Politechnika Śląska w Gliwicach
Wydział Automatyki, Elektroniki i Informatyki

Programowanie Komputerów 2

Operacje morfologiczne

autor	Łukasz Ważny
prowadzący	dr inż. Adam Gudyś
rok akademicki	2017/2018
kierunek	Informatyka
rodzaj studiów	SSI
semestr	2
termin laboratorium / ćwiczeń	poniedziałek, 10:15 – 11:45
grupa	1
sekcja	1
termin oddania sprawozdania	2018-06-20
data oddania sprawozdania	2018-06-19

1 Treść zadania

Napisać program wykonujący operacje morfologiczne (dylatacja, erozja, otwarcie, domknięcie, top hat, bottom hat) na obrazach binarnych. Program wywoływany ma być przez wiersz poleceń z parametrami:

- plik wejściowy z obrazem, na którym mają być wykonane operacje morfologiczne;
- plik wyjściowy, do którego ma być zapisany przekształcony obraz;
- plik z elementem strukturalnym;
- nazwa operacji do wykonania.

2 Analiza zadania

W związku z tym, że literatura opisująca zagadnienia przekształceń morfologicznych różnie przedstawia pewne detale dotyczące algorytmów operacji morfologicznych, poniżej opiszę, co dokładnie rozumiem poprzez każdą z operacji. Obraz binarny traktuję jako macierz białych i czarnych pikseli. Element strukturalny, to również macierz z tylko jednym czarnym pikselem, będącym punktem centralnym.

2.1 Dylatacja

Operację tę oznaczę na potrzeby tej analizy przez $dyl(src)$, gdzie src, to macierz poddawana dylatacji. Polega ona na tym, że do każdego elementu macierzy „przykładany" jest punkt centralny elementu strukturalnego. Następnie sprawdzane jest, czy w sąsiedztwie badanego elementu macierzy w granicach elementu strukturalnego znajduje się przynajmniej jeden czarny piksel. Jeżeli tak, badany element staje się też czarnym pikselem. W wyniku tej operacji, obiekty złożone z czarnych pikseli rozrastają się, zostają zalepione dziury oraz znikają detale.

2.2 Erozja

Tę operację oznaczę jako $ero(src)$, gdzie src, to macierz poddawana erozji. Można powiedzieć, że jest to operacja dualna do dylatacji. Algorytm jest niemalże identyczny. Różnica polega na tym, że zamiast sprawdzania, czy w obrębie „przyłożonego" do badanego elementu macierzy elementu strukturalnego znajduje się przynajmniej jeden czarny piksel; sprawdzane jest czy znajduje się przynajmniej jeden biały piksel. Jeżeli tak - badany element macierzy staje się białym pikselem. W konsekwencji, obiekty złożone z czarnych pikseli, które w porównaniu z elementem strukturalnym są duże, zmiejszają

się; natomiast te małe całkowicie znikają; dziury powiększają się, przyjmując kształt elementu strukturalnego.

2.3 Otwarcie

Operacja otwarcia, to dylatacja macierzy będącej wynikiem erozji macierzy pierwotnej. Symbolicznie można tę operację przedstawić jako:

$$otw(src) = dyl(ero(src)),$$

gdzie *src*, to macierz poddawana otwarciu. W wyniku tej operacji obiekty złożone z czarnych pikseli zachowują podobny rozmiar, są natomiast trochę „wygładzone", to jest usuwane są „wystające" z nich elementy. Ponadto operacja ta posiada ciekawą własność, polegającą na tym, że macierz powstała po wielokrotnym jej zastosowaniu jest identyczna z tą po jednokrotnym zastosowaniu otwarcia.

2.4 Domknięcie

Operacja domknięcia, działa odwrotnie do otwarcia, to jest najpierw przeprowadzana jest dylatacja macierzy pierwotnej, a następnie erozja wyniku dylatacji. Symbolicznie:

$$dom(src) = ero(dyl(src)),$$

gdzie *src*, to macierz poddawana domknieciu. W jej wyniku zostają usunięte wszelakie dziury mniejsze od elementu strukturalnego, natomiast obiekty złożone z czarnych pikseli, które znajdują się dosyć blisko siebie mogą zostać „połączone".

