Records of Computational Graphic Project

a problematic yet dogged approach

Student Name: Jia Yanxin
Student ID:3180105708
Date of Submission:08/09/2021

Data Source and Introduction

To automatically generate the borders of my graph, I have referred to MATLAB tools such as *bwperim*, which uses specific matrix operators to extract border pixels. To simplify the process afterwards, I set the connectivity of pixels to 4, which means any extracted pixel is connected to the closed border on its 4 sides. The extraction result, saved in matrix form, is shown in figure 1.

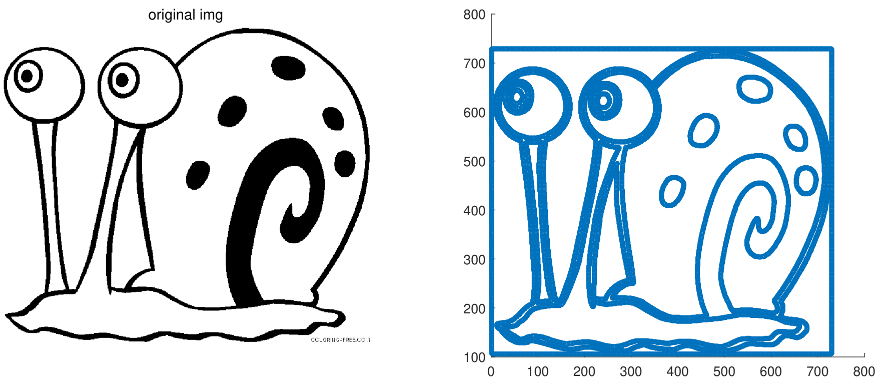


Figure 1: The preprocessed borderline

1 Interpolation and inclusion relations

To separate different closed curves from the extracted border pixel, I used Python (which I am more familiar with) to generate the results.

```
M=[]
 (a,b)=pic.shape
 def search(i,j,M):
     flag=0
     for M_str in M:
         if (i,j) in M_str:
             flag+=1
     return flag
 def search2(i,j,M):
     flag=0
     if (i,j) in M:
             flag+=1
     return flag
```

```python
def decide(i,j,m_str,array_ind):
    if (i,j) in array_ind and not search2(i,j,m_str):
        return True
    else:
        return False

def neighbor(i,j,x,y):
    if (x==i-1 and y==j) or (x==i+1 and y==j) or (x==i and y==j-1) or (x==i and y==j+1)\
    or (x==i-1 and y==j-1) or (x==i-1 and y==j+1) or (x==i+1 and y==j+1) or (x==i+1 and y==j-1):
        return True
    else:
        return False
def closure(i,j,m_str):
    l=len(m_str)
    flag=0
    for k in range(0,int(l/2)):
        print(l)
        if neighbor(m_str[k][0],m_str[k][1],i,j):
            flag+=1
    if flag:
        return True
    else:
        return False

for (i,j) in array_ind:
    #dot=pic[i][j]
    if not search(i,j,M):
        #a closed curve not recorded
        m_str=[(i,j)]
        #the form of following tab is [False, True, False, True]
        while 1:
            dic=dict({0:(i-1,j),1:(i,j+1),2:(i+1,j),3:(i,j-1)})

            lst=[decide(i-1,j,m_str,array_ind),decide(i,j+1,m_str,array_ind),\
                decide(i+1,j,m_str,array_ind),decide(i,j-1,m_str,array_ind)]

            if lst.count(True):
                (i,j)=dic[lst.index(True)]
                m_str.append((i,j))

            elif lst.count(True)==0 and not closure(i,j,m_str):
                print((i,j))
                (i,j)=m_str[-2]
                print((i,j))

            elif lst.count(True)==0 and closure(i,j,m_str):
                M.append(m_str)
                print(0)
                #print(m_str)
                break
```

Myfile

1.1 Cubic Interpolation

I select interpolation knots from the border pixels based on constant distance $d = 30$. To represent closed curves more accurately, the x-and y-coordinate were separately interpolated as functions of $t \in [0, 1]$. I used python.matplotlib to generate graph of interpolation, which you may see in here or by running the program.

