Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Instytut Informatyki

Rok akademicki 2013/2014

Praca dyplomowa inżynierska

Bartosz Domagała

Wektoryzacja obrazów rastrowych przedstawiających obiekty architektoniczne

Opiekun pracy:
prof. dr hab. inż. Jan Zabrodzki

Ocena .....

.....
Podpis Przewodniczącego
Komisji Egzaminu Dyplomowego

Specjalność: Informatyka –
Inżynieria systemów
informatycznych

Data urodzenia: 13 maja 1991 r.

Data rozpoczęcia studiów: 22 luty 2011 r.

Życiorys

Nazywam się Bartosz Domagała, urodziłem się 13 maja 1991 roku w Sieradzu. W roku 2004 ukończyłem Niepubliczną Szkołę Podstawową w Sieradzu, a w roku 2007 Gimnazjum im. Jana Pawła II w Sieradzu. Naukę kontynuowałem w II Liceum Ogólnokształcącym im. Stefana Żeromskiego w Sieradzu, które ukończyłem zdając maturę w 2010 roku. W lutym roku 2011 rozpocząłem studia na wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej na kierunku Informatyka wybierając w toku studiów specjalizację Inżynieria Systemów Informatycznych. Od maja 2013 roku pracuję zawodowo jako programista w języku C++.

.....
podpis studenta

Egzamin dyplomowy

Złożył egzamin dyplomowy w dn. .....

Z wynikiem .....

Ogólny wynik studiów .....

Dodatkowe wnioski i uwagi Komisji .....

.....

Streszczenie

Praca podejmuje problem wektoryzacji obrazów, skupiając się na tych, które przedstawiają obiekty architektoniczne. Opisuje również działanie interaktywnego programu „Ptah" i istotne informacje na temat zaimplementowanego w nim algorytmu, służącego do wektoryzacji.

W pracy znajdują się również podstawowe informacje o przetwarzaniu obrazów rastrowych. Praca zawiera także opis i propozycję implementacji algorytmu Canny'ego służącego do wykrywania krawędzi, będącego częścią tworzonego programu.

W ramach pracy przeprowadzono badania wydajnościowe, które pozwoliły określić jakiego rodzaju ograniczenia programowo-sprzętowe są kluczowe z punktu widzenia możliwości programu.

Słowa kluczowe: *wektoryzacja, grafika wektorowa, grafika rastrowa, obiekty architektoniczne, algorytm Canny'ego.*

Abstract

Title: *Vectorization of raster images presenting the architectural objects*

The work addresses the problem of vectorization images, focusing on those that represent architectural objects. It also describes the process of creating an interactive program „Ptah" and connect vital information about the vectorization and activity of algorithm.

The text also contains basic information about the processing of raster images in vectorization. Thesis includes a description of the proposal and implementation of the Canny algorithm to detecting the edges and creating as part of the program.

Permormance tests have been conducted as a part of the work and allowed to determine what kind of limitations of software and hardware are crucial from the point of view of the possibilities of the program.

Key words: *vectorization, raster images, vector images, architectural objects, Canny algorithm.*

Spis treści

1. Wprowadzenie

Wektoryzacja, zwana również trasowaniem, czyli zmiana rastrowego opisu obrazu na opis wektorowy, to nadal otwarty temat w grafice komputerowej i przetwarzaniu obrazów cyfrowych. Opis wektorowy obrazu wiąże się przede wszystkim z brakiem strat jakościowych bez względu na skalowanie i różnego rodzaju przekształcenia. Daje to duże możliwości przy edycji i obróbce obrazów. Głównym problemem jest tu opracowanie uniwersalnego algorytmu, który zapewniłby zadowalające efekty w ograniczonym czasie. Stąd powstaje wiele rozwiązań dla obrazów z konkretnej dziedziny.

Obrazy przedstawiające obiekty architektoniczne da się zamknąć w ramy pewnych wzorców. Podstawową cechą jest to, że w pewnym, niedużym uproszczeniu obiekty na nich ukazane składają się z figur płaskich oraz odcinków, które można zapisać za pomocą funkcji. Skutkuje to potencjalną możliwością opracowania algorytmu, który sprostałby wektoryzacji tego typu obrazów i pozwolił na uzyskanie satysfakcjonujących wyników.

Celem niniejszej pracy jest opisanie implementacji algorytmu wektoryzacji obrazów przedstawiających obiekty architektoniczne i sprawdzenie jego praktycznego działania. Algorytm ma za zadanie najpierw odpowiednio przetworzyć obraz cyfrowy, a następnie wyznaczyć jego krawędzie, by na koniec zapisać je w postaci funkcji i uzyskać opis wektorowy. Ponadto cały proces powinien być wykonany w sensownym dla człowieka czasie.

2. Podstawy teoretyczne

2.1. Wstęp

Dwuwymiarowy obraz cyfrowy możemy opisać na dwa sposoby: rastrowy oraz wektorowy. Różnią się one przede wszystkim sposobem zapisu informacji o obrazie. Wiąże się to z różnymi możliwościami przy wszelakich operacjach i przekształceniach na obrazach danego typu oraz pociąga za sobą konkretne konsekwencje. Dla przykładu grafika rastrowa przy skalowaniu obrazu traci swoją jakość, natomiast grafika wektorowa nie. Z drugiej strony obraz rastrowy w wielu wypadkach zajmuje mniej pamięci niż obraz wektorowy. Podobnych przykładów możemy znaleźć znacznie więcej.

Te różnice między typami opisów obrazów wiążą się przede wszystkim z możliwością operowania na obiektach. W zależności od zagadnienia i konkretnego zastosowania opis wektorowy może być lepszy bądź gorszy od rastrowego i *vice versa*.

2.2. Grafika rastrowa

Obrazy rastrowe składają się z pikseli, które mają konkretne położenie i konkretną barwę. Najprostszym zapisem obrazu tego typu jest raster (bitmapa), który zawiera informacje o każdym pikselu w obrazie. W praktyce najczęściej stosowane są różnego rodzaju kompresje, co wiąże się ze znacznym zmniejszeniem wielkości pliku. Zależnie od typu kompresji (bezstratna lub stratna) możemy zatracić pewne informacje o obrazie lub nie. W związku z różnymi sposobami kompresji oraz potrzebami dotyczącymi grafiki cyfrowej powstało wiele formatów graficznych.

— **BMP (Bit Map)**
Jeden z najprostszych w zapisie formatów statycznej (tzn. bez animacji) grafiki rastrowej, polegający na zapamiętaniu barwy każdego piksela w obrazie. Opcjonalnie stosuje się w nim bezstratną kompresję RLE.

— **JPG/JPEG (Joint Photographic Expert Group)**
Format graficzny statycznych obrazów rastrowych, który korzysta ze stratnej kompresji JPEG uzyskując dzięki niej mniejsze rozmiary pliku ale tracąc bezpowrotnie część informacji o obrazie. Słabiej prezentuje się w wypadkach gdy w obrazie występują duże obszary jednolitego koloru. Nie obsługuje kanału alfa.

— **GIF (Graphics Interchange Format)**

Stworzony przez firmę CompuServe w 1987 roku format grafiki rastrowej, stosującej bezstratną kompresję algorytmem słownikowym. Obsługuje kanał alfa. Główne problemy z tym formatem dotyczyły patentu na algorytm kompresji, jednak w 2003 roku został on ostatecznie zniesiony.

— **PNG (Portable Network Graphics)**
Format stworzony jako, wolna od patentów, odpowiedź na format GIF w 1995 roku. Stosuje bezstratną kompresję oraz obsługuje przezroczystość. Podobnie jak GIF jest często używany w grafikach na stronach internetowych.

Jakość obrazu rastrowego jest określana przez całkowitą liczbę pikseli (wielkość obrazu) oraz ilości informacji przechowywanych w każdym pikselu (głębia koloru) [5]. Grafika tego typu na co dzień wykorzystywana jest w różnego rodzaju urządzeniach elektronicznych jak aparaty cyfrowe czy monitory.

2.3. Grafika wektorowa

Grafika wektorowa jest to typ obrazu, w którym operuje się obiektami definiowanymi za pomocą wierzchołków bądź punktów sterujących. Każdy obiekt tworzący obraz jest oddzielną instancją, której można nadać parametry: wymiary, położenie, wypełnienie kolorem czy gradientem, grubość, stopień przezroczystości, itp. Konkretne atrybuty zależne są od stosowanego standardu opisu grafiki wektorowej. Podobnie jak w grafice rastrowej mamy tutaj dostępne różne przestrzenie barw.

— **SVG (Scalable Vector Graphics)**
Format grafiki wektorowej statycznej oraz animowanej, stworzonej w roku 1999 przez W3C, nie objęty żadnym patentem. Stworzony został głównie pod kątem witryn internetowych. SVG należy do rodziny XML, dzięki czemu może być integrowany z innymi językami jak choćby XHTML.
Poza opisem standardowych obiektów, format ten umożliwia także opisywanie filtrów i masek przezroczystości.

— **SWF**
Format stworzony przez firmę Adobe i przez nią opatentowany. Pliki w tym formacie mogą zawierać animacje lub aplety o różnym stopniu interakcyjności i funkcjonalności. W przeciwieństwie jednak do obrazów w formacie SVG mogą zawierać treści multimedialne (audio oraz wideo).

Powyższe przykłady to oczywiście tylko mała część wielu dostępnych formatów wektorowych. Każdy z nich jest stosowany w zależności od danej sytuacji.

Główną zaletą obrazów wektorowych jest ich skalowalność (bez utraty jakości) oraz prostota opisu, tzn. operuje się na znacznie mniejszym zbiorze punktów niż w grafice rastrowej, co *de facto* skraca czas niektórych obliczeń. Niestety przy dużej liczbie obiektów bądź znacznej szczegółowości obrazu, plik graficzny staje się proporcjonalnie duży i wymaga znacznie więcej pamięci.

Ze względu na powyższe cechy grafika wektorowa jest używana często w schematach technicznych, wykresach czy logach różnego rodzaju.

2.4. Globalne progowanie obrazów

Progowanie jest jedną z najprostszych metod segmentacji, oznaczającej podział obrazu na obszary spełniające dane kryterium jednorodności [2]. Jest to przykład przekształcenia punktowego, w którym wartość poszczególnego piksela obrazu stanowi wynik operacji algebraicznej przeprowadzonej tylko na tym pikselu. Nie są brane pod uwagę punkty z jego otoczenia.

Progowanie polega na porównywaniu każdego kolejnego punktu w obrazie z zadaną wartością progową. Jej odpowiedni dobór może skutkować wyodrębnieniem obszarów o określonych właściwościach.

Podstawowe i zarazem najmniej skomplikowane progowanie to binaryzacja, czyli podział pikseli (w zależności od progu) na dwie grupy. Binaryzacja zdefiniowana jest wzorem:

$$P(x,y) = \begin{cases} 1, & P(x,y) > t \\ 0, & P(x,y) \leq t \end{cases} \tag{2.1}$$

gdzie *P(x,y)* - piksel o współrzędnych *(x,y)*, a t – próg binaryzacji.

W zależności od potrzeb możemy definiować progowanie inaczej: zmieniając wartości dla konkretnych przedziałów czy dodając kolejne progi. W ostatnim wypadku mamy do czynienia z progowaniem wielokryteryjnym bądź wieloprogowaniem. Za jego pomocą obraz zostaje podzielony na segmenty o różnych poziomach jasności:

$$P(x,y) = \begin{cases} 0, & P(x,y) \in D_0 \\ 1, & P(x,y) \in D_1 \\ \dots & \\ n-1, & P(x,y) \in D_{(n-1)} \\ n, & wp.p. \end{cases} \tag{2.2}$$

gdzie każde D_i jest danym podzbiorem poziomów jasności, *i = 0,1,. . . ,n.*

Meritum jest tutaj dobór odpowiednich progów. Jeśli dla całego obrazu wybieramy taką samą wartość progową – wtedy mamy do czynienia z progowaniem globalnym. Czasami jednak ze względu, np. na nierównomierne oświetlenie sceny potrzebujemy zdefiniować konkretny próg dla lokalnych obszarów. Znalezienie optymalnego progu jest dość trudne.

2.5. Filtry cyfrowe

Filtry cyfrowe w przeciwieństwie do operacji punktowych, przeprowadzane są kontekstowo [3]. Oznacza to, że aby wyznaczyć wartość jednego piksela obrazu wynikowego należy dokonać obliczeń na większej liczbie pikseli obrazu źródłowego, które są w otoczeniu rozważanego piksela. W praktyce filtry są stosowane przede wszystkim do [3]:

— stłumienia w obrazie niepożądanego szumu,
— poprawy ostrości obrazu,
— usunięcia określonych wad obrazu,

— poprawianie obrazu o złej jakości technicznej.

Filtr jest pewną wieloargumentową funkcją matematyczną, przekształcającą jeden obraz na drugi, zmieniając po kolei każdy piksel [2]. Mogą one wykonywać operacje według pewnej liniowej kombinacji pikseli bądź bazować na nieliniowej funkcji obrazu wejściowego. W związku z tym rozróżniamy filtry liniowe oraz nieliniowe.

Filtr jest liniowy jeśli funkcja go opisująca jest addytywna.

Ważnym pojęciem jest splot funkcji, który zdefiniowany w skończonej dziedzinie staje się filtrem [3].

W związku z tym, że w komputerowej analizie obrazu dziedzina funkcji jasności $J_{w(x,y)}$ jest dwuwymiarowa i dyskretna wzór na splot wygląda następująco:

$$J_w(x,y) = \frac{\sum_{i,j \in K} J(x-i, y-j) w(i,j)}{\sum_{i,j \in K} w(i,j)}, \tag{2.3}$$

gdzie $J_{w(x,y)}$ to dyskretna funkcja jasności obrazu źródłowego, K - otoczenie rozważanego piksela, $w(i,j)$ – wagi otoczenia piksela (x,y). Jak wynika ze wzoru 2.3 obliczanie wartości punktu w obrazie wynikowym otrzymuje się poprzez sumowanie elementów obrazu $J_{(x-i,y-j)}$ z odpowiednimi wagami $w(i,j)$ wokół punktu (x,y). Dla każdego piksela obrazu operacja ta musi być wykonana oddzielnie.

Filtry dolnoprzepustowe

Filtry dolnoprzepustowe tłumią składowe widma sygnału o dużej częstotliwości [2]. Są stosowane do usuwania zakłóceń impulsowych i szumów z obrazu.

Najczęściej przyjmuje się, że szum występujący w obrazie jest nieskorelowany, addytywny i równomierny bądź ma rozkład Gaussa. Można w takim wypadku stosować prosty filtr uśredniający na bazie splotu obrazu z funkcją opisaną maską o danych współczynnikach, np.

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

Przedstawiona maska filtra usuwa drobne zakłócenia obrazu, a także wygładza krawędzie. Zarazem jednak powoduje „rozmycie" konturów obiektów oraz pogarsza wyrazistość kształtów znajdujących się na obrazie. Wielokrotnie stosowanie powyższego filtra spowoduje zmniejszanie ostrości obrazu oraz zanikanie elementów o małych rozmiarach. Zwiększenie środkowego współczynnika w masce pozwala zwiększyć stopień wpływu przetwarzanego piksela na końcowy wynik, co zmniejsza negatywne skutki filtrowania. Przykład zastosowania filtra dolnoprzepustowego przedstawia rysunek 2.1

Filtry górnoprzepustowe

Filtry górnoprzepustowe tłumią składowe o małych częstotliwościach. Zwiększają ostrość obrazu, co wiąże się również ze zwiększeniem szumów. Filtry tego typu służą do podkreślenia elementów takich jak kontury i krawędzie (w ogólności:

Rysunek 2.1. Zastosowanie filtra dolnoprzepustowego

elementy charakteryzujące się szybkimi zmianami jasności). Oto przykład maski filtra górnoprzepustowego:

$$\begin{matrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{matrix}$$

W grupie filtrów górnoprzepustowych możemy wyróżnić między innymi filtry wykrywające krawędzie. Są one bardzo ważne w procesie analizy obrazu.

Krawędzią nazywamy granicę pomiędzy dwoma obszarami o różnych odcieniach jasności. W praktyce oznacza to, że przejścia między tymi obszarami mogą być określone na podstawie różnic odcieni jasności pikseli. Ogólną ideą leżącą u podstaw większości technik wykrywania krawędzi jest porównywanie lokalnych pochodnych [2]. Pierwsza pochodna służy do stwierdzenia obecności krawędzi w obrazie, druga natomiast pozwala stwierdzić czy piksel krawędzi leży po jej jasnej czy ciemnej stronie. Przykład zastosowania takiego filtra przedstawia rysunek 2.2.

