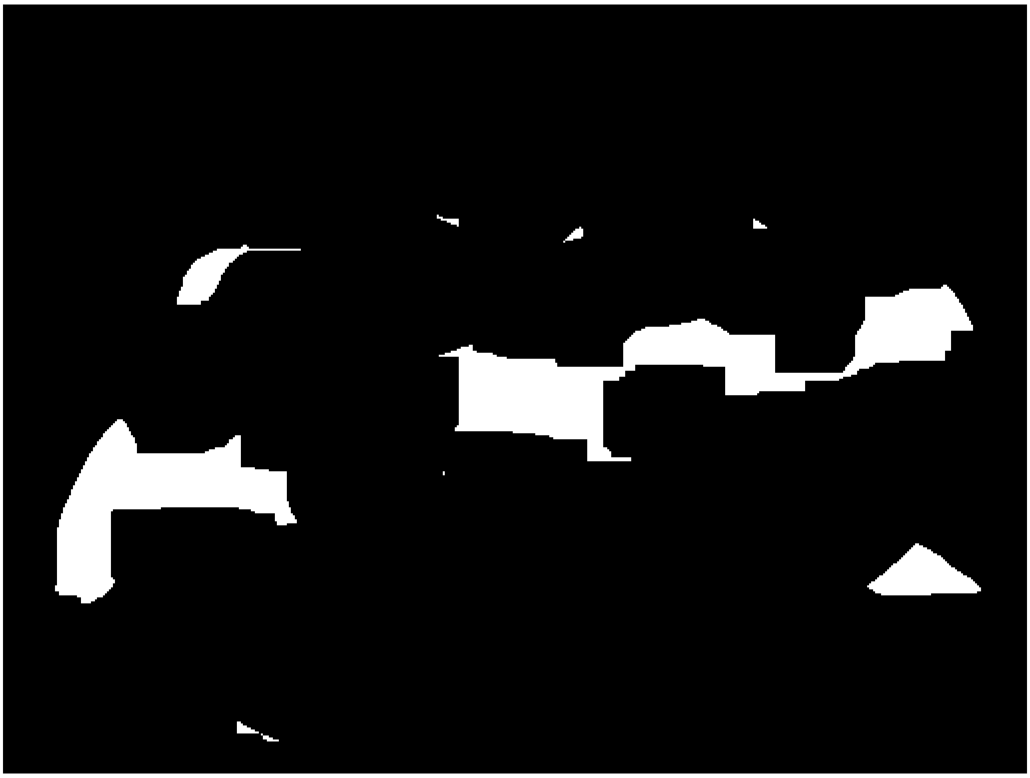
Operacje kontekstowe

Akceleracja Algorytmów Wizyjnych

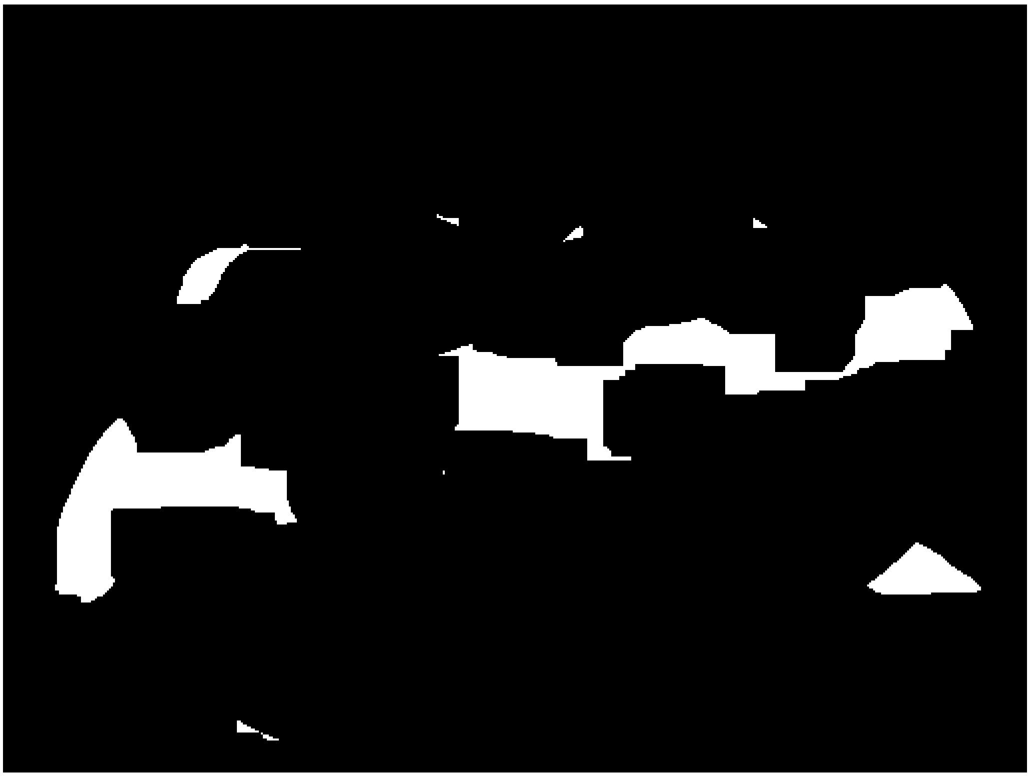
Jan Rosa 13.03.2025

Obraz wejściowy

Obrazy wyjściowe Erozja i Dylatacja



Drawing 1: Erozja Kwadrat 33x33



Drawing 2: Erozja Kwadrat 33x33 - Matlab

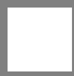
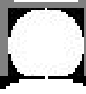
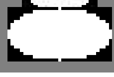

Drawing 3: Erozja Elipsą


Drawing 4: Erozja Elipsą - Matlab

Drawing 5: Dylatacja Elipsą

Drawing 6: Dylatacja Elipsą - MatLab

Kernele operacji morfologicznych

Typ Kernela	Obraz Kernela
Kwadrat 3x3	
Kwadrat 33x33	
Koło 33x33	
Elipsa 33x17	

Kernel OpenCL

Erozja dowolnym prostokątem

```
__kernel void sobel_filter(__read_only image2d_t inputImage, __write_only image2d_t outputImage)
{
	const int2 coord = (int2)(get_global_id(0), get_global_id(1));

	float treshold = 70.0;
	int4 binarized = (int4)(0);

	const float4 conv_val = (float4){0.2989, 0.5870, 0.1140, 0.0};

	int2 square_dim = (int2)(3, 3);

	uint4 eroded_pixel = (uint4)1;
	float4 pixel = (float4)(0);
	float pixel_gray = 0;
	//if(coord.x == 50 && coord.y == 50 )
	{
		for(int row = (coord.x) - (square_dim.x/(int)2); row <= (coord.x) + (square_dim.x/(int)2); ++row){
			for(int col = coord.y - (square_dim.y/(int)2); col <= coord.y + (square_dim.y/(int)2); ++col){
				pixel = convert_float4(read_imageui(inputImage, imageSampler, (int2)(row, col)));
				pixel_gray = dot(pixel, conv_val);
				binarized = step(treshold, pixel_gray);
				eroded_pixel &= convert_uint4(binarized);

			}
			// printf("\n");
		}
		//printf("\n\r\n\r");
```

```
        }

        // write_imageui(outputImage, coord, convert_uint4(pixel));
        write_imageui(outputImage, coord, UCHAR_MAX*eroded_pixel);

}
```