2.5 Top hat

Symbolicznie operację tę można zdefiniować jako:

$$th(src) = src - otw(src),$$

gdzie *src*, to macierz poddawana operacji top hat. W tym miejscu warto sprecyzować, co oznacza użyte we wzorze odejmowanie. Otóż odejmowanie w powyższym przypadku interpretuję jako powrównywanie ze sobą kolejnych odpowiadających sobie pikseli z macierzy *src* oraz $otw(src)$ i jeżeli obydwa porównywane piksele są czarne, to odpowiadający im piksel macierzy wynikowej $th(src)$ jest biały. W przypadku każdego innego zestawienia pikseli, odpowiadający piksel macierzy $th(src)$ jest kopiowany z macierzy *src*. W ten

sposoób w wyniku dostajemy macierz, w której czarne piksele, to lokalne maksima macierzy źródłowej, czyli te czarne piksele, które w swoim najbliższym sąsiedztwie mają bardzo dużo pikseli białych; czyli głównie dosyć małe obiekty, pojedyncze piksele oraz kawałki krawędzi dużych obiektów.

2.6 Bottom hat

Symbolicznie operację tę można zdefiniować jako:

$$bh(src) = dom(src) - src,$$

gdzie *src*, to macierz poddawana operacji bottom hat. Odejmowanie jest interpretowane tak samo, jak w przypadku top hat. Bottom hat działa podobnie jak top hat, przy czym w wyniku bottom hat czarne piksele macierzy wynikowej, to lokalne minima macierzy źródłowej, czyli te białe piksele, które w swoim najbliższym sąsiedztwie mają bardzo dużo pikseli czarnych, czyli głównie małe dziury, wąskie przerwy pomiędzy obiektami, czasami również drobne kawałki krawędzi obiektów.

3 Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Należy przekazać do programu: nazwę pliku wejściowego, wyjściowego, nazwę elementu strukturalnego, nazwę operacji. Dostępne nazwy operacji, to:

```
-dylatacja
-erozja
-otwarcie
-domkniecie
-top_hat
-bottom_hat
```

Podanie nieprawidłowej nazwy operacji powoduje wyświetlenie komunikatu:

```
Nieprawidlowa nazwa operacji!
```

Plik wejściowy oraz element strukturalny może być obrazem w dowolnym formacie graficznym obsługiwanym przez bibliotekę `stb_image`. Plik wyjściowy może już istnieć (zostanie wtedy nadpisany) lub nie (wtedy zostanie utworzony nowy plik), jego format może być dowolnym formatem graficznym obsługiwanym przez bibliotekę `stb_image_write`. Przykładowe uruchomienie programu:

```
program.exe zdj_mazury.jpg mazury_po_erozji.png elem_strukt.bmp -erozja
```

Podanie liczby parametrów innej niż cztery powoduje wyświetlenie komunikatu:

```
Nieprawidlowa liczba parametrow!
```

Podanie nieprawidłowej nazwy pliku wejściowego powoduje komunikat:

```
Nie udalo sie otworzyc pliku wejsciowego!
```

Podanie nieprawidłowej nazwy elementu strukturalnego:

```
Nie udalo sie otworzyc elementu strukturalnego!
```

Jeżeli element strukturalny jest wadliwy, czyli nie ma w nim punktu centralnego, wyświetla się komunikat:

```
Nie znaleziono punktu centralnego w elemencie strukturalnym!
```

4 Specyfikacja wewnętrzna

Program został podzielony na dwa pliki: `main.c` - zawierający główną funkcję programu oraz `morfologia.c` - zawierający funkcje wykonujące odpowiednie przekształcenia morfologiczne. Oczywiście stworzyłem również plik `morfologia.h`, zawierający deklaracje funkcji znajdujących się w `morfologia.c`. Korzystam również z dwóch bibliotek graficznych mieszczących się w całości w plikach nagłówkowych `stb_image.h` oraz `stb_image_write.h`. Warto też zaznaczyć, że czarny piksel jest przez program interpretowany jako piksel o wartości mniejszej niż 10, natomiast biały - większej niż 245.

4.1 Ogólna struktura programu

W funkcji głównej pierwsze około 60 linijek odpowiedzialne jest za sprawdzenie wszystkiego, co jest niezbędne do uruchomienia odpowiedniej funkcji wykonującej operację morfologiczną. Również otwierane są pliki zawierające obraz źródłowy oraz element strukturalny. Potem wywoływana jest funkcja wykonująca operację wskazaną przez użytkownika na obrazie źródłowym. Następnie przekształcony przez funkcję obraz jest zapisywany do pliku o nazwie podanej przez użytkownika. Na końcu zwalniane są zasoby związane z otwartymi plikami oraz buforem, do którego zapisywany był wynik odpowiedniej operacji.

