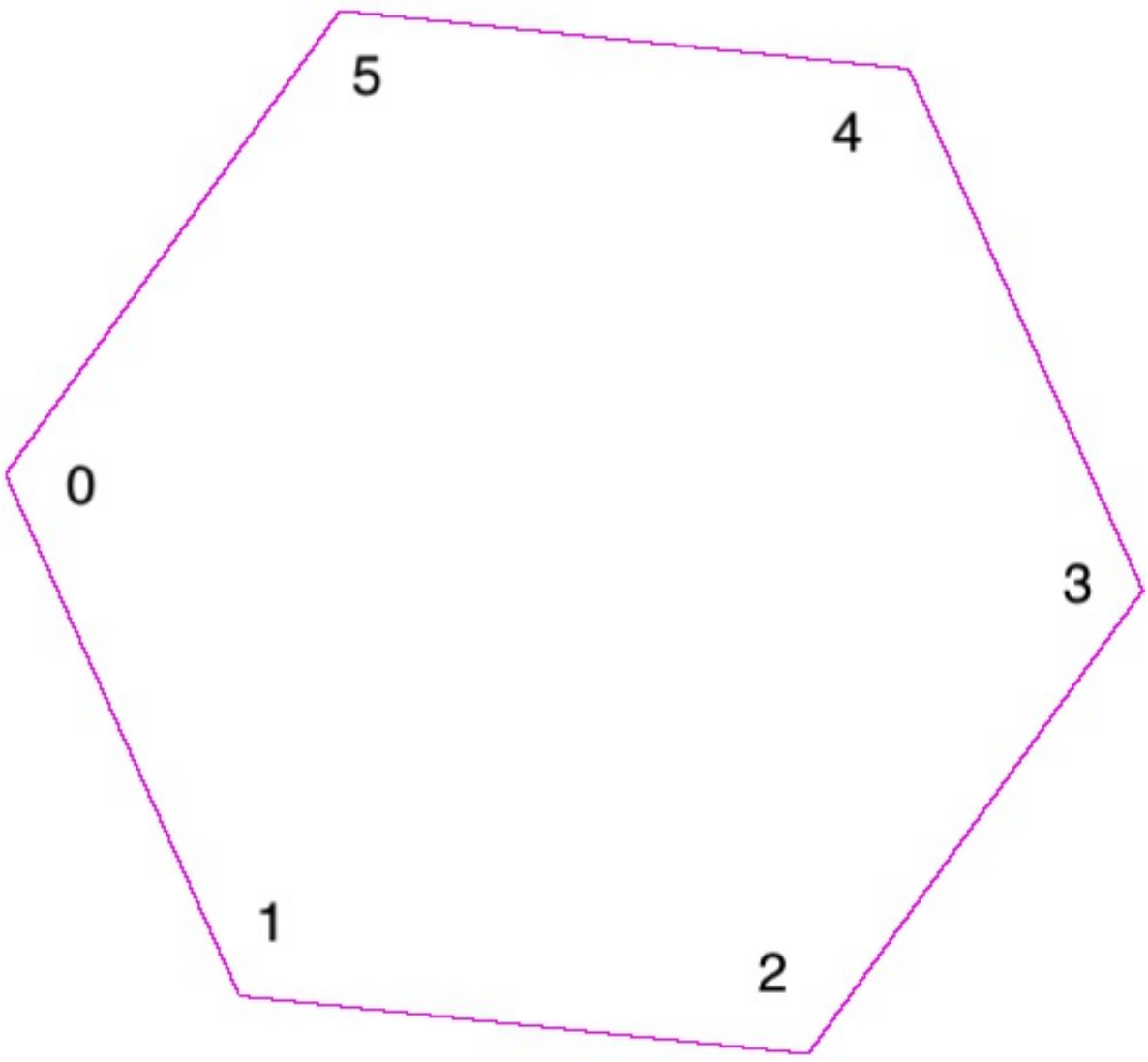
Each operation can be interpreted as done to the point P(x,y) or to the vector OP. The available operations are the following:

- Addition: Represented with + or +=.
- Substraction: Represented with - or -=.
- Negation: The unary operator -.
- Multiplication: There are three different types:
  i. By a scalar: Represented with *= and saved in the Point itself.
  ii. Vectorial product: Represented with ^, returns the modul and sign of the vectorial product.
  iii. Dot product: Represented with *.
- Equality: Represented with ==.