Erozja dowolną elipsą

```
__kernel void sobel_filter(__read_only image2d_t inputImage, __write_only image2d_t
outputImage)
{
        const int2 coord = (int2)(get_global_id(0), get_global_id(1));

        float treshold = 70.0;
        int4 binarized = (int4)(0);

        const float4 conv_val = (float4){0.2989, 0.5870, 0.1140, 0.0};

        int2 square_dim = (int2)(10, 20);

        uint4 eroded_pixel = (uint4)1;
        float4 pixel = (float4)(0);
        float pixel_gray = 0;
        //if(coord.x == 50 && coord.y == 50 )
        {
                for(int row = (coord.x) - (square_dim.x/(int)2); row <= (coord.x) +
(square_dim.x/(int)2); ++row){
                        for(int col = coord.y - (square_dim.y/(int)2); col <= coord.y +
(square_dim.y/(int)2); ++col){
                                int x2 = (row - coord.x)*(row - coord.x);
                                int a2 = (square_dim.x*square_dim.x/4);
                                int y2 = (col - coord.y)*(col - coord.y);
                                int b2 = (square_dim.y*square_dim.y/4);
                                if(      ((float)x2)/((float)a2) + ((float)y2)/((float)b2) <= 1)
```

```
                    {
                        pixel = convert_float4(read_imageui(inputImage, 
imageSampler, (int2)(row, col)));
                        pixel_gray = dot(pixel, conv_val);
                        binarized = step(treshold, pixel_gray);
                        eroded_pixel &= convert_uint4(binarized);
                        //printf("row = %x, col = %x; \n\r", row, col);
                        // printf("8");
                    }
                    else
                    {
                        // printf(" ");
                    }
                }
                // printf("\n");
            }
            //printf("\n\r\n\r");
        }

        // write_imageui(outputImage, coord, convert_uint4(pixel));
        write_imageui(outputImage, coord, UCHAR_MAX*eroded_pixel);

}
```

Dylatacja dowolną elipsą

```
__kernel void sobel_filter(__read_only image2d_t inputImage, __write_only image2d_t 
outputImage)
{
        const int2 coord = (int2)(get_global_id(0), get_global_id(1));

        float treshold = 70.0;
        int4 binarized = (int4)(0);
```

```
        const float4 conv_val = (float4){0.2989, 0.5870, 0.1140, 0.0};


        int2 square_dim = (int2)(10, 20);


        uint4 dilated_pixel = (uint4)0;
        float4 pixel = (float4)(0);
        float pixel_gray = 0;
        //if(coord.x == 50 && coord.y == 50 )
        {
                for(int row = (coord.x) - (square_dim.x/(int)2); row <= (coord.x) +
(square_dim.x/(int)2); ++row){
                        for(int col = coord.y - (square_dim.y/(int)2); col <= coord.y +
(square_dim.y/(int)2); ++col){
                                int x2 = (row - coord.x)*(row - coord.x);
                                int a2 = (square_dim.x*square_dim.x/4);
                                int y2 = (col - coord.y)*(col - coord.y);
                                int b2 = (square_dim.y*square_dim.y/4);
                                if(     ((float)x2)/((float)a2) + ((float)y2)/((float)b2) <= 1)
                                {
                                        pixel = convert_float4(read_imageui(inputImage,
imageSampler, (int2)(row, col)));
                                        pixel_gray = dot(pixel, conv_val);
                                        binarized = step(treshold, pixel_gray);
                                        dilated_pixel |= convert_uint4(binarized);
                                        //printf("row = %x, col = %x; \n\r", row, col);
                                        // printf("8");
                                }
                                else
                                {
                                        // printf(" ");
                                }
                        }
                        // printf("\n");
```

```
        }
        //printf("\n\r\n\r");
    }

    // write_imageui(outputImage, coord, convert_uint4(pixel));
    write_imageui(outputImage, coord, UCHAR_MAX*dilated_pixel);

}
```

Filtr maksymalny

```
__kernel void sobel_filter(__read_only image2d_t inputImage, __write_only image2d_t
outputImage)
{
    const int2 coord = (int2)(get_global_id(0), get_global_id(1));

    float treshold = 70.0;
    int4 binarized = (int4)(0);

    const float4 conv_val = (float4){0.2989, 0.5870, 0.1140, 0.0};

    int2 square_dim = (int2)(10, 10);

    unsigned int maximal_pixel = 0;
    float4 pixel = (float4)(0);
    float pixel_gray = 0;
    //if(coord.x == 50 && coord.y == 50 )
    {
        for(int row = (coord.x) - (square_dim.x/(int)2); row <= (coord.x) +
(square_dim.x/(int)2); ++row){
            for(int col = coord.y - (square_dim.y/(int)2); col <= coord.y +
(square_dim.y/(int)2); ++col){
                int x2 = (row - coord.x)*(row - coord.x);
                int a2 = (square_dim.x*square_dim.x/4);
```

```
                        int y2 = (col - coord.y)*(col - coord.y);
                        int b2 = (square_dim.y*square_dim.y/4);
                        if(     ((float)x2)/((float)a2) + ((float)y2)/((float)b2) <= 1)
                        {
                                pixel = convert_float4(read_imageui(inputImage,
imageSampler, (int2)(row, col)));
                                pixel_gray = dot(pixel, conv_val);
                                maximal_pixel = (maximal_pixel < pixel_gray) ? pixel_gray :
maximal_pixel;
                                //printf("minimal = %d; \n\r", maximal_pixel);
                                // printf("8");
                        }
                        else
                        {
                                // printf(" ");
                        }
                }
                // printf("\n");
        }
        //printf("\n\r\n\r");
    }



    // write_imageui(outputImage, coord, convert_uint4(pixel));
    write_imageui(outputImage, coord, (uint4){maximal_pixel, maximal_pixel, maximal_pixel,
0});


}
```

