

# Laboratorium przetwarzanie obrazów

Jak rozszerzyć program o nowe algorytmy

## 1 Gdzie znajdują się interesujące nas rzeczy.

Jeżeli chcemy dodać nowy algorytm, należy wprowadzić zmiany w plikach:

- **App.hpp** dodanie do enum **AlgSelected**, nowego algorytmu. Oraz opcjonalnie deklaracje metod pomocniczych.
- **App.cpp** dokonanie zmian w metodach **DrawAlgMenuElements**, **DrawParametersPopup**, **LaunchAlgorithms**, **ResetParameters(opcjalne)**
- **Algorithms.hpp** dodanie deklaracji nowej funkcji analogicznie do już tam zawartych. Dodanie nowych parametrów do **ParametersStruct**, jeżeli są wymagane. Dodanie kolejnego enum, jeżeli będzie potrzebne.
- **Algorithms.cpp** implementacja nowego algorytmu.

## 2 App.hpp

Pierwszym krokiem w dodaniu nowego algorytmu jest dodanie go do enum **AlgSelected**.

```
...//·if·new·needed·just·add·it
...//·then·update·DrawAlgMenuElements
...//·DrawParametersPopup
...//·LaunchAlgorithms
...//·then·implement·the·function·in·Algorithms.hpp·Algorithms.cpp
enum AlgSelected
{
    None,
    Negative,
    Brighten,
    Contrast,
    Exponentiation,
    LeveledHistogram,
    Binarization,
    LinearFilter,
    MedianFilter,
    Erosion,
    Dilatation,
    Skeletonization,
    Hough
};
```

Rysunek 1: Enum AlgSelected

### 3 App.cpp

Następnie należy wprowadzić zmiany w metodach.

#### 3.1 DrawAlgMenuElements

```
void App::DrawAlgMenuElements()
{
    //if new needed just add it
    if (ImGui::MenuItem("Negatyw", NULL, algorithmSelected == Negative))
    {
        selectedAlgorithmName = "Negatyw";
        algorithmSelected = Negative;
    }
    if (ImGui::MenuItem("Rozjaśnij", NULL, algorithmSelected == Brighten))
    {
        selectedAlgorithmName = "Rozjaśnij";
        algorithmSelected = Brighten;
    }
    if (ImGui::MenuItem("Kontrast", NULL, algorithmSelected == Contrast))
    {
        selectedAlgorithmName = "Kontrast";
        algorithmSelected = Contrast;
    }
    if (ImGui::MenuItem("Potęgowanie", NULL, algorithmSelected == Exponentiation))
    {
        selectedAlgorithmName = "Potęgowanie";
        algorithmSelected = Exponentiation;
    }
    if (ImGui::MenuItem("Wyrównanie histogramu", NULL, algorithmSelected == LeveledHistogram))
    {
        selectedAlgorithmName = "Wyrównanie histogramu";
        algorithmSelected = LeveledHistogram;
    }
    if (ImGui::MenuItem("Binaryzacja", NULL, algorithmSelected == Binarization))
    {
        selectedAlgorithmName = "Binaryzacja";
        algorithmSelected = Binarization;
    }
    if (ImGui::MenuItem("Filtры Liniowe", NULL, algorithmSelected == LinearFilter))
    {
        selectedAlgorithmName = "Filtры Liniowe";
        algorithmSelected = LinearFilter;
    }
    if (ImGui::MenuItem("Filtры medianowe", NULL, algorithmSelected == MedianFilter))
    {
        selectedAlgorithmName = "Filtры medianowe";
        algorithmSelected = MedianFilter;
    }
}
```

Rysunek 2: Metoda DrawAlgMenu

Tutaj należy dodać na końcu kolejny **if** analogicznie do już zawartych.  
Każdy **if** jest po to, aby wykryć kliknięcie w dany element w menu wyboru algorytmów.



