Przetwarzanie obrazów binarnych

—

Łukasz Ważny

Założenia projektu

- obraz binarny (czarny piksel - wartość 0, biały - wartość 255)
- obraz ulega przetworzeniu za pomocą algorytmu dylatacji
- biblioteka odpowiada jedynie za sam algorytm (reszta w c#)
- główny szkielet programu w c#, algorytm w c oraz assembler

Algorytmy morfologiczne obrazów binarnych

- dylatacja
- erozja
- otwarcie
- domknięcie
- hit-or-miss
- pocienianie
- pogrubianie
- transformata odległościowa
- top hat
- bottom hat

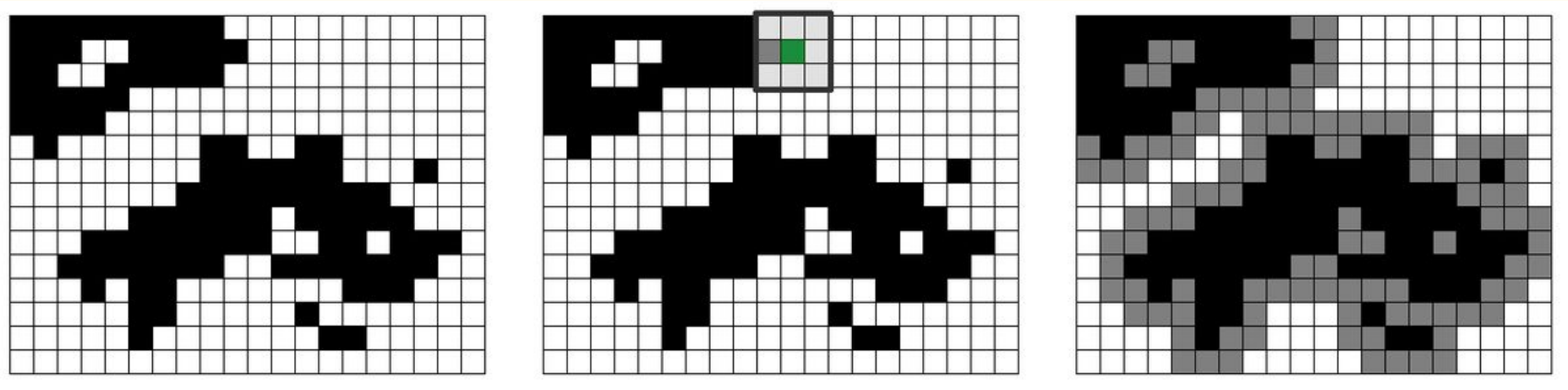
Dylatacja

$$I \oplus B \equiv \{p \in U_1 | \tilde{B} \cap I \neq \emptyset\}$$

$$I \oplus B \equiv \bigcup_{p \in B} (I + p).$$

$I(x, y) \in \{0, 1\}$ $U_1 = \{p | I(p) = 1\}$ $p = (x, y),$ ***B*** - element strukturalny

Dylatacja



Przykłady

Element strukturalny: 3x3
Punkt centralny: x = 1, y = 1

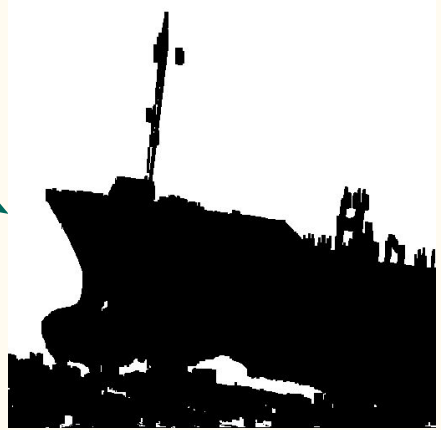
Element strukturalny: 6x6
Punkt centralny: x = 3, y = 3

