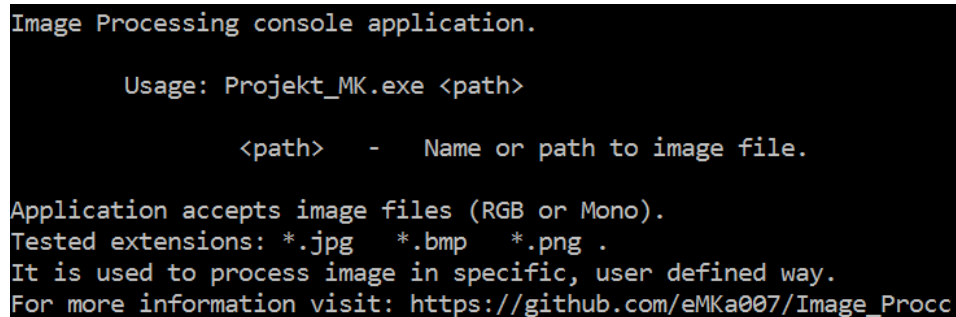


## Opis Aplikacji:

Aplikację uruchamia się z poziomu środowiska Visual Studio, bądź wiersza poleceń wskazując na skompilowany plik *Projekt\_MK.exe*



```
Image Processing console application.

Usage: Projekt_MK.exe <path>

    <path>    -    Name or path to image file.

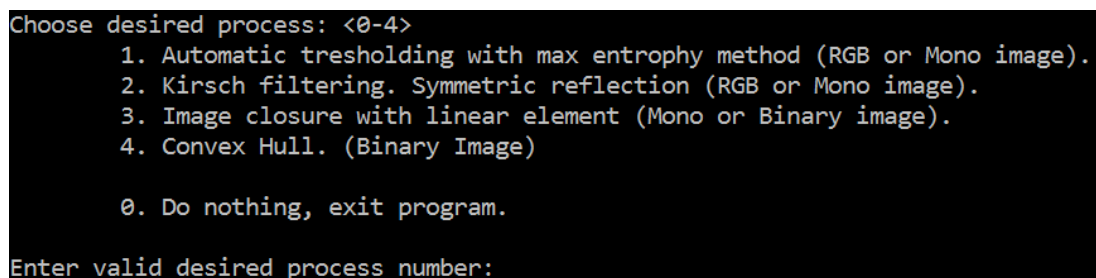
Application accepts image files (RGB or Mono).
Tested extensions: *.jpg *.bmp *.png .
It is used to process image in specific, user defined way.
For more information visit: https://github.com/eMKa007/Image\_Procc
```

Rys 1. Wiadomość w przypadku uruchomienia programu bez argumentu.

Ścieżkę do obrazu poddawanego przekształceniom podajemy jako argument wywołania ( może być to ścieżka względna lub bezwzględna ) np.:

*Projekt\_MK.exe ../TestImages/grayscale.png*

Następnie jesteśmy proszeni o wybór odpowiedniego przekształcenia. W tym miejscu poczynione jest odważne założenie-użytkownik będzie wprowadzał poprawne obrazy do przekształceń (np. dla Convex Hull – tylko obrazy binarne ).



```
Choose desired process: <0-4>
  1. Automatic thresholding with max entropy method (RGB or Mono image).
  2. Kirsch filtering. Symmetric reflection (RGB or Mono image).
  3. Image closure with linear element (Mono or Binary image).
  4. Convex Hull. (Binary Image)

  0. Do nothing, exit program.

Enter valid desired process number:
```

Rys 2. Wybór przez użytkownika odpowiedniego przekształcenia.

W dalszych krokach następuje już praca nad wprowadzonym obrazem lub, w przypadku opcji 3 ( Zamknięcie elementem liniowym ) użytkownik jest proszony o wprowadzenie długości i kąta odchylenia od osi OX.

Przetworzony obraz jest zapisywany w miejscu obrazu źródłowego pod nazwą obrazu źródłowego z dopiskiem odnoszącym się do konkretnego przekształcenia.

Implementacja napisana jest w języku C++ i wykorzystuje przestrzeń nazw System::Drawing. Konieczne jest użycie CLR ( ang. Common Language Runtime Support), oraz dodanie odnośników 'system' oraz 'system.drawing' z platformy .NET.

Zaimplementowane przekształcenia:

1. Automatyczne progowanie metodą maksymalnej entropii.
2. Filtracja Kirscha. Brzeg- odbicie symetryczne.  
Dla obrazów RGB, każda warstwa osobno.
3. Zamknięcie elementem liniowym o zadanej długości i nachyleniu.
4. Wypukłe otoczenie.