Rysunek 3: Menu wyboru algorytmów

### 3.2 DrawParametersPopup

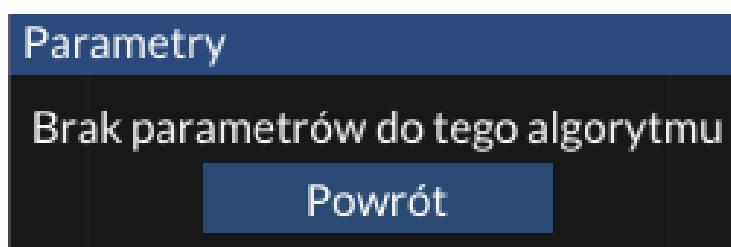
Tutaj należy do **switch** dodać nowy przypadek. W zależności od algorytmu ilość i typ parametrów mogą się różnić.

```
void App::DrawParametersPopup()
{
    ImVec2 center = ImGui::GetMainViewport()->GetCenter();
    ImGui::SetNextWindowPos(center, ImGuiCond_Appearing, ImVec2(0.5f, 0.5f));

    // can not be opened if thread is running
    if (ImGui::BeginPopupModal("Parametry", NULL, ImGuiWindowFlags_AlwaysAutoResize))
    {
        // if new needed just add it
        switch (algorithmSelected)
        {
            case None:
                ImGui::Text("Nie wybrano algorytmu");
                break;
            case Negative:
                ImGui::Text("Brak parametrów do tego algorytmu");
                break;
            case Brighten:
                ImGui::SliderInt("0 ile rozjaśnić/przyciemnić?", &params.value, -255, 255);
                break;
            case Contrast:
                ImGui::SliderFloat("0 ile zmienić kontrast?", &params.contrast, -0.1, 5.0);
                break;
            case Exponentiation:
                ImGui::SliderFloat("Wartość alfa", &params.alfa, 0.1, 3.0);
                break;
            case LeveledHistogram:
                ImGui::Text("Brak parametrów do tego algorytmu");
                break;
            case Binarization:
                DrawBinarizationParams();
                break;
            case LinearFilter:
                DrawLinearFilterParams();
                break;
            case MedianFilter:
                DrawMedianfilterParams();
                break;
            case Erosion:
                DrawErosionParams();
                break;
            case Dilatation:
                DrawDilatationParams();
                break;
        }
    }
}
```

Rysunek 4: Metoda DrawParametersPopup

Jeżeli algorytm nie ma parametrów, należy dodać: **ImGui::Text("Brak parametrów dla tego algorytmu.")**  
Tak będzie wyglądało okienko.



Rysunek 5: Okienko bez parametrów.

W innym przypadku należy dostosować okienko. Oto kilka przykładów:

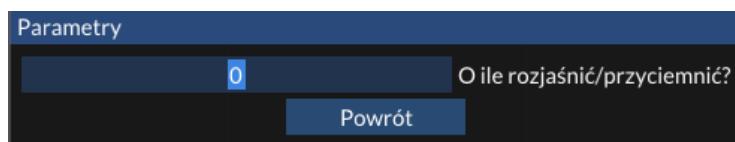
### 3.2.1 Pojedynczy parametr.

Na Rysunku 4 było widać, że dla niektórych algorytmów pojawiała się obsługa tylko jednego parametru na przykład:

**ImGui::SliderInt("O ile rozjaśnić/przyciemnić?", &params.value, -255, 255)**

Najpierw jest opis slidera, potem wartość, do której zapisany będzie wynik, następnie ograniczenia dolne i górne.

A tak będzie wyglądało okienko.