2. Class ConvexPolygon

Attributes

1. Hull: It is a vector of the consecutive points of the convex hull starting at the left-most one and going in the positive direction of rotation.
   It can be retrieved with Hull() method.



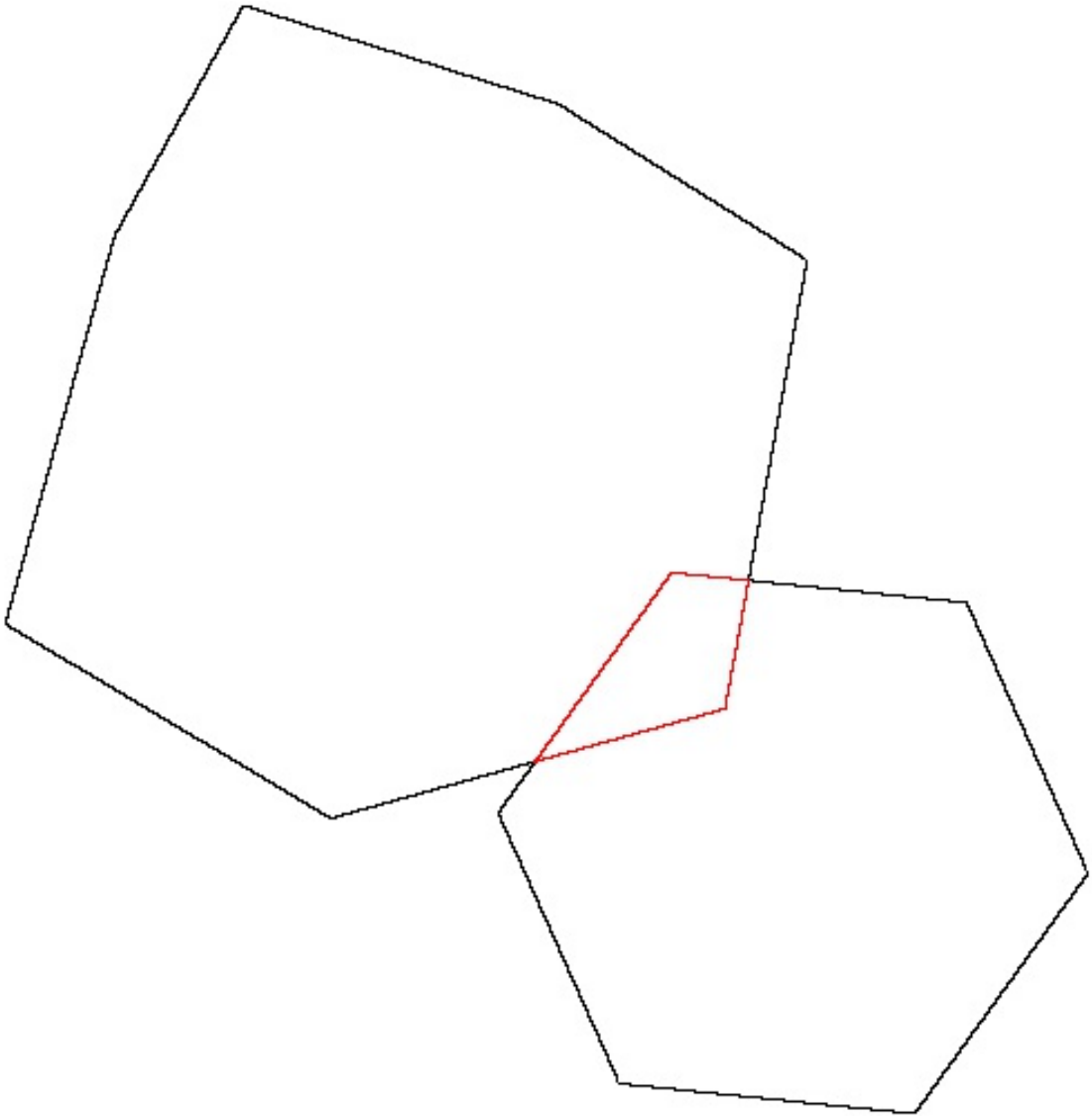
2. Color: A vector with three doubles representing the colors Red, Green, and Blue. With values from 0 to 1.
   It can be retrieved with the getcol() method.