Element strukturalny: 3x3
Punkt centralny: x = 0, y = 0

Przykłady

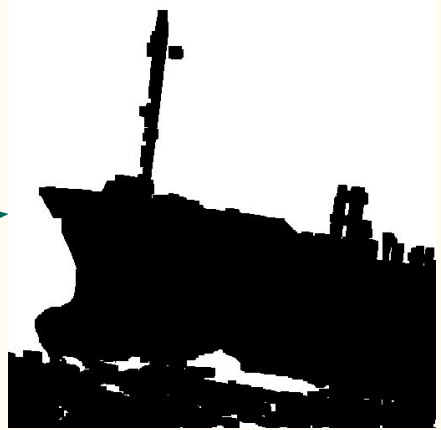
Element strukturalny: 3x15
Punkt centralny: x = 7, y = 1

Element strukturalny: 15x3
Punkt centralny: x = 1, y = 7

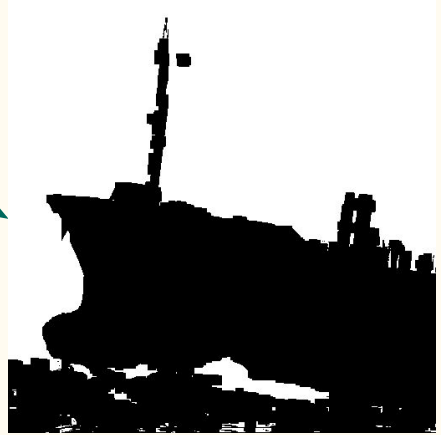


Przykłady

Element strukturalny: 10x10
Punkt centralny: x = 0, y = 0



Element strukturalny: 10x10
Punkt centralny: x = 9, y = 9

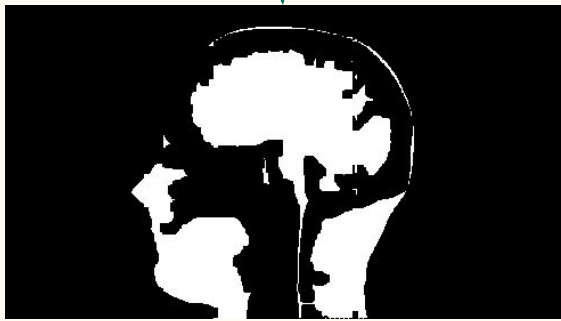


Przykłady

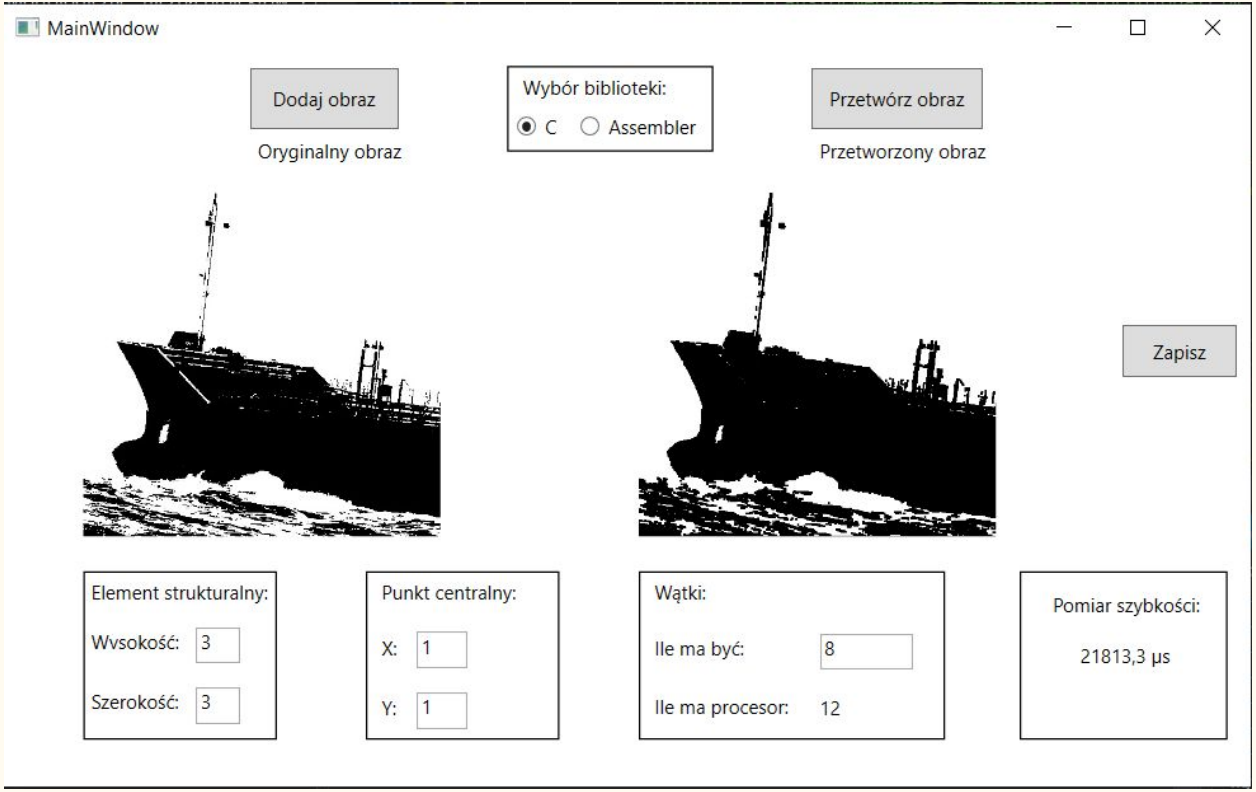
Element strukturalny: 5x5
Punkt centralny: x = 2, y = 2