Rysunek 6: Pojedynczy parametr

### 3.2.2 Kilka parametrów

Czasami może być potrzebne kilka parametrów do algorytmu. Takich jak rodzaj metody lub liczba progów. Wtedy można do zrobić w ten sposób.

```

void App::DrawBinarizationParams()
{
    // set if auto or manual
    ImGui::Text("Wybierz czy chcesz ustawić samemu\nczy automatycznie");
    ImGui::RadioButton("Ręcznie ustaw próg", &params.method, Algorithms::None);
    ImGui::SameLine();
    ImGui::RadioButton("Metoda Gradientowa", &params.method, Algorithms::Gradient);
    ImGui::SameLine();
    ImGui::RadioButton("Metoda iteracyjna", &params.method, Algorithms::Histogram);
    if (params.method == Algorithms::BinarizationMethod::None)
    {
        ImGui::Separator();
        ImGui::Text("Wybierz ilość progów");
        ImGui::RadioButton("Jeden próg", &params.boundCount, 1);
        ImGui::SameLine();
        ImGui::RadioButton("Dwa proggi", &params.boundCount, 2);
        ImGui::Separator();
        ImGui::Text("Ustawa prógi");
        ImGui::SliderInt("t", &params.lowerBound, 0, 255);
        if (params.boundCount == 2)
            ImGui::SliderInt("t1", &params.upperBound, 0, 255);
    }
}

```

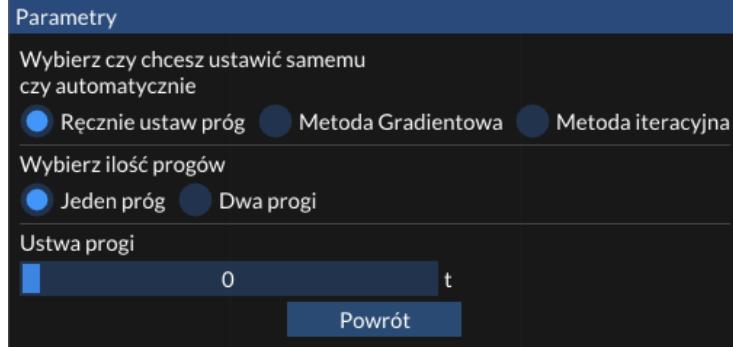
Rysunek 7: Kilka parametrów

Na początku metody pomocniczej znajduje się pierwszy wybór (metody), obsługiwany jest przez funkcję **ImGui::RadioButton()**. Funkcja ta przyjmuje jako parametry:

- **const char \*label** nazwa opcji
- **int \*v** wskaźnik do zmiennej z obecnie wybranym stanem
- **int v \_button** numer stanu przypisany do opcji (najlepiej zdefiniowany w enum)

**ImGui::SameLine()** informuje, żeby umieścić elementy w tej samej linii. **ImGui::Separator()** tworzy linię separującą elementy. Następnie znajduje się kolejny wybór (ilości progów), zrobiony analogicznie do wyboru metody. Jedak pojawi się on tylko wtedy gdy ustawiamy progi ręcznie. Również, jeżeli ustawiamy progi ręcznie pojawią się dwa slider'y do wyboru wartości progu.

Okienko będzie wyglądało tak:



Rysunek 8: Okienko kilku parametrów

### 3.2.3 Parametry tabelkowe

Przy niektórych algorytmach parametrem jest macierz (maska albo element strukturalny).

```
void App::DrawLinearInputArray()
{
    if (ImGui::BeginTable("Maska", params.linearFilterSize, ImGuiTableFlags_Borders))
    {
        for (int row = 0; row < params.linearFilterSize; row++)
        {
            ImGui::PushID(row);
            ImGui::TableNextRow();
            for (int col = 0; col < params.linearFilterSize; col++)
            {
                ImGui::TableSetColumnIndex(col);
                ImGui::PushItemWidth(ARRAY_INPUT_WIDTH);
                std::string s = "#" + std::to_string(col);
                if (params.linearFilterSize == Algorithms::MatrixSize::S3x3)
                    ImGui::InputInt(s.c_str(), &params.linearMask3x3[row][col]);
                else if (params.linearFilterSize == Algorithms::MatrixSize::S5x5)
                    ImGui::InputInt(s.c_str(), &params.linearMask5x5[row][col]);
                else
                    ImGui::InputInt(s.c_str(), &params.linearMask7x7[row][col]);
            }
            ImGui::PopID();
        }
        ImGui::EndTable();
    }
}
```

Rysunek 9: Tabelka w kodzie

Aby rozpocząć tabelkę, należy użyć funkcji `ImGui::BeginTable()`, i musi się ona znajdować w instrukcji warunkowej. Funkcja ta przyjmuje następujące parametry:

- `const char *str_id` nazwa tablicy
- `int columns` ilość kolumn

- **ImGuiTableFlags flags** flagi dotyczące wyglądu tablicy
- **const ImVec2 &outer\_size** rozmiar tablicy
- **float inner\_width** wewnętrzna szerokość komórek

W naszym przypadku interesować będą nas pierwsze 3-4 parametry.