Rysunek 2.2. Zastosowanie filtra górnoprzepustowego

Ze względu na to, że gwałtowna zmiana jasności wyznacza krawędź, do wydzielenia jej z obrazu często stosuje się **metody gradientowe**. Duża wartość gradientu funkcji jasności w danym punkcie wskazuje, że jest to punkt krawędzi [2]. **Gradient** funkcji jasności obrazu $f(x, y)$ w punkcie (x, y):

$$\nabla f = \begin{bmatrix} G_1 \\ G_2 \end{bmatrix} \tag{2.4}$$

$$G_1 = \frac{\partial f}{\partial x} \approx f(x+1,y) - f(x,y), \quad G_2 = \frac{\partial f}{\partial y} \approx f(x,y+1) - f(x,y) \tag{2.5}$$

Wektor gradientu można opisać za pomocą dwóch parametrów:

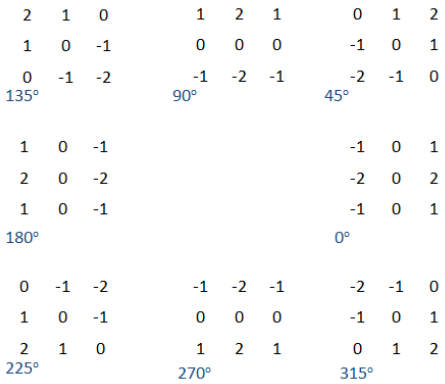
— *modułu, określającego wartość (siłę) krawędzi*

$$|\nabla f| = \sqrt{(G_1(x,y)^2 + G_2(x,y)^2)} \tag{2.6}$$

— *kierunku wektora gradientu (kąta), względem którego kąt konturu jest prostopadły*

$$\rho = arg\frac{G_2}{G_1} \tag{2.7}$$

Do obliczania gradientu obrazu wykorzystuje się wiele metod, między innymi **operatory Sobela**. Operator aproksymuje pierwszą pochodną, a gradient może być estymowany dla ośmiu różnych kierunków i największa wartość z nich wskazuje kierunek gradientu.



Rysunek 2.3. Maski Sobela

Operatory Sobela używane są jako detektory do wykrywania składowych krawędzi o orientacji poziomej, pionowej i skośnych. Maski Sobela przedstawia rysunek 2.3.

2.6. Wykrywanie krawędzi - algorytm Canny'ego

Algorytm Canny'ego jest metodą pozwalającą wykrywać krawędzie w obrazie. Jego twórca założył, że błędy detekcji polegające na wykryciu krawędzi nieistniejącej są jednakowo niepożądane jak niewykrycie krawędzi istniejącej. W związku z tym przyporządkował im równe wagi. Ponadto założył, że każda krawędź posiada stały przekrój poprzeczny i stałą orientację. Metoda Canny'ego korzysta z pierwszej oraz drugiej pochodnej. Ma ona trzy równoczesne cele [2]:

— *wykrycie poprawnej lokalizacji krawędzi*
— *minimalizacja liczby błędnych detekcji krawędzi*
— *minimalizowanie liczby lokalnych maksimów tworzonych przez szum wokół prawdziwego konturu*

2.6.1. Wygładzanie obrazu

Pierwszym etapem algorytmu Canny'ego jest wygładzanie obrazu za pomocą filtra Gaussa. Zgodnie z założeniem w artykule [1] Johna Canny'ego z 1986 roku, obraz jest zaszumiony co wynika z kwantyzacji, próbkowania i niedokładności czytników. W związku z tym, w celu wygładzenia obrazu, stosowany jest operator gaussowski. Za pomocą stałej σ można ustalić minimalną wielkość szczegółów w obrazie, które są istotne dla konkretnego zastosowania. Rysunek 2.4 przedstawia obraz po zastosowaniu przedstawionej maski Gaussa:

$$\begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix}$$

Rysunek 2.4. Zastosowanie filtra Gaussa

2.6.2. Obliczenie gradientu

Kolejnym etapem w algorytmie Canny'ego jest obliczenie gradientu w kierunkach poziomym, pionowym i ukośnym [3]. Można użyć do tego na przykład operatorów Prewitta, Robertsa czy Sobela. Następnie za pomocą wzorów 2.6 oraz 2.7 można obliczyć wynikową moc i kąty gradientu. W przypadku mocy należy pamiętać o normalizacji. Natomiast jeśli chodzi o kąt detekcji to jest on zaokrąglany do jednego z czterech przypadków: 0^o, 45^o, 90^o i 135^o.

2.6.3. Tłumienie niemaksymalne

Przy używaniu tradycyjnych metod gradientowych, krawędzie wykryte w obrazie są stosunkowo grube. Jest to związane z tym, że wynik działania operatora jest

zwykle porównywany z progiem. W przypadku gdy próg zostanie przekroczony to dany punkt zostaje uznany za krawędź.

Algorytm Canny'ego ma za zadanie uzyskać krawędzie o grubości jednego piksela. W tym celu szuka centrum konturu i odrzuca szum związany z tym konturem. W tym wypadku punkt krawędzi to punkt, którego moc jest maksymalna w danym kierunku gradientu. Dzięki temu eliminowane są piksele, które zostałyby zaliczone do krawędzi nawet po operacji progowania. Algorytm postępowania w tym wypadku wygląda następująco [2]:

1. Sprawdź czy przez każdy piksel sąsiadujący z pikselem badanym przechodzi krawędź (dla 8 kierunków). Jeżeli nie – to nadaj mu wartość 0.
2. Dla niezerowego piksela krawędzi sprawdź dwa sąsiednie piksele wskazane przez kierunek gradientu.
3. Jeżeli moduł gradientu któregokolwiek z sąsiadów jest większy od badanego piksela, to zaznacz badany piksel do usunięcia.
4. Przejdź do następnego piksela w kierunku prostopadłym do gradientu. Zaznacz wszystkie zbadane piksele.

2.6.4. Progowanie z histerezą

Ostatnim krokiem algorytmu jest progowanie z histerezą. Pomimo operacji wygładzanie filtrem Gaussa obraz najprawdopodobniej nadal będzie zawierał wiele fałszywych krawędzi, które wywołane są przez szum, bądź fragmenty tekstur. Jednakże kontrast tych krawędzi będzie mały.

W progowaniu z histerezą w algorytmie Canny'ego używane są dwa progi: wysoki – T_H oraz niski - T_L , gdzie T_H, $T_L \in (0,1)$. Najczęściej cała procedura zaczyna się od wyszukiwania piksela przyjmującego wartość wyższą niż wysoki próg. W efekcie wszystkie punkty spełniające to założenie zadeklarowane są jako pozytyw. Wszystkie piksele o wartościach mieszczących się między progami – połączone bezpośrednio z pozytywami, albo znajdujące się w serii z pikselami mieszczącymi się pomiędzy progami, są również deklarowane jako pozytywy. Pozostałe piksele, nie spełniające powyższych warunków, uznane są za negatyw i usuwane [2].

$$y = \begin{array}{ll} f(x,y) > T_H & pozytyw(1) \\ T_L < f(x,y) < T_H & negatyw(0) \;\; lub \;\; pozytyw(1) \\ f(x,y) < T_L & negatyw(0) \end{array} \tag{2.8}$$

2.6.5. Podsumowanie

Wszystkie kolejne kroki algorytmu pozwalają nam uzyskać obraz końcowy, ukazujący krawędzie o grubości jednego piksela. *Meritum* całego procesu jest dobranie odpowiednich parametrów w poszczególnych krokach, które mogą być bardzo różne w zależności od obrazu. To głównie od nich zależy efekt końcowy. W wielu przypadkach dobór konkretnych zmiennych może być związany z metodą prób i błędów.

2.7. Segmentacja obrazu

Segmentacja jest to podział obrazu na spójne fragmenty, różniące się w pewien sposób między sobą. Czasami także oznacza wyodrębnienie interesujących obiektów z tła. „Spójne" oznacza tu obszary o tym samym kolorze, jasności bądź podobnej teksturze. Najczęściej segmentacja polega na dzieleniu obrazu na dwie grupy: tło i obiekty [4].

Segmentacja krawędziowa oparta jest na granicach między obszarami, a jej wynikiem jest zbiór krawędzi.

Jedną z prostszych metod segmentacji jest śledzenie krawędzi obiektów, piksel po pikselu. Następnie aproksymuje się dany zbiór pikseli na odcinki.

Często przed zastosowaniem segmentacji może być pomocne rozpoznawanie obrazów, które pozwala na wykrycie konkretnych kształtów i obiektów.

2.8. Wektoryzacja obrazów rastrowych

Wektoryzacja jest to w ogólności proces przetworzenia rastrowego opisu obrazu na opis wektorowy. Samo wektoryzowanie może odbywać się na dwa sposoby: ręcznie, gdzie odrysowuje się na bitmapie krawędzie i kształty, które potem zostają zamienione na matematycznie opisane proste, krzywe i obsługiwane wielokąty bądź automatycznie, gdzie odpowiednie algorytmy same analizują i przekształcają obraz rastrowy na wektorowy.

Efektem procesu wektoryzacji dla wielu skomplikowanych obrazów bitmapowych jest zniekształcony i pozbawiony szczegółów obraz wektorowy. Im prostsze elementy składają się na bitmapę tym większa szansa, że zostanie ona poprawnie i wiernie przekształcona [6]. Warto tu zaznaczyć, że celem samej wektoryzacji nie jest dokładne odwzorowanie obrazka - żadne dostępne narzędzie nie potrafi tego zrobić.

Zmiana informacji rastrowej na wektorową jest nietrywialna, głównie ze względu na zmianę sposobu reprezentacji danych, a także w związku z faktem, że jest to przekształcenie niejednoznaczne, tzn. w szczególności może istnieć wiele reprezentacji wektorowych odpowiadających jednej reprezentacji rastrowej. Wynika to między innymi z faktu, iż każdy wektor może być reprezentowany jako suma mniejszych wektorów. Ogólnie nie jest to pożądane z punktu widzenia dalszego wykorzystania informacji, aczkolwiek może być rezultatem działania algorytmu wektoryzacji [7].

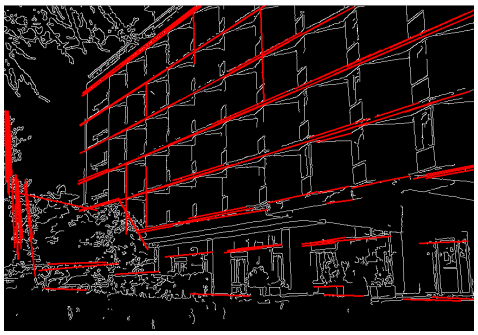
Już od wielu lat proponowane są różne techniki pozwalające na wektoryzację obrazów rastrowych. Doprowadziło to do powstawania wielu komercyjnych pakietów oprogramowania zajmującego się tym problemem. Wszystkie te systemy zapewniają całkiem akceptowalne wyniki na prostych, niezbyt zaszumionych bitmapach. Niemniej w przypadku większych obrazów, dodatkowo gorszej jakości, rezultaty są niezadowalające (na przykład w przypadku rysunków technicznych) i prowadzą do dużych kosztów całego procesu [13][14]. Ponadto, największym wymogiem obecnych metod jest odpowiednie ustawienie wartości zestawu parametrów, które dość często ustalane są czysto empirycznie.

Większość metod wektoryzacji opiera się na paru krokach, wliczając w to wykrywanie krawędzi oryginalnego obrazu, segmentację oraz zapis tychże krawędzi w postaci odcinków, a także różnego rodzaju metody poprawiające zapis obrazu

wynikowego. Te ostatnie skupiają się głównie na dodaniu jakiejś wiedzy kontekstowej, na przykład proste heurystyki poprawiające obraz wektorowy, łączenie linii, które są blisko siebie (i zostały np. rozdzielone przez brakujący piksel) czy biorąc pod uwagę charakter bitmapy - metody stopniowo upraszczające cały rezultat. Pomimo, że takie podejście może znacznie poprawić „surowy wynik" algorytmu, wiąże się to z wprowadzeniem dodatkowych progów i parametrów [15].

Wektoryzacje bitmap można z grubsza podzielić na trzy typy, w zależności od techniki podstawowej:

— **dopasowanie modelu parametrycznego**
Opiera się na użyciu modelu liniowego do detekcji linii w obrazie. Najbardziej ogólną i znaną techniką jest tutaj transformacja Hougha, która jednak nie jest zbyt szeroko stosowana, nie licząc niektórych bardzo prostych bądź szczególnych przypadków.
Transformacja Hougha jest transformacją globalną i addytywną obrazu. Charakteryzuje się szybkim czasem przetwarzania, ma jednak duże wymagania pamięciowe. J. Song i M.R. Lyu [26] zaprezentowali wektoryzację przy użyciu transformacji Hougha - przystosowaną jednak do analizy rysunków.
Inne modele parametryczne są stosowane w konkretnych i specyficznych przypadkach [23][25].



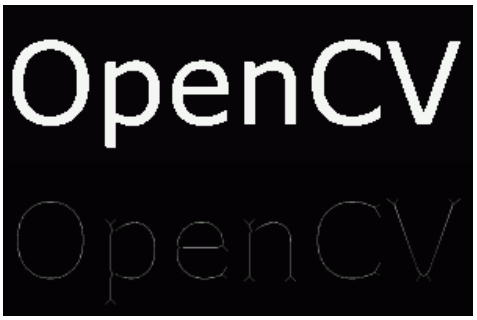
Rysunek 2.5. Przykład zastosowania transformacji Hougha za pomocą biblioteki OpenCV

— **śledzenie i dopasowanie konturów**
Jest to alternatywna technika badana pod koniec lat 80. ubiegłego wieku [16][17]. Polega na umieszczeniu na obrazie konturu konkretnego obiektu (bądź obiektów). Kontur jest następnie iteracyjnie przemieszczany do pobliskich krawędzi i dopasowywany do obrazu. Ta metoda zapewnia prawidłowe i dokładne wyniki, jeśli linie są proste. Niestety jest dość wrażliwa na wszelkie szumy i w przypadku skomplikowanych obiektów zupełnie się nie sprawdza, nawet pomimo długiego czasu obliczeń [18].

— **szkieletyzacja**
Jest ona zdecydowanie najczęściej stosowaną metodą przy wektoryzacji rastra [19][20][21][22]. Cała idea skupia się na obliczeniu osiowych punktów (szkieletów) obiektów w analizowanym obrazie. Szkielet figury jest najczęściej znacznie od niej mniejszy, niemniej całkowicie odzwierciedla jej topologiczne własności.
Kiedy już szkielet jest obliczony, problem wektoryzacji zredukowany jest do segmentacji dwuwymiarowej dyskretnej krzywej. Niestety problem komplikują przejścia i połączenia między grubymi kształtami (które mają postać bocznych

Rysunek 2.6. Przykład działania śledzenia i dopasowania konturów za pomocą biblioteki OpenCV

„gałązek" linii szkieletu) [24], bez względu na użytą metodę szkieletyzacji.



Rysunek 2.7. Zastosowanie szkieletyzacji za pomocą biblioteki OpenCV

Wszystkie powyższe metody mają tylko ograniczone szanse na odpowiednie przetwarzanie konturów i są wrażliwe na szumy. Stąd w obecnej postaci nie są najlepszym sposobem na wektoryzację większości obiektów architektonicznych.

Obecnie jest dostępnych wiele programów umożliwiających ogólną wektoryzację obrazów, między innymi:

— **Inkscape**
Jest to darmowy program do tworzenia i edycji grafiki wektorowej, stworzony w ramach projektu GNU. Udostępnia również narzędzie do wektoryzacji map bitowych. Wykorzystuje do tego silnik Portrace autorstwa Petera Selingera. Silnik ten interpretuje czarno-białe bitmapy, a następnie tworzy zestaw krzywych. Aktualnie Inkscape posiada trzy typy filtrów wejściowych przekształcających obrazek do akceptowanego formatu: rozdzielenie jasności, wykrywanie krawędzi oraz kwantyzacja koloru. Do każdej opcji mamy możliwość dopasowania konkretnych parametrów.
— **Adobe Illustrator**
Płatny odpowiednik programu Inkscape, firmy Adobe. Posiada obecnie więcej opcji i możliwości jeśli chodzi o narzędzie wektoryzacji niż Inkscape co w teorii pozwala na uzyskanie lepszych wyników.

2.9. Koncepcja rozwiązania

Pierwszym pomysłem na temat pracy inżynierksiej była ogólna wektoryzacja obrazów rastrowych. Postanowiłem skupić się jedynie na wektoryzacji obrazów należących do jednej, określonej dziedziny. Po wielu przemyśleniach i zbieraniu informacji na temat samego zagadnienia, wybór padł na obiekty architektoniczne.

W początkowym etapie pracy zacząłem gromadzić informacje, które mogłyby posłużyć mi w wymyśleniu odpowiedniego algorytmu, czytałem książki oraz artykuły na temat grafiki komputerowej i przetwarzania obrazów. Skupiłem się wtedy na znalezieniu odpowiedniego algorytmu do detekcji krawędzi, bo to od niego miało zależeć dalsze powodzenie całego procesu. Wybrałem algorytm Canny'ego, który mimo większego skomplikowania niż inne rozwiązania, pozwala na uzyskanie zadowalających i dość dokładnych wyników, a dodatkowo wykrywał krawędzie o grubości jednego piksela co miało być kluczowe przy późniejszej wektoryzacji.