Element strukturalny: 7x7
Punkt centralny: x = 3, y = 3

Element strukturalny: 3x3
Punkt centralny: x = 1, y = 1

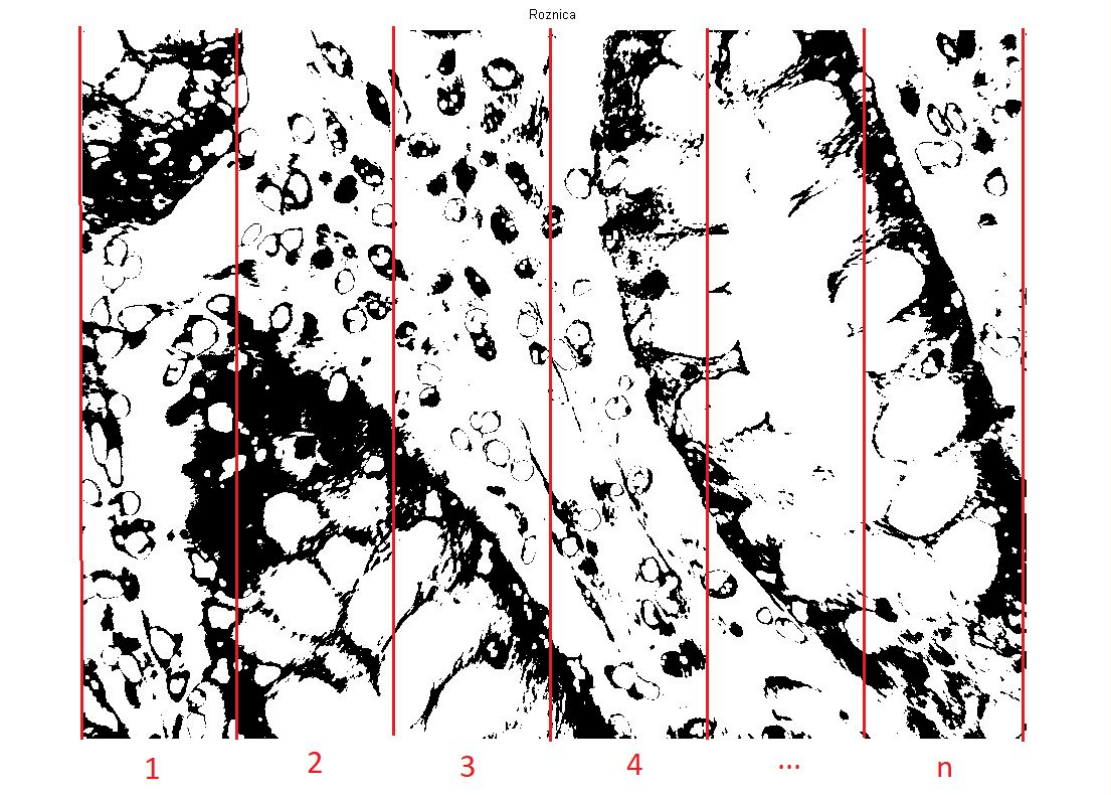


GUI (WPF)


MainWindow
Dodaj obraz
Wybór biblioteki:
C
Assembler
Przetwórz obraz
Oryginalny obraz
Przetworzony obraz
Zapisz
Element strukturalny:
Wysokość:
3
Szerokość:
3
Punkt centralny:
X:
1
Y:
1
Wątki:
Ile ma być:
8
Ile ma procesor:
12
Pomiar szybkości:
21813,3 µs

Wątki

Dla n wątków:



Wątki

```
public ThreadsManaging(BitmapImage image, Params parameters)
{
    this.Threads = new List<Thread>();
    this.transformedImages = new List<IntPtr>();
    this.parameters = parameters;
    this.imagesIntPtr = SplitImage(image, parameters.NrOfThreads);
    this.imWithIndices = new List<ImageWithIndex>();
    for (int i = 0; i < parameters.NrOfThreads; i++)
    {
        unsafe
        {
            transformedImages.Add(new IntPtr(null));
        }
        imWithIndices.Add(new ImageWithIndex(imagesIntPtr[i].GetPtr(), i, imagesIntPtr[i].GetWidth()));
        Thread t = new Thread(performOperation)
        {
            Name = "" + i
        };
        Threads.Add(t);
    }

}
```

Wątki (mała dygresja)

```
public void performOperation(Object image)
{
    try
    {
        ImageWithIndex im = (ImageWithIndex) image;
        //IntPtr transformedImage = Marshal.AllocHGlobal(im.image);
        IntPtr transformedImage;
        //c function
        if (!(bool)parameters.Function)
        {
            transformedImage = dilatationC(im.image, im.width, parameters.ImageHeight,
            parameters.ElemWidth, parameters.ElemHeight, parameters.CentrPntX, parameters.CentrPntY);
            this.transformedImages[im.index] = transformedImage;
        }
        //asm function
        else
        {
            transformedImage = dilatationAsm(im.image, im.width, parameters.ImageHeight,
                parameters.ElemWidth, parameters.ElemHeight, parameters.CentrPntX, parameters.CentrPntY);
        }

        Marshal.FreeHGlobal(im.image);
    }
```

Wątki (mała dygresja)

```
[DllImport(@"D:\studia\JAproj\OperacjeMorfologiczne\c_function\CFunction.dll", CallingConvention = CallingConvention.Cdecl)]
1 odwołanie
private static extern IntPtr dilatationC(IntPtr image, int imageWidth, int imageHeight, int elemWidth, int elemHeight,
    int centrPntX, int centrPntY);

[DllImport(@"D:\studia\JAproj\OperacjeMorfologiczne\asm_function\asm_function.dll")]
Odwołania: 2
private static extern IntPtr dilatationAsm(IntPtr image, int imageWidth, int imageHeight, int elemWidth, int elemHeight,
    int centrPntX, int centrPntY);
```