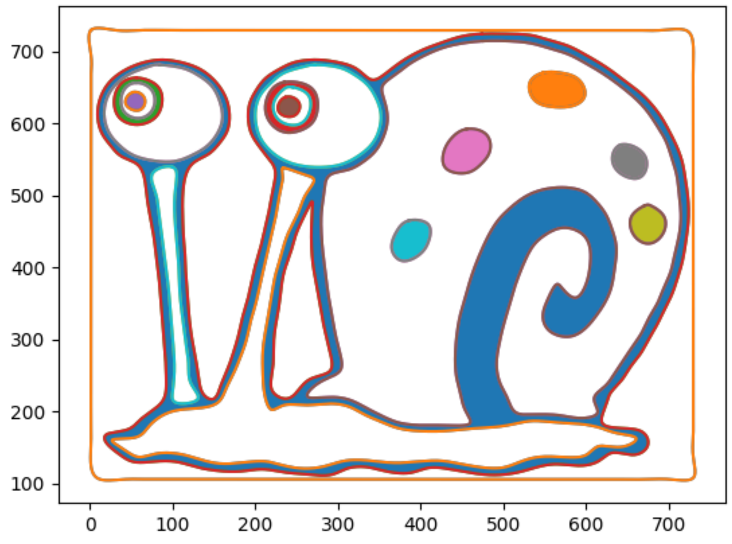


Figure 2: The filled graph after interpolation.

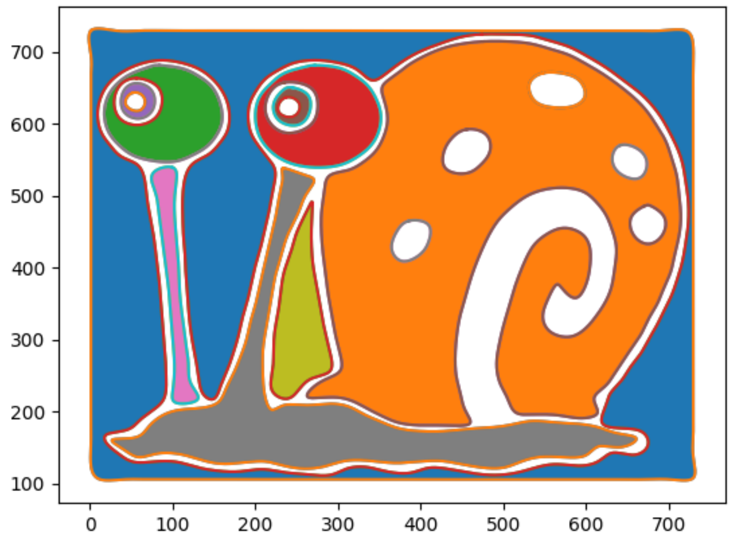
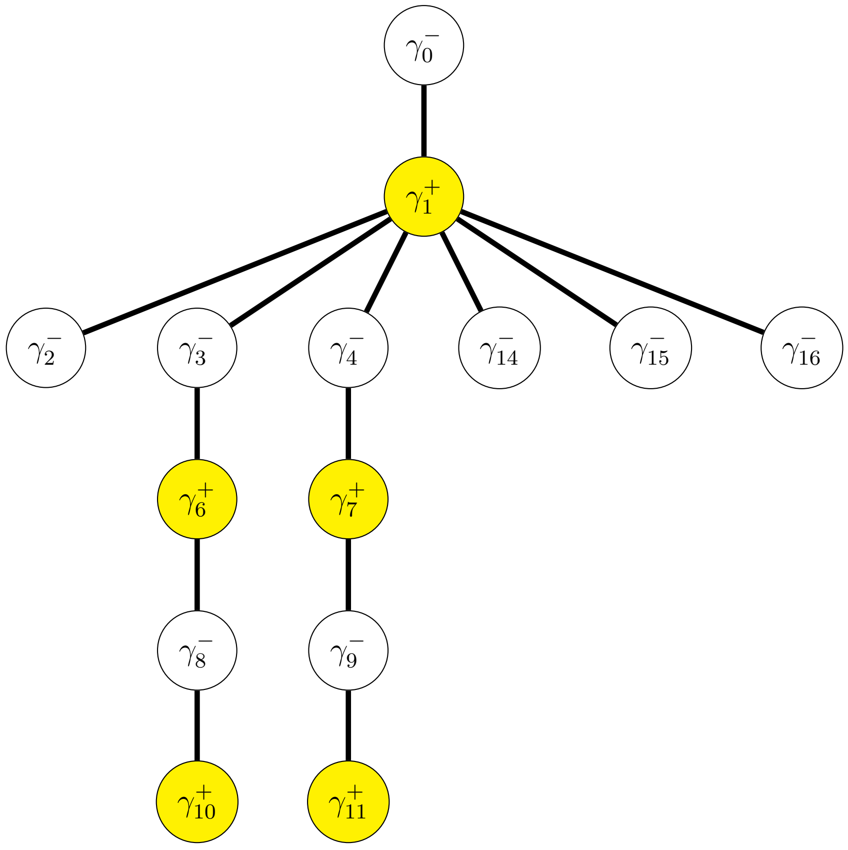
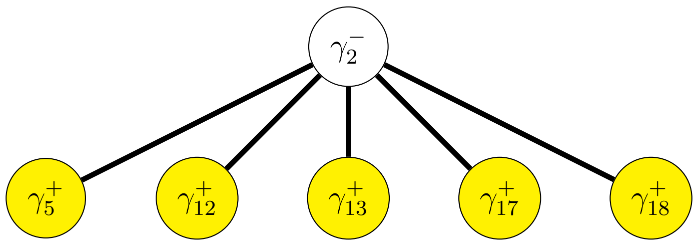