W związku z tym, że program miał być napisany w C++, potrzebna była biblioteka umożliwiająca operacje na obrazach rastrowych i wektorowych. Atutem byłaby także możliwość stworzenia w prosty sposób przyjaznego użytkownikowi interfejsu graficznego. Mój wybór padł na bibliotekę Qt, która bez problemu spełniała powyższe założenia.

Rozważałem użycie gotowych rozwiązań przy pomocy biblioteki OpenCV, która posiada między innymi zaimplementowany algorytm Canny'ego. Jednak ze względu na ostateczną wydajność algorytmu, chciałem mieć maksymalną możliwość modyfikacji funkcji w autorskim rozwiązaniu, a nie jedynie dopasowywać parametry. Dlatego też wszystkie metody przetwarzania obrazu zaimplementowane w programie są napisane przeze mnie (z pomocą biblioteki Qt i metod pozwalających na dostęp do danych obrazu oraz jego wyświetlanie).

Kolejnym etapem było znalezienie odpowiedniej metody do segmentacji krawędziowej. Był to chyba największy problem w trakcie pisaniu całej pracy, ze względu na zadziwiająco małą ilość konkretnych informacji w tym temacie. Owszem, segmentacja krawędziowa jest wspominana w wielu książkach i pracach, niemniej w większości ogólnie, bez konkretnej propozycji rozwiązania. Bardziej szczegółowe informacje można było znaleźć dopiero w literaturze związanej z rozwiązaniem konkretnego problemu jak na przykład wektoryzacja obrazu wizyjnego robota czy wektoryzacja map topograficznych.

W trakcie implementacji samego algorytmu tworzyłem już interfejs graficzny, który miał ułatwić sam proces przetwarzania obrazu oraz umożliwić zmianę niektórych parametrów algorytmów. Kiedy wszystko było już gotowe zacząłem ostatecznie sprawdzać działanie na różnych obrazach przedstawiających obiekty architektoniczne. Algorytm spełniał swoje zadanie, niemniej trochę czasu zajmowało dobranie konkretnych wartości parametrów dla danego obrazu, aby wynik był zadowalający.

2.10. Wymagania funkcjonalne

1. Program „Ptah" powinien umożliwiać wektoryzację obrazów rastrowych przedstawiających obiekty architektoniczne w następujących formatach:
   — *Windows Bitmap (BMP)*
   — *Graphic Interchange Format (GIF)*
   — *Joint Photographic Expert Group (JPG/JPEG)*

— *Portable Network Graphics (PNG)*

2. Użytkownik powinien mieć możliwość zmiany parametrów poszczególnych algorytmów w programie.

3. Program powinien zapisywać obraz wynikowy w formacie SVG.

2.11. Wymagania niefunkcjonalne

1. Aplikacja napisana w języku C/C++ przy pomocy biblioteki Qt i kompilatora g++ 4.7.

2. Interfejs programu powinien być intuicyjny i nieskomplikowany w obsłudze przez standardowego użytkownika. W razie błędów czy nieodpowiednich działań powinien informować użytkownika o przyczynach zaistniałej sytuacji.

3. Program powinien informować użytkownika o postępie prac i fazie, w której się obecnie znajduje.

3. Implementacja

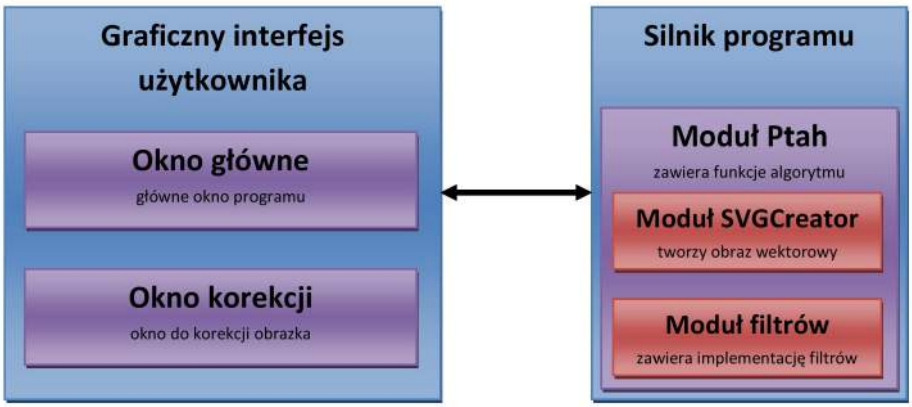
3.1. Opis programu

3.1.1. Wymagania

Program działa na systemach operacyjnych z rodziny Linux. Wymaga biblioteki Qt w wersji przynajmniej 5.2.0. Ilość wymaganej wolnej pamięci RAM zależna jest od wielkości przetwarzanego obrazu.

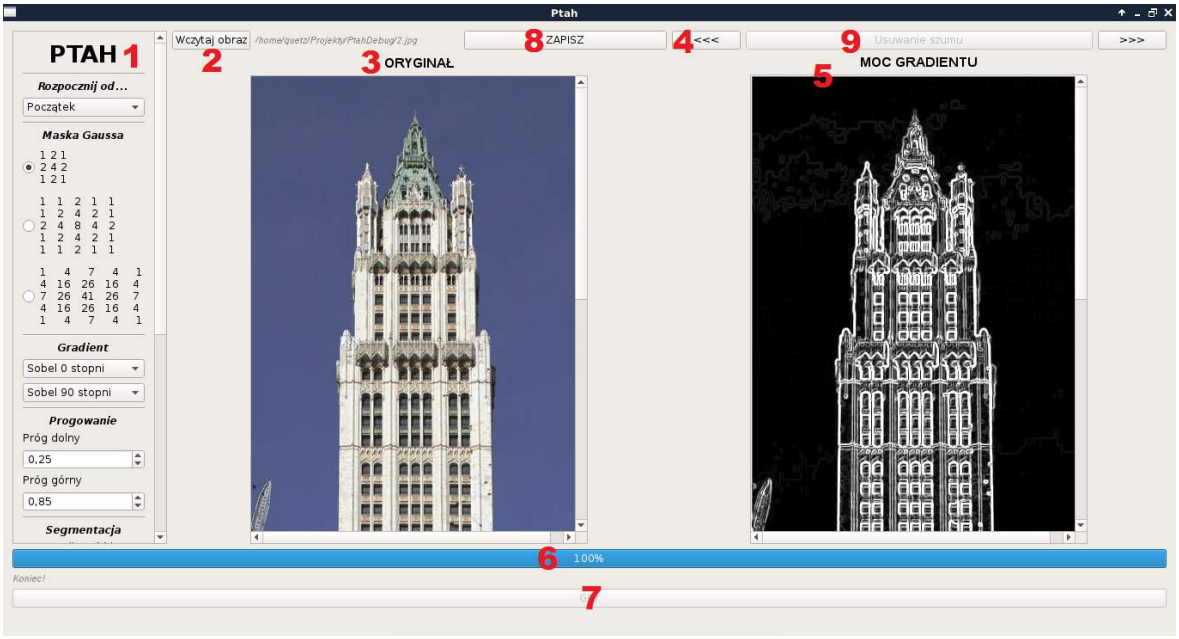
3.1.2. Opis okna programu

Program Ptah implementuje algorytm wektoryzacji obrazów rastrowych przedstawiających obiekty architektoniczne. Rysunek 3.1 przedstawia schemat programu, a rysunek 3.2 przedstawia jego główne okno.

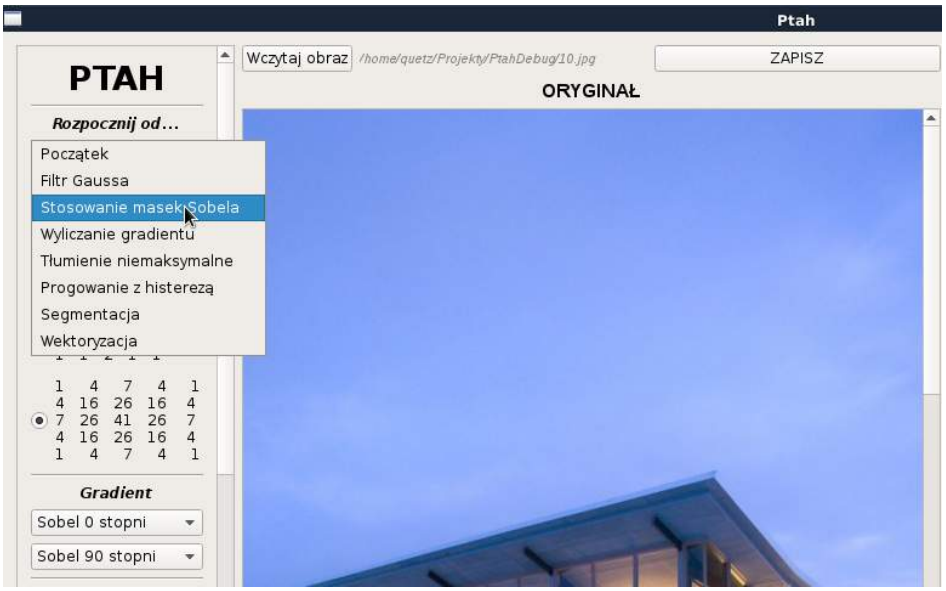


Rysunek 3.1. Struktura programu *Ptah*

1. **Panel sterowania** – pozwala na ustawienie parametrów i wartości związanych z przebiegiem algorytmu, a także wybór od jakiego momentu algorytm ma się wykonywać (Rysunek 3.3); jest aktywny dopiero po wczytaniu obrazu,
2. **Wybór obrazu** – opcja pozwala wybrać obraz rastrowy z dysku w formatach: .jpg, .bmp, .png bądź .gif (Rysunek 3.4). Po wczytaniu (udanym bądź nie) wyświetla się odpowiednia informacja,
3. **Oryginał** – wyświetla oryginał wczytanego obrazka, co pozwala w widoczny sposób porównać konkretny krok algorytmu z pierwowzorem,
4. **Klawisze nawigacji** – po wykonanym algorytmie pozwalają na nawigację między kolejnymi fazami procesu i jego podgląd krok po kroku,
5. **Obraz** – miejsce gdzie wyświetla się wczytany obraz bądź obraz po konkretnej fazie algorytmu (zależnie od kroku, w którym się znajdujemy); każdy krok jest podpisany nagłówkiem,

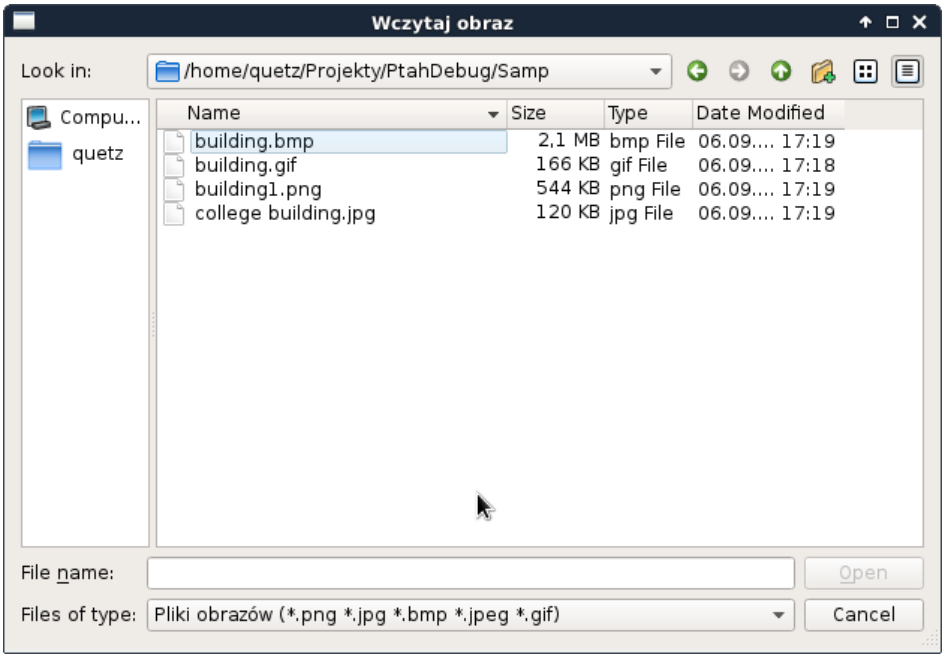


Rysunek 3.2. Okno programu *Ptah*



Rysunek 3.3. Okno wyboru kroku algorytmu

6. **Pasek postępu** – określa postęp algorytmu w trakcie jego działania oraz wyświetla informację o tym co aktualnie robi program,
7. **„Uruchom algorytm”** – przycisk uruchamiający algorytm na wybranym obrazie,
8. **Zapis** – przycisk zapisujący wyświetlony w danej chwili obraz (w formacie .png),
9. **Edytuj** – opcja pozwalająca ręcznie usunąć szumy bądź dodać brakujące krawędzie na obrazie wynikowym algorytmu Canny’ego.



Rysunek 3.4. Okno wyboru obrazu

3.2. Zastosowanie algorytmu Canny'ego

Pierwszym krokiem w implementowanym algorytmie jest wykrycie krawędzi w obrazie za pomocą metody Canny'ego. Każdy krok na tym etapie może wpłynąć na efekt końcowy dlatego tak ważny jest odpowiedni dobór parametrów.

3.2.1. Przekształcenie do skali odcieni szarości

W przypadku algorytmów wykrywających krawędzie (takich jak algorytm Canny'ego), warto dla uproszczenia i zmniejszenia ilości obliczeń przekształcić barwy w oryginalnym obrazie na odcienie szarości. Najprościej zrobić to poprzez uśrednienie trzech składowych wartości: czerwieni, zieleni i niebieskiego (RGB) dla danego piksela. Następnie każdej składowej przypisać wyliczoną średnią. Jest to oczywiście pewne uproszczenie, ale na potrzeby algorytmu spokojnie wystarczy. Całą procedurę powtarzamy dla każdego piksela w obrazie. Wynik procedury pokazany jest na rysunku 3.5.

$$B_{odc.szarosci} = G_{odc.szarosci} = R_{odc.szarosci} = \frac{R_{oryginalne} + G_{oryginalne} + B_{oryginalne}}{3} \tag{3.1}$$

3.2.2. Zastosowanie filtra Gaussa

Kolejny krok jest związany z rozmyciem Gaussa. W programie mamy do wyboru jeden z trzech wariantów pokazanych na rysunku 3.6.

Jest to ważny krok algorytmu, niemniej te trzy maski w zupełności wystarczają do uzyskania odpowiedniego obrazu do dalszej obróbki. W zależności od obrazu, każdy z filtrów pozwala na mniej lub bardziej zadowalające efekty. Na rysunku 3.7 mamy przykład zastosowania każdej z masek.

Rysunek 3.5. Zmiana barw pikseli obrazu na odcienie szarości poprzez uśrednianie.

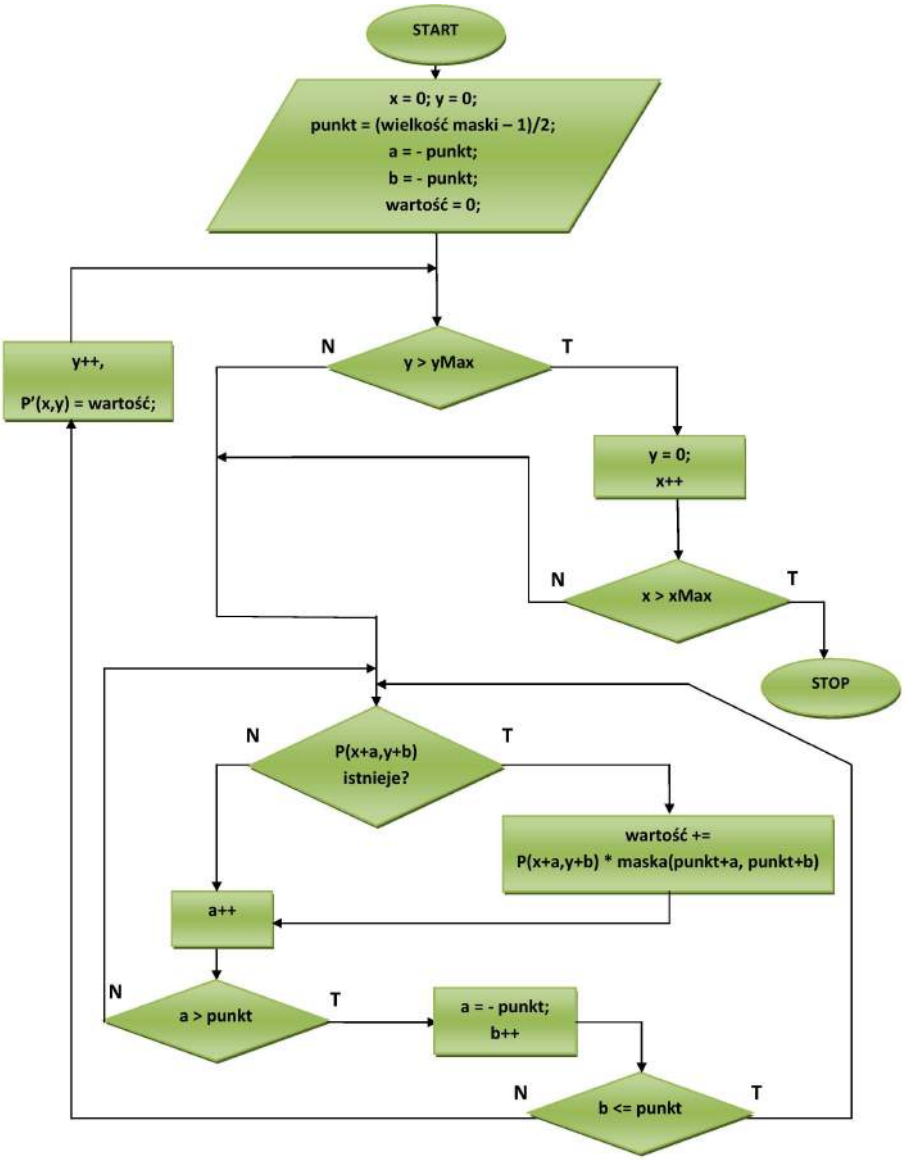
```
1 2 1
2 4 2
1 2 1

1  1  2  1  1
1  2  4  2  1
2  4  8  4  2
1  2  4  2  1
1  1  2  1  1

1   4   7   4   1
4  16  26  16   4
7  26  41  26   7
4  16  26  16   4
1   4   7   4   1
```

Rysunek 3.6. 3 maski Gaussa dostępne w programie Ptah.