#### Ad. 1) Automatyczne progowanie metodą maksymalnej entropii.

Wynikiem tego przekształcenia jest uzyskanie obrazu binarnego o progu dobranym w taki sposób aby maksymalizował funkcję entropii obliczaną przy pomocy wzoru (1):

$$H(k) = \frac{-1}{N_k(tło)} \sum_{x,y=1}^k (A_{x,y} * \ln A_{x,y}) - \frac{1}{N_k(obiekt)} \sum_{k+1}^{size(N)} (A_{x,y} * \ln A_{x,y})$$

$k$  – aktualna wartość progu binaryzacji

$N_k(tło)$  – ilość pikseli tła,  $\leq k$

$N_k(obiekt)$  – ilość pikseli obiektu,  $\geq k$

$A_{x,y}$  – prawdopodobieństwo napotkania piksela na obrazie o zadanej jasności

$size(N)$  – ilość przedziałów histogramu

Eq 1. Równanie funkcji której krok o wartości maksymalnej brany jest jako próg binaryzacji.

Pierwszym krokiem, jeśli zadany obrazem jest obraz RGB, jest sprowadzenie go do odcieni szarości. W przypadku obrazów Grayscale, krok ten nie jest konieczny.

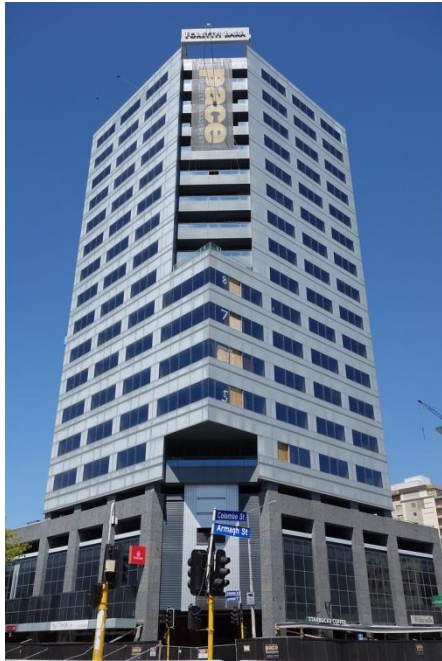
Tak przygotowany obraz możemy poddać dalszym obliczeniom. Wyliczany jest histogram, a następnie przy użyciu funkcji **ComputeMaxEntropy()** obliczana jest wartość zmiennej **ThresholdVal** przechowującej progową wartość odcienia szarości dla której funkcja entropii posiadała największą wartość. Wartość ta jest wykorzystywana do binaryzacji obrazu według założenia (2):

$$Jasność\ Piksela = (Jasność\ Piksela \leq Próg\ Binaryzacji ? 0 : 255)$$

Eq 2. Warunek binaryzacji piksela o zadanej wartości. (składnia C++)

Funkcja **ComputeHistogramEntropy()** jest funkcją pomocniczą wykorzystywaną do obliczenia entropii według wzoru Eq. 1. dla zadanego przedziału histogramu.

Obraz po przekształceniu zapisywany jest do katalogu w którym znajduje się obraz wejściowy pod nazwą zdjęcia wejściowego z dopiskiem “\_AutoThreshold.bmp”.



Rys 3. Obrazy rgb/mono wejściowy. Obrazy wyjściowe po progowaniu z wykorzystaniem maksymalnej entropii.

## Ad. 2) Filtracja Kirscha. Brzeg- odbicie symetryczne.

Przekształcenie to wykorzystywane jest do detekcji krawędzi. Zrealizowane jest poprzez filtrację ośmioma maskami które są względem siebie obrócone o  $45^\circ$ . Po wykonaniu takiego zabiegu maksymalna wartość amplitudy przypisywana jest centralnemu pikselowi, dla którego wykonywane było sumomnożenie.

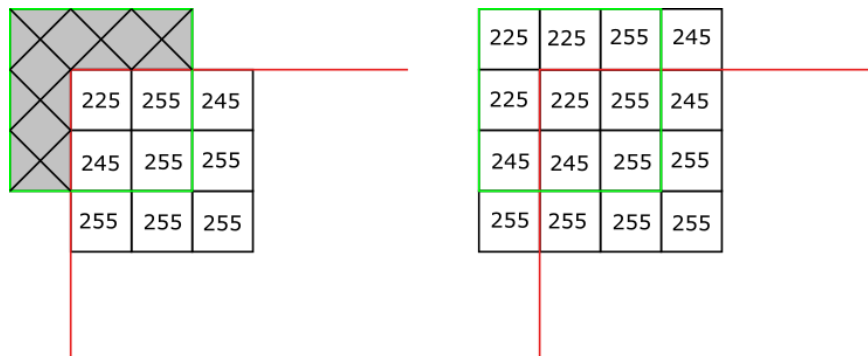
Maski wykorzystywane przy filtracji przedstawiają się następująco:

$$\begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} \quad \begin{bmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{bmatrix} \quad \begin{bmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{bmatrix} \quad \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{bmatrix} \\
 \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{bmatrix} \quad \begin{bmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{bmatrix} \quad \begin{bmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{bmatrix} \quad \begin{bmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}$$

Rys 4. Maski wykorzystywane w Filtracji Kirscha.