Wątki

```
public static List<IntPtrWithSize> SplitImage(BitmapImage image, int n)
{
    int width = image.PixelWidth / n;
    Bitmap bmpImage = Converter.BitmapImage2Bitmap(image);
    List<IntPtrWithSize> result = new List<IntPtrWithSize>();
    for (int i = 0; i < n; i++){
        int widthOfThisPart = width;
        if (i == n - 1)
        {
            int extraWidth = image.PixelWidth - (width * n);
            widthOfThisPart += extraWidth;
        }
        Rectangle cropArea = new Rectangle(0 + i * width, 0, widthOfThisPart, image.PixelHeight);

        Bitmap bmpCrop = bmpImage.Clone(cropArea,
            bmpImage.PixelFormat);

        byte[] bmpBytes = Converter.BitmapToBytes(bmpCrop);
        int size = Converter.GetBytesSize(bmpBytes);
        IntPtr cropIntPtr = Converter.ByteToIntPtr(Converter.BitmapToBytes(bmpCrop));

        IntPtrWithSize intPtrWithSize = new IntPtrWithSize(cropIntPtr, size, widthOfThisPart);
        result.Add(intPtrWithSize);
    }
    return result;
}
```

Wątki

```
public static BitmapImage MergeImage(List<IntPtrWithSize> images, int heightOfOneImage) {
    List<byte[]> imageBitmapsBytes = new List<byte[]>();
    int width = 0;
    foreach (var image in images)
    {
        byte[] imageBytes = Converter.IntPtrToBytes(image.GetPtr(), image.GetSize());
        imageBitmapsBytes.Add(imageBytes);
        width += image.GetWidth();
    }


    byte[] finalImageBytes = new byte[heightOfOneImage * width];

    for (int i = 0; i < heightOfOneImage; i++) {
        for(int j = 0; j < images.Count; j++)
        {
            Buffer.BlockCopy(imageBitmapsBytes[j], images[j].GetWidth() * i, finalImageBytes, width * i + j * images[0].GetWidth(),
                images[j].GetWidth());
        }
    }

    Bitmap bitmapImage = Converter.BytesToBitmap(finalImageBytes, width, heightOfOneImage);

    return Converter.Bitmap2BitmapImage(bitmapImage);
}
```

Pomiar szybkości

```
ThreadsManaging action = new ThreadsManaging(_originalImage, parameters);

//pomiar czasu
Stopwatch watch = new Stopwatch();
watch.Start();
//operation
action.start();
watch.Stop();
speed.Text = ((((double) watch.ElapsedTicks) / ((double) Stopwatch.Frequency)) * 1000 * 1000) +
             " \u00b5s";
```

```
public void start() {
    for (int i = 0; i < parameters.NrOfThreads; i++) {
        Threads[i].Start(imWithIndices[i]);
    }
    for (int i = 0; i < parameters.NrOfThreads; i++) {
        Threads[i].Join();
    }
}
```

