

**UNIWERSYTET WARMIŃSKO-MAZURSKI W OLSZTYNIE
WYDZIAŁ MATEMATYKI I INFORMATYKI**

Kierunek: Informatyka

Patryk Kowalewski

**Implementacja algorytmu do wektoryzacji
grafiki rastrowej**

Praca inżynierska wykonana
w Katedrze Metod Matematycznych Informatyki
pod kierunkiem
dra inż. Bartosza Nowaka

Olsztyn, 2021 rok

UNIVERSITY OF WARMIA AND MAZURY IN OLSZTYN
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Field of Study: Computer Science

Patryk Kowalewski

**Implementation of raster graphics
vectorization algorithm**

Engineer's Thesis is performed
in the Department of Mathematical
Methods in Computer Science
under supervision of
PhD Bartosz Nowak

Olsztyn, 2021

Spis treści

Streszczenie	2
Abstract	3
Wprowadzenie	4
Rozdział 1. Podstawy teoretyczne	5
1.1. Wstęp	5
1.2. Grafika rastrowa	5
1.3. Grafika wektorowa	6
1.4. Algorytm Canny'ego	7
1.4.1. Redukcja szumu	7
1.4.2. Szukanie natężenia gradientu obrazu	7
1.4.3. Usuwanie niemaksymalnych pikseli	7
1.4.4. Progowanie z histerezą	7
1.4.5. Podsumowanie	8
1.5. Wektoryzacja	8
1.6. Technologie	9
Rozdział 2. Implementacja	10
2.1. Wymagania	10
2.2. Wykorzystanie algorytmu Canny'ego	10
2.2.1. Przygotowanie obrazu	10
2.2.2. Rozmycie gaussowskie	11
2.2.3. Wyznaczenie pochodnej kierunkowej	11
2.2.4. Tłumienie niemaksymalne pikseli	11
2.3. Wektoryzacja	12
2.3.1. Zapis grafiki wektorowej	12
2.4. Interfejs graficzny	13
2.4.1. Wybór obrazu	14
2.4.2. Wektoryzacja	15
2.4.3. Komunikaty	16
Rozdział 3. Testy	17
3.1. Jednostka testowa	17
3.2. Test wydajności	17
3.2.1. Obrazy testowe	17
3.2.2. Wyniki	19
3.3. Test jakości	19
3.3.1. Porównanie	19
3.3.2. Skomplikowany obraz	21
Rozdział 4. Podsumowanie	22
Bibliografia	23
Spis rysunków	24

Streszczenie

Praca jest poświęcona zagadnieniu wektoryzacji obrazów rastrowych. Opisuje również zaimplementowany algorytm, służący do wektoryzacji. W pracy znajdują się również informacje o przetwarzaniu obrazów rastrowych oraz różnice pomiędzy grafiką rastrową a wektorową. Praca zawiera także opis i sposób implementacji algorytmu Canny'ego służącego do wykrywania krawędzi, który jest kluczową częścią programu.

Słowa kluczowe: grafika komputerowa, grafika wektorowa, wektoryzacja, algorytm Canny'ego.

Abstract

Implementation of raster graphics vectorization algorithm

The thesis is devoted to the issue of vectorization of raster images. It also describes the implemented algorithm for vectorization. The thesis also contains information on the processing of raster images and the differences between raster and vector graphics. The paper also describes and implements the Canny edge detection algorithm, which is a key part of the program.

Keywords: computer graphics, vector images, vectorization, Canny's algorithm.

Wprowadzenie

Wektoryzacja (trasowanie) to proces, który ma na celu zamianę grafiki rastrowej na grafikę wektorową. W procesie wektoryzacji rastry opisujące daną bitmapę zostają zgrupowane w większe obiekty wektorowe.

Grafikę wektorową cechuje przede wszystkim brak utraty jakości związanej ze skalowaniem lub innymi przekształceniami obrazu.

Głównym celem pracy jest opisanie implementacji algorytmu służącego do wektoryzacji obrazów rastrowych oraz sprawdzenie jego praktycznego działania przy użyciu interfejsu graficznego stworzonego do łatwej i intuicyjnej obsługi programu.

Pierwszym etapem algorytmu jest przetworzenie obrazu, następnym krokiem jest wyznaczenie krawędzi, które później są zapisywane w postaci funkcji by na końcu zapisać je jako grafikę wektorową.

Rozdział 1

Podstawy teoretyczne

1.1. Wstęp

Grafika rastrowa to obraz, który składa się z pikseli, które w dużej ilości tworzą spójny i jednolity obraz. Wraz z powiększaniem ulega on rozmyciu ze względu na ograniczoną rozdzielcość. Natomiast grafika wektorowa to obraz zbudowany z figur geometrycznych, odcinków, krzywych. Charakteryzuje się łatwością zmiany parametrów i atrybutów elementów, oraz skalowaniem bez utraty jakości. Za to obraz rastrowy często zajmuje mniej miejsca na dysku niż obraz wektorowy. Także oba te typy dwuwymiarowych obrazów cyfrowych mają swoje zalety. Dlatego w zależności od sytuacji grafika wektorowa może okazać się lepsza lub gorsza od grafiki rastrowej.

1.2. Grafika rastrowa

Potocznie nazywana bitmapą to przedstawienie obrazu przy pomocy prostokątnej siatki kolorowych pikseli. Grafika rastrowa jest aktualnie najpopularniejszą formą wyświetlania obrazu cyfrowego. Zdjęcia z aparatu cyfrowego są przykładem grafiki rastrowej. [1]

W systemach komputerowych grafika rastrowa jest przechowywana w sposób skompresowany (stratnie lub bezstratnie) albo nieskompresowany w wielu formatach plików graficznych.

Popularne formaty plików z grafiką rastrową:

BMP (Bit Map) - Jeden z najprostszych formatów przechowywania obrazów rastrowych.

