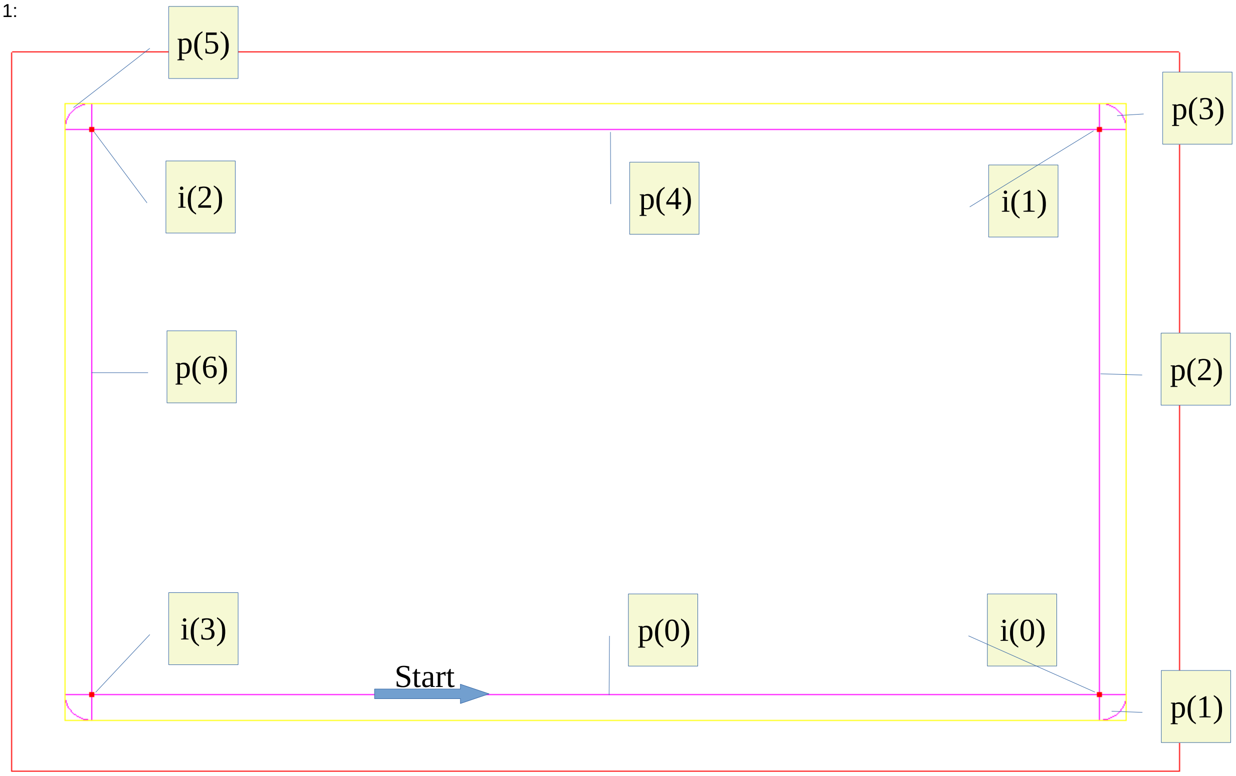
Grotius Pocket Algoritme Approach, Designed and written by a linux user, Github Grotius-CNC

Date: 25-06-2020

Preproccesing:	Split up all primitives as lines, arc’s, that are passing intersection points to primitives beween Intersection points, the list must be ordered.	
Given data:	1.Primitive numbers (p), a primitive can be a line, arc, linestrip, spline etc	
	2.Primitive end intersection (I)	
Algoritme rules:	1. Incrementing a sector number, may only be incremented to a number not containing the closed sector list	
	2. Decrementing a vector number, must be decremented to a number not containing the closed sector list	
	3.A sector is closed when the primitive end intersection is seen for the second time, the I number is send to The cs list	
	4. When a primitive has a end intersection, This is marked in the Action (a) list as S++ (sector increase) or S-- (sector decrease)	
	5. When a Intersection (I) is spotted for the first time, The action is S++. When a intersection (I) is spotted for the second time, the action is S--	
C++ input data :	std::vector<std::vector<int>> id;	2d container that holds the Primitive number and the primitive end intersection
	id[index][0] = primitive (p)	The first dataplace will be the primitive (p) number
	id[index][1] = primitive end intersection (I)	The second dataplace will be The primitive intersection (I) number
C++ output data :	std::vector<std::vector<std::vector<int>>> od;	3d container that holds the Area, the sector number, including their primitives
	od[index][0] = area (s)	The first dataplace will contain the Area of the sector
	od[index][0] = sector (s)	The second dataplace will contain the sector number (s)
	od[index][1] = primitive (p)	The thirth dataplace will be the primitive (p) number
C++ closed sector data :	std::vector<int> cs;	Closed sector list.
C++ variable	int a;	Action, 0=no action, 1=increase, 2=decrease
	int s;	Sector, will hold the current sector

Example 1:



Primitives (p)	Primitive end intersection (I)	Sector (s)	Action (a)	Closed Sectors (cs)
0	0	0	s++	
1	0	1	s--	1
2	1	0	s++	
3	1	2	s--	2
4	2	0	s++	
5	2	3	s--	3
6	3	0	s++	
7	3	4	s--	4

The color flags are defining the steps to do at each row:

Solution :