Rysunek 3.7. Przykład zastosowania 3 dostępnych masek na tym samym obrazie.

Cała procedura filtrowania jest dość trywialna co pokazuje schemat blokowy na rysunku 3.8. $P(i,j)$ oznacza piksel oryginalnego obrazu o współrzędnych i i j, $P'(i,j)$ piksel obrazu wynikowego, natomiast $xMax$ oraz $yMax$ to szerokość i wysokość obrazu (pomniejszone o 1 ze względu na to, że zaczynami iterację od 0). Zmienne a i b pozwalają na łatwiejsze filtrowanie obrazu i upraszczają schemat blokowy. Działa to w następujący sposób: na początku obliczamy wartość zmiennej $punkt$, która zależna jest od wielkości maski. Określa ona maksymalną odległość

pikseli od piksela badanego w każdym z ośmiu kierunków (dla maski $3x3$ wyniesie 1, a dla maski $5x5$ - 2, itd.). Zmienne a i b zaczynają od wartości $-punkt$. Następnie są kolejno inkrementowane i dodawane do współrzędnej badanego piksela, dopóki nie osiągną wartości $punktu$. Ponadto z każdą inkrementacją zmiennej a lub b sprawdzane jest czy współrzędne piksela nie przekraczają zakresu obrazu. Warto tutaj zaznaczyć, że dla pikseli granicznych maska jest „obcinana" i nowa wartość wyliczana jest tylko na podstawie istniejących sąsiadów.



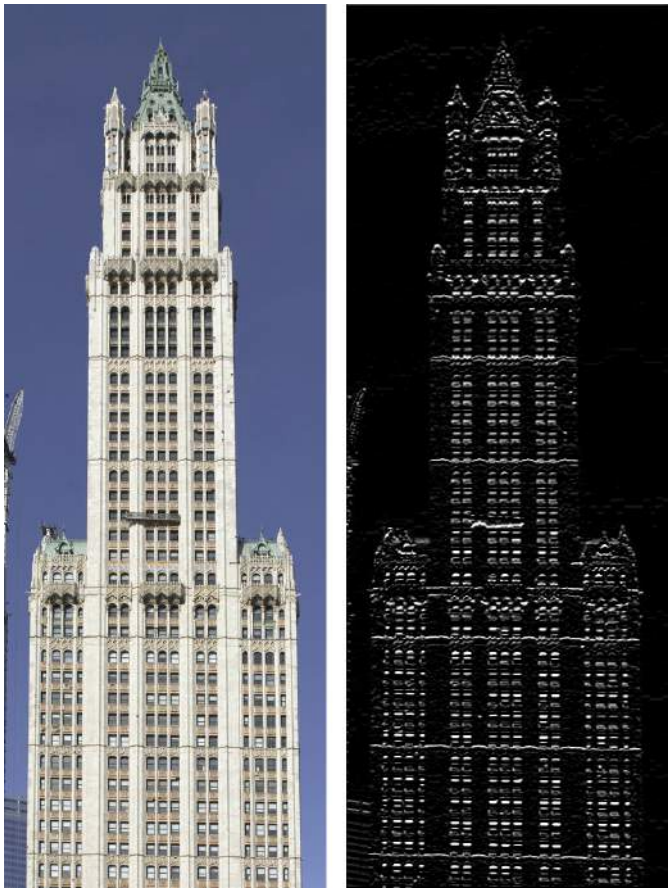
Rysunek 3.8. Schemat blokowy filtrowania

Wartości wyliczone dla każdego piksela obrazu wynikowego z tego etapu mogą mieć wartości wykraczające poza zakres $[0, 255]$, dlatego też do wyświetlenia obrazu pomocniczego (takiego, który może w programie zobaczyć użytkownik) są one normalizowane, tzn. wartościom mniejszym od 0 przypiswane jest 0, a większym niż 255 przypisywane jest 255. Do dalszych obliczeń i przekształceń są jednak używane

dane nieznormalizowane. Należy o tym pamiętać, ponieważ liczenie na wartościach znormalizowanych może skutkować gorszymi wynikami.

3.2.3. Zastosowanie kierunkowych filtrów Sobela

W programie mamy do wyboru cztery różne maski Sobela: $0^o, 45^o, 90^o$ oraz 135^o - odpowiadające czterem różnym kierunkom. W trakcie jednej iteracji algorytmu możemy na raz wybrać dowolne dwa z nich. Zastosowanie filtrów Sobela odbywa się w programie według tego samego algorytmu co zastosowanie filtrów Gaussa (pokazanego na schemacie 3.8). Przykładowe wyniki przedstawiają rysunki 3.9 oraz 3.10.



Rysunek 3.9. Przykład zastosowania maski Sobela (0^o).

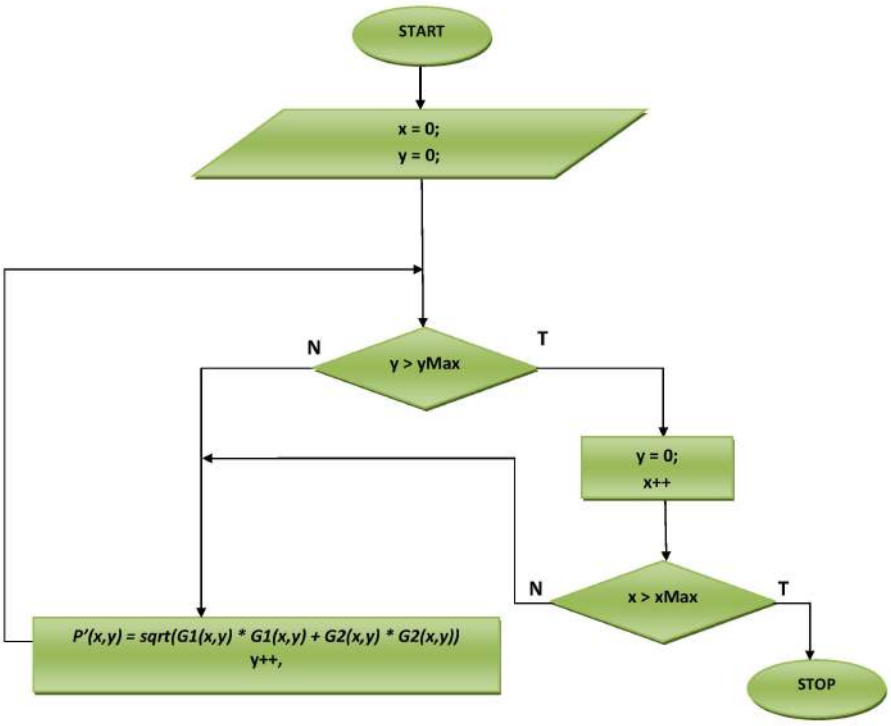
Podobnie jak i w poprzednim wypadku do wyświetlenia pomocniczego obrazu wynikowego należy znormalizować wartości.

3.2.4. Obliczanie mocy i kierunku gradientu

Z wyliczonymi wartościami pikseli po filtrowaniu dwoma maskami Sobela możemy przystąpić do wyliczania mocy gradientu, zgodnie ze wzorem 2.6. W programie jest to zaimplementowane zgodnie ze schematem blokowym widocznym na rysunku 3.11, gdzie $G1$ oznacza obraz po zastosowaniu pierwszej maski Sobela (dowolnie wybranej), a $G2$ drugiej.

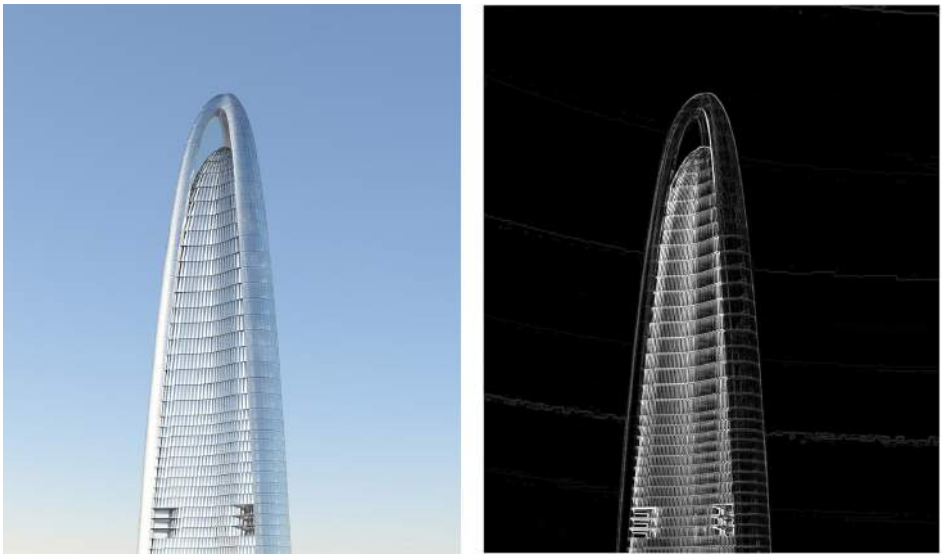
Rysunek 3.10. Przykład zastosowania maski Sobela (90^o).

Wyniki algorytmu przedstawiają rysunki 3.12 oraz 3.13.



Rysunek 3.11. Schemat blokowy obliczania mocy gradientu

Kolejnym krokiem jest wyliczanie kierunku gradientu. Wyliczoną ze wzoru 2.7 wartość przyporządkowujemy do jednego z czterech kierunków. Schemat blokowy przedstawia rysunek 3.14. D oznacza macierz kierunków gradientu w obrazie (otrzymaną za pomocą wzoru 2.7).Podzielenie wartości *arcus tangensa* przez π i pomnożenie przez 180 wynika ze zmiany jednostki z radianów na stopnie.



Rysunek 3.12. Moc gradientu przy maskach Sobela 0^o i 90^o.

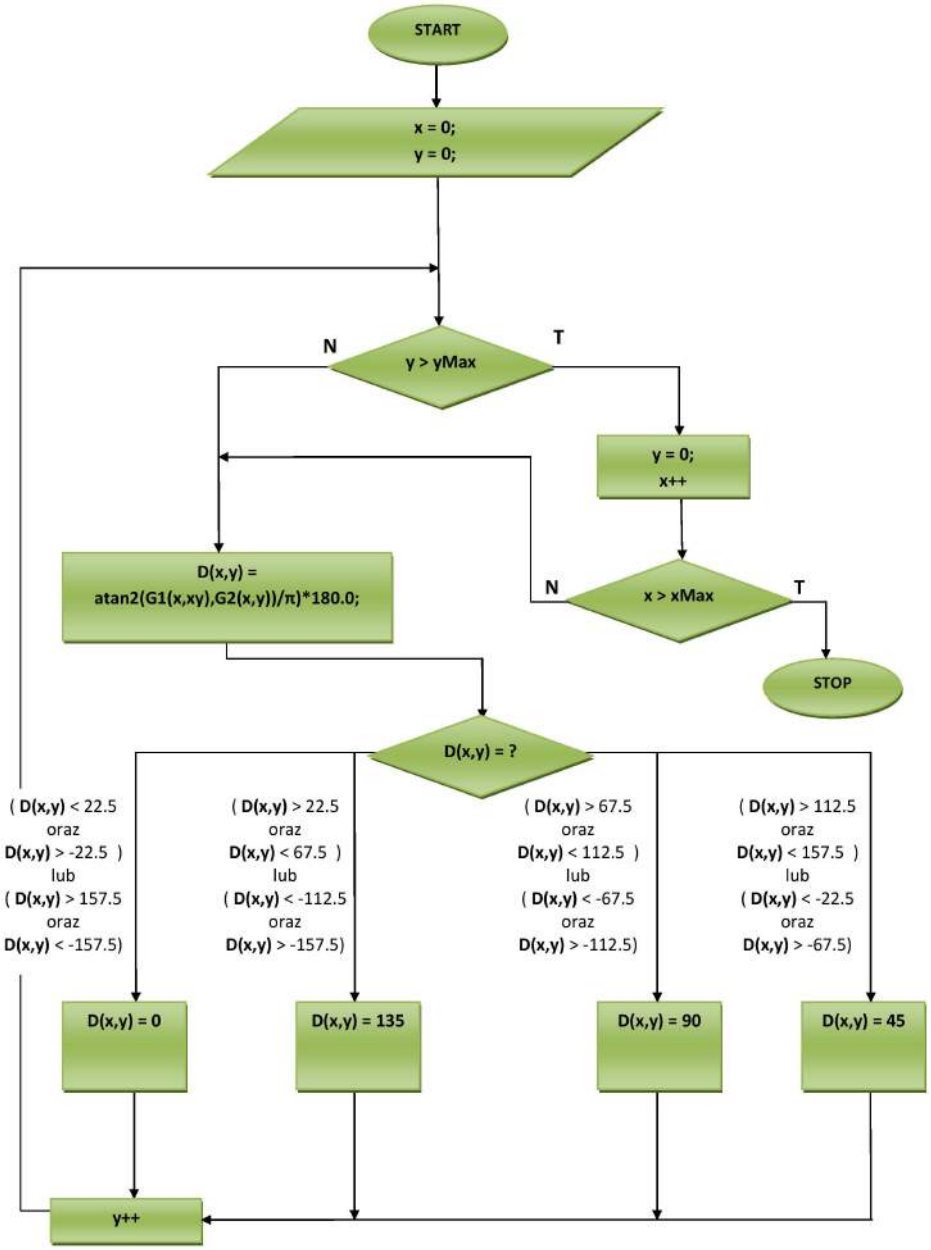
Rysunek 3.13. Moc gradientu przy maskach Sobela 0^o i 90^o.

W celach poglądowych program tworzy „obraz kierunków" gradientu i każdy z czterech kierunków oznacza inną barwą. Pozwala to użytkownikowi prześledzić postęp algorytmu i zrozumieć późniejsze efekty wykrywania krawędzi.

Pomimo, że wykrywane są jedynie cztery kierunki, powinno być to wystarczające nawet w wypadku znajdowania krawędzi pod kątami 30^o czy 60^o. Rysunek 3.15 pokazuje wynik algorytmu Canny'ego z zastosowaniem algorytmu ze schematu 3.14 oraz filtrów Sobela 0^o i 90^o.