Constructors

- Default: The hull is void and the color is black.
- Normal: With a vector of points, the convex hull will be computed and then saved. No option to indicate the color this way.
- Modifiers: The method setcol() lets you change the color of the polygon.
    - Precondition: The vector given must contain three doubles.

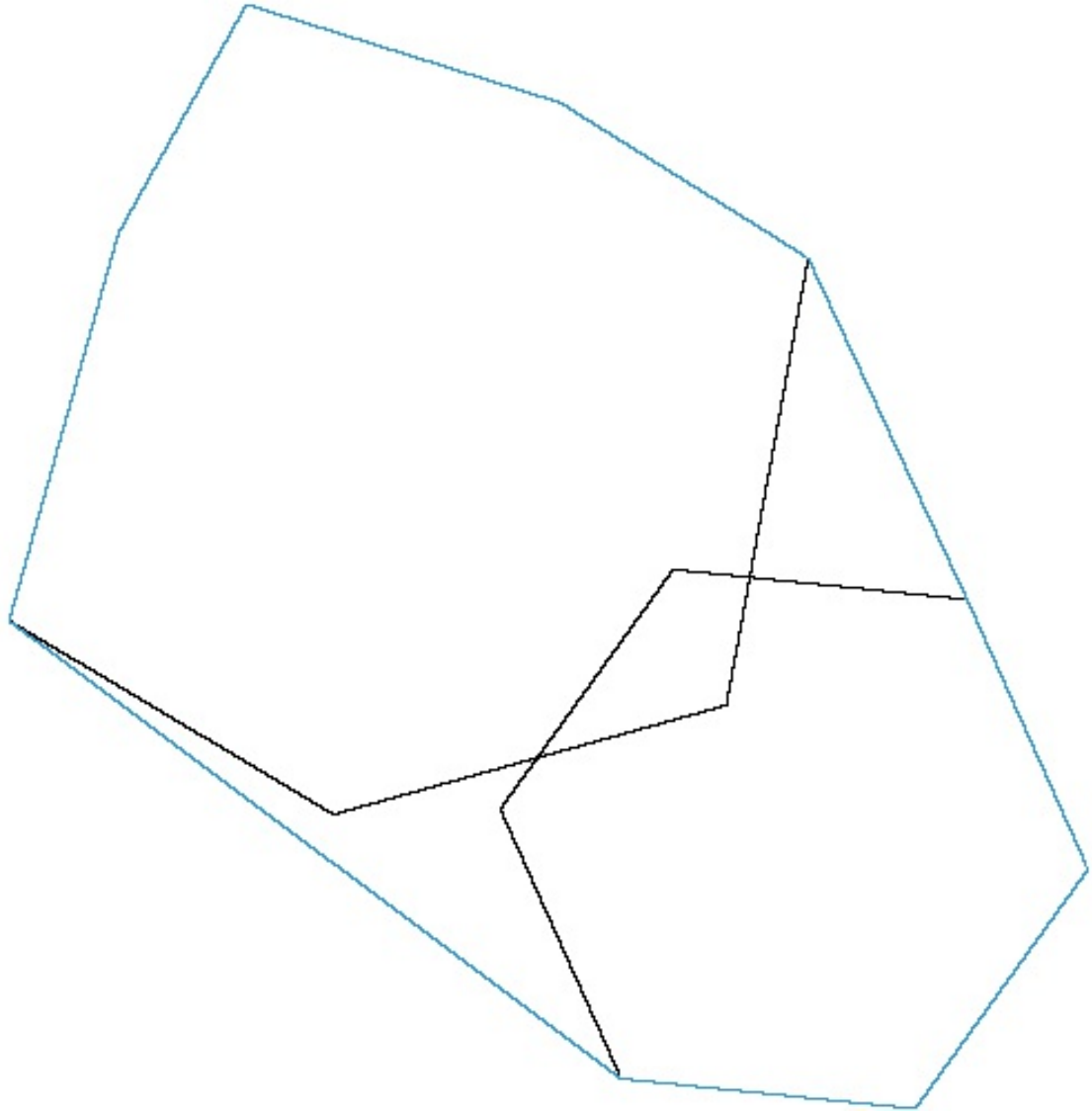
Public Methods and Operators

- Intersection: Represented by *. Returns the intersection of two convex polygons.
    - Cost: `O((m+n)log(m+n))`
    - Algorithm
      Compute the intersection points of both polygons
      Compute the vertices of each polygon that are inside the other polygon
      Call the constructor on those points.



- Union: Represented by +. Returns the convex union of two convex polygons.
    - Cost: `O((m+n)log(m+n))`
    - Algorithm

Just call the constructor of the class with all the points.



- Bounding Box(bbox): Returns a rectangle (convex polygon) that contains all the given convex polygons.
  - Cost: `O(n)`
  - Algorithm
    Compute the minimum and maximum x-y coordinates.
    From them create the LL, LR, UR, UL corners.
    Finally, call the constructor with those four points.
- isInside: Tells if a convex polygon is inside another.
  - Cost: `O(n log(m))`
  - Algorithm
    For each point check if it is inside the polygon. If one point is not inside, the program ends.
- num_vert: Returns the number of vertices.
- num_edges: Returns the number of edges.
- area / perimeter / centroid: Returns the area / perimeter / centroid.
- isRegular: Tells if a polygon is regular. With an uncertainty of 1%.