4.2 Szczegółowy opis implementacji funkcji

```
1 int pkt_centralny(unsigned char *elem, int elem_width,
    int elem_height, int * pkt_centralny_x, int *
    pkt_centralny_y);
```

Funkcja pkt_centralny szuka współrzędnych punktu centralnego w elemencie strukturalnym. Przyjmuje parametry:

elem	wskaźnik na otwarty plik z elementem strukturalnym;
elem_width	szerokość elementu strukturalnego;
elem_height	wysokość elementu strukturalnego;
pkt_centralny_x	wskaźnik na zmienną, w której ma być zapisana współrzędna x punktu centralnego;
pkt_centralny_y	wskaźnik na zmienną, w której ma być zapisana współrzędna y punktu centralnego.

Funkcja zwraca wartość 0, jeżeli udało się znaleźć punkt centralny (jego odpowiednie współrzędne po wywołaniu funkcji znajdują się w zmiennych, na które wskazują wskaźniki pkt_centralny_x oraz pkt_centralny_y); natomiast jeżeli się nie udało znaleźć tegoż punktu, zwracana jest wartość -1.

Funkcje:

```
1 unsigned char * dylatacja(const unsigned char * plik_we
    , int plik_we_width, int plik_we_height, int
    elem_width, int elem_height, int pkt_centr_x, int
    pkt_centr_y);
```

```
1 unsigned char * erozja(const unsigned char * plik_we,
    int plik_we_width, int plik_we_height, int
    elem_width, int elem_height, int pkt_centr_x, int
    pkt_centr_y);
```

```
1 unsigned char * otwarcie(const unsigned char * plik_we,
     int plik_we_width, int plik_we_height, int
    elem_width, int elem_height, int pkt_centr_x, int
    pkt_centr_y);
```

```
1 unsigned char * domkniecie(const unsigned char *
    plik_we, int plik_we_width, int plik_we_height, int
    elem_width, int elem_height, int pkt_centr_x, int
    pkt_centr_y);
```

```
unsigned char * top_hat(const unsigned char * plik_we,
    int plik_we_width, int plik_we_height, int
    elem_width, int elem_height, int pkt_centr_x, int
    pkt_centr_y);
```

```
unsigned char * bottom_hat(const unsigned char *
    plik_we, int plik_we_width, int plik_we_height, int
    elem_width, int elem_height, int pkt_centr_x, int
    pkt_centr_y);
```

przyjmują jednakowe parametry:

plik_we	wskaźnik na otwarty plik z obrazem źródłowym, który chcemy poddać operacji o nazwie takiej, jak nazwa funkcji;
plik_we_width	szerokość obrazu źródłowego;
plik_we_height	wysokość obrazu źródłowego;
elem_width	szerokość elementu strukturalnego;
elem_height	wysokość elementu strukturalnego;
pkt_centr_x	współrzędna x punktu centralnego;
pkt_centr_y	współrzędna y punktu centralnego.

Każda z nich wykonuje operację o nazwie takiej, jak nazwa funkcji, na pliku źródłowym i zwraca otrzymany wynik (przekształcony obraz) w formie wskaźnika na dane, które można zapisać do pliku w formacie graficznym.

Pierwsze dwie funkcje - dylatacja oraz erozja - alokują pamięć na bufor, który następnie jest przez te funkcje zwracany. Pamięć ta zwalniana jest potem w funkcji głównej.