3.2.5. Tłumienie niemaksymalne pikseli

Po wyliczeniu macierzy kierunków gradientu oraz mocy gradientu należy zastosować tłumienie niemaksymalne pikseli (opisane w rozdziale 2.6.3). Każdy punkt

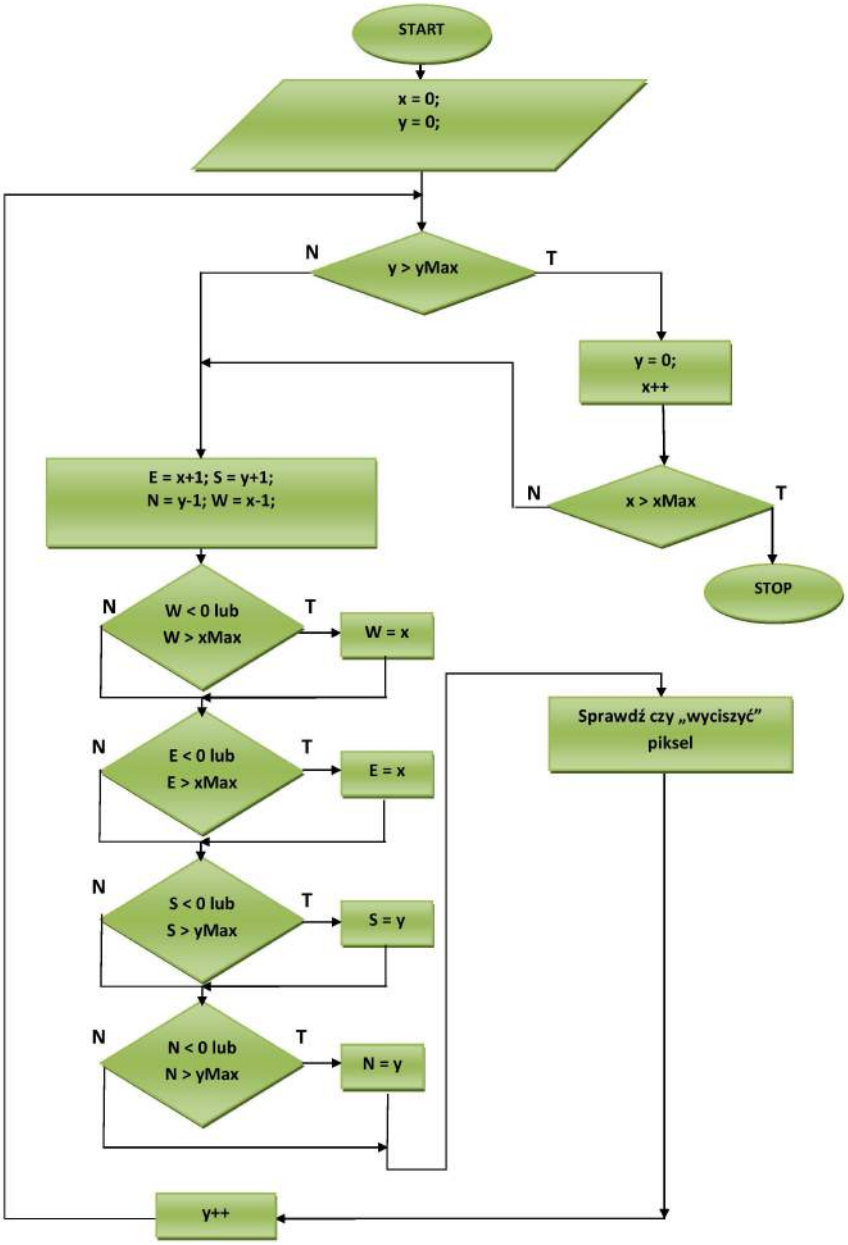


Rysunek 3.14. Schemat blokowy obliczania kierunku gradientu

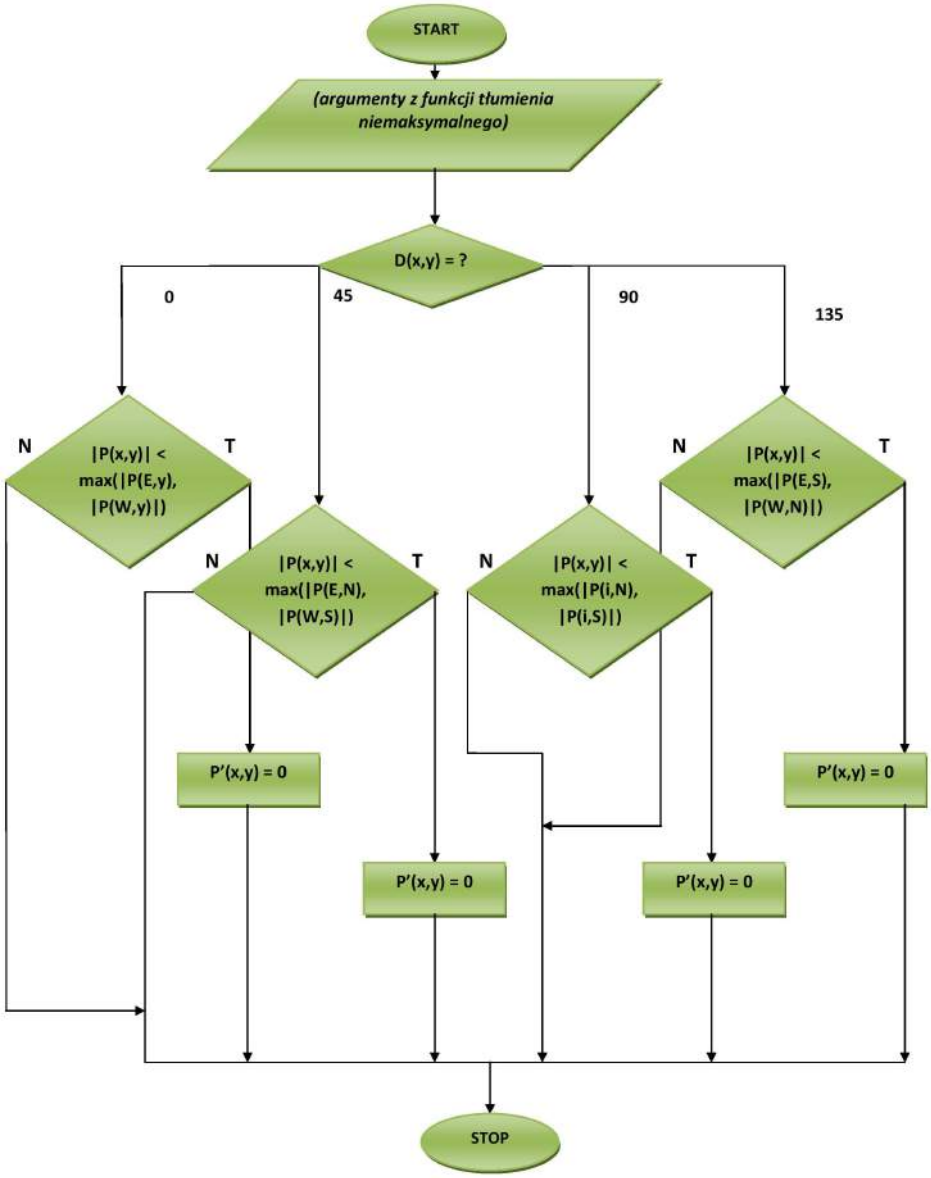
Rysunek 3.15. Wynik algorytmu Canny'ego dla obrazu o krawędziach pod kątami innymi niż 0^o, 45^o i 90^o

obrazu jest porównywany z dwoma sąsiadami - zależnie od jego kierunku gradientu. Schemat blokowy na rysunku 3.16 przedstawiaja rozwiązanie zaimplementowane w programie „Ptah". Nowe zmienne, które doszły do schematów blokowych to E, W, S oraz N. Są to po prostu zmienne pomocnicze przy określaniu kierunków (od angielskich nazw czterech stron świata), służące do sprawdzania sąsiadów badanego piksela. Jeśli któraś ze współrzędnych E, W, S lub N wychodzi poza zakres obrazu, tj. jest mniejsza niż 0 lub większa niż $xMax$ bądź $yMax$ (w zależności od przypadku), przypisywana jest jej wartość x bądź y badanego piksela.

Funkcję *sprawdź czy „wyciszyć" piksel* pokazuje schemat blokowy 3.17. „Wyciszenie" piksela to nadanie mu wartości 0.



Rysunek 3.16. Schemat blokowy tłumienia niemaksymalnego



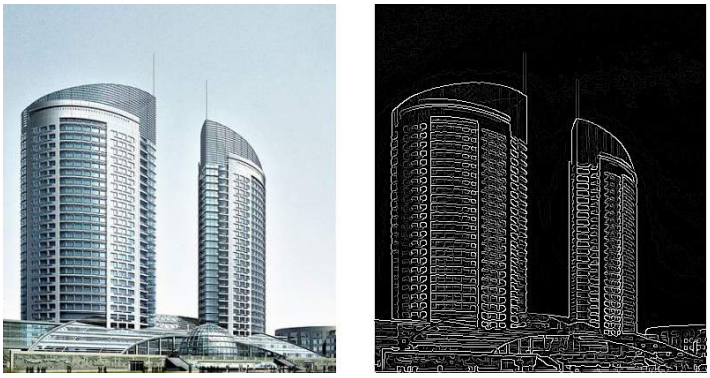
Rysunek 3.17. Schemat blokowy algorytmu sprawdzania czy piksel powinien być wyciszony

Rysunki 3.18 i 3.19 przedstawiają przykładowe efekty tego procesu.

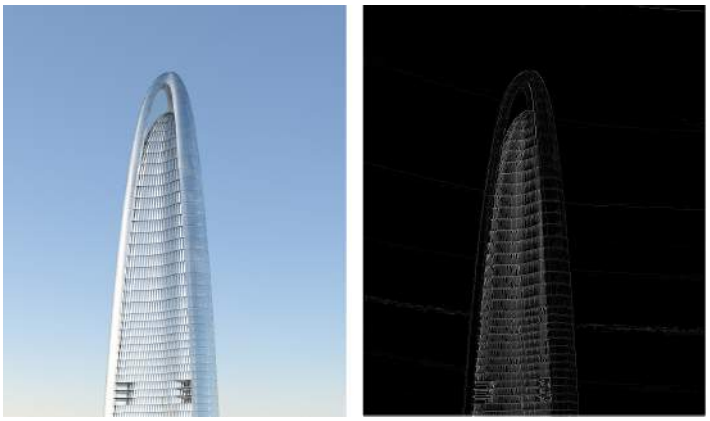
3.2.6. Progowanie z histerezą

Na wynik ostatniego kroku w algorytmie Canny'ego mamy wpływ poprzez wartości progów: T_l oraz T_h. Możemy je ustalić przed uruchomieniem całej procedury algorytmu.

Sama procedura zaimplementowana jest w sposób przedstawiony na rysunku 3.20. Zasada działania jest następująca: jeśli wartość piksela jest większa do górnego progu jest on „zapalany" - przypisywana jest mu wartość 255. Natomiast jeśli jest mniejsza od górnego ale większa od dolnego musi mieć przynajmniej jednego sąsiada, który spełnia pierwszy warunek. Pozostałe piksele są „wyciszane".



Rysunek 3.18. Przykład zastosowania tłumienia niemaksymalnego w programie Ptah (próg dolny: 0.65, próg górny: 0,9).



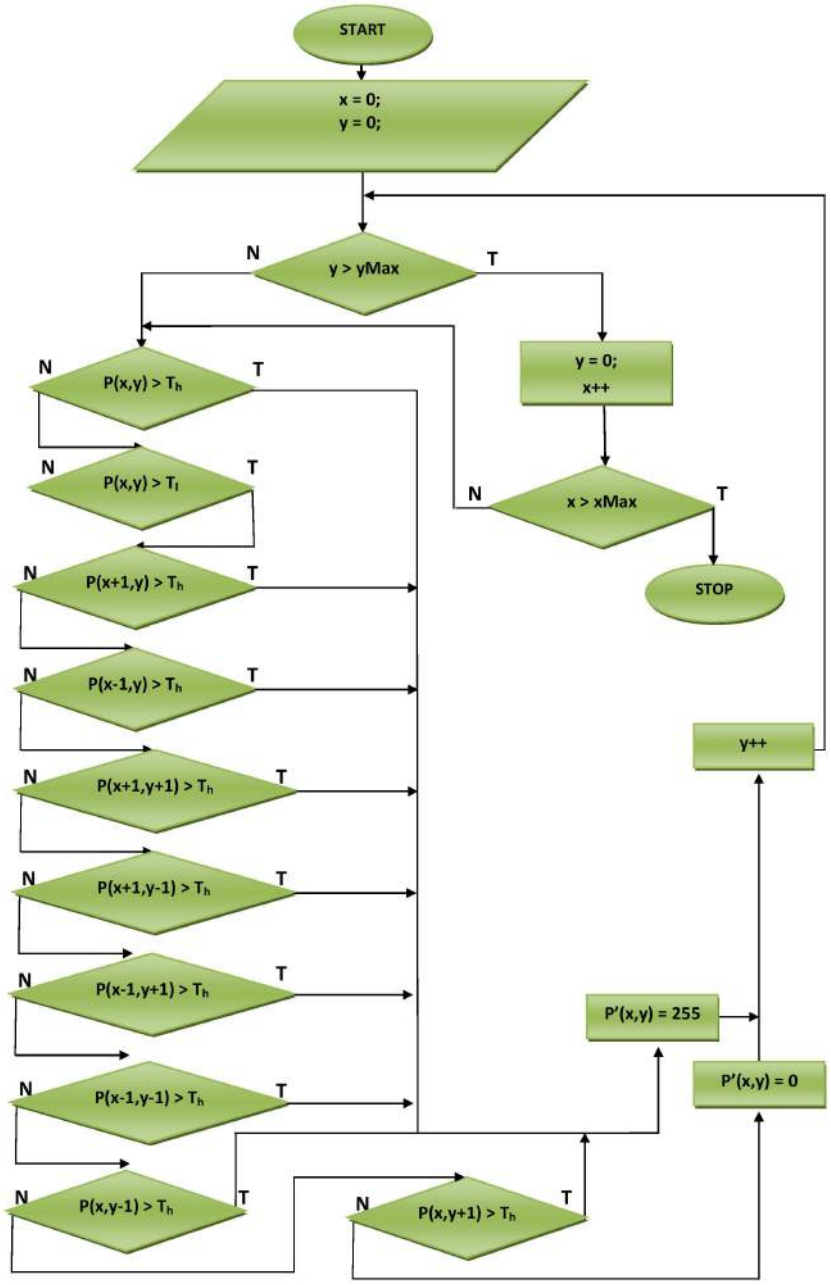
Rysunek 3.19. Przykład zastosowania tłumienia niemaksymalnego w programie Ptah (próg dolny: 0.85, próg górny: 0,9).

Należy pamiętać o przeskalowaniu progu do przedziału wartości na jakim operujemy - w przypadku programu Ptah wartość ustawiona przez użytkownika jest mnożona przez 255 - czyli długość przedziału wartości (ze względu na działania w systemie RGB).

Na koniec całego procesu algorytm wykonuje jeszcze jedną iterację po całym obrazie i zamienia czarne piksele na białe, a białe na czarne. Jest to ostateczna forma obrazu po całym algorytmie Canny'ego, gotowa do wektoryzacji.

3.2.7. Ręczna poprawa obrazu

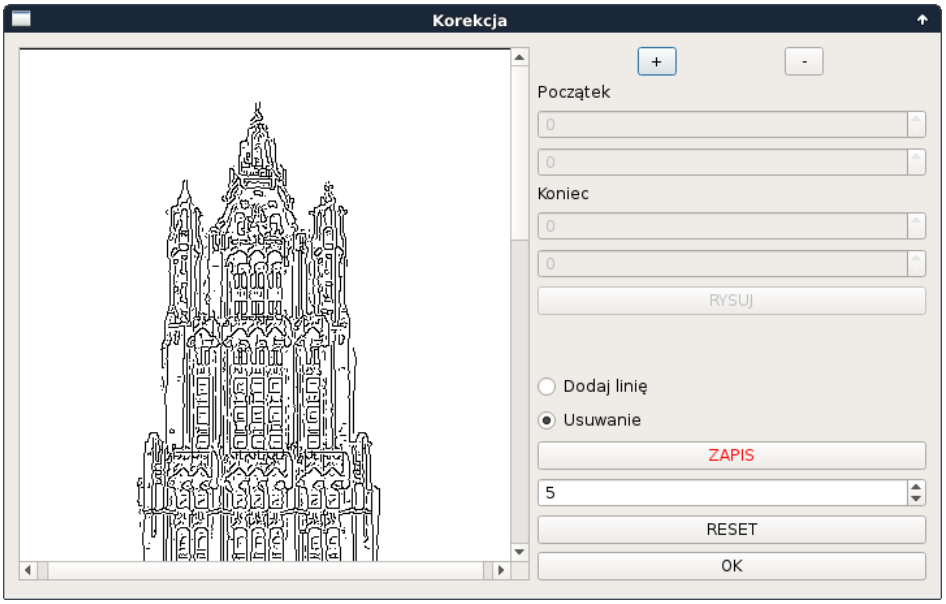
Niestety często zdarza się, że mimo wielu prób ustawienia odpowiednich wartości parametrów wynik wektoryzacji nie jest zadowalający. W takim wypadku program „Ptah" umożliwia użytkownikowi „ręczne" usuwanie szumów bądź dodawanie krawędzi (Rysunek 3.21). Ponieważ na tym etapie obraz jest binarny (piksele przyjmują jedynie wartości 0 (krawędź) bądź 255 (tło)) wszelka edycja jest dość trywialna, a zarazem ma ogromny wpływ na utworzone później wektory.