Figure 3: The filled graph after interpolation(reversed).

1.2 Inclusion Relation

Because my set contains 19 curves in total and would look horrendous if all displayed in one Hasse diagram, the following contain disintegration of the overall picture into smaller structures.





(γ_0 is negatively oriented and would not affect the total modelling result of this shape, but for safety reasons, we consider it as $\mathcal{J}_1$.) The realizable spadjor is $\mathcal{J} = \cup_{k=1}^{11}$,with atom spadjors as $\mathcal{J}_1 = \{\gamma_0\}$,$\mathcal{J}_2 = \{\gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_{14}, \gamma_{15}, \gamma_{16}\}$,$\mathcal{J}_3 = \{\gamma_6, \gamma_8\}$,$\mathcal{J}_4 = \{\gamma_7, \gamma_9\}$,$\mathcal{J}_5 = \{\gamma_{10}\}$,$\mathcal{J}_6 = \{\gamma_{11}\}$,$\mathcal{J}_7 = \{\gamma_5\}$,$\mathcal{J}_8 = \{\gamma_{12}\}$,$\mathcal{J}_9 = \{\gamma_{13}\}$,$\mathcal{J}_{10} = \{\gamma_{17}\}$,$\mathcal{J}_{11} = \{\gamma_{18}\}$.Except for $\mathcal{J}_1$, all other atom spadjors are of type $\mathcal{J}^+$.
If correctly run, the result of the program would resemble figure 5,6,7.
The matrix M in figure ref1 indicates which curve includes which. If $M(i,j) = 1$, then it means curve γ_j is included by curve γ_i. The values on column j indicates how many curves, and which ones, are including γ_j, making it easy to be translated into Hasse diagrams. Figure 6 displays which curves are included by the curve γ_i. Figure 7 displays the orientation of curves, calculated with shoelace formula. All numbering of curves correspond to that in the Hasse diagram. Due to precision problems, I failed to solve the inclusion relation by solving cubic equations, thus I extracted points of enough density from the border of each interpolation curve to derive point-to-polygon inclusion relation. I used the traditional method of drawing one line segment from the dot and count the number of intersection between the line segment and the