Zawiera w sobie prostą kompresję bezstratną RLE (która nie musi być użyta), informację o użytych kolorach. Obsługuje tryby RGB oraz RGBA.

JPEG (Joint Photographic Expert Group) - algorytm stratnej kompresji grafiki rastrowej, wykorzystany w formacie plików graficznych o tej samej nazwie. W skutek tej kompresji plik ma mniejsze rozmiary ale część informacji o obrazie jest bezpowrotnie tracona. Format ten nie obsługuje kanału alfa.

GIF (Graphics Interchange Format) - format pliku graficznego z kompresją bezstratną.

Pliki tego typu są powszechnie używane na stronach WWW. Pozwala na tworzenie prostych animacji, a ponieważ obsługuje kanał alfa możliwe jest zdefiniowanie koloru tła jako przezroczyste.

PNG (Portable Network Graphics) - rastrowy format plików graficznych oraz system bezstratnej kompresji danych graficznych. Obsługuje stopniowaną przezroczystość (kanał alfa) oraz 48-bitową głębię kolorów czyli 16 bitów na kanał koloru. Dzięki temu można zapisać bezstratnie dowolne grafiki.

Jakość obrazu rastrowego jest określana przez całkowitą liczbę pikseli (wielkość obrazu) oraz ilości informacji przechowywanych w każdym pikselu (głębokość koloru). Nie można zwiększyć rozmiaru obrazu bez jednoczesnego zmniejszenia jego ostrości. Grafika rastrowa jest wykorzystywana do zapisywania zdjęć i realistycznych obrazów.

1.3. Grafika wektorowa

Rodzaj grafiki komputerowej, której obraz jest opisany za pomocą figur geometrycznych, umiejscowionych w matematycznie zdefiniowanym układzie współrzędnych. Inna nazwa grafiki wektorowej to grafika obiektowa, ponieważ obraz jest opisany za pomocą obiektów, które są zbudowane z podstawowych elementów, czyli prostych figur geometrycznych takich jak odcinki, krzywe, okręgi i wielokąty. Każdy obiekt jest opisany za pomocą parametrów, w zależności od figury są to inne współrzędne, np. w przypadku odcinka są to współrzędne jego końców. Obiekty takie mają określone atrybuty takie jak: grubość i kolor linii, kolor lub gradient wypełnienia figury, stopień przezroczystości. Atrybuty zależą głównie od stosowanego standardu opisu grafiki wektorowej.

Popularne formaty grafiki wektorowej:

SVG (Scalable Vector Graphics) - uniwersalny format dwuwymiarowej grafiki wektorowej (statycznej i animowanej). Format SVG powstał z myślą o zastosowaniu na stronach WWW. W formacie tym oprócz standardowych obiektów można opisywać filtry, maski przezroczystości, wypełnienia gradientowe itp. SVG należy do rodziny XML, dzięki czemu można w nim użyć języków skryptowych oraz szablonów stylów.

EPS (Encapsulated Postscript) - format plików, będący podzbiorem języka PostScript, służący do przechowywania pojedynczych stron grafiki wektorowej w postaci umożliwiającej osadzanie ich w innych dokumentach.

PDF (Portable Document Format) - format plików służący do prezentacji, przenoszenia i drukowania treści tekstowo-graficznych, stworzony przez firmę Adobe Systems.

Istnieje oczywiście więcej formatów, warte wspomnienia są formaty popularne przy projektach logo, AI (Adobe Ilustrator) i CDR (pakiet CorelDRAW).

W przeciwnieństwie do grafiki rastrowej, grafika wektorowa jest grafiką w pełni skalowalną, co oznacza, iż obrazy wektorowe można nieograniczenie powiększać oraz zmieniać ich proporcje bez uszczerbku na jakości. W przypadku grafiki rastrowej obrót obrazu może go zniszczyć powodując utratę jakości (w szczególności, jeśli nie jest to obrót o wielokrotność kąta prostego). Należy zaznaczyć, że operacja rasteryzacji jest wykonywana zawsze przed jakimkolwiek obracaniem grafiki wektorowej na monitorze czy drukarce. Istnieją jednakże urządzenia takie jak plotery dla których opis wektorowy jest naturalnym sposobem działania. Grafika wektorowa sprawdza się najlepiej, gdy zachodzi potrzeba stworzenia grafiki, czyli obrazu mającego stosunkowo małą liczbę szczegółów, nie zaś zachowaniu foto realizmu obecnego na zdjęciach. Stosowana jest najczęściej wykorzystywana w schematach technicznych, mapach, planach lub logach. Podczas korzystania z komputera można spotykać się z grafiką wektorową częściej niż mogłoby się to wydawać. Stosowana jest ona m.in. w fontach, graficznych interfejsach użytkownika systemów operacyjnych oraz grach komputerowych i wideo do opisu grafiki trójwymiarowej. [2]

1.4. Algorytm Canny'ego

Algorytm Canny'ego służy do wykrywania krawędzi w obrazie, metoda jest wielostopniowa, a celem jest określenie optymalnego algorytmu detekcji. [3]

Optymalny jest wtedy gdy posiada następujące cechy:

dobra detekcja - algorytm powinien wykryć jak nawięcej krawędzi

dobre umiejscowienie - znaleziona krawędź powinna znajdować się jak najbliżej oryginalnej krawędzi

minimalna odpowiedź - krawędź powinna być oznaczona tylko raz

1.4.1. Redukcja szumu

Detektor krawędzi Canny'ego wykorzystuje filtr, który bazuje na pierwszej pochodnej funkcji Gaussa, ponieważ jest czuły na szum występujący w nieobrobionym obrazie. Efektem działania tego filtra jest lekko rozmyty obraz.