Następnie należy w pętli podwójnej utworzyć wszystkie elementy.

W zewnętrznej pętli (dla wierszy) używamy ImGui::TableNextRow() aby przejść do kolejnego wiersza. Następnie w wewnętrznej pętli (dla kolumn) ustawiamy kurSOR na kolumnę przy użyciu ImGui::TableSetColumnIndex(col) i dodajemy element.

Aby dodać nowy element, można użyć różnych elementów na przykład:

- ImGui::Text() - do wyświetlenia elementu bez możliwości wprowadzenia.
- ImGui::InputInt() - do wprowadzenia liczby całkowitej z powiązaniem do elementu o odpowiednim indeksie.
- ImGui::Checkbox() - do oznaczenia elementu, którego wartości mogą przyjmować true/false.

Tak będą wyglądać tablice w kolejności takiej, jak na liście:

**Parametry**

Wybierz rozmiar filtru

3x3  5x5  7x7

---

Wybierz rodzaj filtru

Uśredniający  Gauss  Sobel Poziomy  
 Sobel Pionowy  Laplasjan  Wyostrzający  
 Własna

---

0	0	1	2	1	0	0
0	3	13	22	13	3	0
1	13	59	97	59	13	1
2	22	97	159	97	22	2
1	13	59	97	59	13	1
0	3	13	22	13	3	0
0	0	1	2	1	0	0

**Powrót**

Rysunek 10: Tablica z samym wyświetlaniem zawartości

**Parametry**

Wybierz rozmiar filtru

3x3  5x5  7x7

---

Wybierz rodzaj filtru

Uśredniający  Gauss  Sobel Poziomy  
 Sobel Pionowy  Laplasjan  Wyostrzający  
 Własna

