QUADTREE PROJEKT

24 MAJA

IO

Autor: Bartłomiej Filipowicz

Algorytm otoczki wypukłej + quadTree

Do stworzenia otoczki wypukłej wykorzystałem bibliotekę scipy oraz numpy, a żeby narysować otrzymany efekt wykorzystałem bibliotekę matplotlib. Tak się prezentuje kod tworzenia samej otoczki:

```
rng = numpy.random.default_rng()
points = 5 * rng.random((15, 2))    # 30 random points in 2-D
hull = ConvexHull(points)
```

Tak natomiast prezentuje się fragment odpowiedzialny za narysowanie otoczki:

```
# plot
plt.plot(points[:,0], points[:,1], '#000000') # insert '#000000' if you
# ^^^ insert 'o' if you want to have colorful dots
for simplex in hull.simplices:
    plt.plot(points[simplex, 0], points[simplex, 1], 'k-')
```

Wypełnienie otoczki załatwiłem jedną linijką:

```
#filling
plt.fill(points[hull.vertices,0], points[hull.vertices,1], '#000000')
```

Omówię teraz sam algorytm quadTree.

Algorytm ten zrealizowałem przy użyciu stworzonej przeze mnie klasy QuadTree oraz biblioteki PIL, dzięki której uzyskałem strukturę danych z dostępem do każdego pojedynczego piksela z obrazka wejściowego.

Tak wygląda konstruktor tej klasy(w obiekcie pix są przechowywane wszystkie piksele):

```
class QuadTree:
    """A class implementing a quadtree."""

    def __init__(self, fileName):
        """Take an image file name as an argument.

        I will conduct the quadTree algorithm by operating on single pixels.
        It will work by implementing recursion on it.
        The image will be divided recursively into four smaller parts(squares) on the
        as a result of the convex hull program (otoczka.py).

        """

        self.im = Image.open(fileName) # Can be many different formats.
        self.pix = self.im.load() # loading pixels of an image
        print(self.im.size)  # Get the width and hight of the image for iterating ove
        # set north west, north east, south west, south east for the computing area
        # I do not want to have the quadTree over the entire image, only where the fi
        self.nw = [81,59]   # north west of an image
        self.ne = [575,59]  # north east of an image
        self.sw = [81,426]  # south west of an image
        self.se = [575,426] # south east of an image
```

„To był świetny projekt do zrealizowania. Doskonale się przy nim bawiłem!”

W destruktorze jest zapisywany wynik działania algorytmu jako plik graficzny:

```
def __del__(self):
    print('Destructor called.')
    self.im.save('quadtree.png')  # Save the modified pixels as .png
```

Nie wkleiłem kodu metody divide(), ponieważ nie zmieściłby się on na stronie(odsyłam do kodu źródłowego). Jest to najważniejsza funkcja w całym programie. Działa w ten sposób, że na początku sprawdza czy mam do czynienia z jednym pikselem. Jeśli tak, funkcja kończy działanie. Jeśli nie, iteruję po wszystkich pikselach danego podkwadratu, dopóki nie trafię na co najmniej jeden piksel biały i czarny. Robię tak, ponieważ quadTree ma działać tylko na brzegach figur, a nie na całych figurach. Jeśli powyższy warunek jest spełniony, to wywołuję funkcję draw(), a następnie rekurencyjnie wywołuję cztery metody divide() dla odpowiednich kwadratów(pamiętając, że linie siatki zajmują jeden piksel). Po wywołaniu tych czterech metod kończę działanie funkcji instrukcją return. Jest to bardzo ważne, ponieważ na początku zapomniałem dodać tę instrukcję i program bardzo długo się wykonywał(jest wtedy wywoływanych bardzo wiele funkcji divide()).
Tak wygląda metoda draw():

