Akademia Górniczo – Hutnicza
*im. Stanisława Staszica*
w Krakowie

**Informatyka**
Wydział Informatyki, Elektroniki
i Telekomunikacji
Bank Pekao S. A. Kraków
65 1240 2294 1111 0010 4756 7266

Technika mikroprocesorowa 2

# *Shape detection*

Janusz Mirosław
Tomasz Nowicki

2014/2015

# Spis treści

# 1. Opis projektu

Celem naszego projektu jest napisanie programu, który będzie rozpoznawał kształty na zdjęciach/plikach graficznych. Program zostanie napisany w języku Java SE 8u25 bez używania gotowych bibliotek lub obiektowości.

Na początku zechcemy aby nasz program rozpoznawał proste kształty – kwadrat, prostokąt oraz trójkąt z wczytanego uprzednio pliku graficznego. W miarę możliwości zechcemy dodać kolejne kształty: koło i elipsę. Ponad to zechcemy wykorzystać w naszym projekcie kamerę internetową. Chcielibyśmy robić nią zdjęcia i rozpoznawać kształty na zrobionym przez nią zdjęciu.

# 2. Harmonogram prac

**Data.....Opis prac**

2014-10-21.....Wykonanie zarysu dokumentacji

2014-11-12.....Program rozpoznaje kształty (trójkąt, kwadrat, prostokąt, koło, elipsa) na zdjęciu wczytanym z pliku graficznego

2014-11-26.....Program rozpoznaje kształty (trójkąt, kwadrat, prostokąt, koło, elipsa) mające wyraźne, gładkie krawędzie na zdjęciu zrobionym kamerką internetową.

2014-12-03.....Program rozpoznaje kształty (trójkąt, kwadrat, prostokąt, koło, elipsa) na zdjęciu zrobionym kamerką internetową. Poprawione zostało wykrywanie figur mających nieregularne, postrzępione krawędzie.

# 3. Opis algorytmu

Na początku wczytujemy zdjęcie z pliku graficznego (dokładniej – używaliśmy plików `.bmp`) do zmiennej typu `BufferedImage` z biblioteki import `java.awt.image.BufferedImage`, od tej chwili nie korzystamy z innych bibliotek.

Następnie piksel po pikselu odczytujemy kolory naszego obrazka i wpisujemy je do tablicy `Color[][] tab_RGB`. W kolejnych krokach będziemy tworzyć nowe tablice i nowe pliki w celu kontroli tego co dzieje się z naszym zdjęciem podczas kolejnych etapów pracy.

Tworzymy nowe zdjęcie w skali szarości wybierając kolory w następujących proporcjach: czerwony 30%, zielony 59%, niebieski 11%.

Kolejny krok to zastosowanie ośmiu operatorów Sobela, które pozwolą nam wykryć piksele znacząco różniące się kolorem od swoich sąsiadów. Jeżeli sąsiedzi mają taki sam kolor to badany piksel otrzymuje kolor zbliżony do czarnego ponieważ macierze zapewnią nam zniesienie się tego samego koloru.

Dostajemy 8 tablic tablic – jedną dla każdego operatora Sobela. Przechodzimy po każdej komórce tych tablic – jeżeli piksel ma kolor zbliżony od czarnego (do R=1, G=1, B=1) to zostaje zmieniony na czarny (R=0, G=0, B=0), w przeciwnym wypadku zostaje zmieniony na biały R=0, G=0, B=0). Po utworzeniu każdej z tablic Sobela usuwamy pojedyncze białe piksele (takie, które nie mają przynajmniej dwóch „białych” sąsiadów). Kolejny krok to nałożenie na siebie tablic w następujący sposób: jeżeli dany piksel w jednej z tablic jest biały to w nowej tablicy ma mieć biały kolor, w przeciwnym wypadku ma być czarny. Na tabeli z sumą usuwamy pojedyncze białe piksele w analogiczny sposób do wyżej opisanego.

W następnym etapie wykonujemy wygładzanie krawędzi. Polega ono na tym, że w obrębie uzyskanych już krawędzi `amount_of_smoothing` razy (parametr ustawiany przez użytkownika na początku programu) przechodzimy po tablicy ze zdjęciem i szukamy pikseli, które mają mniej niż `neighbors_smoothed_edge` białych sąsiadów. Jeżeli znajdziemy piksel spełniający ten warunek, to zmieniamy jego kolor na czarny, ponieważ jest to piksel „wystający” z krawędzi i nie pozwoli nam on na poprawne wykrycie kształtu. Jeżeli piksel nie spełnia tego warunku to staje się biały (działanie profilaktyczne).