1.4.2. Szukanie natężenia gradientu obrazu

Krawędź skierowana może być w różnych kierunkach, dlatego algorytm Canny'ego oblicza gradient względem krawędzi poziomych i pionowych oraz przekątnych krawędzi. [4]

Sobel to operator używany do obliczania gradientu obrazu. Operator aproksymuje pierwszą pochodną, a gradient może być estymowany dla osmiu różnych kierunków. Największa wartość z nich wskazuje kierunek gradientu. Detekcja kierunku odbywa się z bliżeniem kąta do 45° . [6]

Algorytm zwraca wartości pierwszej pochodnej dla kierunku poziomego (\mathbf{G}_y) i kierunku pionowego (\mathbf{G}_x), które mogą być wykorzystane we wzorach przedstawionych na Rysunku 1.1.

$$\mathbf{G} = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2} \quad \Theta = \arctan\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

Rysunek 1.1. Wzory na gradient oraz kierunek krawędzi

1.4.3. Usuwanie niemaksymalnych pikseli

Nie-maksymalne tłumienie jest techniką ścieniania krawędzi, zapewniając w ten sposób ich ciągłość.

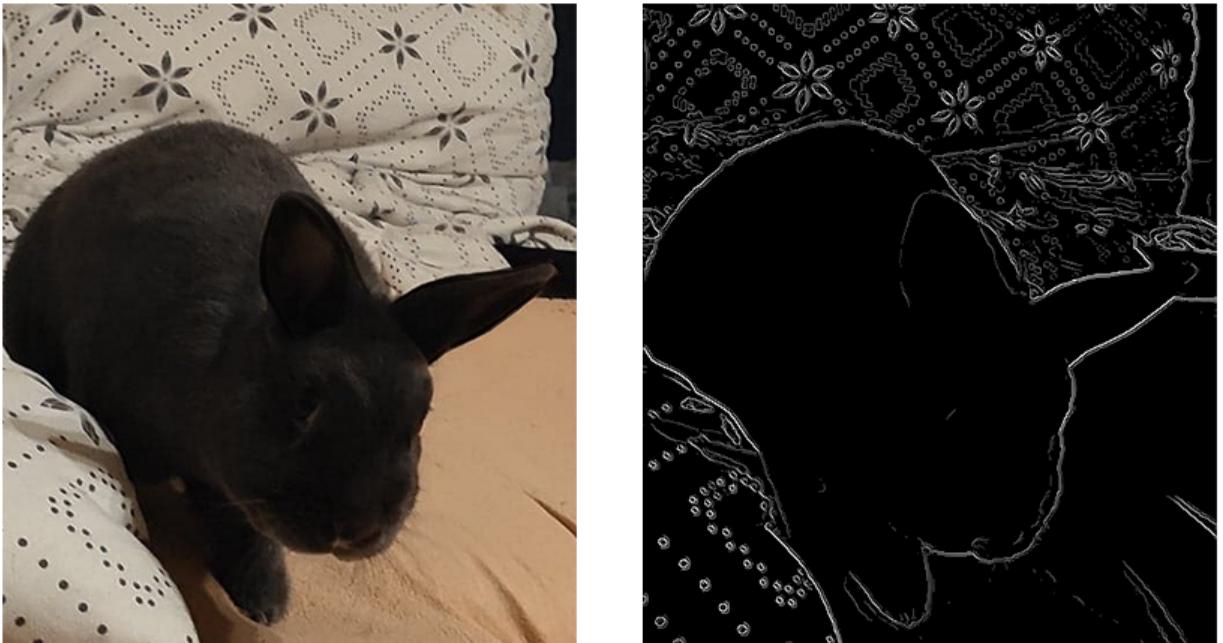
Algorytm sprawdza czy przez każdy sąsiadujący piksel względem piksela badanego przechodzi krawędź. Jeżeli nie przechodzi, piksel ten otrzymuje wartość 0. W przeciwnym przypadku algorytm sprawdza dwa sąsiednie piksele wskazane przez kierunek gradientu, w celu ustalenia czy moduł gradientu jest większy od piksela badanego. Gdy wartość badanego piksela jest większa od obu sąsiadów, zostaje zachowana, w przeciwnym przypadku algorytm przypisuje temu pikselowi wartość 0. [5]

1.4.4. Progowanie z histerezą

Ostatni krok ma celu sprawdzenie czy na obrazie istnieją jakieś fałszywe krawędzie. Progowanie prowadzi do usunięcia nieistotnych krawędzi, o wartościach pikseli poniżej ustalonego dolnego progu.

1.4.5. Podsumowanie

Po przejściu wszystkich kroków algorytmu uzyskujemy obraz wyjściowy, który przedstawia krawędzie z obrazu wejściowego. Obraz końcowy może się znacznie różnić w zależności od dobranych parametrów. W bardziej skomplikowanych obrazach potrzeba wielu prób przed uzyskaniem najlepszego efektu. Na Rysunku 1.2 został przedstawiony algorytm Canny'ego w praktyce.



Rysunek 1.2. Przykład zastosowania algorytmu Canny'ego

1.5. Wektoryzacja

Wektoryzacja to proces polegający na przetworzeniu obrazu rastrowego w obraz wektorowy, czyli krawędzie na bitmapie zostają zapisane w postaci prostych, krzywych oraz wielokątów. Wynikiem wektoryzacji jest więc przedstawienie kształtu, który był na oryginalnym obrazie. Wektoryzacja najlepiej sprawdza się przy prostych obrazach, na których nie wiele szczegółów.

Nie ma jednej idealnej metody na wektoryzację obrazów rastrowych, dlatego też powstało wiele firm postanowiło stworzyć własne opgramowanie, które w większości przypadków dają zadawalające efekty oraz możliwość ręcznej edycji obrazu po wektoryzacji.

Pomimo tego że nie ma najlepszej metody spora część z nich opiera się na tym samym schemacie, w skład którego wchodzi wykrycie krawędzi, zapis krawędzi w postaci krzywych oraz w wielu przypadkach funkcji, które mają na celu poprawić obraz końcowy.