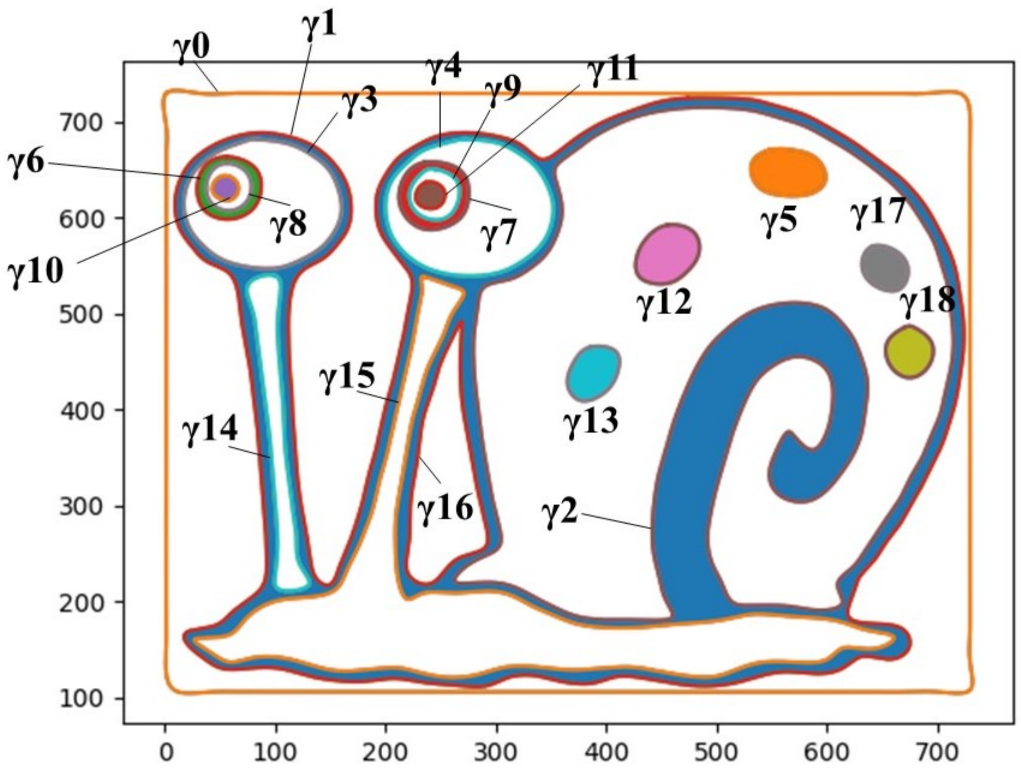
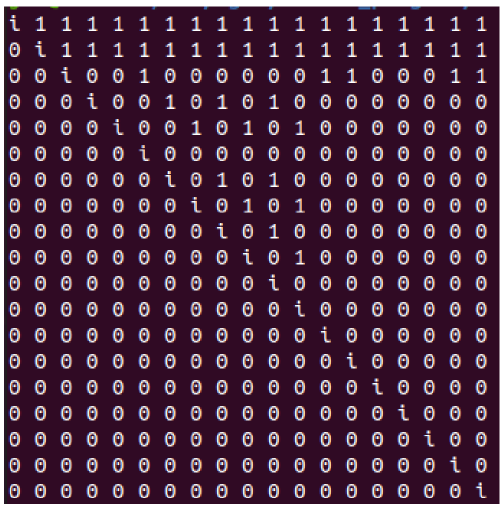
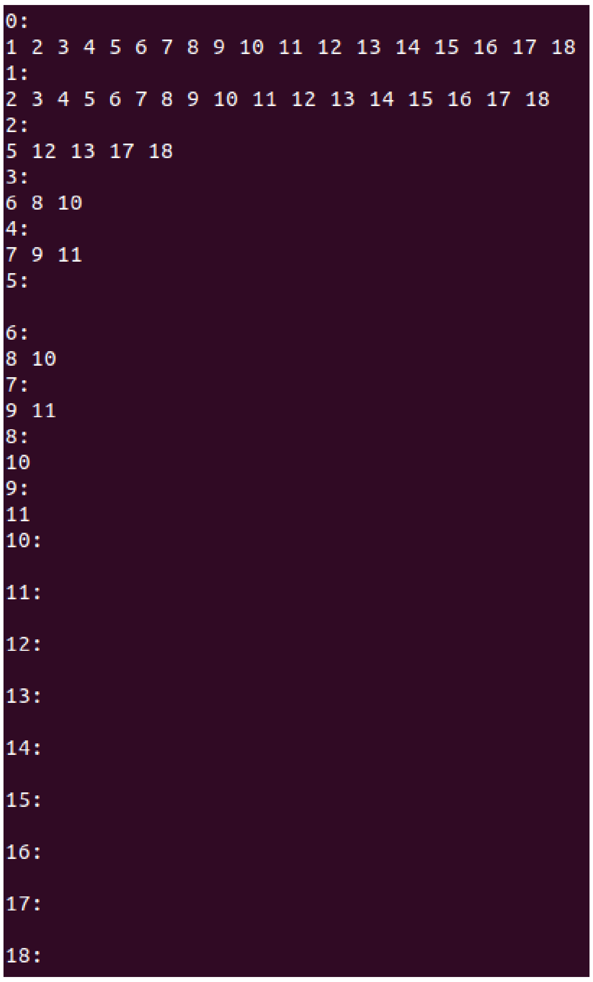


Figure 4: The filled graph after interpolation, represented by realizable spadjor. I did not draw the orientation of each curve since it would not look very pretty in this messy pic. The numbering of curves is just the same as that in Hasse diagrams, which may be of reference.