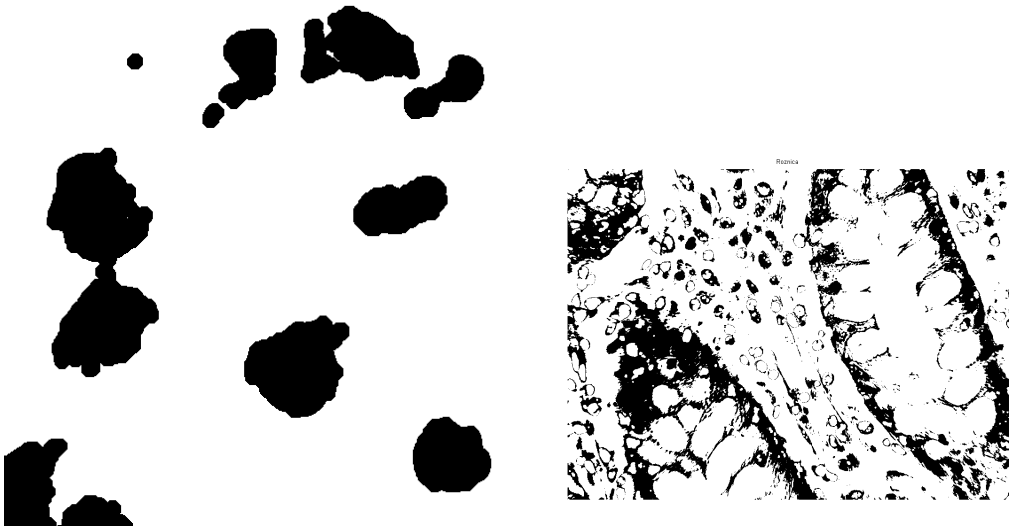
Kolejne dwie funkcje - otwarcie oraz domkniecie - korzystają dwukrotnie z funkcji poprzednich, alokując tym samym pamięć dwa razy. Jeden z zaalokowanych zasobów zwalniany jest jeszcze w trakcie działania funkcji, inny jest zwalniany w funkcji głównej.

Funkcja top_hat również alokuje dwukrotnie pamięć, korzystając przy tym jednak tylko raz z poprzednich funkcji. Również jeden z zasobów zwalniany jest w trakcie działania funkcji - inny w funkcji głównej.

Ostatnia z funkcji - bottom_hat - korzysta tylko raz z poprzednich funkcji, alokując pamięć, która zwalniana jest w funkcji głównej.

5 Testowanie

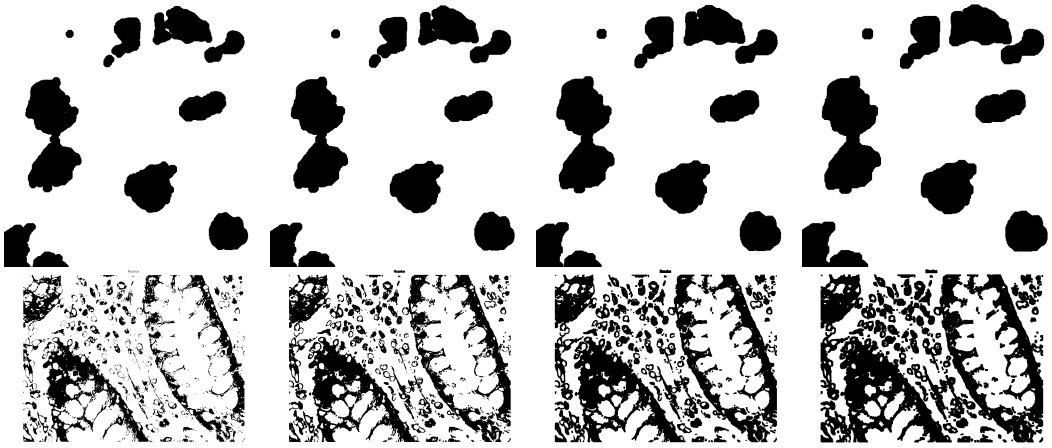
Program został przetestowany na następujących obrazach:



Użyty element strukturalny do testowania, to obraz binarny o rozmiarze 3 x 3, z czarnym pikselem, będącym punktem centralnym, w samym środku.