Obraz można wektoryzować również ręcznie. W tym celu należy dokonać pewnych pomiarów, a następnie ręcznie zapisać plik wyjściowy. Dobrym przykładem może być wektoryzacja rysunku technicznego. Ilustracja z reguły ma wiele geometrycznych kształtów oraz tekst. Dodatkowo format wektorowy SVG umożliwia łatwe wprowadzanie tekstu (nawet indeksów dolnych i górnych). [7]

1.6. Technologie

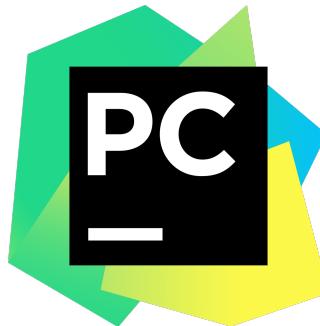
Python to język programowania wysokiego poziomu o rozbudowanym pakiecie bibliotek standardowych, którego ideą przewodnią jest czytelność i klarowność kodu źródłowego. Jego składnia cechuje się przejrzystością i zwięzłością. Elegancka składnia i dynamiczne pisanie sprawiają, że jest to idealny język do tworzenia skryptów i szybkiego tworzenia aplikacji w wielu obszarach na większości platform. Interpreter Pythona można łatwo rozszerzyć o nowe funkcje i typy danych zaimplementowane w C lub C++. Python nadaje się również jako język rozszerzający dla dostosowywalnych aplikacji. [8]

OpenCV to biblioteka funkcji wykorzystywanych podczas obróbki obrazu, oparta na otwartym kodzie i zapoczątkowana przez Intel. Autorzy skupią się na przetwarzaniu obrazu w czasie rzeczywistym.

Biblioteka została stworzona w języku C, lecz istnieją nakładki umożliwiające korzystanie z niej również w językach C++, C#, Python, Java, JavaScript. Wszystkie nowe rozwiązania i algorytmy pojawiają się w języku C++.

OpenCV-Python korzysta z biblioteki Numpy, która jest wysoce zoptymalizowana do operacji numerycznych ze składnią w stylu MATLAB. Wszystkie struktury tablic OpenCV są konwertowane do i z tablic Numpy. Ułatwia to również integrację z innymi bibliotekami używającymi Numpy, takimi jak Matplotlib.[11]

PyCharm to zintegrowane środowisko programistyczne (**IDE**) dla języka programowania Python firmy JetBrains. Logo programu przedstawione na Rysunku 1.3



Rysunek 1.3. Logo programu PyCharm

Jest to oprogramowanie wieloplatformowe pracujące na platformach systemowych: Microsoft Windows, GNU/Linux oraz macOS. Program wydawany jest w wersji Professional Edition, która jest oprogramowaniem własnościowym oraz w wolnej wersji Community Edition, która pozbawiona jest jednak części funkcjonalności w porównaniu z wersją własnościową. [9][10]

Podstawowe funkcje:

- Inteligentny edytor kodu z podświetleniem składni dla języka Python.
- Edycja i wsparcie dla HTML, CSS oraz JavaScript.
- Graficzny debugger dla Pythona.
- Integracja z systemami kontroli wersji.
- Wbudowany terminal.
- Możliwość instalacji dodatkowych pluginów.

Rozdział 2

Implementacja

2.1. Wymagania

Użytkownik programu powinien być w stanie samodzielnie dokonać prostej wektoryzacji obrazu rastrowego przy użyciu interfejsu graficznego oraz możliwość zmiany parametrów wpływających na obraz końcowy.

2.2. Wykorzystanie algorytmu Canny'ego

Implementacja zaczyna się od wykrycia krawędzi przy użyciu metody Canny'ego. Poszczególne parametry są ustawiane przez użytkownika w interfejsie graficznym.

2.2.1. Przygotowanie obrazu

W celu zmniejszenia ilości obliczeń na wstępie obraz jest przekształcany do skali odcieni szarości. Obraz w skali szarości przedstawiony na Rysunku 2.1.



Rysunek 2.1. Obraz w skali odcieni szarości

2.2.2. Rozmycie gaussowskie

W tym kroku następuje modyfikacja obrazu za pomocą filtra Gaussa w celu zmniejszenia szumów i zakłóceń. Rysunek 2.2 przedstawia obraz po zastosowaniu rozmycia Gaussowskiego.



Rysunek 2.2. Obraz po zastosowaniu filtru Gaussa

2.2.3. Wyznaczenie pochodnej kierunkowej

Następnie obliczane są pochodne w kierunkach x i y , w tym celu wykorzystany zostaje operator Sobel.

2.2.4. Tłumienie niemaksymalne pikseli

Tłumienie niemaksymalne jest zastosowane by upewnić się, że pikselem krawędzi jest piksel, dla którego wartość gradientu jest maksymalna w kierunku wyznaczonym przez gradient. Dla piksela aktualnie przeglądanego, algorytm przeszukuje sąsiednie piksele w czterech kierunkach. Wartość piksela zostaje zachowana gdy jest większa od jego sąsiadów, natomiast w przeciwnym przypadku wartość tego piksela zostaje zmieniona na zero.

Przy zachowaniu zbyt dużej ilości niemaksymalnych pikseli, krawędź zostaje pogrubiona, by tego uniknąć algorytm wykonuje dodatkowo progowanie o podwójnym stopniu. W interfejsie graficznym przed uruchomieniem algorytmu, wybieramy dolną wartość progu.



Rysunek 2.3. Obraz po zastosowaniu algorytmu Canny'ego

2.3. Wektoryzacja

Po zastosowaniu algorytmu Canny'ego, obraz (Rysunek 2.3) jest gotowy do procesu wektoryzacji. W tym celu zostaje wykorzystana funkcja szukająca konturów oparta o kod łańcuchowy. Kontury to po prostu krzywe łączące wszystkie ciągłe punkty, mające ten sam kolor lub intensywność. Kontury są użytecznym narzędziem do analizy kształtów oraz wykrywania i rozpoznawania obiektów.