- Cost: `O(n)`
- Algorithm
  First, check if all the edges are equal.
  Then, if all the angles are the same.
  Everything is done with a tolerance of 1%, which means that two magnitudes are equal if they differ in less than 1% of its magnitude.

Private Methods and Static functions

All of the above is done using this functions:

- Min/Max: There is a whole set of functions to compute the maximum / minimum of the x/y-coordinate of many points, and return either the value or the index to the point with that coordinate.

  - Cost: `O(n)`
  - Algorithm
    Default algorithm to look for a maximum/minimum in any set given any order.

- Simple Polygon: Given a representation of a set of points, it changes the representation to be that of a polygon with does not cut into himself.

  - Cost: `O(n log(n))`
  - Algorithm
    First, move the point with the smallest x-coordinate to the beginning.
    After that, change the axis frame such that the latter point goes to the origin.
    Then, sort all the points by their slope.
    Finally, undo the change in the axis frame.

- Graham Scan: Given a set of points that represents a simple polygon, construct its convex hull.

  - Cost: `O(n)`
  - Algorithm
    Start in the point with smallest x-coordinate (which by the previous algorithm is the first one).
    Now, iteratively before adding any new point, check if the last added point forms an angle greater than 180 degrees with the surrounding vertices.
    If so, remove it. Otherwise, add the new point.

- Interior Points: There is a private method that check if a given point in inside a given convex polygon.