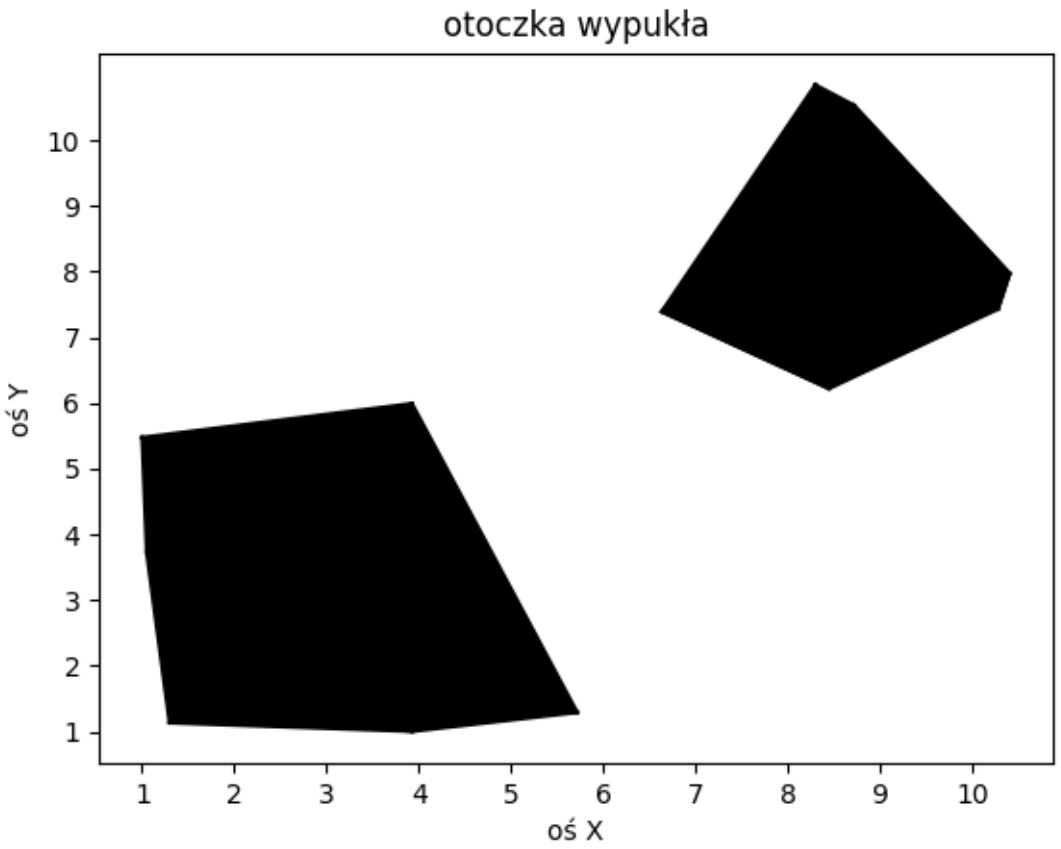
```
def draw(self, nw, ne, sw, se):
    """Draw a representation of the quadtree on the image."""

    imgLine = ImageDraw.Draw(self.im)
    imgLine.line([ (int(((ne[0] - nw[0]) / 2 ) + nw[0] ), ne[1]) , (int(((ne[0] - nw[0]) / 2 ) + nw[0] ), se[1]) ],
    imgLine.line([ (nw[0], int(((sw[1] - nw[1]) / 2 ) + nw[1] )) , (ne[0], int(((sw[1] - nw[1]) / 2 ) + nw[1] )) ],
```