2.3.1. Zapis grafiki wektorowej

Do zapisu obrazu w postaci wektorowej wykorzystana została biblioteka Matplotlib.

Na podstawie znalezionych konturów, są rysowane wektory przy uwzględnieniu rozmiarów oryginalnego obrazu.

Pętla która jest odpowiedzialna za rysowanie wektorów:

```
for k in kontury:  
    plt.plot(k[:, 0, 0], h - k[:, 0, 1], 'k', linewidth=1.2)
```

Na końcu obraz zostaje zapisany do pliku w formacie svg:

```
plt.autoscale(tight=True)  
plt.axis('off')  
plt.savefig(nazwa + '.svg', format="svg", bbox_inches='tight',  
transparent=True, pad_inches=0)
```

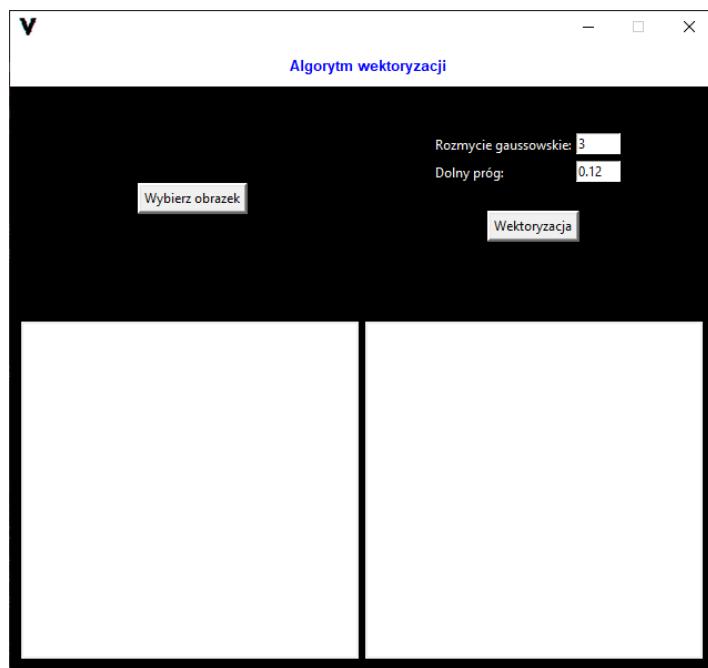
Wynik widoczny jest na Rysunku 2.4.



Rysunek 2.4. Obraz po wektoryzacji

2.4. Interfejs graficzny

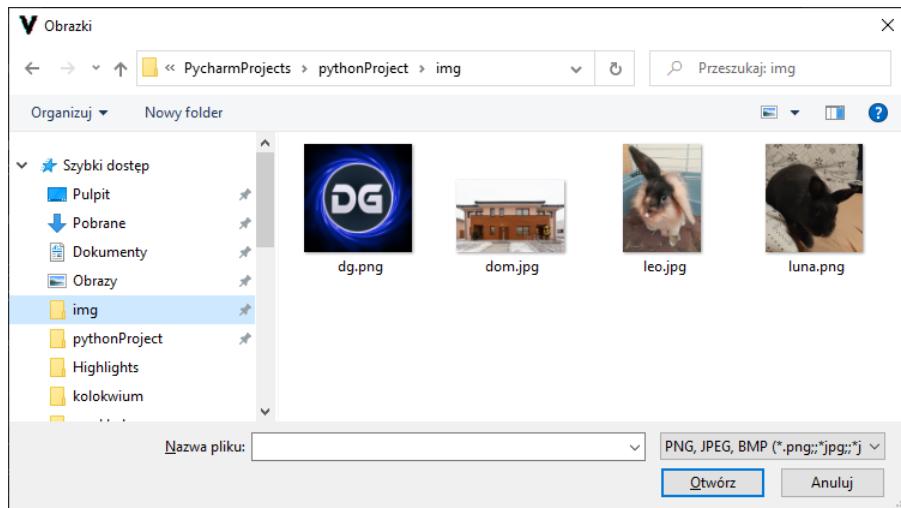
Interfejs graficzny (Rysunek 2.5) (**GUI**) powstał przy użyciu modułu Tkinter, który jest podstawowym interfejsem graficznym w Phytonie.



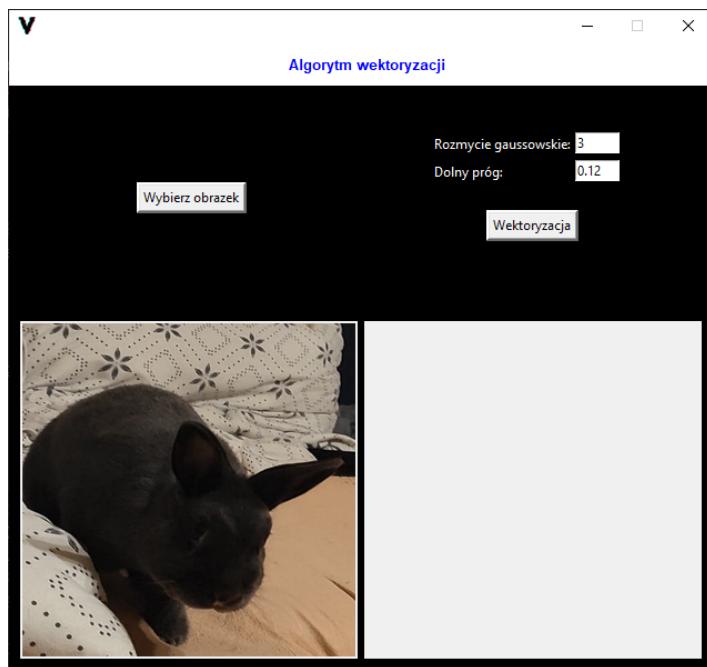
Rysunek 2.5. Wygląd GUI po uruchomieniu programu

2.4.1. Wybór obrazu

Po wciśnięciu przycisku "Wybierz obrazek", otwiera się nowe okno (Rysunek 2.6), w którym możemy wybrać obraz do wektoryzacji. Dostępne są trzy popularne formaty grafiki rastrowej: PNG, JPEG oraz BMP. Po wybraniu obrazu widzimy jego podgląd w głównym oknie interfejsu (Rysunek 2.7).