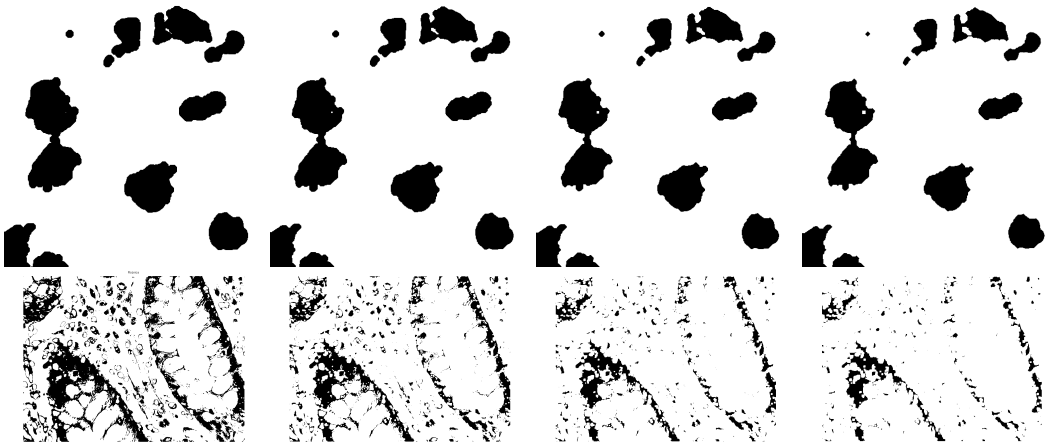
Poniższe obrazy będące wynikami testów zostały tak zmniejszone, aby każda „grupa" badanych obrazów zmieściła się w jednym wierszu. Dzięki temu po powiększeniu, np. dwukrotnym, dokumentu, porównywanie ich powinno być ułatwione. Na początku każdego wiersza znajduje się oryginalny obraz. Obok niego kolejne obrazy poddawane odpowiednim operacjom. Obrazy użyte do testowania można ściągnąć ze stron: `http://docplayer.pl/docs-images/27/11147745/images/88-0.png`, `http://docplayer.pl/docs-images/27/11147745/images/34-0.png`

Najpierw obydwa obrazy zostały poddane trzykrotnej dylatacji:



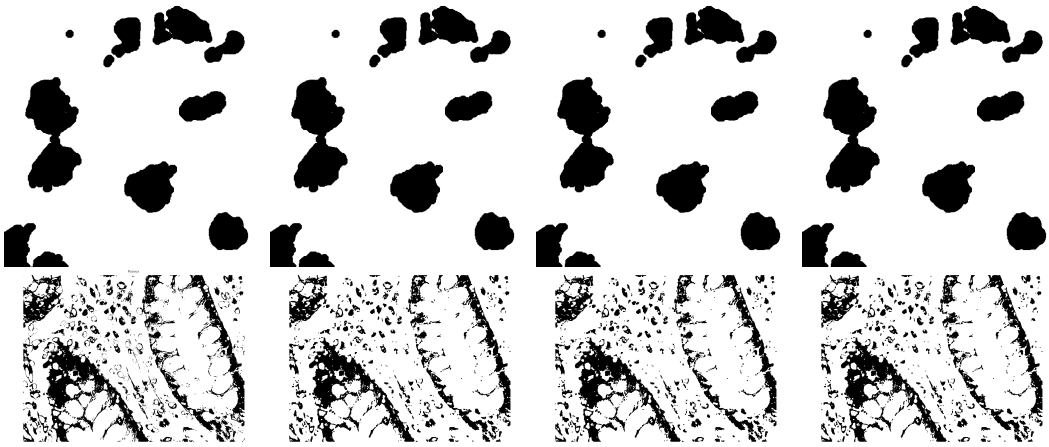
Jak widać, czarne obiekty w obydwóch przypadkach powiększyły się, zniknęły też dziury.

Kolejna przeprowadzona operacja to erozja:



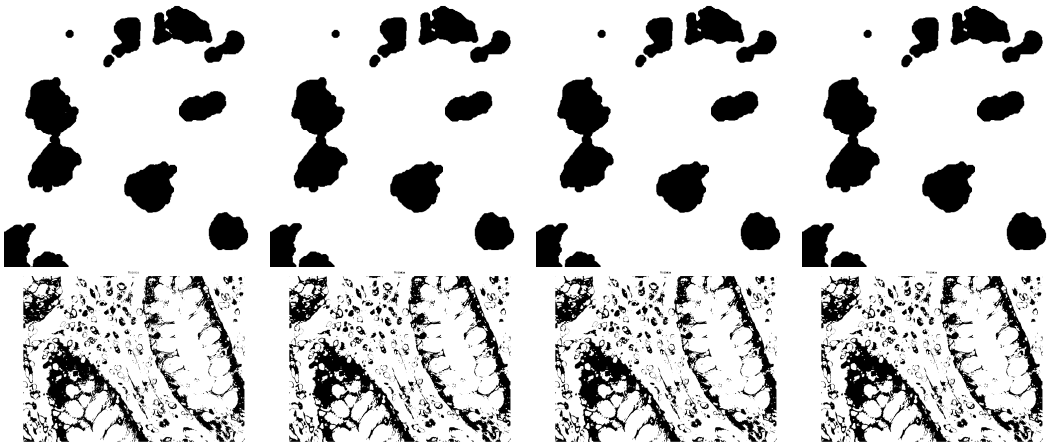
W wyniku tej operacji, widać to szczególnie w przypadku drugiego obrazu, obiekty „skurczyły" się, te bardzo małe prawie znikły, natomiast dziury powiększyły się i zbliżyły się do kształtu elementu strukturalnego, co lepiej jest widoczne w przypadku pierwszego obrazu (potrzebne około trzykrotne powiększenie, aby lepiej to zauważyć)