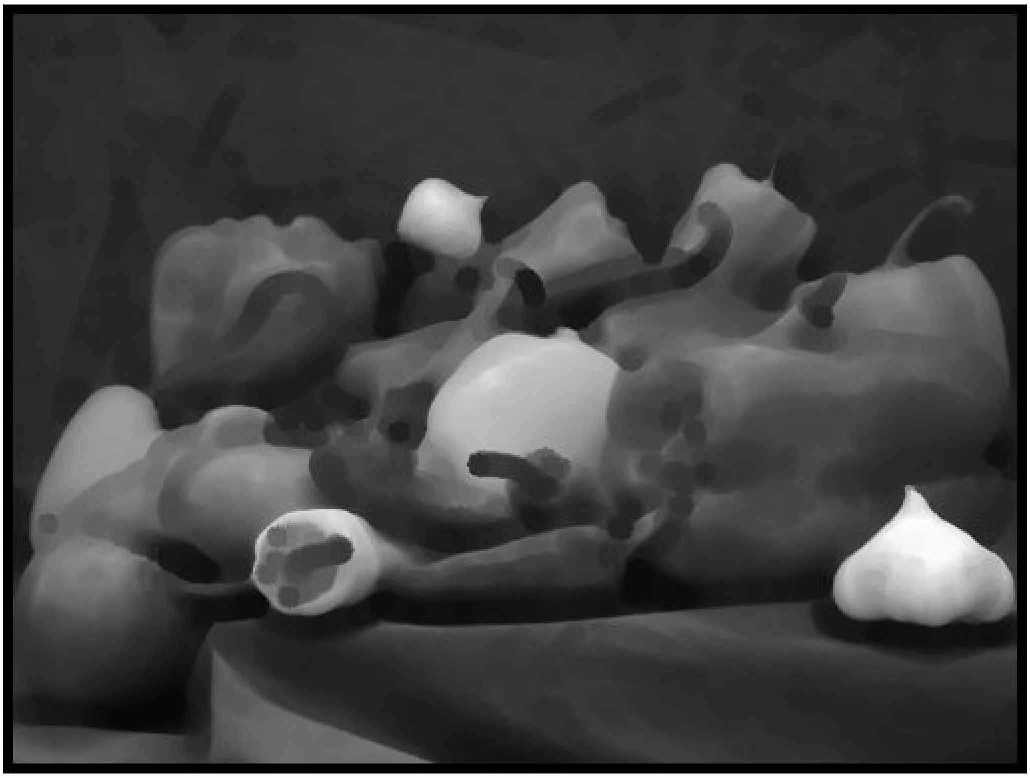
Filtr Minimalny

```
__kernel void sobel_filter(__read_only image2d_t inputImage, __write_only image2d_t
outputImage)
{
```