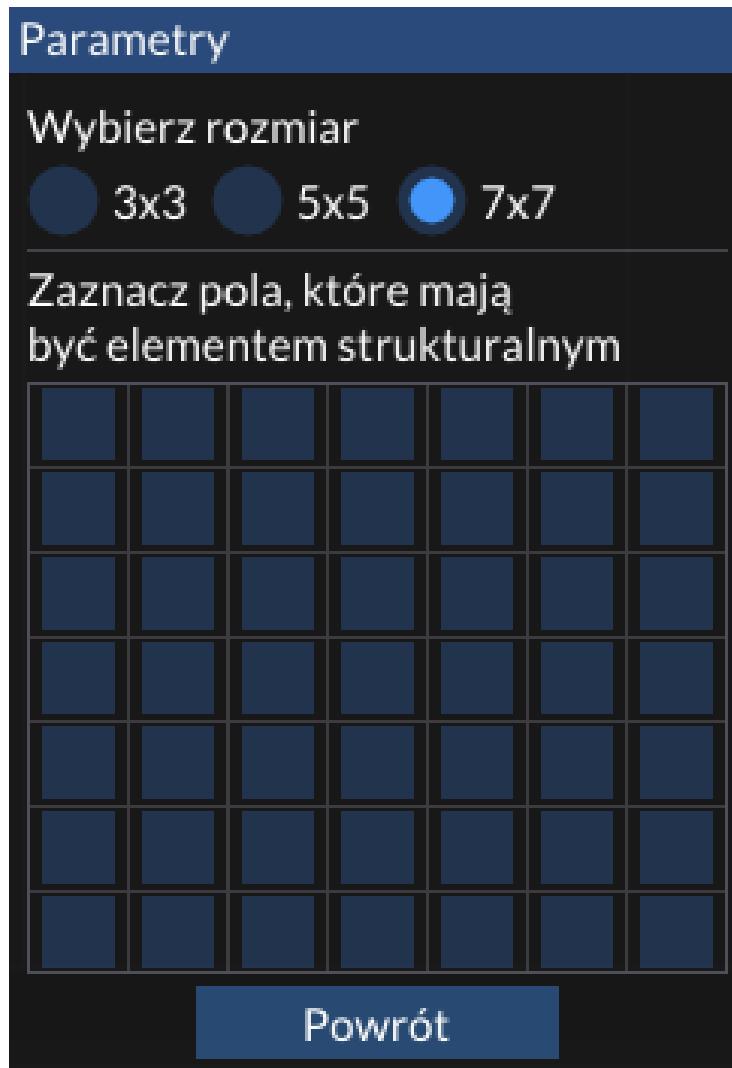
---

Ustaw własną maskę

0	-	+	0	-	+	1	-	+	2	-	+	1	-	+	0	-	+	0	-	+
0	-	+	3	-	+	13	-	+	22	-	+	13	-	+	3	-	+	0	-	+
1	-	+	13	-	+	59	-	+	97	-	+	59	-	+	13	-	+	1	-	+
2	-	+	22	-	+	97	-	+	159	-	+	97	-	+	22	-	+	2	-	+
1	-	+	13	-	+	59	-	+	97	-	+	59	-	+	13	-	+	1	-	+
0	-	+	3	-	+	13	-	+	22	-	+	13	-	+	3	-	+	0	-	+
0	-	+	0	-	+	1	-	+	2	-	+	1	-	+	0	-	+	0	-	+

**Powrót**

Rysunek 11: Tablica z wprowadzaniem liczb całkowitych



Rysunek 12: Tablica z checkboxami

Na koniec tablice należy zakończyć używając ImGui::EndTable().  
**Dodatkowe uwagi:**

- Jeżeli tablica ma służyć do wprowadzania danych należy użyć **ImGui::PushID(nr wiersza)** - przed ImGui::TableNextRow() i **ImGui::PopID()** - po wewnętrznej pętli.
- Jeżeli ustawiamy ID należy najpierw użyć **std::string s = "##" + std::to\_string(col)**, aby ustalić to ID, a następnie przekazać je do funkcji od elementu na przykład **ImGui::Checkbox(s.c\_str(), &a3x3[row][col])**.

- Jeżeli nie potrzebujemy opisów kolumn, można użyć **ImGui::TableSetupColumn("##")** dla każdej kolumny.
- Jeżeli chcemy ustawić szerokość elementu użyjemy **ImGui::TableSetupColumn("##", ImGuiTableColumnFlags\_WidthFixed, ARRAY\_ITEM\_WIDTH)**  
Tablica powinna mieć podany stały rozmiar w **ImGui::BeginTable()**.
- Jeżeli tablica nie ma ustawionego rozmiaru na początku można określić szerokość elementu możemy użyć **ImGui::PushItemWidth(ARRAY\_INPUT\_WIDTH)**.
- Metoda **DrawInputArray()**, może być wykorzystywana, jeżeli chcemy utworzyć kolejne tablice z checkboxami.
- **Jeżeli fragment kodu od obsługi parametrów jest zbyt długi, można go przenieść do metody pomocniczej.**