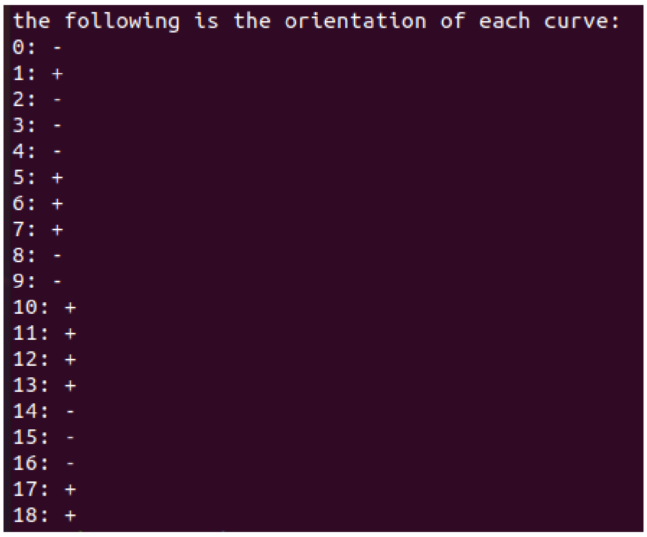
```
i 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 i 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 i 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1 1
0 0 0 i 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0
0 0 0 0 i 0 0 1 0 1 0 1 0 0 0 0 0 0 0
0 0 0 0 0 i 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 i 0 1 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 i 0 1 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 i 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 i 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 i 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 i 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 i 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 i 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 i 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 i 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 i 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 i 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 i
```

Figure 5: The filled graph after interpolation.



```
0:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
1:
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
2:
5 12 13 17 18
3:
6 8 10
4:
7 9 11
5:

6:
8 10
7:
9 11
8:
10
9:
11
10:

11:

12:

13:

14:

15:

16:

17:

18:
```

Figure 6: The filled graph after interpolation.



```
the following is the orientation of each curve:
0: -
1: +
2: -
3: -
4: -
5: +
6: +
7: +
8: -
9: -
10: +
11: +
12: +
13: +
14: -
15: -
16: -
17: +
18: +
```

Figure 7: The filled graph after interpolation.

polygon in question on one side of the dot (in this case, the right side). Since there are no overlapping curves in input data, if polygon A includes more then 50% of the edge points of polygon B, I regard that $A > B$.
The form of input is .txt file with its each line constituting of the four parameters of interpolation ($x/y = a + bt + ct^2 + dt^3, t \in [0, 1]$) for x and y coordinates each. You may input all curves in the same file and separate the curves with "#", in which case one should activate the line "pyread()" in "main()" to separate the curves into different .txt files. **NOTE: PLEASE DON'T DELETE RESULT.TXT** because it is the input in "inclusion". "pyread()" takes result.txt and separates them into curve_(x/y)i.txt, which may be read directly by the main program.

2 Boolean Algebra of Yin Sets

This part is largely based on [2], thus I would not give redundant details about its realization. In a word, one first extract all intersected segments, then group them together to make out intersection regions; the complement operation plots the curves backward, while the union operation first extract the intersected segments of both complements then reverse them and group them together.
To successfully run this (ragged) package, you should have **Python3.8 (along with matplotlib) installed in your system and accessible to g++**. I have not learned how to pack them together yet. And, before running any program(intersection/union/complement) in this pack, one should first check if there exist a result.txt in the folder. "result.txt" is for saving the data to be plotted. If the "result.txt" remained from the last run of this package, be **SURE to DELETE RESULT.TXT** before starting. Sorry for the inconvenience, but I am really quite unfamiliar with these operations.
The validity of this package is demonstrated by some graphs contained in the folder and (some) shown below in figure 8,9,10,11. If you wish to test it on other data, be sure to change the "num1" and "num2" on head of each .cpp file into the actual number of curves.

References

[1] Computational Geometry Algorithms and Applications, Mark de Berg et al., Third Edition.

[2] BOOLEAN ALGEBRA OF TWO-DIMENSIONAL CONTINUA WITH ARBITRARILY COMPLEX TOPOLOGY, Q. Zhang and Z. Li, MATHEMATICS OF COMPUTATION, Volume 89, number 325.