- Cost: `O(log(n))`
- Algorithm It is a divide & conquer algorithm. The base case for n=3 is done by checking if the point is to the left of the three sides. For any other n: Take the point in the middle. Consider the segment formed by the first point and the latter. There are two cases
  a. The point is at the right, so forget about the last n/2 points.
  b. The point is at the left, so forget about the first n/2 points.

  Finally, apply recursion over the remaining n/2 points.

- Interior Points(second version): Tells which points of a polygon are inside another polygon.

  - Cost: `O(n log(m))`
  - Algorithm
    Just apply the function above for every point.

- Intersection line-polygon: Computes the point(s) in which a line intersects a polygon. This one is a bit extense, the first part is an sketch, and the second is in detail.

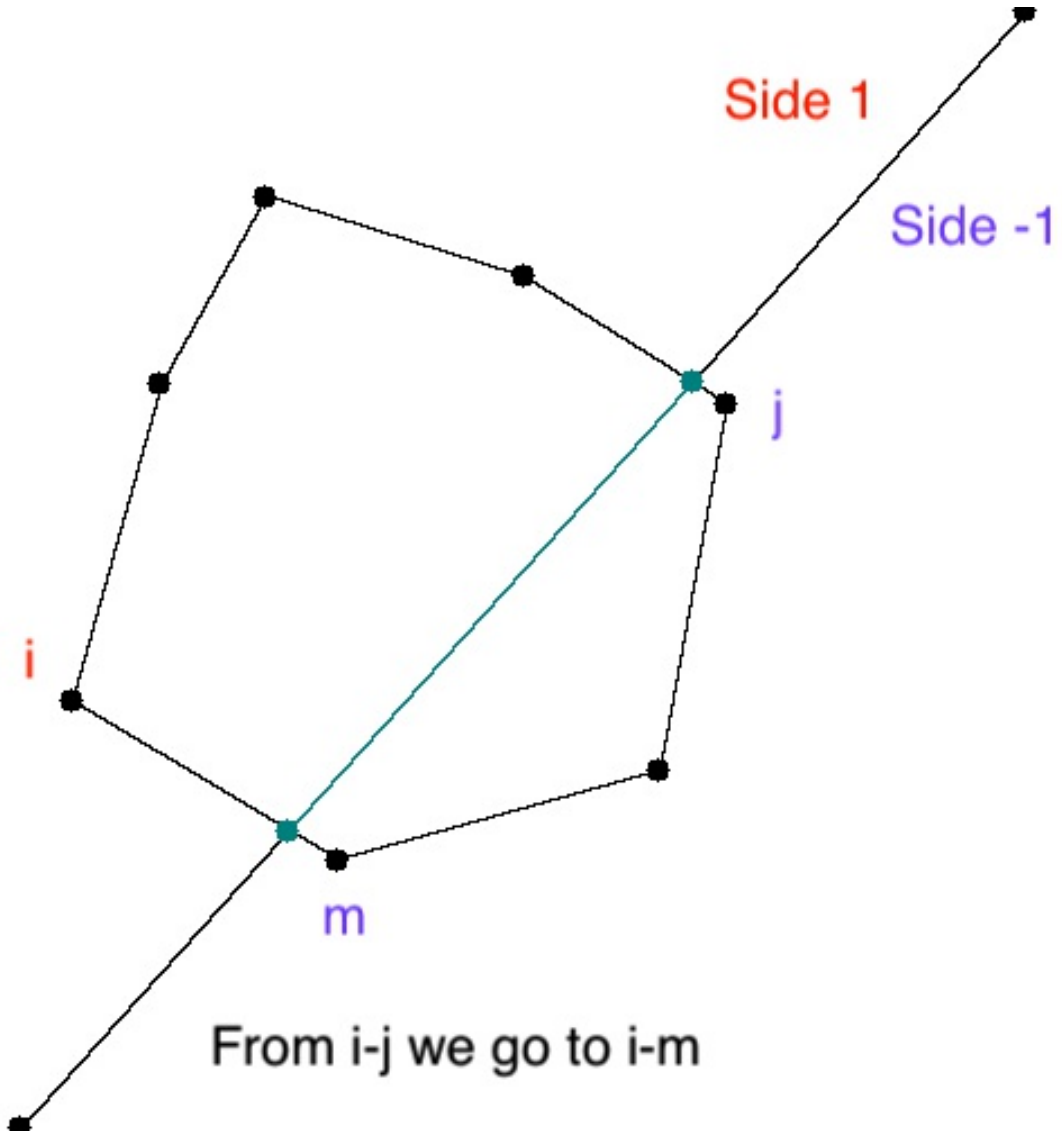
  - Cost: `O(log(n))`
  - Algorithm
    Base case: The polygon is a dygon, so the intersection can be done analytically using Cramer's rule.
    Case n: We need to find two consecutive points, one at each side of the line. In order to do that, again use divide & conquer to reduce the problem to one of size n/2.