Kod w C

```c
EXPORT unsigned char* dilatationC(const unsigned char* image, int imageWidth, int imageHeight, int elemWidth, int elemHeight, int centrPntX, int centrPntY){

    unsigned char* buffer = (unsigned char*)malloc(imageHeight*imageWidth * sizeof(unsigned char));

    //kopiuje dane z image do bufora
    for (int j = 0; j < imageHeight - 1; j++)
    {
        for (int i = 0; i < imageWidth; i++)
        {
            buffer[i + j * imageWidth] = image[i + j * imageWidth];
        }
    }

    for (int h = 0; h < imageHeight; h++)
    {
        for (int w = 0; w < imageWidth; w++)
        {
            //sprawdzam, czy w obrobie elementu strukturalnego sa piksele o wartosci < 10, jeloli tak - piksel o wsp. w i h,
            //przyjmuje wartość 0
            if (h <= centrPntY && w <= centrPntX)
            {
                int czy_jest = 0;
                for (int j = 0; j < elemHeight - h && h + j < imageHeight && !czy_jest; j++)
                {
                    for (int i = 0; i < elemWidth - w && w + i < imageWidth && !czy_jest; i++)
                    {
                        if (image[i + j * imageWidth] < 10)
                        {
                            czy_jest = 1;
                            buffer[w + h * imageWidth] = 0;
                        }
                    }
                }
            }
            else
            {
                if (h <= centrPntY)
                {
                    int czy_jest = 0;
                    for (int j = 0; j < elemHeight - h && h + j < imageHeight && !czy_jest; j++)
                    {
                        for (int i = 0; i < elemWidth && w + i < imageWidth && !czy_jest; i++)
                        {
                            if (image[w - centrPntX + i + j * imageWidth] < 10)
                            {
                                czy_jest = 1;
                                buffer[w + h * imageWidth] = 0;