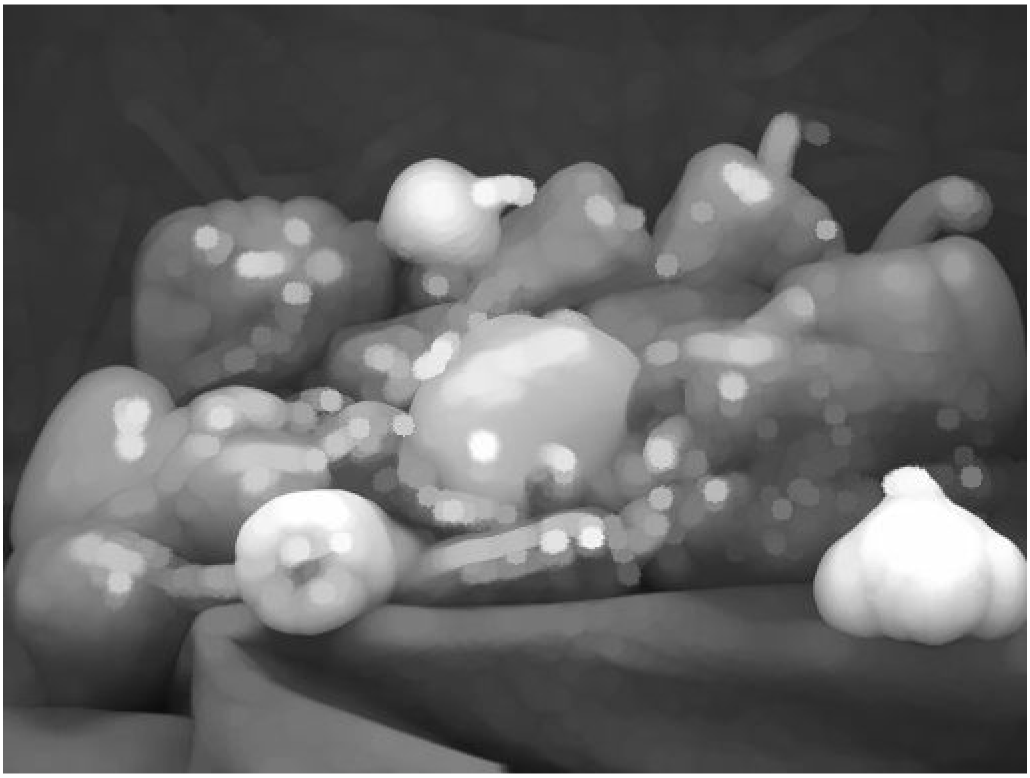
```
	const int2 coord = (int2)(get_global_id(0), get_global_id(1));

	float treshold = 70.0;
	int4 binarized = (int4)(0);

	const float4 conv_val = (float4){0.2989, 0.5870, 0.1140, 0.0};

	int2 square_dim = (int2)(10, 10);

	unsigned int minimal_pixel = UINT_MAX;
	float4 pixel = (float4)(0);
	float pixel_gray = 0;
	//if(coord.x == 50 && coord.y == 50 )
	{
		for(int row = (coord.x) - (square_dim.x/(int)2); row <= (coord.x) + (square_dim.x/(int)2); ++row){
			for(int col = coord.y - (square_dim.y/(int)2); col <= coord.y + (square_dim.y/(int)2); ++col){
				int x2 = (row - coord.x)*(row - coord.x);
				int a2 = (square_dim.x*square_dim.x/4);
				int y2 = (col - coord.y)*(col - coord.y);
				int b2 = (square_dim.y*square_dim.y/4);
				if(	((float)x2)/((float)a2) + ((float)y2)/((float)b2) <= 1)
				{
					pixel = convert_float4(read_imageui(inputImage, imageSampler, (int2)(row, col)));
					pixel_gray = dot(pixel, conv_val);
					minimal_pixel = (minimal_pixel > pixel_gray) ? pixel_gray : minimal_pixel;

					//printf("minimal = %d; \n\r", minimal_pixel);
					// printf("8");
				}
				else
```

```
                    {
                        // printf(" ");
                    }
                }
                // printf("\n");
            }
            //printf("\n\r\n\r");
        }



        // write_imageui(outputImage, coord, convert_uint4(pixel));
        write_imageui(outputImage, coord, (uint4){minimal_pixel, minimal_pixel, minimal_pixel, 0});

}
```