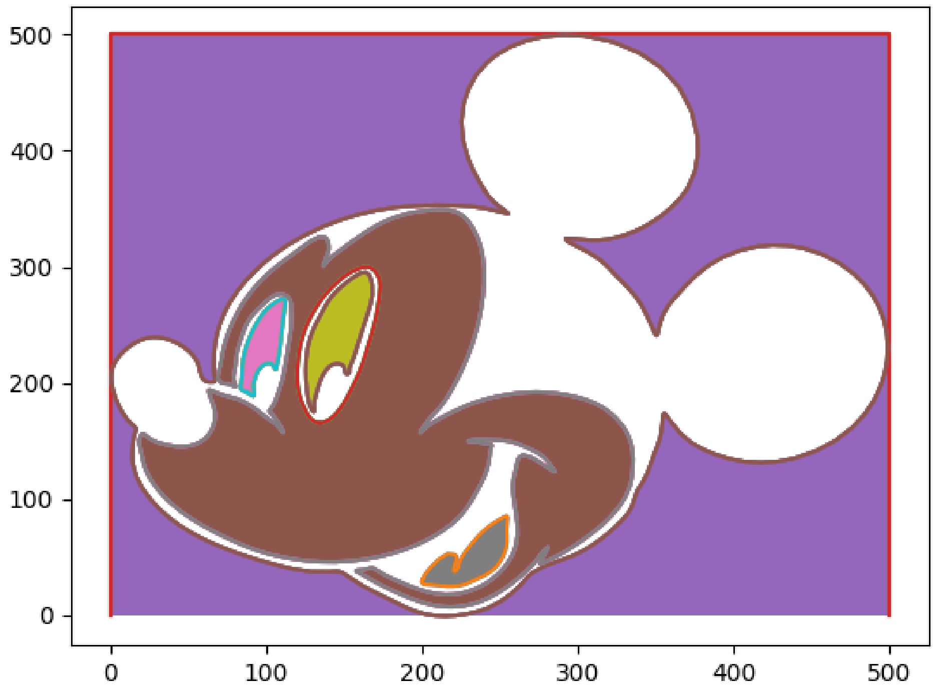


Figure 8: Example for complement operation.

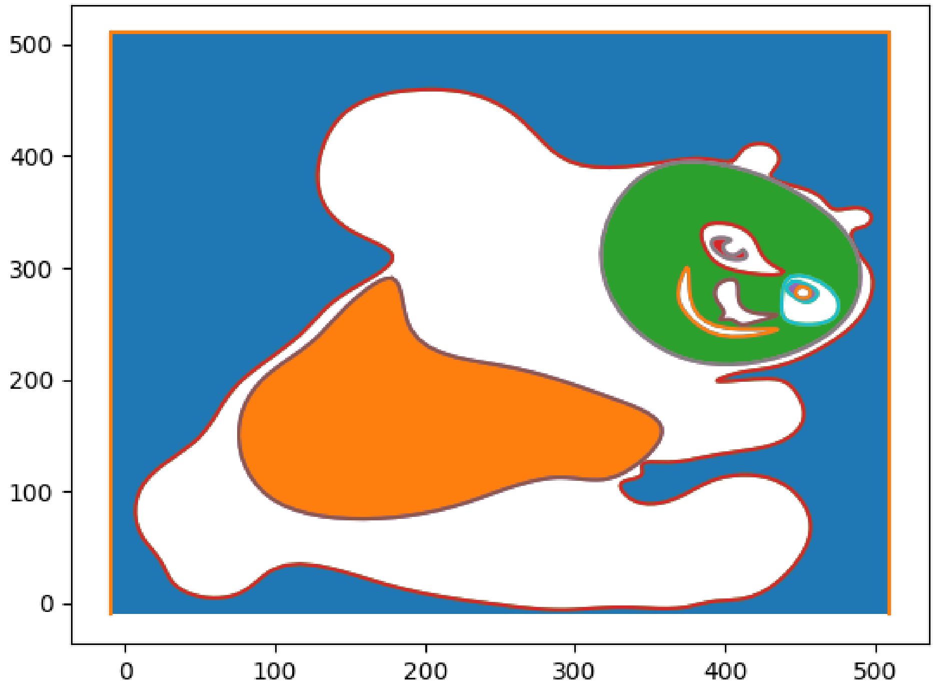


Figure 9: Example for complement operation.