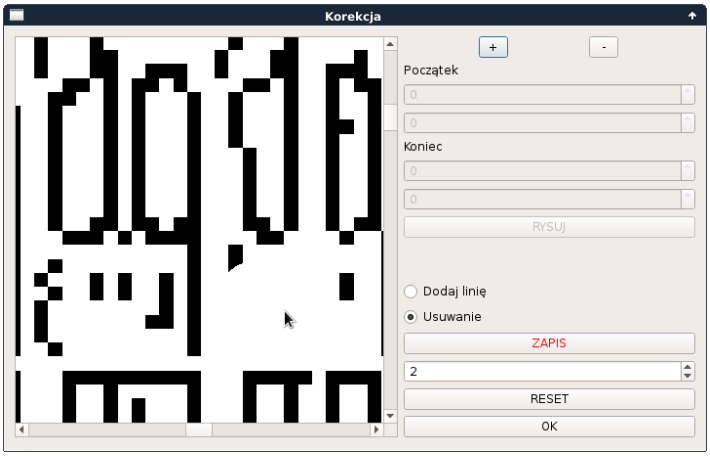
Rysunek 3.20. Schemat blokowy progowania z histerezą

W oknie edycji mamy możliwość skalowania obrazka (za pomocą przycisków „+" oraz „-"), co pozwala na dokładniejszą poprawę. Dzięki opcji usuwania pikseli możemy myszką „wymazać" zbędne punkty (Rysunek 3.22). Dostępna jest możliwość ustawienia promienia obszaru „wymazywania".

Druga z opcji dostępnych w edytorze to możliwość dodawania podłużnych linii (Rysunek 3.23) do obrazu. Po wyborze opcji „Dodaj linię" musimy wskazać dwa punkty w obszarze obrazka, które mają być punktem końcowym i początkowym odcinka. Istnieje też możliwość dodania linii bezpośrednio podając współrzędne punktów a następnie klikając przycisk „RYSUJ".



Rysunek 3.21. Okno ręcznej edycji w programie „Ptah”



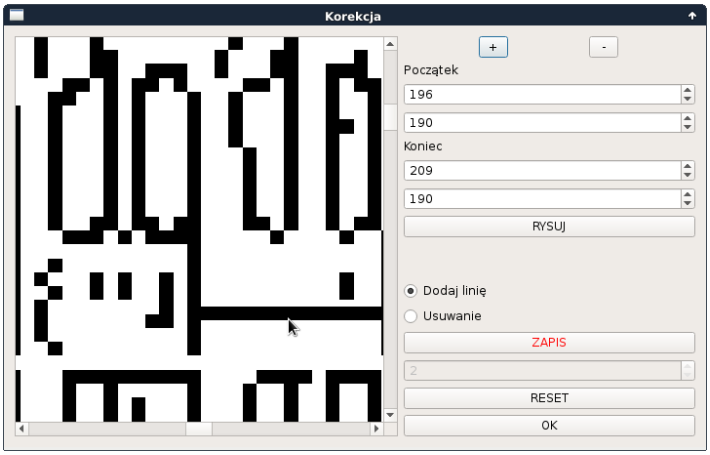
Rysunek 3.22. Przykład opcji usuwania pikseli za pomocą okna edycji

Zmiany zatwierdzamy przyciskiem „ZAPIS”, a w razie błędu czy pomyłki możemy kliknąć „RESET” w celu przywrócenia stanu obrazu pierwotnego.

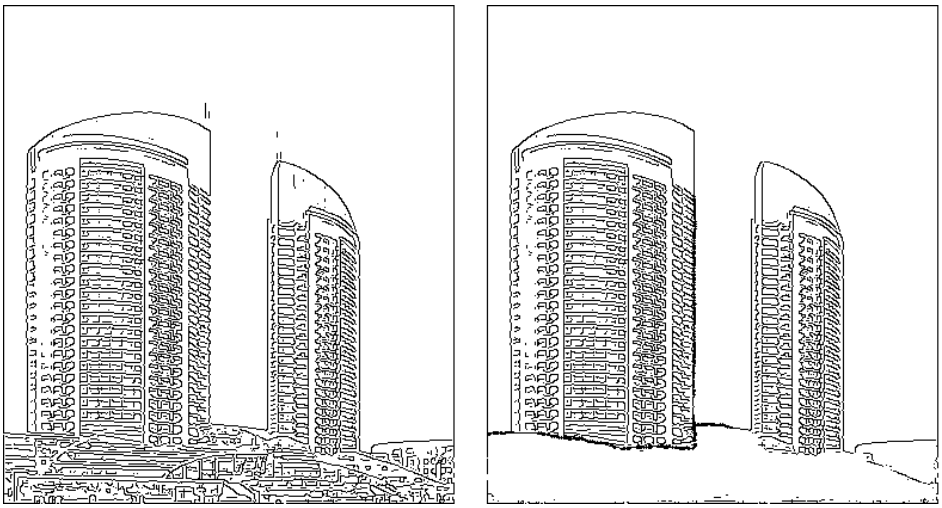
Po niezbędnych poprawkach użytkownik może wywołać algorytm od momentu segmentacji i uzyskać bardziej satysfakcjonujący obraz końcowy. Przykłady poprawek jakie można samemu wprowadzić przedstawiają rysunki 3.24 oraz 3.25.

3.3. Wektoryzacja

Wykorzystywany przeze mnie algorytm wektoryzacji opiera się częściowo na idei kodu Freemana i metodzie śledzenia krawędzi, czyli znajdowania przebiegu linii poprzez wyszukiwanie kolejnych „zapalonych” (w tym wypadku czarnych – RGB(0,0,0)) pikseli w sąsiedztwie aktualnie przetwarzanego [8].



Rysunek 3.23. Przykład opcji dodawania linii za pomocą okna edycji

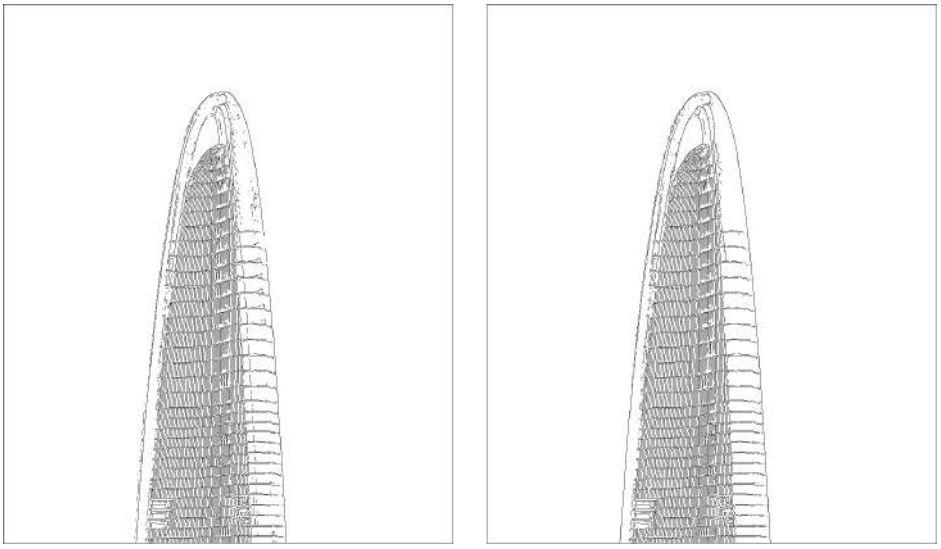


Rysunek 3.24. Obraz przed i po „ręcznej" edycji użytkownika.

Cała procedura zaczyna się od znalezienia czarnego piksela nienależącego do żadnego skonstruowanego wcześniej wektora. Jego współrzędne są współrzędnymi początku i końca wektora. Nazwijmy go pikselem „startowym" (Wzór 3.2).

$$\vec{v_b} = \begin{bmatrix} x_0, & y_0 \\ x_0, & y_0 \end{bmatrix} \tag{3.2}$$

Następnie przeszukiwane jest najbliższe sąsiedztwo końca wektora i wyszukiwane „zapalone" piksele. W przypadku znalezienia takiego punktu, sprawdzamy czy utworzony w ten sposób wektor jest właściwy. W tym wypadku oznacza to sprawdzanie czy każdy dotychczasowo odwiedzony punkt nie jest oddalony od potencjalnego wektora bardziej niż predefiniowane dopuszczalne odchylenie d_{max}. Możemy to sprawdzić za pomocą wzoru 3.3. $p(n)$ oznacza każdy kolejny piksel (punkt) należący do potencjalnego wektora, a zmienne x oraz y - współrzędne punktów.



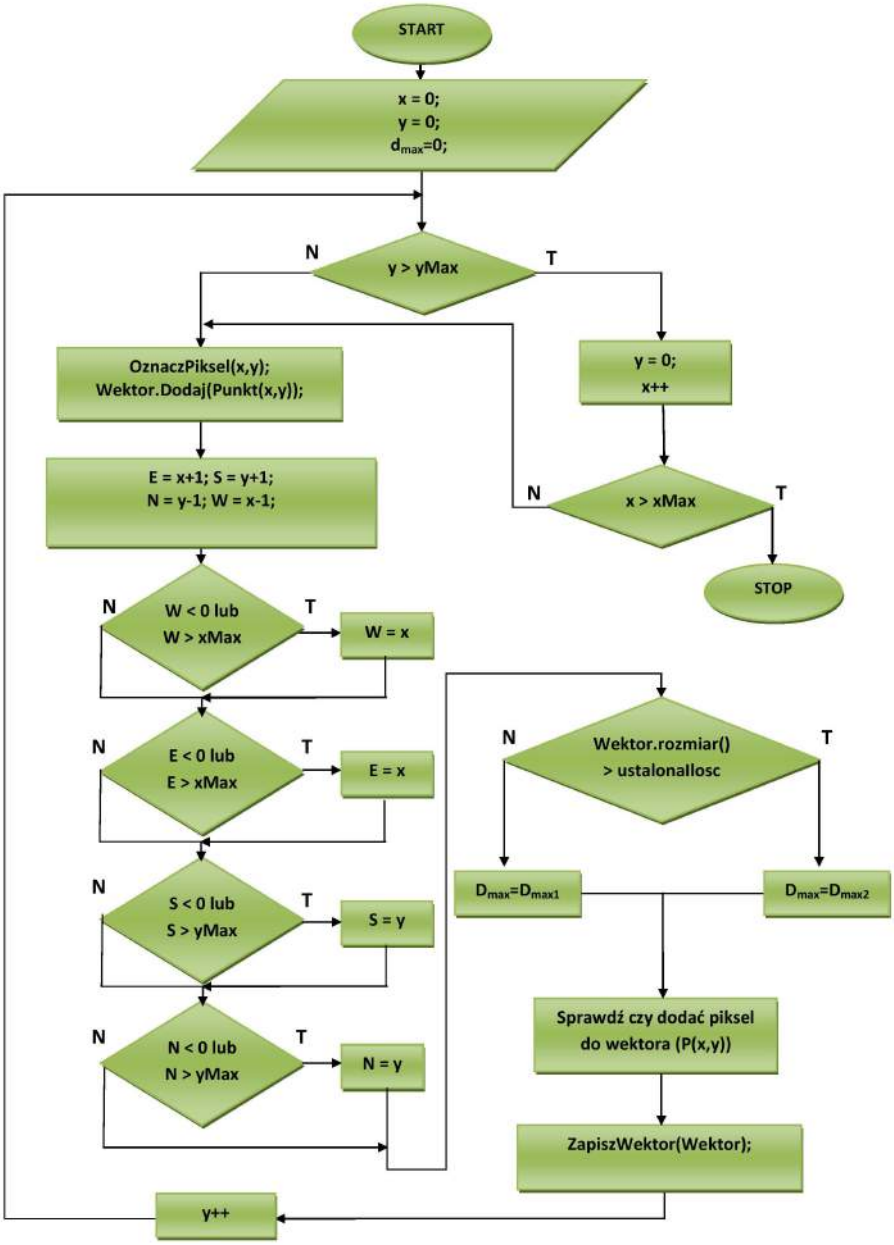
Rysunek 3.25. Obraz przed i po „ręcznej" edycji użytkownika.

$$\forall p(n) \begin{cases} dla: \left|\frac{y_0-y_h}{x_0-x_h}\right| < 1; \left|\frac{y_0-y_h}{x_0-x_h} * (p_x(n) - p_x(0)) + p_y(0) - p_y(n)\right| < d_{max} \\ dla: \left|\frac{y_0-y_h}{x_0-x_h}\right| \geq 1; \left|\frac{x_0-x_h}{y_0-y_h} * (p_x(n) - p_x(0)) + p_y(0) - p_y(n)\right| < d_{max} \end{cases} \tag{3.3}$$

Jeśli punkty spełniają powyższe kryterium, punkt uznawany jest za nowy koniec wektora i algorytm działa dalej, wykonując kolejne iteracje. W przeciwnym razie wektor jest zwracany i następuje ponowne wyszukanie „zapalonego" piksela, który stanie się początkiem nowego wektora. Tutaj dochodzi jeszcze kwestia długości otrzymanych wektorów. W przypadku, gdy obraz po zastosowaniu algorytmu Canny'ego zawiera jeszcze jakieś szumy (np. kilkupikselowe krawędzie, które nie należą do szkieletu budynku) warto określić minimalną długość przyjmowanych wektorów. Program Ptah umożliwia to poprzez parametr przyjmujący wartości od 1-1000. W zależności od obrazu może to usunąć powstałe podczas wcześniejszego przetwarzania szumy, bądź spowodować nieciągłość prawdziwych krawędzi.

W programie zostało zrealizowane to w sposób następujący: współrzędne każdego znalezionego piksela (tworzącego poprawny wektor) wpisywane są do tablicy. W momencie końca rozbudowy danego wektora, tablica jest zapisywana, aczkolwiek w kolejnych krokach wektoryzacji interesują nas tylko elementy: pierwszy i ostatni, które są zarazem początkiem i końcem wektora. Pozostałe piksele służą zobrazowaniu użytkownikowi do jakiego wektora został przyporządkowany dany punkt. Schematy blokowe algorytmu pokazuje rysunek 3.26. Schemat na rysunku 3.27 pokazuje przebieg funkcji ze schematu 3.26 sprawdzającej czy piksel powinien zostać dodany do wektora.

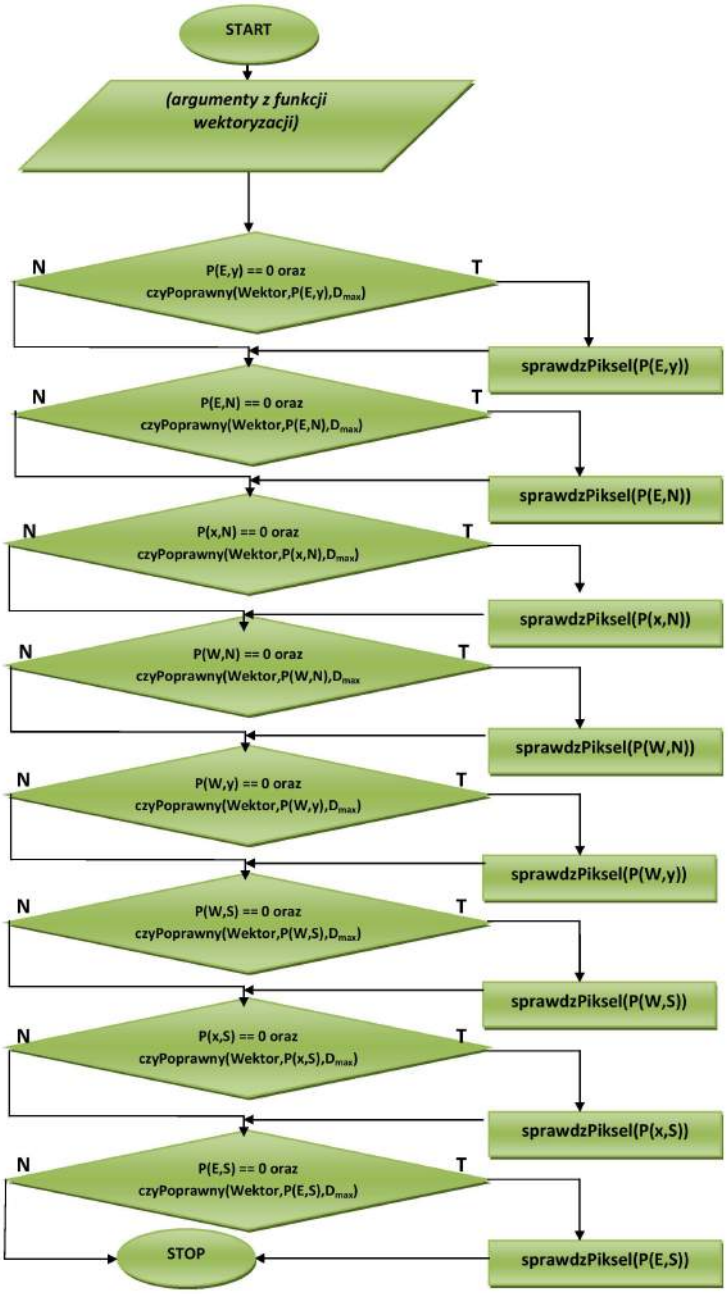
Takie rozwiązanie zapewnia elastyczność w poszukiwaniu odcinków dopuszczając pewne odchylenia pikseli od idealnej prostej. Jest to ważne, choćby ze względu na dyskretny charakter ich położenia. Ponadto ogranicza poszukiwania do punktów leżących w określonym obszarze marginesu wzdłuż budowanego wektora.