    ```
    cut_points(P, Q, polygon T, i, j)
    ```

  Invariant: Those two point we look for are always between the index i and j.

  Inductive step: Let `m = (i+j)/2`, then either call the function between `[i..m]` or between `[m..j]`. Following a different criteria for each case:

  i. `T[i]` is in a different side of the line than `T[j]`, then `T[m]` is either in the same side as `T[i]` or as `T[j]`, therefore choose the one in the different side.
  ii. Both `T[i]` and `T[j]` are in the same side. If `T[m]` is in another side, both `[i..m]` and `[m..j]` are valid. Else, look at `T[m−1]` and `T[m+1]` to see in which way you are getting closer to the line and choose that way.



3. Polygon Calculator

This are the ones given by Jordi Petit, but they are listed again for clarification:

> `polygon` `print` `save` `load` `list`
>
> `area` `perimeter` `vertices` `centroid`
>
> `setcol` `draw` `frame`
>
> `intersection` `union` `inside` `bbox`

I added the command `frame`, to include the possibility to renderize an image with more resolution. But keep in mind that a frame bigger than 5000 x 5000 will be pretty slow.

Error handling

The possible errors are encoded with a number from 0 to 4. If one appears, something has gone wrong and isn't working as you would expect, the command won't terminate but maybe it will have done something you don't really want, so it is adviceable to restart the calculator.

0. Wrong number or type of arguments
1. Invalid polygon identifier
2. Not enough parameters
3. Invalid command
4. Wrong format

There are also two warnings, but are rarely used. If one appears, you can ignore them freely.

0. Not enough parameters
1. Nothing to show

Known bugs:

- The function `polygon`, won't give any error when one of the x-coordinates is not convertible to double, but it will add to the polygon the points mentioned before it.

Advice

The commands `save` and `draw`, will overwrite the given files. Don't overwrite any source code or needed image.

---

4. Tests

The Makefile provides some commands for checking some concrete features of the classes. For example, `make test_point` and `make test_conv` will check the methods for each class are well implemented. If they are executed, no output from those commands should be expected. If you want to use this commands to check your own tests, you'll need to modify the format of your tests.

For more details of this and other commands to check either the functionality or the efficiency, read the README.md in the folder `tests`.

---

5. Compilation

The command `make compile` will do all the work.

Know errors and fixes

> `Polygon_Calculator.cc:8:10: fatal error: 'pngwriter.h' file not found`
>
> `#include <pngwriter.h>`

If this or something similar shows up, it is because either you haven`t installed the library PNGwriter, or you have installed it in another place. If you have installed it, but don't remember where, do the following to fix the problem. First, check the flags for the compiler are the correct ones:

> `CXXFLAGS = -Wall -std=c++11 -O2 -DNO_FREETYPE -I $(HOME)/libs/include`
>
> `LIBS = -L $(HOME)/libs/lib -l PNGwriter -l png`

If they already are like that, you'll need to find TWO files in your computer and their respectives directories. The first is `libPNGwriter.a`. To find it, use this command

