

Laboratorium przetwarzanie obrazów

Radosław Piotrowicz

Informacje potrzebne do implementacji

1 Gdzie znajdują się interesujące nas rzeczy.

Wszystkie algorytmy należy implementować w **Algorithms.cpp**.

W **Algorithms.hpp** znajdują się wszystkie deklaracje funkcji.

W **Algorithms.hpp** znajduje się również definicja struktury z parametrami do algorytmów (niektóre nie wymagają parametrów).

```
..void CreateNegative(Image *outputImage);  
..void BrightenImage(Image *outputImage, ParametersStruct *params);  
..void Contrast(Image *outputImage, ParametersStruct *params);  
..void Exponentiation(Image *outputImage, ParametersStruct *params);  
..void LevelHistogram(Image *outputImage);  
..void Binarization(Image *outputImage, ParametersStruct *params);  
..void LinearFilter(Image *outputImage, ParametersStruct *params);  
..void MedianFilter(Image *outputImage, ParametersStruct *params);  
..void Erosion(Image *outputImage, ParametersStruct *params);  
..void Dilatation(Image *outputImage, ParametersStruct *params);  
..void Skeletonization(Image *outputImage);  
..void Hough(Image *outputImage, ParametersStruct *params);
```

Rysunek 1: Deklaracje funkcji

```
· struct ParametersStruct
· {
·     ···· // Brighten ·· Darken
·     ···· int value = 0;
·     ···· // Contrast
·     ···· float contrast = 1.0;
·     ···· // Exp
·     ···· float alfa = 1.0;
·     ···· // Binarization
·     ···· int boundCount = 1;
·     ···· int lowerBound = 0;
·     ···· int upperBound = 0;
·     ···· int method = None;
·     ···· // Linear Filters
·     ···· int linerFilterS = Average;
·     ···· int linearFilterSize = S3x3;
·     ···· std::array<std::array<int, 3>, 3> linearMask3x3 = AVERAGE_3x3;
·     ···· std::array<std::array<int, 5>, 5> linearMask5x5 = AVERAGE_5x5;
·     ···· std::array<std::array<int, 7>, 7> linearMask7x7 = AVERAGE_7x7;
·     ···· // Median Filters
·     ···· int medianFilterS = Full;
·     ···· int medianFilterSize = S3x3;
·     ···· std::array<std::array<bool, 3>, 3> medianMask3x3 = MEDIAN_3x3;
·     ···· std::array<std::array<bool, 5>, 5> medianMask5x5 = MEDIAN_5x5;
·     ···· std::array<std::array<bool, 7>, 7> medianMask7x7 = MEDIAN_7x7;
·     ···· // Erosion
·     ···· int erosionElementSize = S3x3;
·     ···· std::array<std::array<bool, 3>, 3> erosionElement3x3 = EMPTY_3x3;
·     ···· std::array<std::array<bool, 5>, 5> erosionElement5x5 = EMPTY_5x5;
·     ···· std::array<std::array<bool, 7>, 7> erosionElement7x7 = EMPTY_7x7;
·     ···· // Dilatation
·     ···· int dilatationElementSize = S3x3;
·     ···· std::array<std::array<bool, 3>, 3> dilatationElement3x3 = EMPTY_3x3;
·     ···· std::array<std::array<bool, 5>, 5> dilatationElement5x5 = EMPTY_5x5;
·     ···· std::array<std::array<bool, 7>, 7> dilatationElement7x7 = EMPTY_7x7;
·     ···· // Hough
·     ···· int maxIndexRo = 0;
·     ···· int maxIndexTheta = 0;
·     ···· int maxHoughVal = 0;
· }
```

Rysunek 2: Struktura danych z parametrami

2 Co jest potrzebne przy implementacji algorytmu

Algorytmy wykonują się w oddzielnym wątku, w związku z czym w funkcji muszą znaleźć się dwie rzeczy.

2.1 Lokalna kopia obrazu i parametrów

```
// local copy
Mutex::GetInstance().Lock();
Image copy = *outputImage;
auto value = params->value;
Mutex::GetInstance().Unlock();
```

Rysunek 3: Kopiowanie potrzebnych zasobów

2.2 Skopiowanie wyników do zasobów współdzielonych oraz informacja, że wątek się zakończył

```
// copy back to output
Mutex::GetInstance().Lock();
Mutex::GetInstance().ThreadStopped();
*outputImage = copy;
copy.ClearImage();
Mutex::GetInstance().Unlock();
```

Rysunek 4: Zatrzymanie i skopiowanie wyników

Można też opcjonalnie dodać odświeżanie i możliwość przerwania wykonywania.

Natomiast nie jest to wymagane.