Stworzenie i wypełnienie masek zrealizowane jest w konstruktorze klasy `KirschFilt` przy pomocy funkcji `FillKirschMasks()`.

Dla pikseli znajdujących się na brzegach obrazu wejściowego nałożenie maski wykraczałoby poza zakres obrazu. Dlatego też, wymagane jest powielenie wszystkich pikseli brzegowych (tj. rozszerzenie obrazu o 2 piksele w każdym kierunku). Realizowane jest to przez funkcję: `ReplicateBorderValues()`.



Rys 5. Problem wyjścia maski poza obszar obrazu bez zreplikowanej granicy.

Następnie funkcja `FiltChannelsRGB()` dla każdego piksela oblicza maksymalną wartość z filtracji ośmioma wcześniej stworzonymi maskami. a następnie przypisuje ją do piksela centralnego dla którego obliczenia te były prowadzone. Jeśli wartość maksymalna przekracza wartość 255. Wpisywana jest 255.

Wyjściowy obraz zapisywany jest do katalogu w którym znajduje się plik wejściowy pod nazwą obrazu wejściowego z dopiskiem “\_KirschFiltration.bmp”.





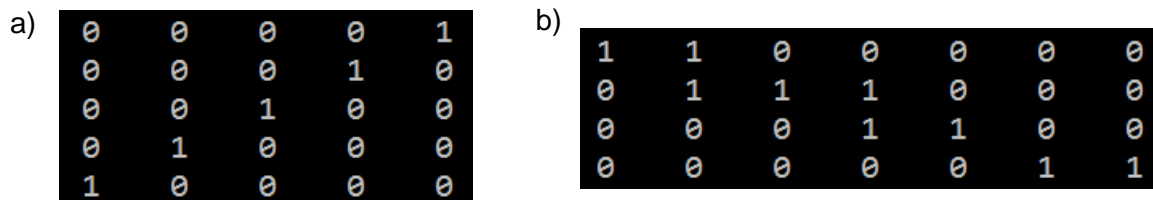
Rys. 6 Wynik filtracji Kirscha dla obrazów RGB oraz w odcieniach szarości.

### Ad. 3) Zamknięcie elementem liniowym o zadanej długości i nachyleniu.

Operacja zamknięcia jest złożeniem dwóch przekształceń- erozji i dylatacji obrazu. Operacja erozji polega na przypisaniu pikselowi dla którego nałożona jest maska wartości minimalnej piksela dla którego wartość elementu strukturalnego jest równa 1. Wynikiem tego działania jest zmniejszenie powierzchni obiektów na obrazie.

Odwrotną do erozji jest operacja dylatacji. W niej do piksela centralnego następuje przypisanie wartości maksymalnej piksela dla którego wartość elementu strukturalnego jest równa 1.

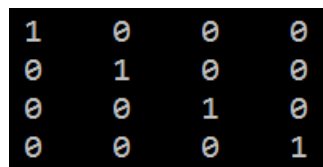
Za stworzenie elementu strukturalnego odpowiedzialna jest funkcja **CreateStructuralElement()** która pobiera od użytkownika informację dotyczącą długości elementu liniowego oraz jego odchylenia od osi OX ( kierunek odwrotny do ruchu wskazówek zegara ). Wyliczana jest potrzebna wielkość tablicy, alokowana pamięć oraz sama tablica jest wypełniana ( wpisywane wartości 1 ) w miejsca przejścia linii o zadanym nachyleniu przez piksel elementu strukturalnego. Dla tak wytworzonego elementu określany jest piksel środkowy ( kotwica ), będący elementem centralnym wykorzystywanym w operacjach erozji i dylatacji.