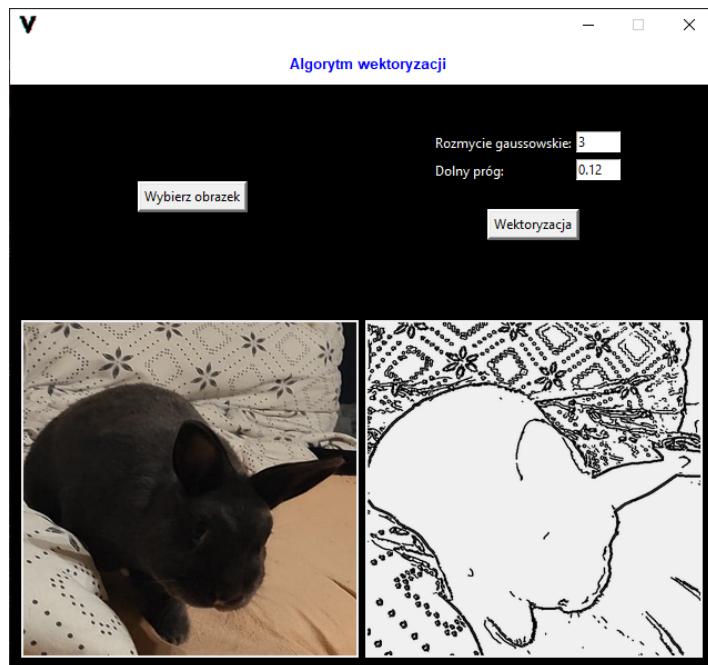
Rysunek 2.6. Okno wyboru obrazu



Rysunek 2.7. GUI po wybraniu obrazu

2.4.2. Wektoryzacja

Paremtry po wpisaniu przez użytkownika zostają zapisane po naciśnięciu przycisku "Wektoryzacja", po czym zostaje uruchomiony algorytm wektoryzacji. Gdy proces dobiegnie końca, wyświetlany jest podgląd obrazu po wektoryzacji oraz zostaje on zapisany w pliku o formacie SVG.

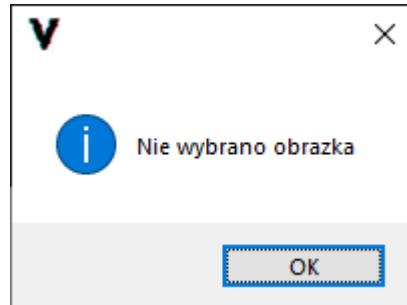


Rysunek 2.8. GUI wektoryzacji

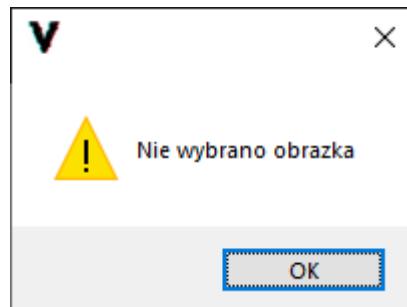
Dodatkowo w celach poglądowych tworzony jest obraz jako grafika rastrowa w formacie PNG. Obraz ten jest wyświetlany w interfejsie graficznym jako podgląd (Rysunek 2.8), ponieważ Tkinter nie obsługuje wyświetlania plików graficznych w formacie SVG.

2.4.3. Komunikaty

Program informuje użytkownika w dwóch przypadkach. Gdy po wcisnięciu przycisku "Wybierz obrazek" użytkownik zamknie okno nie wybierając żadnego obrazu (Rysunek 2.9) lub w przypadku naciśnięcia przycisku "Wektoryzacja" nie wybierając wcześniej obrazu (Rysunek 2.10).

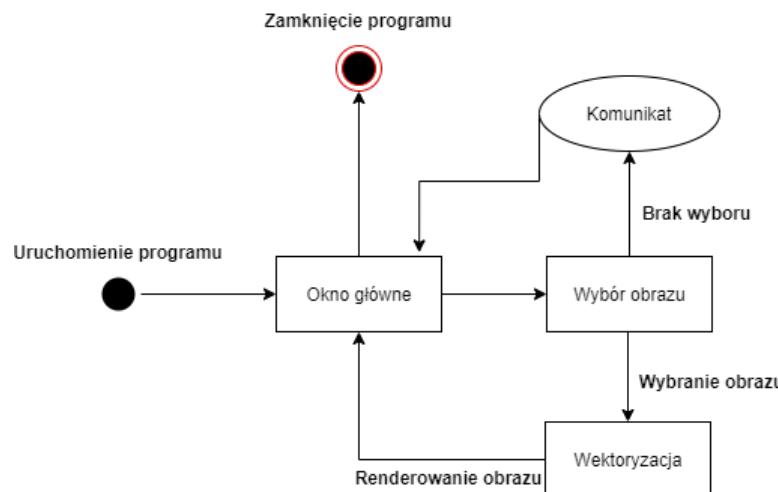


Rysunek 2.9. Zamknięcie okna bez wyboru obrazu



Rysunek 2.10. Naciśnięcie "Wektoryzuj" bez wybranego obrazu

Program jest prosty i czytelny w obsłudze, działa według podanego schematu przedstawionego na Rysunku 2.11.