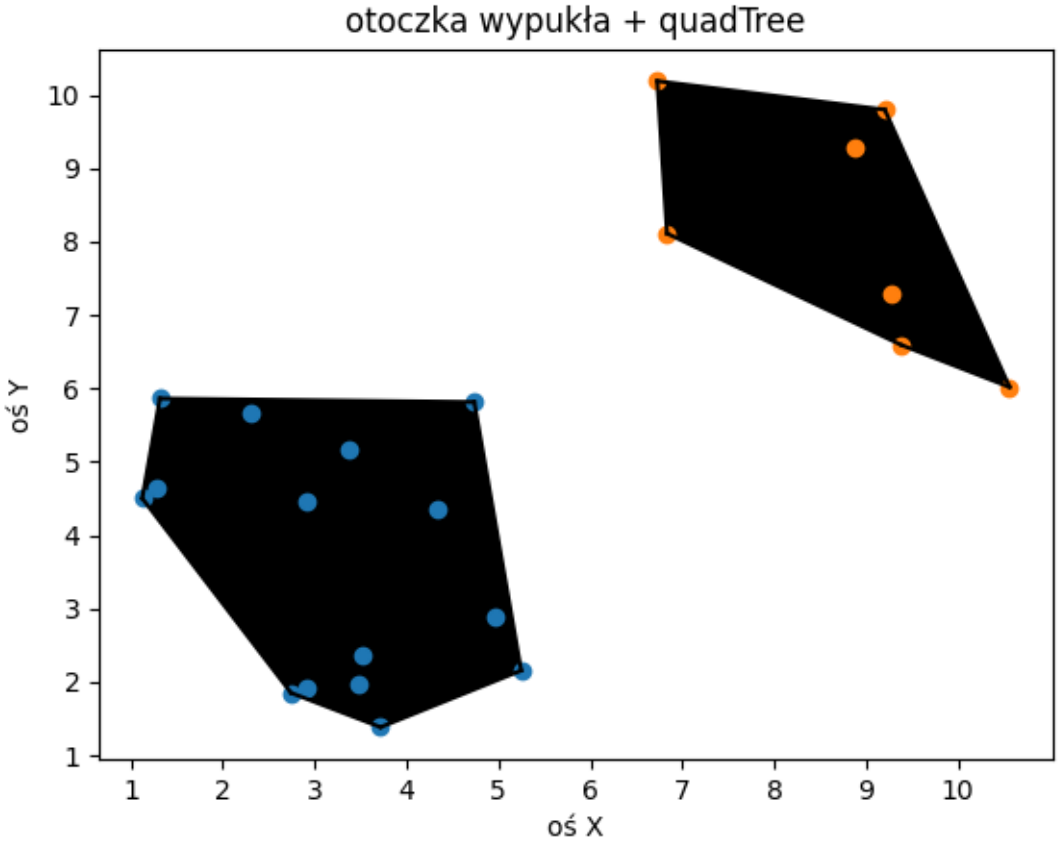
Sam program główny jest bardzo schludny i przejrzysty:

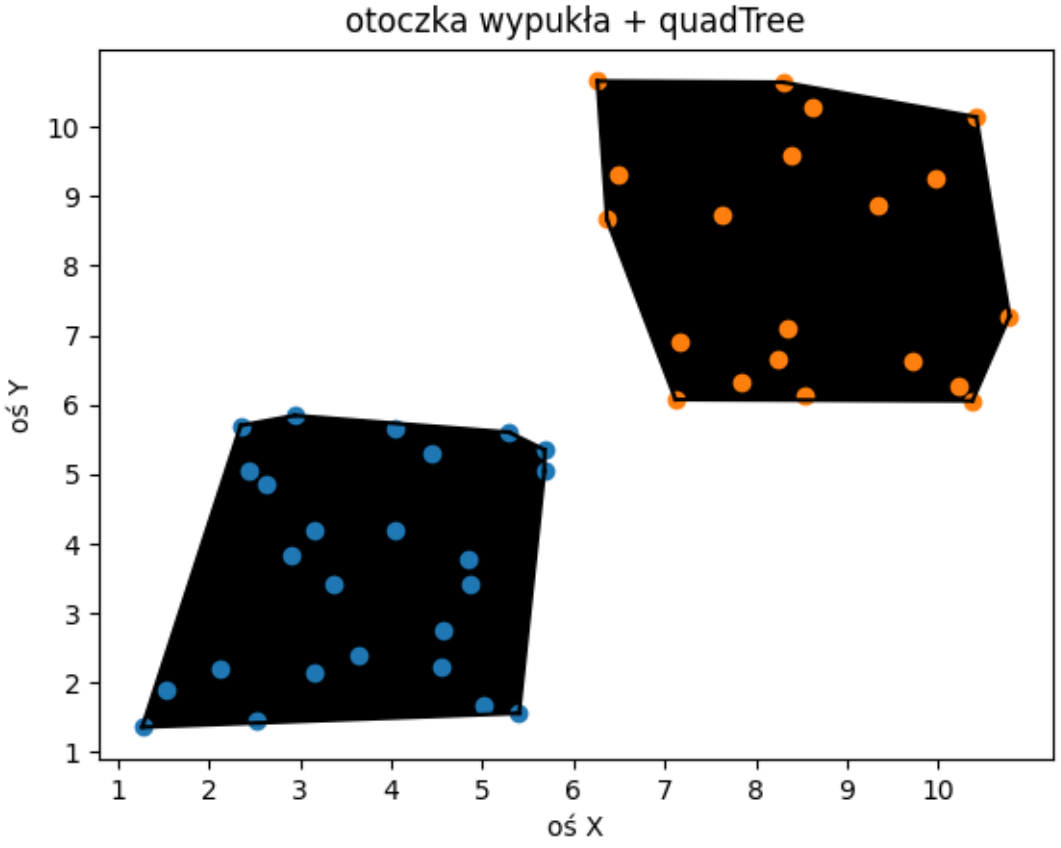
```
n.py > ...
from quadtree import QuadTree

myfileName = 'wykres.png' # input image file

qtree = QuadTree(myfileName)

qtree.divide(qtree.nw, qtree.ne, qtree.sw, qtree.se)
```

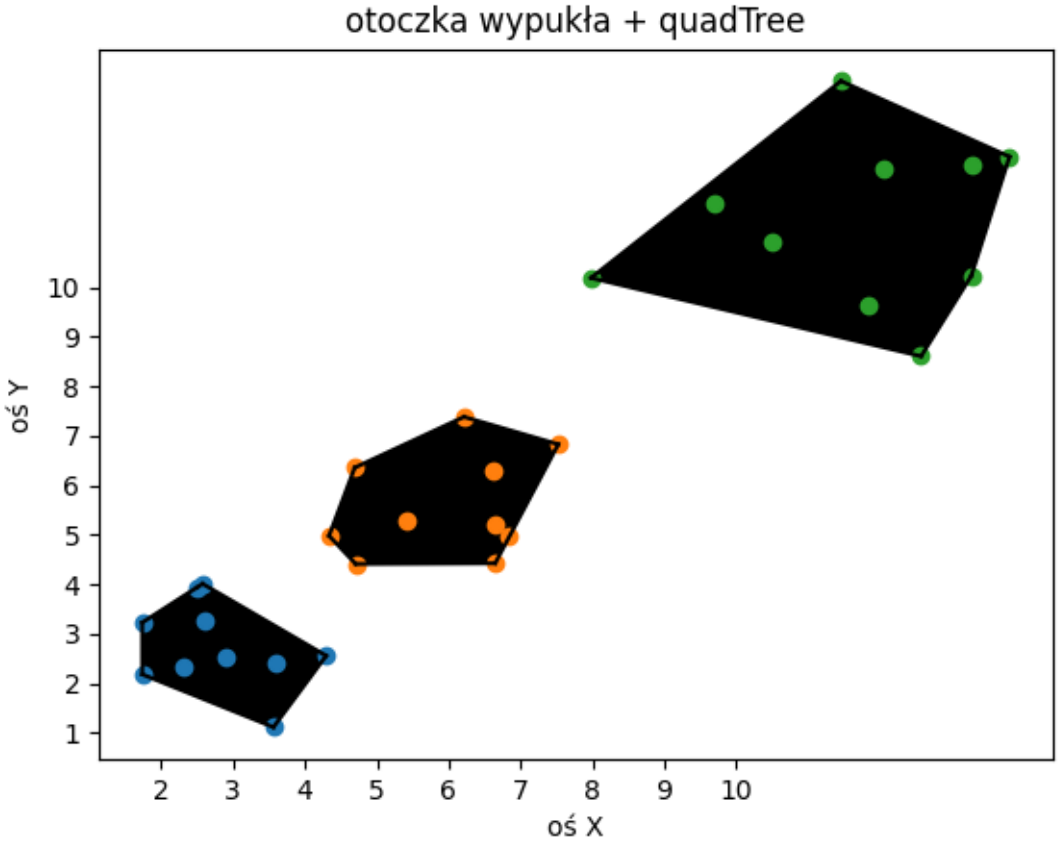
A oto wyniki działania programu.