> `find / -name libPNGwriter.a 2>/dev/null`

It is going to last a few minutes, because it is looking in all your directories. Whenever a directory is printed like, for example: `/Users/087024/libs/lib/libPNGwriter.a` you can interrupt the process by doing `ctrl+c`. Now, next to the flag `-L` put the directory you found. In the example above it is `/Users/087024/libs/` and you should put:

> `LIBS = -L` *`/Users/087024/libs/lib/`* `-l PNGwriter -l png`

The other file you need to find is `pngwriter.h`. Again use `find` to find it, like this:

> `find / -name pngwriter.h 2>/dev/null`

The new directory you found goes with the flag `-I`. For example, if the directory is `/Users/087024/libs/include/`, you should put:

> `CXXFLAGS = -Wall -std=c++11 -O2 -DNO_FREETYPE -I` *`/Users/087024/libs/include`*

If none of this worked, reinstall the package PNGwriter in the folder `$HOME/libs` and leave the flags as they were in the beginning.

If you don't know how to install PNGwriter, Jordi Petit has a fantastic tutorial in his github.

---

6. Makefile

The command `make` shows all the available commands, which are:

- compile: Compiles the class Point, ConvexPolygon, and the Polygon_Calculator
- use: Executes the Polygon_Calculator.exe program
- test: Check the Polygon_Calculator with some test cases
- test_time: Shows the time it needs to execute each method for different sizes.
- test_point: Check the class Point. For more detail go to the README.md in the folder tests.
- test_conv: Check the class ConvexPolygon. For more detail go to the README.md in the folder tests.
- open_point: Opens the files of the tests of the class Point. Only valid for visual Studio Code. (Change `code` for `geany` if you are using Linux)
- open_conv: Opens the files of the tests of the class ConvexPolygon. Only valid for visual Studio Code. (Change `code` for `geany` if you are using Linux)
- open: Opens the files of the tests of the Polygon_Calculator. Only valid for visual Studio Code. (Change `code` for `geany` if you are using Linux)
- clean: Removes all the `.o`, `.exe`, and `.out` files.

---

7. Additional notes and Clarifications

Uncertainties

In many functions the uncertainty imposed by Jordi Petit is too low. I decided to change it to one a bit more flexible. My uncertainties are measured in percentages, this means the margin depends on the magnitude of the value. This way, for bigger numbers, there is a bigger absolute margin, and smaller for small values. This method is similar to the one physics use, and it is because is more trustful.

Why the clockwise() method seems counterclockwise?

Because this is the way things should be. Having our clocks move in the negative sense of rotation while the times moves forward is absurd. But, the occidental culture is built upon history, not science and math. That is why you think my function clockwise has the wrong name. To support that this view is not unique, follow this link.

Notation in the algorithm explanation

The name of the variables is asigned following this rule:

`n` is associated to the number of vertices of the first polygon, and `m` to the second.

LL: lower left
LR: lower right
UR: upper right
UL: upper left

Proofs of correctness

It isn't provided any proof of the correctness of any algorithm although they are not obviously correct. If you don`t believe it, try some difficult cases, but keep in mind the ones that are provided are quite general and difficult.

Text format

The function `load`, only accepts `.txt`, `.inp`, or `.dat` files, but it can be modified to accept any kind of file.

Tips for reading the Code

Start from the bottom. The code is written in a way that explains himself(almost).

When reading an algorithm, first read the comments and then check the program does what it says.

Remember this notation: T stands for polygons or vectors of points; P, Q and other uppercase letters stand for points.

The static functions are placed immediately above of where they are used. Or if they are used by many other functions they are at the top.

Min/Max functions are copy-paste of the first one, so there is no need to read in detail each of one. Disgracefully, there are many function along the code that exhibits this feature, but I couldn't do better because although they are very similar, I didn't find a way to encapsulate them even more.