### 3.3 LaunchAlgorithms

W tej metodzie należy dodać kolejny przypadek analogicznie do już istniejących. Można też wywołać funkcje w wątku głównym, natomiast wtedy nie będzie możliwości przerwania, a interfejs nie będzie odpowiadał. Należy też użyć **Mutex::GetInstance().ThreadStopped()**, aby popup pokazujący się w trakcie wykonywania nie wyświetlał.

```

void App::LaunchAlgorithms()
{
    ... // if new needed just add a new one
    switch (algorithmSelected)
    {
        case Negative:
            algorithmThread = std::thread(&Algorithms::CreateNegative, &outputImage);
            break;
        case Brighten:
            algorithmThread = std::thread(&Algorithms::BrightenImage, &outputImage, &params);
            break;
        case Contrast:
            algorithmThread = std::thread(&Algorithms::Contrast, &outputImage, &params);
            break;
        case Exponentiation:
            algorithmThread = std::thread(&Algorithms::Exponentiation, &outputImage, &params);
            break;
        case LeveledHistogram:
            algorithmThread = std::thread(&Algorithms::LevelHistogram, &outputImage);
            break;
        case Binarization:
            algorithmThread = std::thread(&Algorithms::Binarization, &outputImage, &params);
            break;
        case LinearFilter:
            algorithmThread = std::thread(&Algorithms::LinearFilter, &outputImage, &params);
            break;
        case MedianFilter:
            algorithmThread = std::thread(&Algorithms::MedianFilter, &outputImage, &params);
            break;
        case Erosion:
            algorithmThread = std::thread(&Algorithms::Erosion, &outputImage, &params);
            break;
        case Dilatation:
            algorithmThread = std::thread(&Algorithms::Dilatation, &outputImage, &params);
            break;
        case Skeletonization:
            algorithmThread = std::thread(&Algorithms::Skeletonization, &outputImage);
            break;
        case Hough:
            algorithmThread = std::thread(&Algorithms::Hough, &outputImage, &params);
            break;
        default:
            break;
    }
}

```

Rysunek 13: Metoda LaunchAlgorithms

### 3.4 ResetParameters

W tej metodzie należy ustawić parametry na wartości domyślne (ustalone w strukturze z parametrami), jeżeli algorytm posiada parametry.

```

void App::ResetParameters()
{
    params.value = 0;
    params.contrast = 1.0;
    params.alfa = 1.0;
    params.boundCount = 1;
    params.lowerBound = 0;
    params.upperBound = 0;
    params.method = Algorithms::BinarizationMethod::None;
    // linear
    params.linearFilterS = Algorithms::LinearFilters::Average;
    params.linearFilterSize = Algorithms::MatrixSize::S3x3;
    params.linearMask3x3 = AVERAGE_3x3;
    params.linearMask5x5 = AVERAGE_5x5;
    params.linearMask7x7 = AVERAGE_7x7;
    // median
    params.medianFilterS = Algorithms::MedianFilters::Full;
    params.medianFilterSize = Algorithms::MatrixSize::S3x3;
    params.medianMask3x3 = MEDIAN_3x3;
    params.medianMask5x5 = MEDIAN_5x5;
    params.medianMask7x7 = MEDIAN_7x7;
    // erosion
    params.erosionElementSize = Algorithms::MatrixSize::S3x3;
    params.erosionElement3x3 = EMPTY_3x3;
    params.erosionElement5x5 = EMPTY_5x5;
    params.erosionElement7x7 = EMPTY_7x7;
    // dilatation
    params.dilatationElementSize = Algorithms::MatrixSize::S3x3;
    params.dilatationElement3x3 = EMPTY_3x3;
    params.dilatationElement5x5 = EMPTY_5x5;
    params.dilatationElement7x7 = EMPTY_7x7;
    // Hough
    params.maxIndexRow = 0;
    params.maxIndexTheta = 0;
    params.maxHoughVal = 0;
}

```

Rysunek 14: Metoda ResetParameters

## 4 Algorithms.hpp

Tutaj należy zadeklarować nową funkcję dla kolejnego algorytmu. Tutaj również znajduje się definicja struktury z parametrami, więc jeżeli trzeba dodać nowe, to również tutaj się znajdują. Można też dodać pomocniczy enum, jeżeli będzie potrzebny.

**Najlepiej zacząć dodanie nowego algorytmu od tego pliku.**

```
..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 15: Deklaracje funkcji

```
.. enum BinarizationMethod
.. {
..     None,
..     Gradient,
..     Histogram
.. };

.. enum LinearFilters
.. {
..     Average,
..     Gauss,
..     SobelHorizontal,
..     SobelVertical,
..     Laplasjan,
..     Sharpening,
..     CustomL
.. };

.. enum MedianFilters
.. {
..     Full,
..     Cross,
..     CustomM
.. };

.. enum MatrixSize
.. {
..     S3x3 = 3,
..     S5x5 = 5,
..     S7x7 = 7
.. };
```

```

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 17: Struktura z parametrami

## 5 Algorithms.cpp

Tutaj należy zdefiniować nowy algorytm. Należy pamiętać, że domyślnie powinien się wykonywać w oddzielnym wątku. Natomiast można funkcje wywołać w wątku głównym.