                            }
                        }
                    }
                }
                else if (w <= centrPntX)
                {
                    int czy_jest = 0;
                    for (int j = 0; j < elemHeight && h + j < imageHeight && !czy_jest; j++)
                    {
                        for (int i = 0; i < elemWidth - w && w + i < imageWidth && !czy_jest; i++)
                        {
                            if (image[h * imageWidth - imageWidth * centrPntY + i + j * imageWidth] < 10)
                            {
                                czy_jest = 1;
                                buffer[w + h * imageWidth] = 0;
                            }
                        }
                    }
                }
                else
                {
                    int czy_jest = 0;
                    for (int j = 0; j < elemHeight && h + j < imageHeight && !czy_jest; j++)
                    {
                        for (int i = 0; i < elemWidth && w + i < imageWidth && !czy_jest; i++)
                        {
                            if (image[w + h * imageWidth - imageWidth * centrPntY - centrPntX + i + j * imageWidth] < 10)
                            {
                                czy_jest = 1;
                                buffer[w + h * imageWidth] = 0;
                            }
                        }
                    }
                }
            }
        }
    }
    return buffer;
}
```

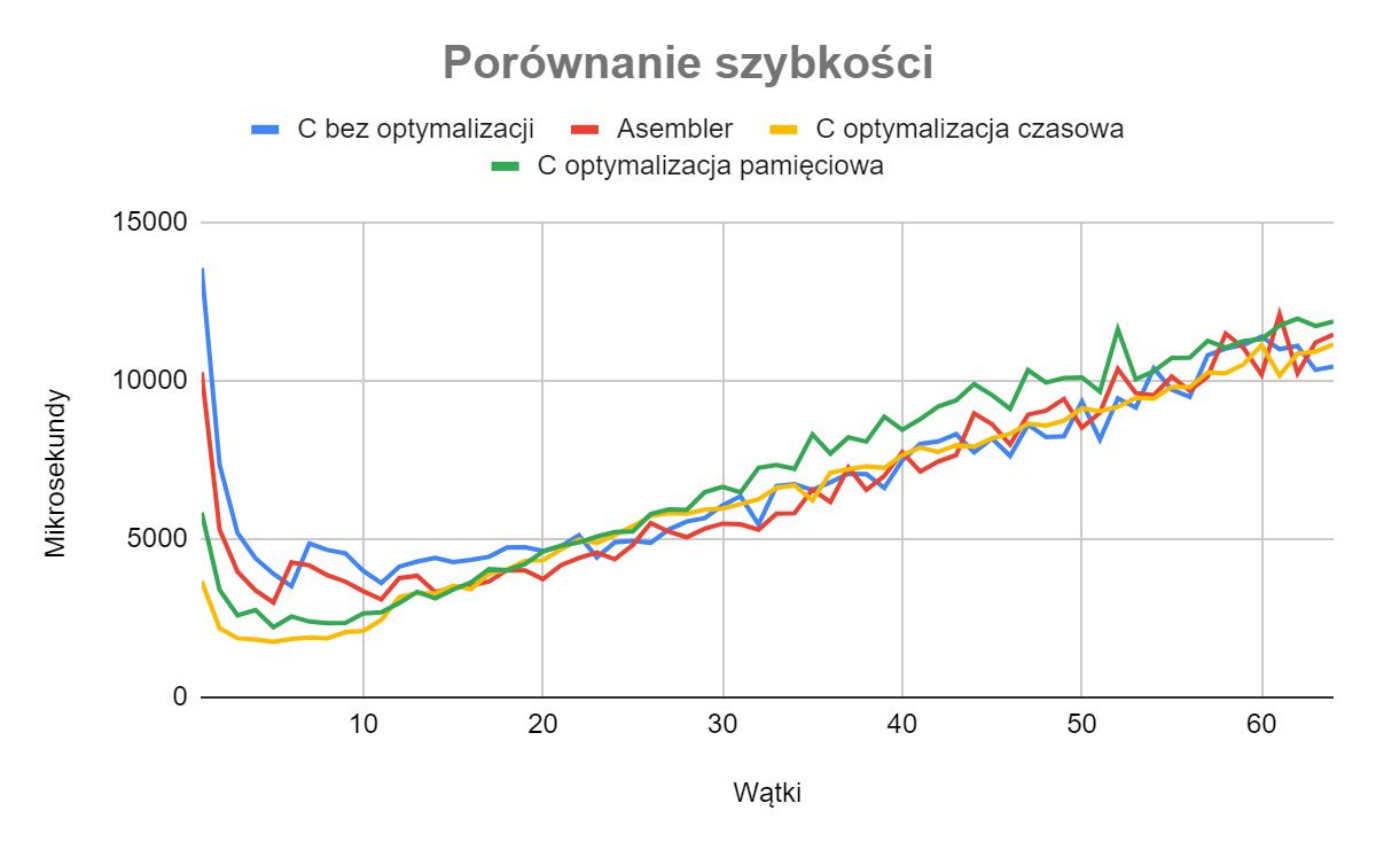
```c
int czy_jest = 0;
for (int j = 0; j < elemHeight && h + j < imageHeight && !czy_jest; j++)
{
    for (int i = 0; i < elemWidth && w + i < imageWidth && !czy_jest; i++)
    {
        if (image[w + h * imageWidth - imageWidth * centrPntY - centrPntX + i + j * imageWidth] < 10)
        {
            czy_jest = 1;
            buffer[w + h * imageWidth] = 0;
        }
    }
}
```