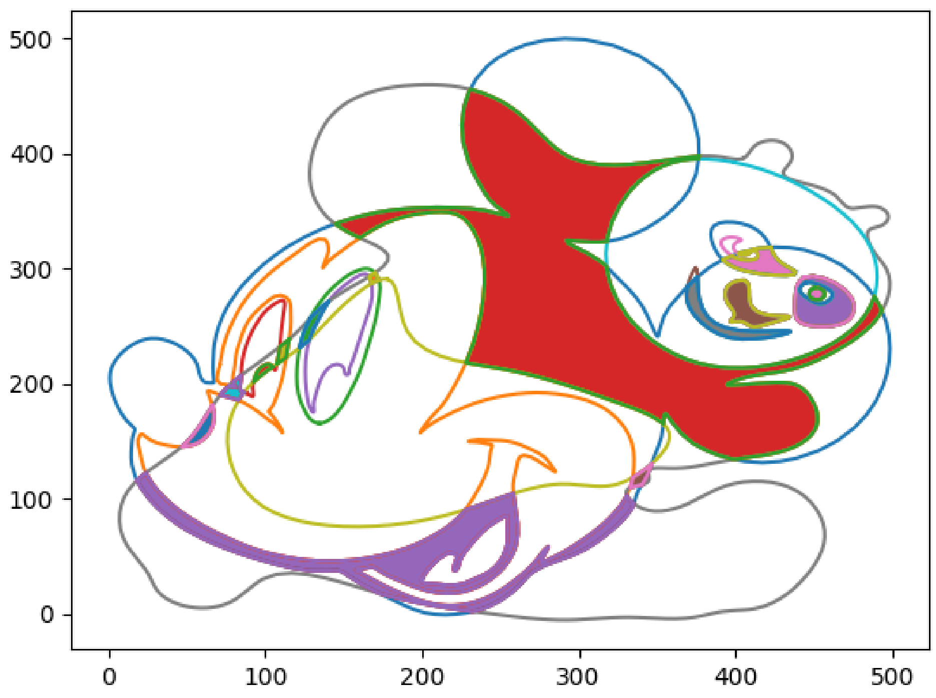


Figure 10: Example for intersection operation.

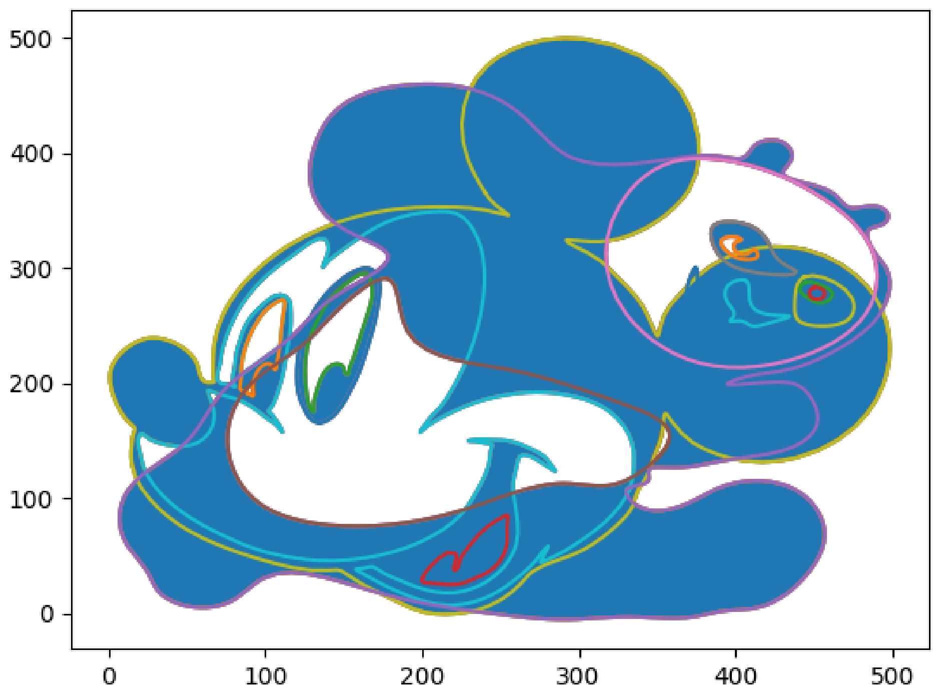


Figure 11: Example for union operation.