Rysunek 2.11. Działanie interfejsu

Rozdział 3

Testy

3.1. Jednostka testowa

Wszystkie testy zostały przeprowadzone na komputerze wyposażonym w procesor AMD FX(tm)-4350 Quad-Core 4.20 GHz, pamięć RAM 16GB, kartę graficzną ASUS R9 280 Series (3GB VRAM) oraz system operacyjny windows 10.

3.2. Test wydajności

Test został wykonany w celu sprawdzenia ile czasu oraz pamięci potrzebuje algorytm. Do tego celu były przeznaczone obrazy w różnych formatach, różną rozdzielczością oraz ilością szczegółów.

3.2.1. Obrazy testowe

Test przeprowadzony został na czterech różnych obrazach (Rysunki 3.1, 3.2, 3.3, 3.4).



Rysunek 3.1. Obraz testowy1



Rysunek 3.2. Obraz testowy2



Rysunek 3.3. Obraz testowy3



Rysunek 3.4. Obraz testowy4

3.2.2. Wyniki

Dla każdego obrazu testowego zostało przeprowadzone 5 prób, aby uzyskać uśrednione wyniki. Test był wykonany dwukrotnie, pierwszy (Rysunek 3.5) przy ustawieniu progu dolnego na wartość 0.1 i drugi (Rysunek 3.6) z wartością progu dolnego ustawioną na 0.2. W obu przypadkach współczynnik rozmycia gaussowskiego wynosił 3.

Próg dolny: 0.1

	Obraz testowy1	Obraz testowy2	Obraz testowy3	Obraz testowy4
Czas wykonania	7.76 s	8.71 s	13.64 s	5.95 s
Zużycie procesora	27%	27%	27%	26%
Wykorzystanie pamięci RAM	130 MB	150 MB	170 MB	140 MB

Rysunek 3.5. Tabela przedstawiająca wyniki przy ustawieniu progu dolnego na 0.1

Próg dolny: 0.2

	Obraz testowy1	Obraz testowy2	Obraz testowy3	Obraz testowy4
Czas wykonania	6.63 s	5.86 s	8.59 s	4.5 s
Zużycie procesora	24%	26%	26%	20%
Wykorzystanie pamięci RAM	140 MB	150 MB	160 MB	120 MB

Rysunek 3.6. Tabela przedstawiająca wyniki przy ustawieniu progu dolnego na 0.2

3.3. Test jakości

3.3.1. Porównanie

Pierwszy test sprawdza jak wypada algorytm w porównaniu z komercyjnym programem do wektoryzacji. Jako obraz wejściowy użyty został obraz (Rysunek 3.1). Natomiast program, który został użyty w celu porównawczym to Adobe Illustrator. Oba obrazy zostały zapisane w formacie SVG.



Rysunek 3.7. Wektoryzacja przy użyciu algorytmu



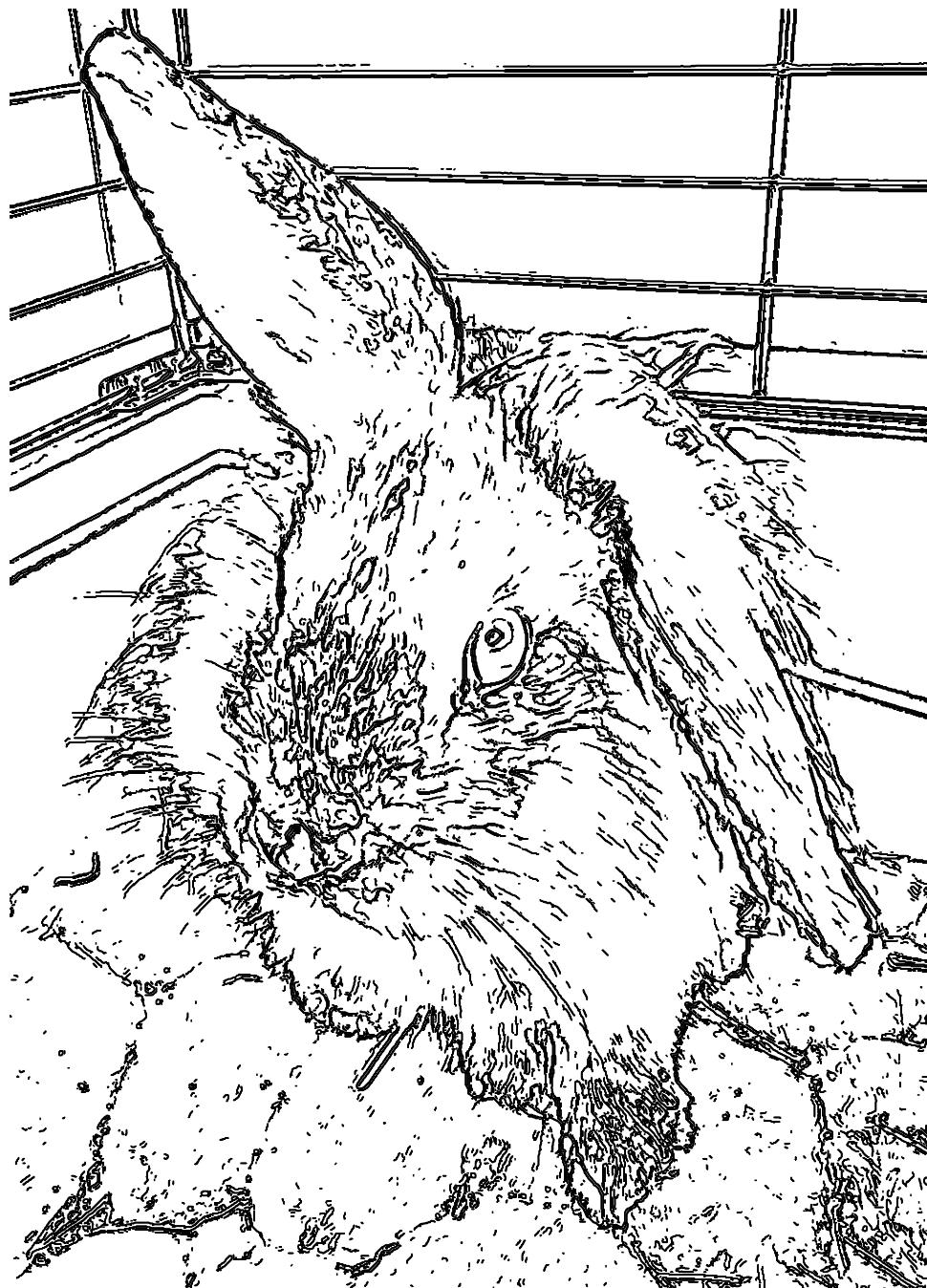
Rysunek 3.8. Wektoryzacja przy użyciu Adobe Illustrator

Wektoryzacja przy użyciu programu Adobe Illustrator została wykonana automatycznie bez ręcznej korekty.