Rysunek 3.26. Schemat blokowy algorytmu tworzącego wektory z wykrytych krawędzi

Na obrazie mogą znajdować się zaokrąglone krawędzie. W takim wypadku algorytm wektoryzacji przedstawi je za pomocą łamanej, czyli zbioru wielu krótkich odcinków połączonych ze sobą. Przykład takiego obrazu widać na Rysunku 3.30.

Znaczenie ma tutaj dobór współczynnika d_{max}. W związku z tym, że dla dużej wartości współczynnika często trudno utworzyć z kilku punktów spełniający założenia wektor, warto na początku przyjąć niższą wartość, a po konstrukcji odcinka z 4-5 pikseli zwiększyć ją [7]. Pozwala to osiągnąć lepsze efekty. W programie Ptah mamy możliwość wyboru dwóch wartości d_{max} i numeru piksela, po którym wartość używana w algorytmie ma się zmienić.

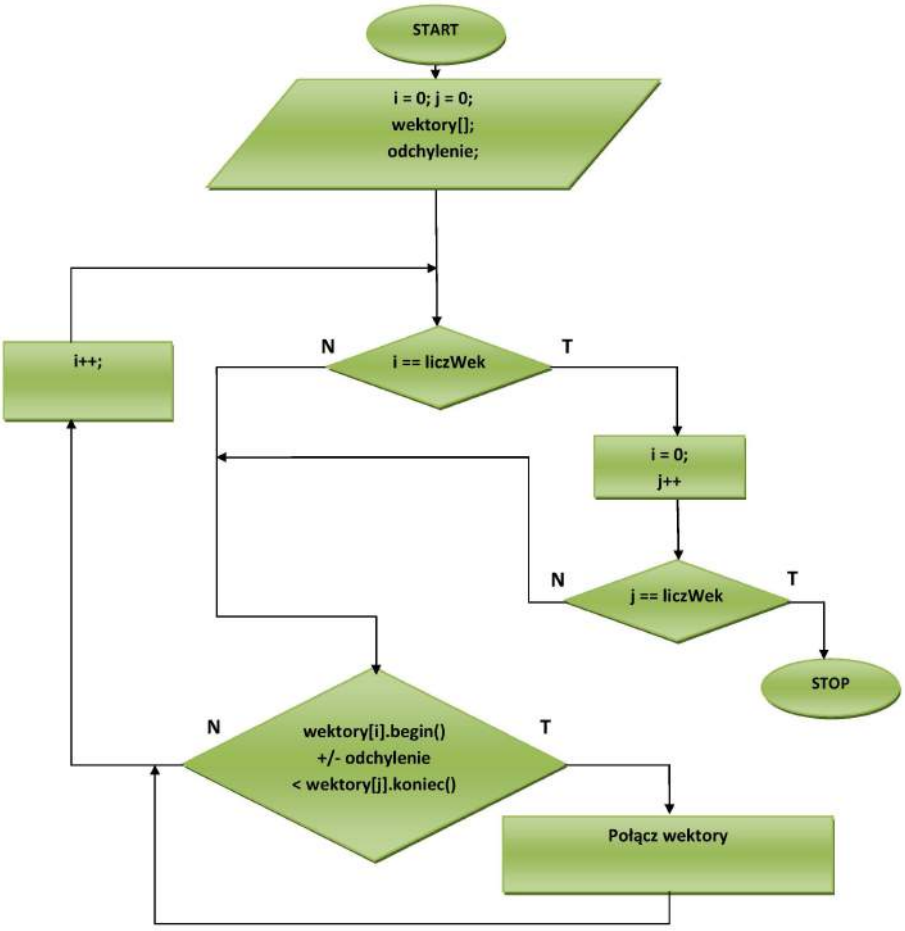


Rysunek 3.27. Część decyzyjna algorytmu z rysunku 3.26

Należy pamiętać o tym, by każdy odwiedzony i dodany do wektora piksel, odrzucać przy budowie kolejnych wektorów – niweluje to problem zapętlenia funkcji rekurencyjnej.

„Surowy obraz" wyjściowy może nie do końca spełniać wymogi - na przykład niektóre krawędzie mogły zostać niepotrzebnie podzielone na kilka wektorów (choćby brak jednego czy dwóch pikseli może ten problem powodować). Funkcja łączenia wektorów pozwala nam na przejrzenie całej listy linii, które znalazł algorytm i w razie możliwości - połączenia ich. W programie „Ptah" mamy możliwość ustalenia

odchylenia w punktach (tzn. na jaką maksymalnie odległość możemy szukać rozłączonych pikseli) oraz ilości iteracji po liście wektorów. Przebieg algorytmu pokazuje schemat blokowy 3.28. *Odchylenie* to parametr ustawiany przez użytkownika - określa on maksymalną odległość o jaką mogą być oddalone od siebie dwa wektory (a dokładnie - punkt początku jednego i punkt końca drugiego). *Wektory* to kontener przechowujący wszystkie znalezione wektory, natomiast *liczWek* to ilość wszystkich wektorów w tym kontenerze. Zmienne i oraz j służą do iteracji.



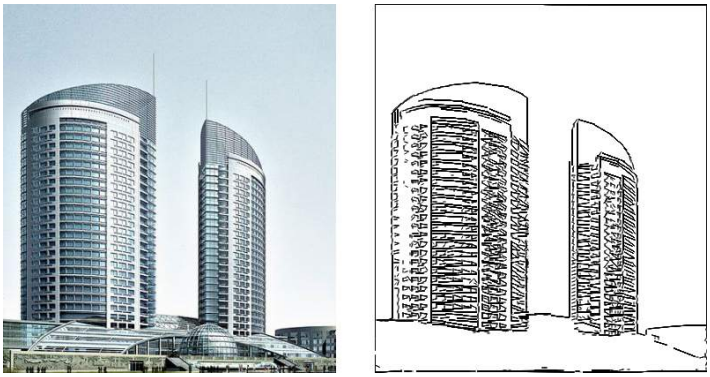
Rysunek 3.28. Schemat blokowy algorytmu łączenia wektorów.

Zgodnie ze schematem 3.28 każdy wektor jest porównywany z pozostałymi. W przypadku gdy dwa wektory mogą być połączone w jeden, tworzony jest nowy o współrzędnych początku pierwszego i współrzędnych końca drugiego, a stare wektory są usuwane.

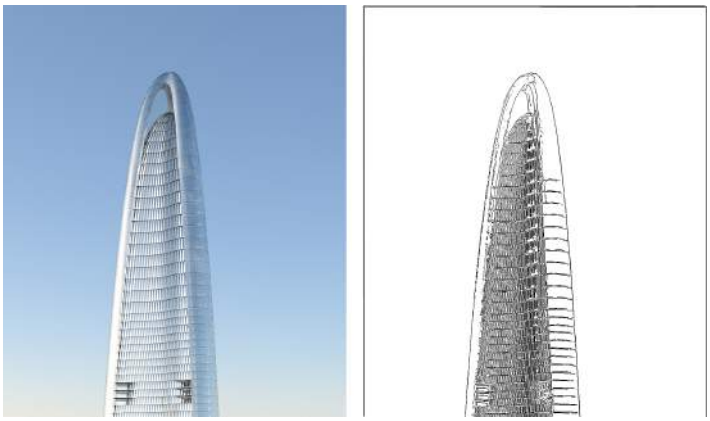
Cały proces dość znacznie wydłuża czas trwania algorytmu. Możemy jednak uzyskać o wiele bardziej przejrzysty obraz wynikowy, a także zmniejszyć jego pamięć (np. niwelując linie o długości jednego piksela i łącząc je z dłuższymi wektorami).

3.4. Zapis w postaci wektorowej

Zapis do postaci wektorowej w programie Ptah przebiega z pomocą biblioteki Qt, która posiada klasę **QSvgGenerator** pozwalającą tworzyć pliki w formacie SVG. W praktyce sprowadza się to do dodawania kolejnych elementów do obiektu tej klasy, a dokładnie podawania współrzędnych początku i końca linii (a tak naprawdę naszych wcześniej wyliczonych wektorów), które są następnie rysowane. Na końcu zapisywany jest plik w formacie SVG o tej samej nazwie co plik źródłowy.



Rysunek 3.29. Przykładowy obraz wynikowy algorytmu wektoryzacji.

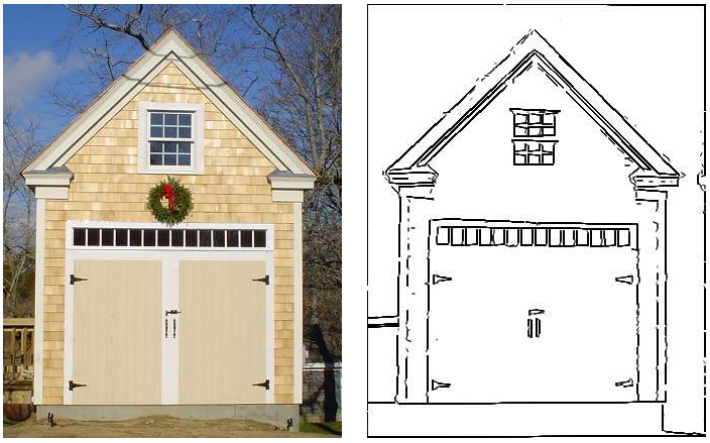


Rysunek 3.30. Przykładowy obraz wynikowy algorytmu wektoryzacji.

Rysunki 3.29, 3.30 oraz 3.31 przedstawiają pierwotny obrazek (po lewej stronie) oraz ostateczny wynik działania algorytmu w programie Ptah (po prawej stronie).

3.5. Wnioski

Liczne testy różnych obrazów przedstawiających obiekty architektoniczne dowodzą, że algorytm nie zawsze radzi sobie z rastrami, na których budynki są w jakiś sposób zasłonięte przez inne elementy (np. drzewa). W zależności od parametrów jakie podamy w konkretnych krokach algorytmu, możemy w takich wypadkach uzyskać fałszywe krawędzie, które nie należą do budynku albo zatracić informację o części krawędzi wektoryzowanego obiektu.



Rysunek 3.31. Przykładowy obraz wynikowy algorytmu wektoryzacji.

Jeśli jednak wynik nadal nas nie zadowala, zostaje nam możliwość ręcznego usunięcia szumów w obrazie wynikowym algorytmu Canny'ego bądź dodanie brakujących krawędzi.

Algorytm dobrze sprawdza się przy obrazach, na których mamy budynek na jednolitym tle, bez zbędnych, innych obiektów oraz z w miarę jednolitym oświetleniem (czasami cienie na budynkach powodowały wykrycie błędnych linii). Algorytm Canny'ego sprawdza się w tym wypadku bardzo dobrze i wykrywa krawędzie prawidłowo. To z kolei wiąże się z lepszą wektoryzacją, podczas której powstaje mniej wektorów, ale za to dłuższych.

4. Badania

4.1. Baza sprzętowa

Wszystkie badania zostały przeprowadzone na komputerze wyposażonym w procesor Intel Core 2 DUO 2,20 GHz, pamięć RAM 3GB, kartę graficzną NVIDIA GeForce GT 320M (1GB) oraz system operacyjny Debian (wersja "Jessy") ze środowiskiem xfce.

4.2. Badanie jakości

Badanie to miało na celu porównanie wyników wektoryzacji użytej w programie „Ptah" z rozwiązaniami dostępnymi w innych programach. Dzięki temu można było zobaczyć mocne i słabe strony użytego algorytmu i na podstawie tego wyciągnąć odpowiednie wnioski.

Rysunek 4.1. Pierwszy obraz badany.

Rysunek 4.2. Drugi obraz badany.

4.2.1. Plan badań jakości

1. Przygotowanie trzech różnych obrazów przedstawiających obiekty architektoniczne w formacie .jpg (rysunki 4.1, 4.2 oraz 4.3).
2. Użycie algorytmów wektoryzacji w programach: Ptah (bez ręcznej edycji) oraz Inkscape (wersja 0.48.4). Program Inkscape został wybrany ze względu na darmową licencję.
3. Analiza i interpretacja wyników.

Poniżej przedstawiam parametry użyte do wektoryzacji obrazów w poszczególnych programach:

1. **Obraz I**
   a) Ptah

Rysunek 4.3. Trzeci obraz badany.

i. Maska Gaussa nr 1
ii. Filtry Sobela - 0^o oraz 90^o
iii. Próg dolny - 0.5, próg górny - 0.9
iv. Minimalna długość wektora - $2pkt.$
v. Odchylenia w tworzeniu wektorów: 0.5 oraz 0.5
vi. Zmiana odchylenia - *nie ma znaczenia*
vii. Odchylenie przy łączeniu wektorów - 1
viii. Liczba iteracji w łączeniu wektorów - 2

b) Inkscape
i. Tryb - wykrywanie krawędzi
ii. Próg 1 - 0.45
iii. Próg 2 - 0.65
iv. Liczba kolorów - 2

2. **Obraz II**

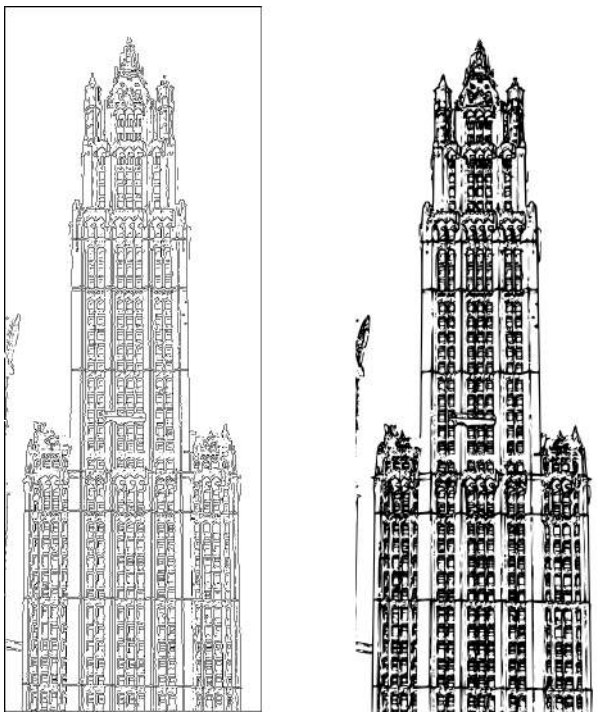
a) Ptah
i. Maska Gaussa nr 2
ii. Filtry Sobela - 0^o oraz 90^o
iii. Próg dolny - 0.3, próg górny - 0.65
iv. Minimalna długość wektora - $3pkt.$
v. Odchylenia w tworzeniu wektorów: 2.5 oraz 1.5
vi. Zmiana odchylenia - 10
vii. Odchylenie przy łączeniu wektorów - 1
viii. Liczba iteracji w łączeniu wektorów - 2

b) Inkscape
i. Tryb - wykrywanie krawędzi
ii. Próg 1 - 0.3
iii. Próg 2 - 0.65
iv. Liczba kolorów - 2

3. **Obraz III**

a) Ptah
   i. Maska Gaussa nr 1
   ii. Filtry Sobela - 0^o oraz 90^o
   iii. Próg dolny - 0.85, próg górny - 0.3
   iv. Minimalna długość wektora - $2pkt.$
   v. Odchylenia w tworzeniu wektorów: 3.0 oraz 1.5
   vi. Zmiana odchylenia - 4
   vii. Odchylenie przy łączeniu wektorów - 0
   viii. Liczba iteracji w łączeniu wektorów - 0
b) Inkscape
   i. Tryb - wykrywanie krawędzi
   ii. Próg 1 - 0.1
   iii. Próg 2 - 0.1
   iv. Liczba kolorów - 2

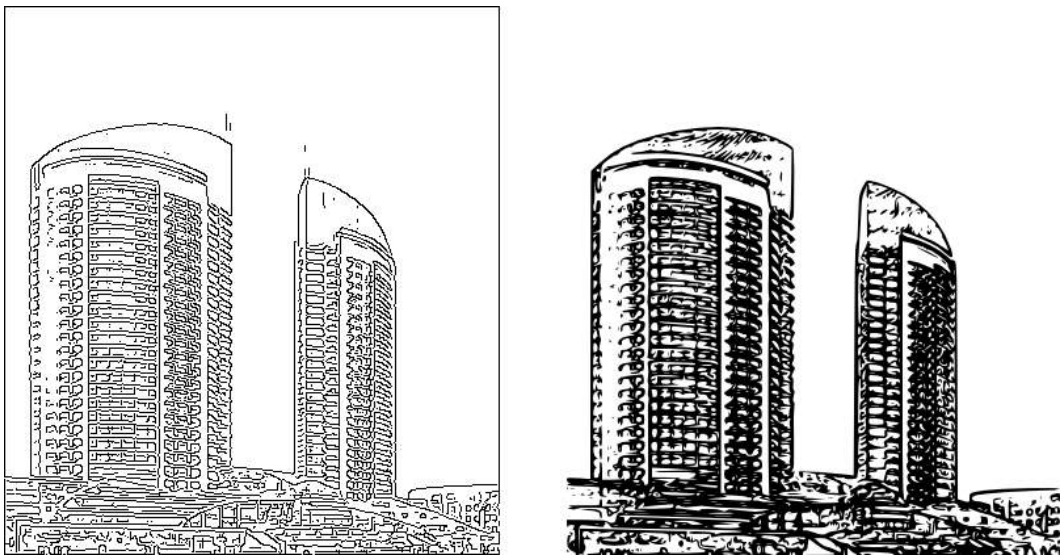
4.2.2. Wyniki badań jakości



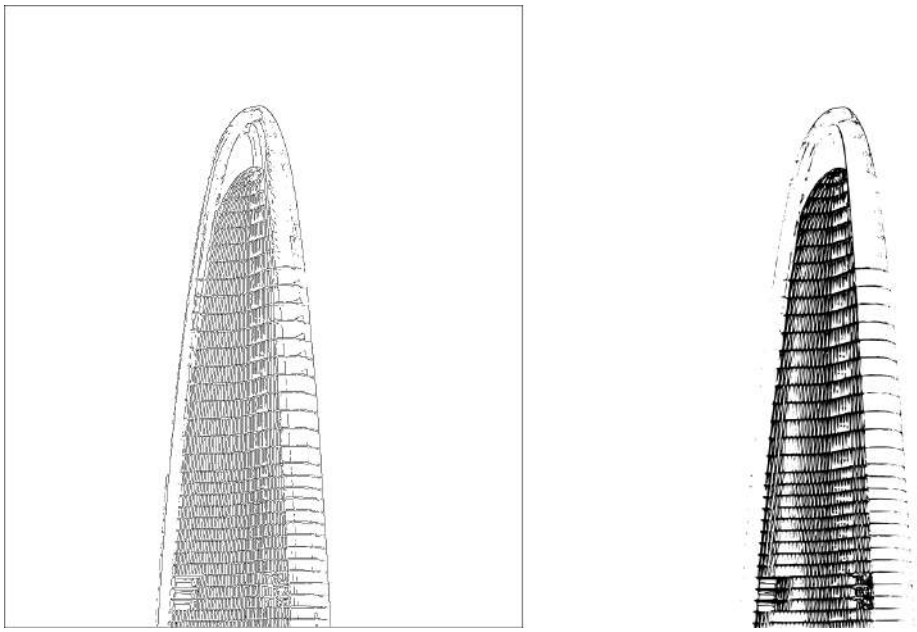
Rysunek 4.4. Wynik wektoryzacji obrazu I w programach (kolejno): Ptah oraz Inkscape

Efekty użycia algorytmów w obu programach przedstawiają rysunki: 4.4, 4.5 oraz 4.6.