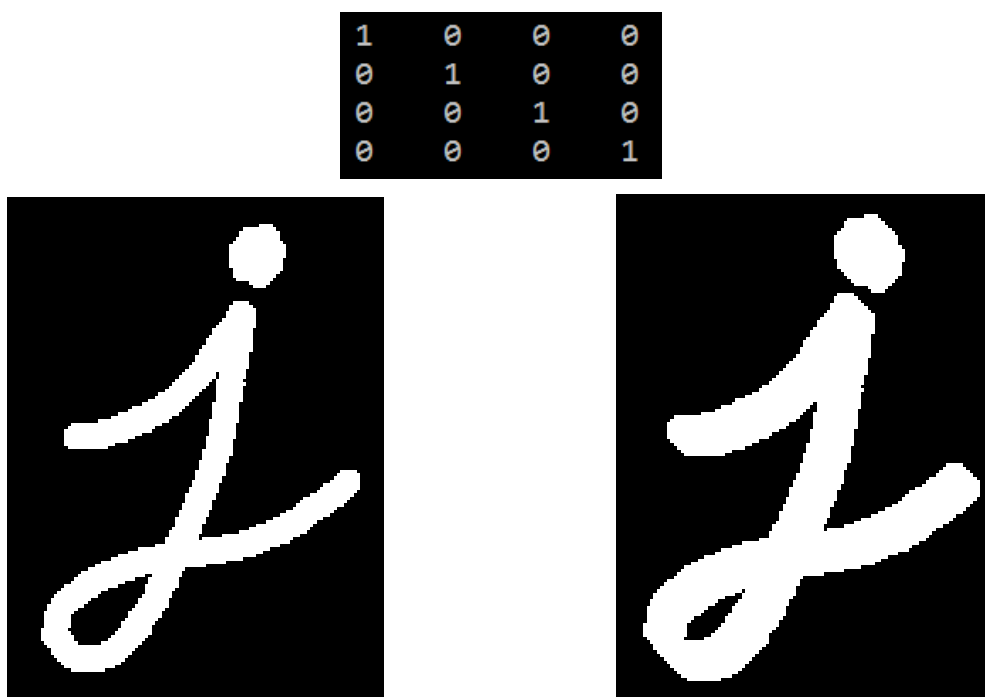
Rys. 7 Przykłady stworzonych elementów strukturalnych.  
a) Długość= 7; kąt =  $45^{\circ}$  b) Długość = 7; kąt= $150^{\circ}$

Operacje erozji i dylatacji zaimplementowane są odpowiednio przez funkcje **Erode()** i **Dilate()**.

- **imopen() = Dilate( Erode() )**
- **imclose()= Erode( Dilate() )**



Rys 8. Element strukturalny oraz przetworzony obraz w odcieniach szarości.  
Przekształcenie: zamknięcie elementem liniowym (długość: 5; nachylenie  $135^{\circ}$ ).



Rys 9. Element strukturalny oraz przetworzony obraz binarny.  
Przekształcenie: zamknięcie elementem liniowym (długość: 5; nachylenie 135°).

#### Ad 4) Wypukłe otoczenie - obraz binarny.

Operacja ta przekształca wejściowy obraz binarny tak, aby wszystkie figury na obrazie były figurami wypukłymi (tj. dowolnie poprowadzona linia prosta wewnątrz figury, w całości zawiera się wewnątrz tej figury). Implementacja tego przekształcenia opiera się na operacji pogrubiania ośmioma maskami obracanymi o kąt 45°.

$$\begin{bmatrix} 1 & 1 & X \\ 1 & 0 & X \\ 1 & X & 0 \end{bmatrix} \quad
 \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & X \\ X & 0 & X \end{bmatrix} \quad
 \begin{bmatrix} 1 & 1 & 1 \\ X & 0 & 1 \\ 0 & X & X \end{bmatrix} \quad
 \begin{bmatrix} X & 1 & 1 \\ 0 & 0 & 1 \\ X & X & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & X & 1 \\ X & 0 & 1 \\ X & 1 & 1 \end{bmatrix} \quad
 \begin{bmatrix} X & 0 & X \\ X & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad
 \begin{bmatrix} X & X & 0 \\ 1 & 0 & X \\ 1 & 1 & 1 \end{bmatrix} \quad
 \begin{bmatrix} 1 & X & X \\ 1 & 0 & 0 \\ 1 & 1 & X \end{bmatrix}$$

Rys 10. Maski wykorzystywane w przekształceniu „Wypukłe otoczenie” (Convex Hull) obrazu binarnego.

Wartość analizowanego piksela centralnego ustawiana jest na 1 (255) bądź 0, w zależności od spasowania maski. Odpowiednie indeksy maski i obrazu wejściowego muszą do siebie pasować:

- wartość 1 (255) na obrazie musi odpowiadać wartości 1 (255) w masce.
- wartość 0 (0) na obrazie musi odpowiadać wartości 0 (0) w masce.
- w przypadku wartości -1 (dokumentacji X) w masce wartość na obrazie może być dowolna, nie jest brana pod uwagę.



Rys. 11. Obraz binarny przed i po przekształceniu "Wypukłe otoczenie".