Kolejny krok to zamalowanie czarnych dziur w krawędziach. Etap ten nie daje nam tak spektakularnych rezultatów jak wygładzanie krawędzi, niemniej nie możemy jednoznacznie zadeklarować, że jest nam niepotrzebny. `amount_of_filling` razy przechodzimy po tablicy ze zdjęciem i szukamy pikseli, które mają mniej niż `neighbors_filled_edges` białych sąsiadów. Jeżeli znajdziemy piksel spełniający ten warunek, to zmieniamy jego kolor na czarny. W przeciwnym wypadku staje się biały (działanie profilaktyczne).

Po zakończeniu wszystkich zabiegów na zdjęciu malujemy na nim czarną ramkę o szerokości jednego piksela. Zapobiega to sytuacji w której badany kształt wychodziłby poza tablicę.

Przechodzimy po całej tablicy i szukamy wierzchołków „startowych” danej figury, czyli takich które leżą najbliżej lewego górnego rogu tablicy. Jeżeli natrafimy na taki wierzchołek to rozpoczynamy okrążać figurę. Poruszamy się w czterech ustalonych przez nas „kierunkach” czyli:
Kierunek pierwszy: w prawo albo w prawo i w dół albo w dół
Kierunek drugi: w dół albo w lewo i w dół albo w lewo
Kierunek trzeci: w lewo albo w lewo i w górę albo w górę
Kierunek czwarty: w górę albo w prawo i w górę albo w prawo

Takie przejście kolejno po kierunkach powinno spowodować powrót do wierzchołka „startowego” (badamy ten fakt z dokładnością do 10% zmiany współrzędnych). W czasie przechodzenia po drodze w każdym kierunku malujemy piksele po których już przeszliśmy na zielono oraz liczymy ich ilość.

Jeżeli znajdziemy się ponownie w wierzchołku startowym to sprawdzamy czy trzy lub cztery wierzchołki mają współrzędne różne od (0, 0) – to ustalone przez nas wartości początkowe. Jeżeli mamy 4 wierzchołki to sprawdzamy czy one się nie pokrywają (z dokładnością do 50 pikseli) ponieważ nie chcemy wykrywać małych obiektów, samotnych linii albo „grubych” krawędzi – byłoby to uciążliwe dla zdjęcia zrobionego kamerą.

Kolejny krok to sprawdzenie warunków czy trafiliśmy na czworokąt czy na trójkąt (sprawdzamy ilość wierzchołków różnych od (0, 0) oraz badamy długości boków naszej figury). Jeżeli mamy cztery wierzchołki to może to być kwadrat, prostokąt, koło albo elipsa. Kwadrat i koło odróżniamy od prostokąta i elipsy na podstawie odległości między odpowiednimi wierzchołkami. Koło i elipsę odróżniamy od kwadratu i prostokąta na podstawie kroków które zrobiliśmy przechodząc po danym boku. Jeżeli zrobiliśmy tyle kroków ile wynosi odległość między wierzchołkami (z dokładnością do 3% dla koła i do 1% dla elipsy) to jest to kwadrat albo prostokąt, a w przeciwnym wypadku koło albo elipsa.

Po znalezieniu dowolnej figury zmieniamy kolory pikseli w jej obrębie oraz prostokąta w który jest wpisana. Nie chcemy ponownie znajdować wierzchołków należących do uprzednio znalezionej figury, ponieważ mogłyby generować nam nieprawidłowe figury.

W celach testowych tworzymy obrazek po przejściu po obwodach figur oraz kolejny po zamalowaniu wszystkich figur.

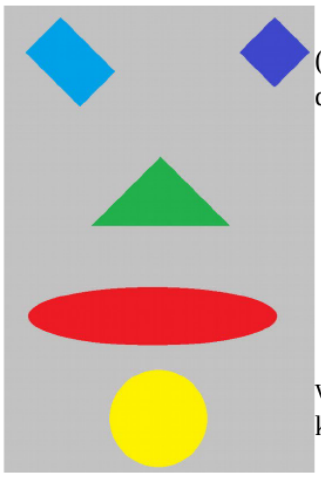
Dla kwadratów i prostokątów na standardowe wyjście wypisujemy współrzędne ich wierzchołków. Dla kół wypisujemy położenie ich środków oraz długości promieni. Dla elips – położenie ich środka oraz długości promieni krótszych i promieni dłuższych.

# 3. Wyniki

Wykrywanie kształtów na zdjęciach wymaga różnych ustawień następujących stałych:

```
int neighbors_white
int neighbors_smoothed_edge
int neighbors_filled_edges
int amount_of_smoothing
int amount_of_filling
```

W związku z tym wyszukanie kształtów na danym zdjęciu wymaga przeprowadzenia od kilku do nawet kilkudziesięciu prób, a to zajmuje mnóstwo czasu. Na kolejnych stronach prezentujemy uzyskane wyniki dla kilku przykładowych zdjęć.