Wynik wektoryzacji obrazu I wygląda „na pierwszy rzut oka" podobnie. Jednak po głębszej analizie widać dokładnie, że w przypadku wyniku z Inkscape'a krawędzie są mniej ostre i bardziej zaokrąglone i w większości grubsze niż jeden piksel, przez co obraz wygląda bardziej spójnie. Ponadto raczej nie widać przerywanych krawędzi. Z drugiej jednak strony zatraca się tu informacja o dokładnym kształcie



Rysunek 4.5. Wynik wektoryzacji obrazu II w programach (kolejno): Ptah oraz Inkscape



Rysunek 4.6. Wynik wektoryzacji obrazu III w programach (kolejno): Ptah oraz Inkscape

danych elementów. Wynik z programu Ptah ma więcej ostrych krawędzi, zachowuje dokładniejszą informację o konkretnych kształtach elementów takich jak na przykład okna czy dach. Niestety widać też w niektórych miejscach fałszywe linie, a także przerwania prawdziwych krawędzi. Można to jednak poprawić ręcznie.

Jeden i drugi algorytm niezbyt dobrze poradził sobie z balkonami tego budynku.

Bardzo podobnie jest w przypadku obrazu II. Tutaj jeszcze lepiej widać jak poprzez algorytm Inkscape'a zatraciła się informacja o kształtach elementów, a krawędzie połączyły się ze sobą.

W wynikowej transformacji obrazu III Inkscape nie zdołał wykryć zewnętrznej krawędzi budynku (mimo bardzo niskiego progu), dodatkowo grubość krawędzi okien zniekształca ich pierwotny kształt. Ptah natomiast bez problemu wykrył krawędzie boczne, a także zachował odpowiedni kształt elementów.

Reasumując, wyniki algorytmu Inkscape'a są bardziej spójne i używając kolokwializmu „przyjemniejsze dla oka". Natomiast wyniki mojego algorytmu zawierają więcej szumów i są mniej spójne, lepiej jednak zachowują kształty elementów i przy obrazach gorszej jakości - lepiej wykrywają krawędzie. Dodatkowo w programie Ptah użytkownik może „ręcznie" poprawić obraz w kolejnych krokach algorytmu i obserwować każdą poszczególną zmianę parametrów, co pozwala na ich lepsze dopasowanie.

W związku z tym w przypadkach, gdy użytkownikowi zależy na dokładniejszym odzwierciedleniu kształtów konkretnych elementów i na wyniku bardziej „technicznym" aniżeli „artystycznym", lepiej sprawdzał się będzie program Ptah. Oczywiście programy komercyjne mogą mieć potencjalnie lepsze metody wektoryzacji, niestety oprogramowanie tego typu jest dość kosztowne.

4.3. Badanie wydajności

Badanie to miało na celu sprawdzenie jaki jest całkowity czas trwania algorytmu, w zależności od jego wielkości i dobranych parametrów. Dodatkowo miało wykryć „wąskie gardła" algorytmu. Ze względu na to, że wartości parametrów takich jak próg dolny czy próg górny przy progowaniu z histerezą czy wybór kierunkowej maski Sobela nie mają realnego wpływu na ilość przeprowadzonych operacji, a w konsekwencji na czas trwania procesu, wziąłem pod uwagę zmianę wielkości maski Gaussa oraz minimalnej liczby pikseli do stworzenia wektora.

4.3.1. Plan badań wydajności

1. Implementacja funkcji mierzącej czas każdego z kroków algorytmu wektoryzacji.
2. Przygotowanie sześciu różnych obrazów przedstawiających obiekty architektoniczne.
3. Zmierzenie czasów dla każdego z obrazów przy masce Gaussa 3x3 oraz przy minimalnej wielkości wektora 5 (pikseli).
4. Zmierzenie czasów dla każdego z obrazów przy masce Gaussa 5x5 oraz przy minimalnej wielkości wektora 1 (piksel).
5. Zapisanie otrzymanych wartości w formie tabeli.
6. Analiza i interpretacja wyników.

4.3.2. Wyniki badań wydajności

Poniższe tabele przedstawiają czasy trwania poszczególnych operacji dla jednego obrazu (w różnych rozmiarach) przedstawiającego obiekty architektoniczne, o danych wymiarach:

— **Obraz 1** - 144480 pikseli *(336x430)*

— **Obraz 2** - 480000 pikseli *(800x600)*
— **Obraz 3** - 730800 pikseli *(525x1392)*
— **Obraz 4** - 1262520 pikseli *(1260x1002)*
— **Obraz 5** - 3871488 pikseli *(2272x1704)*
— **Obraz 6** - 9720000 pikseli *(3600x2700)*

Kolejne kolumny tabeli oznaczają numer obrazka i czas jaki zajęła dana czynność w milisekundach.

Procedura	**1**	**2**	**3**	**4**	**5**	**6**
Skala szarości	13	43	60	113	336	863
Filtrowanie Gaussa	37	128	186	334	1042	2806
Filtrowanie Sobela	75	256	373	691	2092	5572
Obliczanie mocy gradientu	15	48	67	127	445	1663
Obliczanie kierunku gradientu	26	85	112	235	801	2249
Tłumienie niemaksymalne	10	32	33	79	240	522
Progowanie z histerezą	19	65	96	169	520	1293
Wektoryzacja	343	896	1666	2808	5632	8243

Wyniki czasowe przy masce Gaussa 3x3 oraz minimalnej liczbie pikseli do zbudowania wektora równej 5

Procedura	**1**	**2**	**3**	**4**	**5**	**6**
Skala szarości	15	42	67	111	349	916
Filtrowanie Gaussa	82	272	424	734	2909	6356
Filtrowanie Sobela	74	251	413	664	2138	6019
Obliczanie mocy gradientu	14	47	70	117	451	1848
Obliczanie kierunku gradientu	25	82	113	223	720	2502
Tłumienie niemaksymalne	8	30	34	73	227	512
Progowanie z histerezą	18	64	99	175	521	1308
Wektoryzacja	392	1000	2346	3369	7348	9435

Wyniki czasowe przy masce Gaussa 5x5 oraz minimalnej liczbie pikseli do zbudowania wektora równej 1

Na początek porównajmy wyniki w tabelach dla tych samych obrazów ale z różnymi parametrami. Możemy zaobserwować, że zwiększenie filtra Gaussa wydłuża czas przetwarzania około dwukrotnie. Natomiast zmniejszenie progu liczby pikseli do budowy wektora wydłuża czas trwania średnio od około 15 do nawet 40 procent. Przy większych obrazach może to stanowczo wydłużyć czas przetwarzania, tak jak w wypadku obrazu piątego, gdzie zmiana parametrów wydłużyła czas o około 3,5 sekundy.

Reszta operacji odbywa się w porównywalnym czasie, praktycznie wręcz identycznym (różnice do parunastu milisekund).

Teraz przyjrzyjmy się wzrostem ogólnego czasu przetwarzania kolejnych obrazów. Przy największych badanych obrazach czas trwania przekracza już 30 sekund. Jako, że wzrost czasu przetwarzania jest dość duży w stosunku do wzrostu ilości pikseli, można zakładać, że obrazy o wielkościach ponad 10 000 000 pikseli mogą być przetwarzane nawet do minuty.

Przy przeprowadzaniu powyższych badań można było zaobserwować jeszcze jedno zjawisko - coraz większe zapotrzebowania pamięciowe. Ze względu na występowanie funkcji rekurencyjnych w obrazie ilość funkcji odkładanych na stos jest dość duża, szczególnie dla obrazów o większych rozmiarach. Dla przykładu: przy przetwarzaniu obrazu nr 6, program potrzebował ponad 1,5GB wolnej pamięci aby wykonać cały algorytm.

5. Podsumowanie

W ramach pracy został zaimplementowany algorytm wektoryzacji obrazów rastrowych przedstawiających obiekty architektoniczne. Na podstawie podanych przykładów widać, że algorytm działa zgodnie z założeniami. Należy jednak wziąć pod uwagę, że zadowalające wyniki są zależne w dużej mierze od parametrów w poszczególnych krokach. Odpowiednie dopasowanie tych zmiennych do konkretnego obrazu może pozytywnie wpłynąć na efekt końcowy.

Algorytm został również przetestowany i przebadany w praktyce pod względem jakościowym i wydajnościowym.

W zależności od kontekstu wektoryzacji algorytm może mieć porównywalne, a czasami nawet dużo lepsze wyniki w odniesieniu do dostępnych narzędzi Open Source typu Inkscape.

Przy wzroście liczby pikseli w obrazie dość szybko rosną wymagania pamięciowe oraz czas trwania całego procesu.

Podsumowując, algorytm zastosowany w programie Ptah jest dobrą alternatywą dla uniwersalnych algorytmów tego typu, głównie jeśli zależy nam na wektoryzacji konturów budynku i zachowaniu dokładnej informacji o kształcie jego poszczególnych elementów. Warto jednak jeszcze raz nadmienić, że wymaga odpowiedniego dostosowania parametrów do każdego indywidualnego przypadku i może wymagać ręcznej edycji przez użytkownika.

W ramach rozwoju algorytmu warto by pomyśleć nad dodaniem funkcji, która automatycznie (tzn. bez konieczności bezpośredniej ingerencji użytkownika) będzie usuwała elementy obrazu niebędące częściami obiektów architektonicznych. Poprawić można by także sam algorytm wektoryzacji, aby poza odcinkami rozpoznawał też łuki czy nawet konkretne figury geometryczne.

Z punktu widzenia interfejsu użytkownika w programie „Ptah” warto byłoby udostępnić możliwość zmiany położenia obiektów oraz ich parametrów po całym algorytmie wektoryzacji.

Bibliografia

[1] John F. Canny, *A Computional Approach to Edge Detection*, http://perso.limsi.fr/vezien/PAPIERS_ACS/canny1986.pdf, 1986.

[2] Witold Malina, Maciej Smiatacz, *Cyfrowe przetwarzanie obrazów*, Akademicka Oficyna Wydawnicza EXIT, Warszawa, 2008.

[3] Rafał Ludwiczuk, *Algorytm Canny'ego detekcji krawędzi w procesie segmentacji obrazów medycznych*, Uniwersytet Marii Curie-Skłodowskiej w Lublinie, http://kis.pwszchelm.pl/publikacje/II/Ludwiczuk.pdf.

[4] Cezary Bołdak, *Cyfrowe przetwarzanie obrazów*, 2008, http://aragorn.pb.bialystok.pl/ boldak/DIP/CPO-W04-v01-50pr.pdf.

[5] Jacek Wiślicki, *Wprowadzenie do grafiki wektorowej*, http://jacenty.iis.p.lodz.pl/draw/wykl_01.pdf.

[6] Piotr Krzysztof Jastrzębski, *Wektoryzacja prostych schematów układów cyfrowych*, http://repo.pw.edu.pl/docstore/download.seam?fileId=WUT307467.

[7] Marek Zachara, *Szybka metoda wektoryzacji krawędzi odcinkami w czasie rzeczywistym*, http://journals.bg.agh.edu.pl/AUTOMATYKA/2006-03/Auto_427-434.pdf.

[8] R.Klette,P.Zamperoni, *Handbook of Image Processing Operators*, Wydawnictwo Wiley, Chichester, England 1996, 332-340.

[9] Marek Zachara, *Real time object tracking and recognition*, MSc Thesis, University of Bristol, England, 1998.

[10] Mariusz Nieniewski, *Segmentacja obrazów cyfrowych. Metody segmentacji wododziałowej*, Akademicka Oficyna Wydawnicza EXIT, Warszawa, 2005.

[11] Włodzimierz Kasprzak, *Rozpoznawanie obrazów i sygnałów mowy*, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 2009.

[12] Jerzy Cytowski, Jerzy Gielecki, Artur Gola, *Cyfrowe przetwarzanie obrazów medycznych. Algorytmy. Technologie. Zastosowania*, Akademicka Oficyna Wydawnicza EXIT, Warszawa, 2008.

[13] A.K. Chhabra, I.T. Phillips, *Performance Evaluation of Line Drawing Recognition Systems*, Proc. 15th Int'l Conf. Pattern Recognition, vol. 4, pp. 4864-4869, 2000.

[14] I.T. Phillips, A.K. Chhabra *Empirical Performance Evaluation of Graphics Recognition Systems*, IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 21, no. 9, pp. 849-870, 1999.

[15] Xavier Hilaire, Karl Tombre *Robust and Accurate Vectorization of Line Drawings*, IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 28, no. 6, 2006.

[16] D. Antoine, S. Collin, K. Tombre *Analysis of Technical Documents: The REDRAW System*, pp. 385-402, Springer Verlag, 1992.

[17] R.W. Smith *Computer Processing of Line Images: A Survey*, Pattern Recognition, vol. 20, no. 1, pp. 7-15, 1987.

[18] K. Tombre, C. Ah-Soon, P. Dosch, G. Masini, S. Tabbone *Stable and Robust Vectorization: How to Make the Right Choices*, Graphics Recognition—Recent Advances, 2000.

[19] E. Bodansky, M. Pilouk *Using Local Deviations of Vectorization to Enhance the Performance of Raster-to-Vector Conversion Systems*, Int'l J. Document Analysis and Recognition, vol. 3, no. 2, pp. 67-72, 2000.

[20] R.D.T. Janssen, A.M. Vossepoel *Adaptive Vectorization of Line Drawing Images*, Computer Vision and Image Understanding, vol. 65, no. 1, pp. 38-56, 1997.

[21] R. Kasturi, S.T. Bow, W. El-Masri, J. Shah, J.R. Gattiker, U.B.Mokate *A System for Interpretation of Line Drawings*, IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 12, no. 10, pp. 978-992, 1990.

[22] V. Nagasamy, N.A. Langrana *Engineering Drawing Processing and Vectorization System*, Computer Vision, Graphics and Image Processing, vol. 49, no. 3, pp. 379-397, 1990.

[23] V. Poulain d'Andecy, J. Camillerapp, I. Leplumey *Kalman Filtering for Segment Detection: Application to Music Scores Analysis*, Proc. 12th Int'l Conf. Pattern Recognition, vol. 1, pp. 301-305, 1994.

[24] H. Blum, R. Nagel *Shape Description Using Weighted Symmetric Axis Features*, Pattern Recognition, vol. 10, pp. 167-180, 1978.

[25] Y. Zheng, H. Li, D. Doermann *A Parallel-Line Detection Algorithm Based on HMM Decoding*, IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 27, no. 5, pp. 777-792, 2005.

[26] J. Song, M.R. Lyu *A Hough Transform Based Line Recognition Method Utilizing Both Parameter Space and Image Space*, Pattern Recognition, vol. 38, no. 4, pp. 539-552, 2005.