Należy zwrócić uwagę na:

### 5.1 Lokalna kopia obrazu i parametrów

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

Rysunek 18: Kopiowanie potrzebnych zasobów

### 5.2 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 19: Zatrzymanie i skopiowanie wyników

Może 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.

### 5.3 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 20: 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 21: Zatrzymanie i ręczne kopiowanie

## 6 ImGui

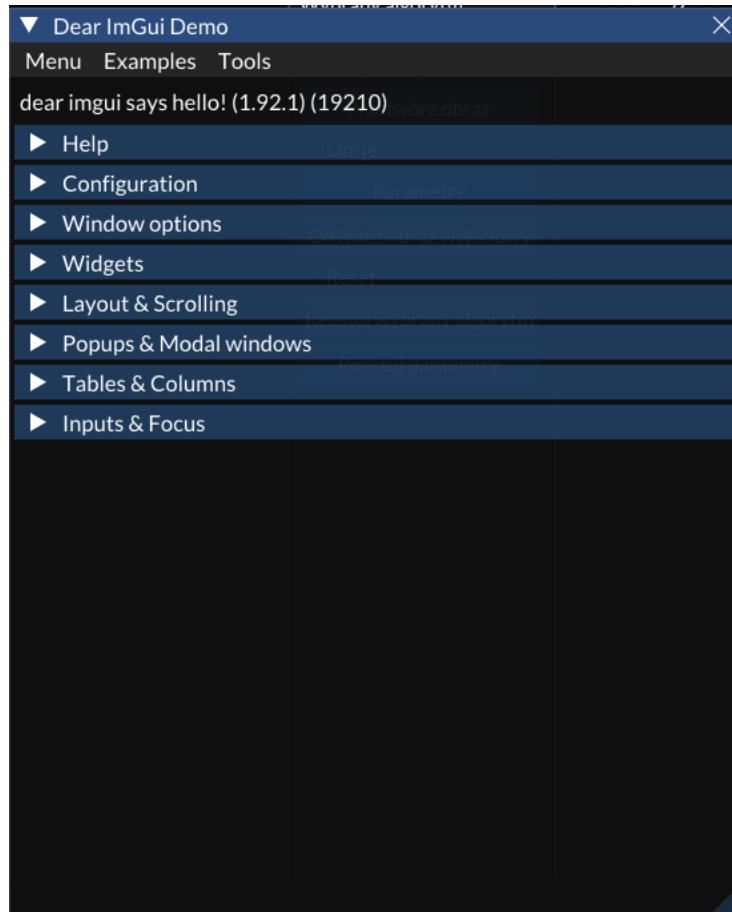
W tej sekcji opisze przydatne informacje dotyczące użytej biblioteki graficznej.

## 6.1 ImGuiDemo

Do programu dołączone jest ImGuiDemo, które zawiera przykłady możliwości biblioteki graficznej.

Znajduje się ono w sekcji **Pomoc->Pokaż ImGuiDemo**

Okno wygląda tak:



Rysunek 22: Okno ImGuiDemo

Przykłady w kodzie znajdują się w pliku **imgui\_demo.cpp**. Plik ten znajduje się w folderze ImGui. W tym pliku powinno znajdować się wszystko, co może być potrzebne przy zapoznawaniu się z biblioteką. Ponadto twórcy ImGui zachęcają do zachowania tego pliku w projektach w celu posiadania przykładów implementacji funkcjonalności.



Rysunek 23: Folder ImGui

## 6.2 Link do biblioteki

Na końcu zostawiam link do Githuba, z biblioteką. Znajdują się tam wszystkie potrzebne informacje.

**Github:** <https://github.com/ocornut/imgui>