Kolejną operacją było otwarcie:



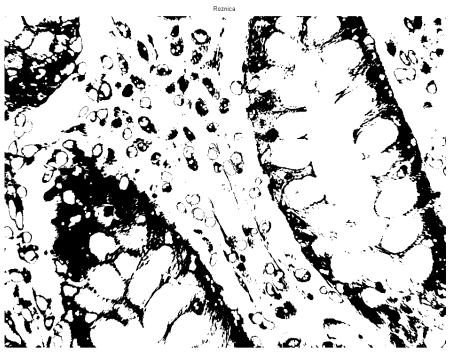
Ze względu na charakter pierwszego obrazu (dosyć gładkie krawędzie obiektów, brak wystających elementów), nie widać w jego przypadku prawie żadnej różnicy. Natomiast w przypadku drugiego obrazu da się zauważyć po około trzykrotnym przybliżeniu dokumentu, że pewne wystające elementy, wąskie łączenia obiektów zostały usunięte; a krawędzie lekko wygładzone. Warto też zauważyć, że kolejne wykonywanie operacji otwarcia nie zmienia obrazu, co jest zgodne z założeniami.

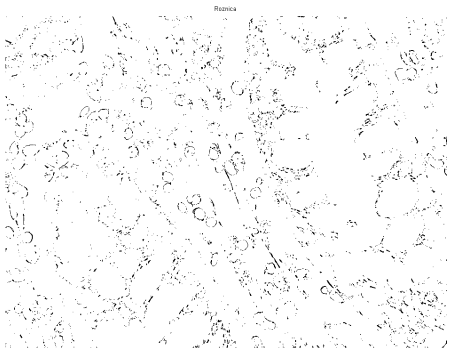
Następna operacja - domknięcie:



Podobnie, jak w przypadku otwarcia, pierwszy obraz nie jest dobrym kandydatem do tej operacji i nie da się zauważyć żadnych zmian. Natomiast w drugim obrazie da się zauważyć (do tego potrzebne jest już czterokrotne powiększenie), że bardzo małe dziury zniknęły, a np. litery wyrazu „Roznica", który znajduje się na samej górze obrazu, zostały ze sobą połączone. Podobnie jak w przypadku otwarcia, kolejne przeprowadzanie operacji domknięcia nic nie zmienia.

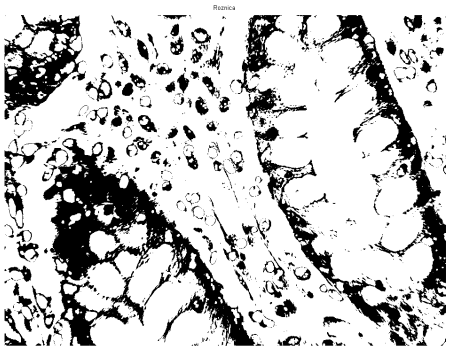
Potem wykonana została operacja top hat:

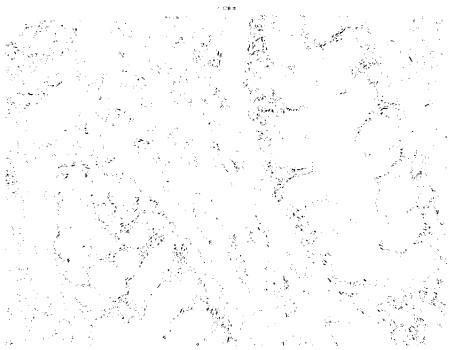




Te białe pole po prawej stronie pierwszego obrazu, to wynik operacji top hat, na nim przeprowadzonej, jak widać program nie znalazł żadnych lokalnych maksimów, co ma sens, ponieważ obiekty z tego obrazu mają dosyć gładkie krawędzie i większość czarnych pikseli ma w swoim otoczeniu przynajmniej połowę również czarnych. Natomiast w przypadku drugiego obrazu lokalne maksima zostały wykryte i jest ich już całkiem dużo.