Obrazy wyjściowe Filtr maksymalny i minimalny

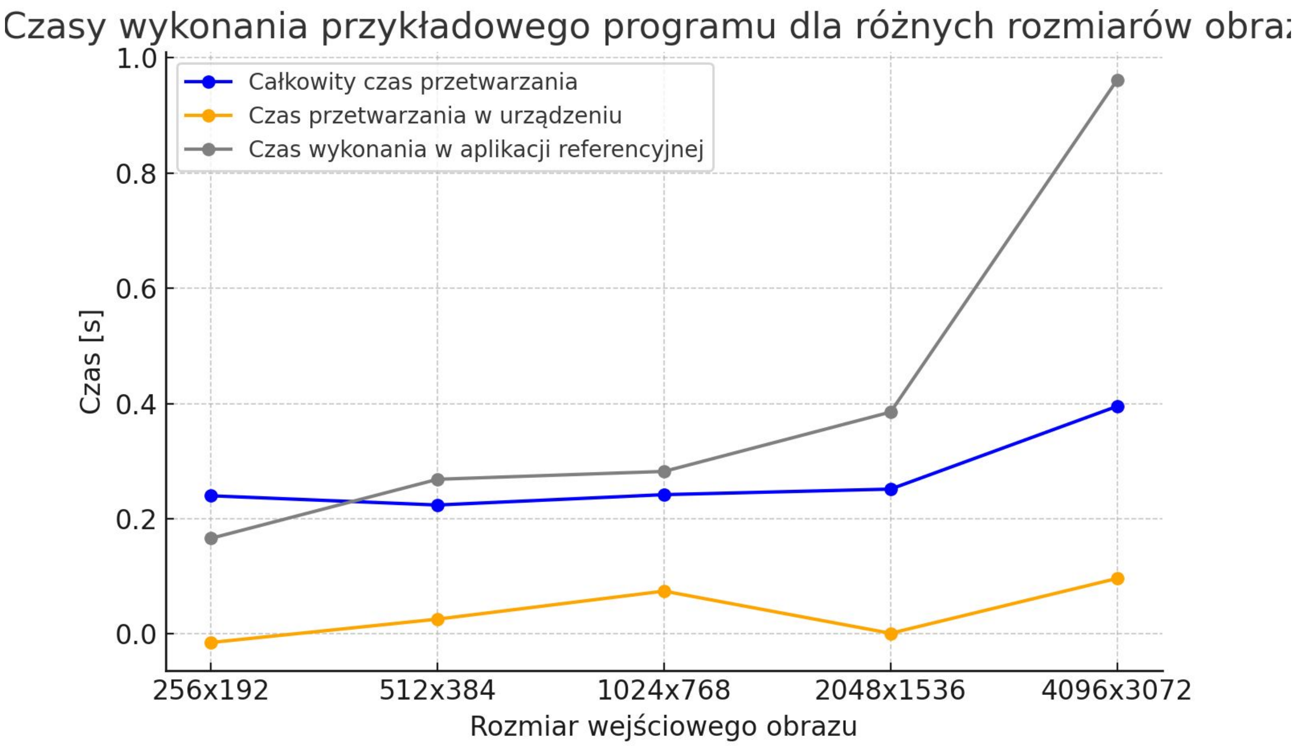


Drawing 7: Filtr minimalny koło



Drawing 8: Filtr maksymalny koło

Czas Przetwarzania obrazów



:

Rozmiar wejściowego obrazu	Całkowity czas przetwarzania [s]	Czas przetwarzania w urządzeniu [s]	Czas wykonania w MatLab [s]
256x192	~0.25	~0.02	~0.20
512x384	~0.27	~0.03	~0.22
1024x768	~0.29	~0.03	~0.25
2048x1536	~0.28	~0.04	~0.40
4096x3072	~0.35	~0.08	~1.00

Wnioski

Implementacja operacji morfologicznych w OpenCL znacząco przyspiesza przetwarzanie obrazów w porównaniu do Matlab, zwłaszcza dla dużych rozmiarów. Wszystkie yniki są zgodne z MATLAB-em, co potwierdza poprawność algorytmów erozji i dylatacji opartych na operatorach \`min\` i \`max\`. Równoległe przetwarzanie minimalizuje wzrost czasu obliczeń wraz z rozmiarem obrazu. Wpływ rozmiaru elementu strukturalnego jest zauważalny – większe struktury zwiększają czas obliczeń, ale OpenCL radzi sobie z tym lepiej niż Matlab.