Automatyczne odświeżanie można włączyć w zakładce Ustawienia. Tam również można ustawić co ile sekund ma się odświeżać

2.3 Jak dodać przerwanie i odświeżanie do funkcji

```
for (int row = 0; row < copy.GetHeight(); row++)
{
    for (int col = 0; col < copy.GetWidth(); col++)
    {
        auto pix = copy.GetPixel(col, row);
        // algorithm code
        copy.SetPixel(col, row, pix);
    }
    Mutex::GetInstance().Lock();
    // if canceled
    if (!Mutex::GetInstance().IsThreadRunning())
    {
        *outputImage = copy;
        copy.ClearImage();
        Mutex::GetInstance().Unlock();
        return;
    }
    // if auto-refresh enabled
    if (Mutex::GetInstance().GetState() == Mutex::AlgorithmThreadRefresh)
    {
        *outputImage = copy;
        Mutex::GetInstance().SetState(Mutex::MainThreadRefresh);
    }
    Mutex::GetInstance().Unlock();
}
```

Rysunek 5: Odświeżanie i anulowanie

W niektórych algorytmach efekt jest lepszy, jeżeli ustawimy ręczne odświeżanie. Wtedy można to zrobić w ten sposób.

```
Mutex::GetInstance().Lock();
// if canceled
if (!Mutex::GetInstance().IsThreadRunning())
{
    *outputImage = fullCopy;
    fullCopy.ClearImage();
    for (int i = 0; i < h; i++)
        pixelsStatus[i].clear();
    pixelsStatus.clear();
    Mutex::GetInstance().Unlock();
    return;
}
// manually refreshed
*outputImage = fullCopy;
Mutex::GetInstance().SetState(Mutex::MainThreadRefresh);
Mutex::GetInstance().Unlock();
```

Rysunek 6: Zatrzymanie i ręczne kopiowanie

3 Opis klas i przydatnych metod

3.1 Klasa Image

Klasa Image jest kontenerem na obrazy, które będziemy przetwarzać.
Najbardziej interesuje nas w niej SDLSurface, czyli struktura z informacjami o pikselach w obrazie.

Zawiera takie przydatne metody jak:

- **Image** operator=(const Image other) - kopiuje całą zawartość innego obiektu
- **int** GetWidth() - zwraca szerokość obrazu
- **int** GetHeight() - zwraca wysokość obrazu
- **int** GetPixelCount() - zwraca ilość pikseli
- **float*** GetLightValues() - zwraca tablice float o długości 256 z wartościami jasności (są też wersje dla poszczególnych składowych)
- **float*** GetDistributor() - zwraca tablice float o długości 256 z wartościami dystrybuanty (są też wersje dla poszczególnych składowych)
- **void** CopyBrightnessHistogram(float *dst) - kopiuje histogram do podanej tablicy
- **void** CopyNormalisedBrightnessHistogram(float *dst) - kopiuje histogram (unormowany do wartości 0.0 - 1.0) do podanej tablicy
- **bool** NoSurface() - czy obraz zawiera poprawnie zainicjowane SDLSurface (jeśli nie to źle)
- **void** SetBlankSurface(int width, int height) - ustaw obraz na biały prostokąt o wymiarach
- **void** RefreshPixelValuesArrays() - odśwież tablice z histogramami i dystrybuantami (po poprawnym zakończeniu wątku z algorytmem wątek główny automatycznie odświeży)
- **Pixel** GetPixel(int x, int y) - zwraca Strukturę Pixel z informacjami o pikselu w danym punkcie
- **void** SetPixel(int x, int y, Pixel pix) - ustawia pixel w danym punkcie na podany
- **void** SetPixelWhite(int x, int y) - ustawia pixel w danym punkcie na kolor biały
- **void** SetPixelBlack(int x, int y) - ustawia pixel w danym punkcie na kolor czarny

- **void** CopyOnlySurfaceAndSize(Image other) - kopiuje tylko surface i wy-miary (użyteczne w algorytmach, w których wartość jednego piksela zależy od wartości jego sąsiadów)

3.2 Struktura Pixel

Ta struktura zawiera wartości R, G, B oraz jasność piksela.

Należy pamiętać, że gdy modyfikujemy piksel, należy ustawić wszystkie **3** składowe.

WARTOŚĆ JASNOŚCI JEST TYLKO DO ODCZYTU. METODA SetPixel NIE BIERZE JEJ POD UWAGĘ.

3.3 Klasa Mutex

Semafor do synchronizacji wątków

- **void** Lock() - opuść semafor
- **void** Unlock() - podnieś semafor
- **bool** IsThreadRunning() - czy wątek przetwarzający ma się dalej wyko-nywać (używane przy przerwaniu wykonywania wątku)
- **void** ThreadStopped() - ustawia, że wątek przetwarzający zakończył wy-konywanie