Różnica między tymi dwoma obrazami jest zauważalna. Wektoryzacja przy użyciu algorytmu (Rysunek 3.7) była bardziej szczegółowa przy kształtach z nieoczywistą krawędzią, natomiast przy użyciu komercyjnego oprogramowania (Rysunek 3.8) oprócz znalezienia krawędzi zostały również wypełnione kształty, które były tego samego koloru.

3.3.2. Skomplikowany obraz

Ten test jakości miał na celu sprawdzenie jak dobrze algorytm radzi sobie z obrazem, na którym istnieje wiele krawędzi w postaci futra królika. Jako obraz testowy został użyty obraz (Rysunek 3.3).



Rysunek 3.9. Obraz testowy 3, próg dolny: 0.1, współczynnik rozmycia gausowskiego: 3

Algorytm dobrze poradził sobie z obrazem testowym. Wykrył krawędzie mimo utrudnionego zadania, a efekt jest widoczny na Rysunku 3.9.

Rozdział 4

Podsumowanie

W ramach pracy został zaimplementowany algorytm wektoryzacji obrazów rastrowych. Program przy użyciu interfejsu graficznego jest prosty w obsłudze oraz spełnia swoje zadanie, czyli poddaje obraz procesowi wektoryzacji. Obraz wyjściowy jest w akceptowalnej jakości, a czas pracy algorytmu nie jest długi. Oczywiście w zależności od skomplikowania obrazu wejściowego oraz dobranych parametrów czas potrzebny na przetworzenie obrazu może się wydłużyć.

Algorytm dobrze sprawdza się przy wektoryzacji logotypów oraz daje zadawalające efekty przy zdjęciach ludzi lub zwierząt.

Podsumowując algorytm spełnia swoje oczekiwania. Świetnie nadaje się do wektoryzacji prostych obrazów, gdy najważniejszym kryterium jest odwzorowanie kształtu.

Oczywiście zaimplementowany algorytm nie jest idealny i najbardziej efektywny, aczkolwiek istnieją opcje na jego ulepszenie i rozbudowanie funkcjonalności.

Bibliografia

- [1] Wikipedia, *Grafika rastrowa*, https://pl.wikipedia.org/wiki/Grafika_rastrowa
- [2] Wikipedia, *Grafika wektorowa*, https://pl.wikipedia.org/wiki/Grafika_wektorowa
- [3] John F. Canny, *A Computational Approach to Edge Detection* 1986, http://perso.limsi.fr/vezien/PAPIERS_ACS/canny1986.pdf
- [4] Wikipedia, *Canny*, <https://pl.wikipedia.org/wiki/Canny>
- [5] Marcin Wilczewski, *Algorytmy graficzne*, <http://www.mif.pg.gda.pl/homepages/marcin/AG2015-16/wyklad5.pdf>
- [6] Irwin Sobel, *History and Definition of the Sobel Operator*, 2014, https://www.researchgate.net/publication/239398674_An_Isotropic_3x3_Image_Gradient_Operator
- [7] Wikipedia, *Image tracing*, https://en.wikipedia.org/wiki/Image_tracing
- [8] Python, *dokumentacja*, <https://docs.python.org/3/tutorial/index.html>
- [9] JetBrains, *PyCharm*, <https://www.jetbrains.com/pycharm/>
- [10] Wikipedia, *PyCharm*, <https://pl.wikipedia.org/wiki/PyCharm>
- [11] OpenCV, *dokumentacja*, https://docs.opencv.org/master/d0/de3/tutorial_py_intro.html

Spis rysunków

1.1.	Wzory na gradient oraz kierunek krawędzi	7
1.2.	Przykład zastosowania algorytmu Canny'ego	8
1.3.	Logo programu PyCharm	9
2.1.	Obraz w skali odcieni szarości	10
2.2.	Obraz po zastosowaniu filtra Gaussa	11
2.3.	Obraz po zastosowaniu algorytmu Canny'ego	12
2.4.	Obraz po wektoryzacji	13
2.5.	Wygląd GUI po uruchomieniu programu	13
2.6.	Okno wyboru obrazu	14
2.7.	GUI po wybraniu obrazu	14
2.8.	GUI wektoryzacji	15
2.9.	Zamknięcie okna bez wyboru obrazu	16
2.10.	Naciśnięcie "Wektoryzuj" bez wybranego obrazu	16
2.11.	Działanie interfejsu	16
3.1.	Obraz testowy1	17
3.2.	Obraz testowy2	17
3.3.	Obraz testowy3	18
3.4.	Obraz testowy4	18
3.5.	Tabela przedstawiająca wyniki przy ustawieniu progu dolnego na 0.1	19
3.6.	Tabela przedstawiająca wyniki przy ustawieniu progu dolnego na 0.2	19
3.7.	Wektoryzacja przy użyciu algorytmu	20
3.8.	Wektoryzacja przy użyciu Adobe Illustrator	20
3.9.	Obraz testowy3, próg dolny: 0.1, współczynnik rozmycia gausowskiego: 3	21