Pierwszy, testowy obrazek utworzony w programie Paint (rozdzielczość 640x960), wszystkie kształty zostały poprawnie wykryte dla następujących wartości parametrów:

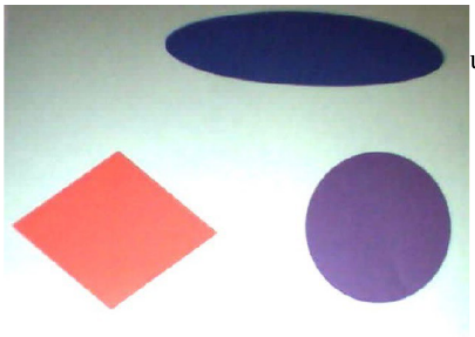
```
int neighbors_white = 4;
int neighbors_smoothed_edge = 5;
int neighbors_filled_edges = 5;
int amount_of_smoothing = 1;
int amount_of_filling = 1;
```

Spostrzeżenie – dla wyraźnych krawędzi nie musimy stosować ani wygładzania krawędzi ani zamalowywania czarnych pikseli w obrębie krawędzi.

Wyjście:

```
Znalazlem prostokat, jego wierzcholki to:
[116; 23]
[227; 136]
[154; 206]
[44; 95]

Znalazlem kwadrat, jego wierzcholki to:
[555; 23]
[626; 96]
[553; 166]
[483; 95]

Znalazlem trojkat, jego wierzcholki to:
[320; 310]
[462; 450]
[180; 450]

Znalazlem elipse
Srodek: [289; 637]
Promien pierwszy: 62
Promien drugi: 255

Znalazlem kolo
Srodek: [312; 847]
Promien: 100

Czas wykonywania programu 4.88[s]
```

Pozostałe zdjęcia zostały wykonane kamerką internetową (rozdzielczość 640x480)



Dla tego zdjęcia nie udało nam się dobrać uniwersalnych parametrów. Parametry:

```
int neighbors_white = 4;
int neighbors_smoothed_edge = 5;
int neighbors_filled_edges = 5;
int amount_of_smoothing = 3;
int amount_of_filling = 5;
```

Wyjście:

```
Znalazlem kwadrat, jego wierzcholki to:
[155; 199]
[292; 311]
[146; 416]
[10; 299]

Czas wykonywania programu3.26[s]
```

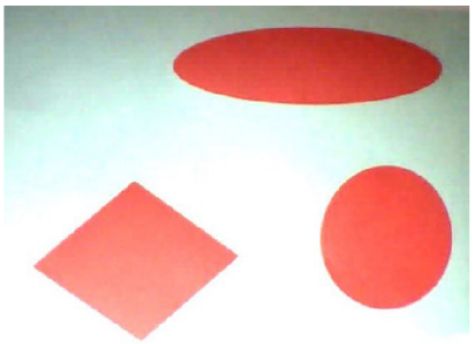
Dla ustawień:

```
int neighbors_white = 4;
int neighbors_smoothed_edge = 5;
int neighbors_filled_edges = 5;
int amount_of_smoothing = 5;
int amount_of_filling = 5;
```

Program wykrył elipsę:

```
Znalazlem elipse
Srodek: [401; 59]
Promien pierwszy: 64
Promien drugi: 193

Czas wykonywania programu 3.41[s]
```

Niestety nie udało nam się wykryć trzeciej figury. Przed wystarczającym wygładzeniem krawędzi następowało jej przerwanie. Wystarczyła zmiana kolorów zdjęcia (patrz następna strona), by wykryć na nim wszystkie kształty.



Parametry:

```
int neighbors_white = 4;
int neighbors_smoothed_edge = 5;
int neighbors_filled_edges = 5;
int amount_of_smoothing = 4;
int amount_of_filling = 5;
```

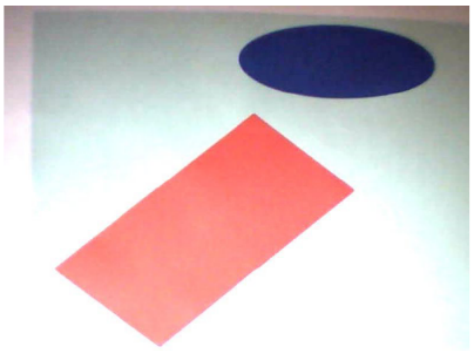
Wyjście:

```
Znalazlem elipse
Srodek: [408; 83]
Promien pierwszy: 55
Promien drugi: 185

Znalazlem elipse
Srodek: [522; 314]
Promien pierwszy: 101
Promien drugi: 93

Znalazlem kwadrat, jego wierzcholki to:
[177; 241]
[320; 344]
[186; 458]
[40; 357]

Czas wykonywania programu 3.36[s]
```

Pojawia się pytanie – dlaczego „koło” zostało wykryte jako elipsa? Jak widać jeden z promieni koła wykrytego jako elipsa jest dłuższy od drugiego o ~9% (przypominamy, że tolerancję ustawiliśmy na 5%). Jest to zrozumiałe z tego względu, że nie jest to obrazek z programu Paint tylko zdjęcie zrobione kamerą. Zmiana ustawień tolerancji na 10% spowodowała wykrycie koła o promieniu równym w przybliżeniu średniej z promieni elipsy.