Najpierw otoczka wypukła:

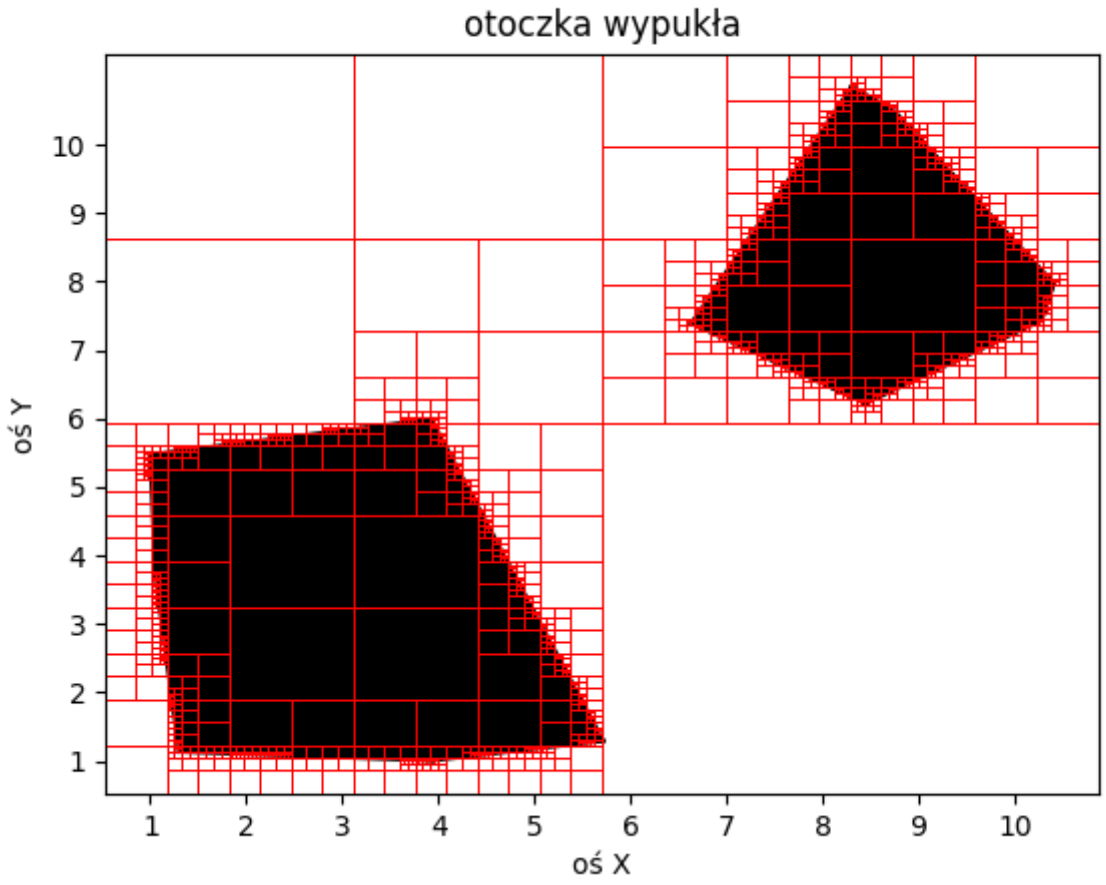


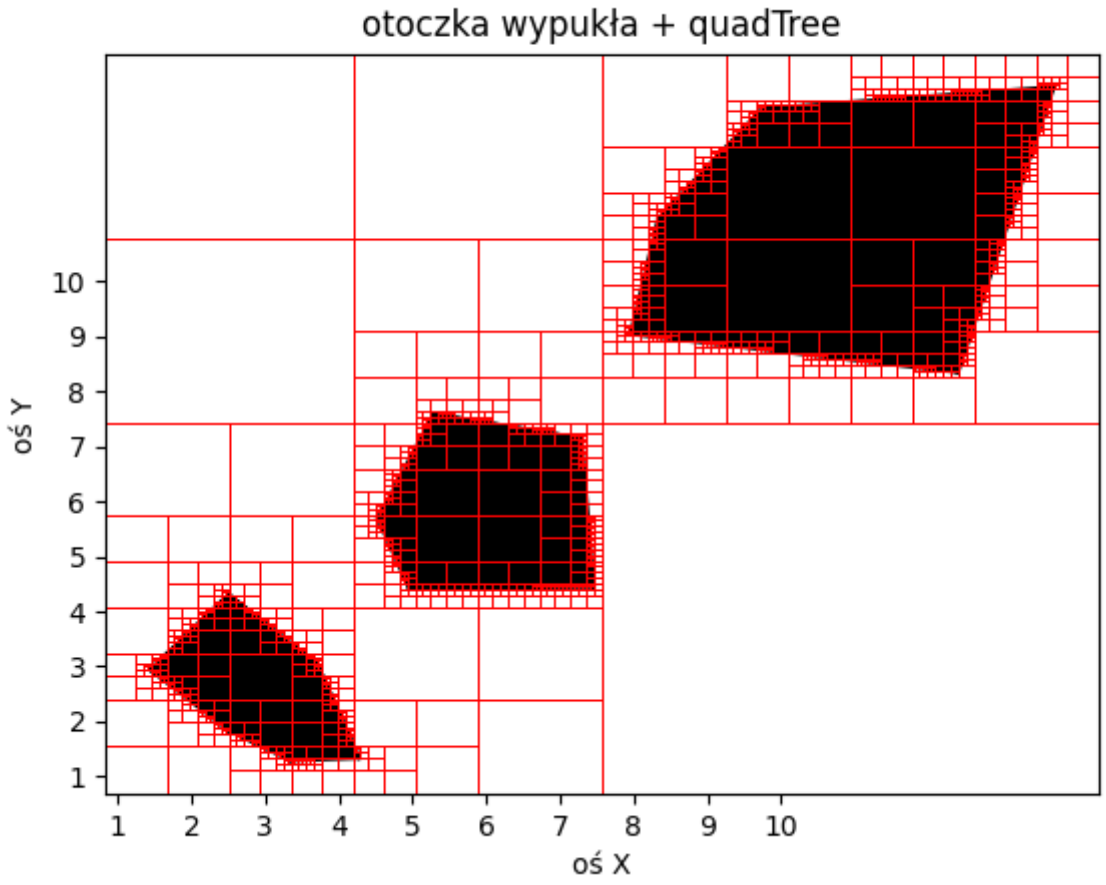

otoczka wypukła + quadTree
oś Y
oś X
1
2
3
4
5
6
7
8
9
10


otoczka wypukła + quadTree
oś Y
oś X
1
2
3
4
5
6
7
8
9
10



Teraz quadTree:


otoczka wypukła
oś Y
oś X
1
2
3
4
5
6
7
8
9
10



Wnioski

Projekt quadTree jest świetnym projektem, który można śmiało dodać do swojego githuba. Ten projekt wciągnął mnie całkowicie – ciężko było oderwać się od komputera, a samo kodzenie było czystą przyjemnością. Jedynie trzeba było się skupić podczas podawania argumentów dla rekurencyjnych wywołań metody divide().