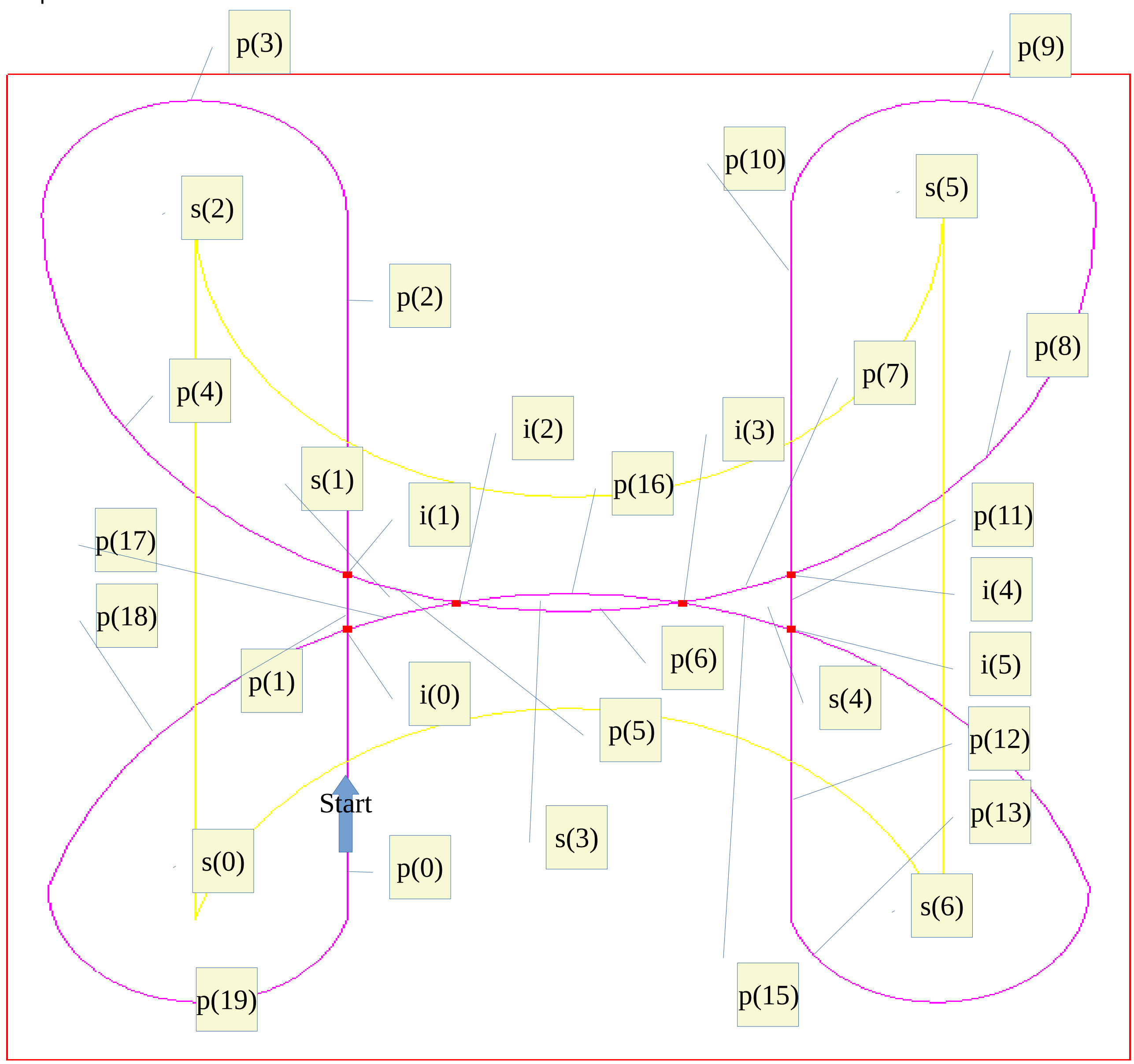
Sector (s)	Primitives (p)	Area (a)	Cw	Ccw
0	0,2,4,6	<0		x
1	1	>0	x	
2	3	>0	x	
3	5	>0	x	
4	7	>0	x	

Comment:

At p(0) we have a end intersection I(0), when we spot a new, unused intersection as I(0), we increase the sector incremented by 1 with a unused sector number. If the sector number was used before, we increment until we have a unused sector number.

At p(1) we spot a already used i(0), so we decrease. We look in the closed sector list, we can not use cs(1), this was used before, We decrease to 0.

Example 2:


p(3)
p(9)
p(10)
s(5)
s(2)
p(2)
p(8)
p(7)
p(4)
i(2)
i(3)
s(1)
p(16)
i(1)
p(11)
p(17)
i(4)
p(18)
p(6)
i(5)
p(1)
i(0)
s(4)
p(5)
p(12)
Start
p(13)
s(3)
s(0)
p(0)
s(6)
p(19)
p(15)

Primitives (p)	Primitive end intersection (I)	Sector (s)	Action (a)	Closed Sectors (cs)
0	0	0	s++	
1	1	1	s++	
2		2		
3		2		
4	1	2	s--	2
5	2	1	s++	
6	3	3	s++	
7	4	4	s++	
8		5		
9		5		
10	4	5	s--	5
11	5	4	s++	
12		6		
13		6		
14	5	6	s--	6
15	3	4	s--	4
16	2	3	s--	3
17	0	1	s--	1
18		0		
19		0		

Comment:
At the line p(6), we see that from the Action (a) we have to increment the section. But 2 already excists in the cs list, so we increment To the value 3. This is quite a tricky one to spot. See the yellow flags.

At the line p(15) we have to decrement the section from value 6 down. Value 5 is in the cs list, so we Decrement to value 4. See the green flags.

Solution :

Sector (s)	Primitives (p)	Area (a)	Cw	Ccw
0	0,18,19	<0		X
1	1,5,17	>0	X	
2	2,3,4	<0		X
3	6,16	<0		X
4	7,11,15	>0	X	
5	8,9,10	<0		X
6	12,13,14	<0		X