Fragment kodu w Asm

```
;kopiowanie argumentów do rejestrów
movd xmm0, rcx                  ;skopiuj 1. argument - src - do rejestru xmm0
mov rax, rdx                    ;skopiuj 2. argument - imageWidth - do rejestru rax
movd xmm1, eax                  ;skopiuj 2. argument - imageWidth - do rejestru xmm1
mov rax, r8                     ;skopiuj 3. argument - imageHeight - do rejestru rax
movd xmm2, eax                  ;skopiuj 3. argument - imageHeight - do rejestru xmm2
movd xmm3, r9                   ;skopiuj 4. argument - elemWidth - do rejestru xmm3
movd xmm4, dword ptr[rsp+40]    ;skopiuj 5. argument - elemHeight - ze stosu do rejestru xmm4
movd xmm5, dword ptr[rsp+48]    ;skopiuj 6. argument - centrPntX - ze stosu do rejestru xmm5
movd xmm6, dword ptr[rsp+56]    ;skopiuj 7. argument - centrPntY - ze stosu do rejestru xmm6

;alokacja pamięci na bufor obrazu docelowego
movsd xmm7, xmm1                ;skopiuj imageWidth do xmm7
pmuludq xmm7, xmm2              ;pomnóż imageWidth * imageHeight
movd eax, xmm7                  ;wynik mnożenia do eax
cdqe                            ;podwojenie
mov rdi, rax                    ;ilość pamięci do zaalokowania wrzucamy do rejestru rdi
extern malloc : PROC
call malloc                     ;alokacja pamięci
movd xmm7, rax                  ;kopiuj wskaźnik do zaalokowanej pamięci do xmm7
```

Testowanie


Porównanie szybkości
C bez optymalizacji
Asembler
C optymalizacja czasowa
C optymalizacja pamięciowa
15000
10000
5000
0
Mikrosekundy
10
20
30
40
50
60
Wątki