Ostatnia wykonana operacja, to bottom hat:





W pierwszym obrazie zostały wykryte minima, ponieważ znajduje się w nim kilka „mikroskopijnych" dziur oraz kilka pikseli znajdujących się w „zagięciach" obiektów, co oznacza, że są osaczone przez wiele czarnych pikseli. Natomiast w przypadku drugiego obrazu mamy już wykrytych lokalnych minimów znacznie więcej, podobnie jak w przypadku operacji top hat.

Program został też sprawdzony pod kątem wycieków pamięci za pomocą wtyczki do Visual Studio o nazwie Visual Leak Detector. Za każdym razem wtyczka wygenerowała raport pozytywny, to jest nie zostały wykryte żadne wycieki. Poniżej zamieszczam kilka przykładowych raportów, po jednym dla każdej operacji morfologicznej.

Dylatacja:

```
Visual Leak Detector read settings from: C:\Program Files (x86)\Visual Leak Dete
Visual Leak Detector Version 2.5.1 installed.
'Projekt.exe' (Win32): Loaded 'C:\Windows\System32\kernel.appcore.dll'. Symbols
The thread 0x3308 has exited with code 0 (0x0).
The thread 0x2550 has exited with code 0 (0x0).
The thread 0x1ed8 has exited with code 0 (0x0).
No memory leaks detected.
Visual Leak Detector is now exiting.
```

Erozja:

```
Visual Leak Detector read settings from: C:\Program Files (x86)\Visual Leak Dete
Visual Leak Detector Version 2.5.1 installed.
'Projekt.exe' (Win32): Loaded 'C:\Windows\System32\kernel.appcore.dll'. Symbols
The thread 0x2af4 has exited with code 0 (0x0).
The thread 0x2998 has exited with code 0 (0x0).
The thread 0x2ea8 has exited with code 0 (0x0).
No memory leaks detected.
Visual Leak Detector is now exiting.
```

Otwarcie:

```
Visual Leak Detector read settings from: C:\Program Files (x86)\Visual Leak Dete
Visual Leak Detector Version 2.5.1 installed.
'Projekt.exe' (Win32): Loaded 'C:\Windows\System32\kernel.appcore.dll'. Symbols
The thread 0x216c has exited with code 0 (0x0).
The thread 0x31d4 has exited with code 0 (0x0).
The thread 0x36f8 has exited with code 0 (0x0).
No memory leaks detected.
Visual Leak Detector is now exiting.
```

Domknięcie:

```
Visual Leak Detector read settings from: C:\Program Files (x86)\Visual Leak Dete
Visual Leak Detector Version 2.5.1 installed.
'Projekt.exe' (Win32): Loaded 'C:\Windows\System32\kernel.appcore.dll'. Symbols
The thread 0x2cc0 has exited with code 0 (0x0).
The thread 0x1eb4 has exited with code 0 (0x0).
The thread 0x2dd8 has exited with code 0 (0x0).
No memory leaks detected.
Visual Leak Detector is now exiting.
```

Top hat:

```
Visual Leak Detector read settings from: C:\Program Files (x86)\Visual Leak Dete
Visual Leak Detector Version 2.5.1 installed.
'Projekt.exe' (Win32): Loaded 'C:\Windows\System32\kernel.appcore.dll'. Symbols
The thread 0x2dd8 has exited with code 0 (0x0).
The thread 0x2cc0 has exited with code 0 (0x0).
The thread 0x17a8 has exited with code 0 (0x0).
No memory leaks detected.
Visual Leak Detector is now exiting.
```

Bottom hat:

```
Visual Leak Detector read settings from: C:\Program Files (x86)\Visual Leak Dete
Visual Leak Detector Version 2.5.1 installed.
'Projekt.exe' (Win32): Loaded 'C:\Windows\System32\kernel.appcore.dll'. Symbols
The thread 0x3170 has exited with code 0 (0x0).
The thread 0x232c has exited with code 0 (0x0).
The thread 0x1610 has exited with code 0 (0x0).
No memory leaks detected.
Visual Leak Detector is now exiting.
```