Wyjście:

```
Znalazlem elipse
Srodek: [408; 83]
Promien pierwszy: 55
Promien drugi: 185

Znalazlem kolo
Srodek: [522; 314]
Promien: 97

Znalazlem kwadrat, jego wierzcholki to:
[177; 241]
[320; 344]
[186; 458]
[40; 357]

Czas wykonywania programu 3.44[s]
```



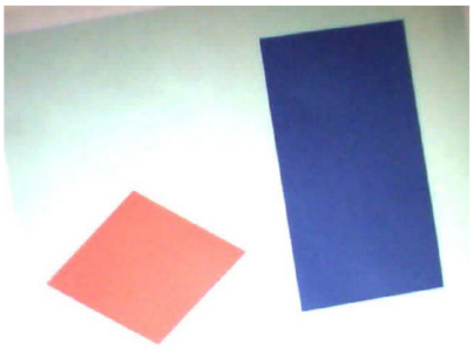
Parametry:

```
int neighbors_white = 4;
int neighbors_smoothed_edge = 5;
int neighbors_filled_edges = 5;
int amount_of_smoothing = 2;
int amount_of_filling = 5;
```

Wyjście:

```
Znalazlem prostokat, jego wierzcholki to:
[424; 183]
[601; 318]
[268; 585]
[86; 450]

Czas wykonywania programu 4.45[s]
```

Niestety nie udało nam się wykryć drugiej figury (elipsy). Przed wystarczającym wygładzeniem krawędzi następowało jej przerwanie. Prawdopodobnie tak jak w przykładzie drugim wystarczyłaby zmiana kolorów figury, np. na czerwony.



Parametry:

```
int neighbors_white = 4;
int neighbors_smoothed_edge = 5;
int neighbors_filled_edges = 5;
int amount_of_smoothing = 1;
int amount_of_filling = 1;
```

Wyjście:

```
Znalazlem kwadrat, jego wierzcholki to:
[178; 252]
[331; 338]
[211; 465]
[57; 380]

Czas wykonywania programu 2.35[s]
```

Niestety nie udało nam się wykryć drugiej figury (elipsy). Przed wystarczającym wygładzeniem krawędzi następowało jej przerwanie. Prawdopodobnie tak jak w przykładzie drugim wystarczyłaby zmiana kolorów figury, np. na czerwony.

# 4. Podsumowanie

Wykrywanie kształtów okazało się nietrywialnym i niekoniecznie łatwo rozwiązywalnym zagadnieniem. Wykrycie kształtów na pliku zrobionym w programie graficznym (np. Paint), w którym kształty mają gładkie, regularne krawędzie oraz nie występują cienie jest łatwe do uzyskania. Operatory Sobela bardzo dobrze radzą sobie z wykryciem zmian kolorów (krawędzi) na takich obrazach, w wyniku dostajemy cienkie i gładkie krawędzie, a to umożliwia nam bezproblemowe przekształcenie ich na figury. Poradziliśmy sobie z każdą narysowaną przez nas w poprawny sposób figurą.

Większą trudność sprawiają zdjęcia zrobione kamerą. Tło nie jest jednolite (ponad to testowaliśmy tła dwukolorowe) a figura ma nieregularne krawędzie, jest nachylona pod różnymi kątami. Kolejną trudnością, którą musieliśmy pokonać były cienie występujące na całym zdjęciu. W tym przypadku nie możemy się pochwalić idealnym wykrywaniem kształtów. Podczas testów przeprowadziliśmy wiele prób z różnymi ustawieniami opisanych wyżej parametrów, a i tak nie zawsze udało się uzyskać pożądany efekt. Ośmielimy się stwierdzić, że w dużej mierze zależy to od koloru tła i figur, niestety nie wykryliśmy żadnej sensownej analogii.

Projekt nauczył nas szerzej patrzeć na pracę ze zdjęciami. Okazało się, że problemu nie da się sprowadzić do polecenia „wykryj różową figurę”, tylko musieliśmy przeprowadzić wiele operacji na zdjęciu a następnie dobierać ustawienia odpowiednich parametrów tak żeby uzyskać gładkie, regularne krawędzie. Po uzyskaniu odpowiednich krawędzi nasz własny algorytm co najmniej dobrze radził sobie z określeniem rodzaju figury.

5. Bibliografia


- http://www.sunshine2k.de/coding/java/Houghtransformation/HoughTransform.html
- http://en.wikipedia.org/wiki/Sobel_operator