Mariusz KAPRUZIAK
ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY, WYDZIAŁ INFORMATYKI,
ul. Żołnierska 52, 71-210 Szczecin

Zmodyfikowana zrandomizowana transformata Hougha w strukturze FPGA

Dr inż. Mariusz KAPRUZIAK

Obronił prace doktorska na Politechnice Szczecińskiej w 2006r. na temat "Opracowanie koncepcji i realizacja dedykowanego procesora radia programowalnego". Obecnie prowadzi badania dotyczące systemów rekonfigurowanych oraz widzenia maszynowego na Politechnice Szczecińskiej.

e-mail: mkapruziak@wi.ps.pl

Streszczenie

Wykonanie pełnej transformaty Hougha wymaga dużej mocy obliczeniowej. Moc obliczeniową można zredukować wybierając losowo tylko niektóre próbki do przetwarzania. Algorytm taki nosi nazwę RHT. Klasycznie, ze względu na intensywne wykorzystanie dynamicznych struktur danych oraz dużą liczbę warunkowo wykonywanych funkcji, zadanie RHT nie implementuje się dobrze w strukturze FPGA. Autor proponuje modyfikację tego algorytmu, tak aby efektywnie można było wykonać taka implementacje. W artykule przedstawiono propozycję struktury procesora dedykowanej dla struktury FPGA i realizującego algorytm RHT.

Słowa kluczowe: RHT, FPGA.

Modified randomized Hough Transform in FPGA

Abstract

Original Hough Transform requires much computational power to calculate parameter space and select proper maxima in that space. Computational power can be reduced by randomly selecting only a subset of points for processing. An algorithm constructed that way is called the RHT (Randomized Hough Transform). Originally, because of intensive use of dynamic memory structures and high number of conditionally executed functions, the RHT algorithm does not fit well in FPGA. The author tries to modify that algorithm in order to be able to implement it efficiently in static FPGA structures. It is achieved by means of rejecting the parameter space, using a line list instead and selecting lines from that list to be paired, extended or checked. Pairing is a procedure of connecting two lines and is a vital part at the initial stage. The extending procedure tries to search if there are more points at left or right side of a line which might extend that line. The checking procedure confirms existence of a line and potentially eliminates it from further processing (when long enough or too short). Having linearly addressed the line list with random selection of lines inside allows not to use dynamic memory structures and improves its FPGA implementation significantly. The FPGA implemented structure of a processor for the proposed algorithm is presented in the paper.

Keywords: RHT, FPGA.

1. Wstęp

Transformata Hougha (HT) jest uznanym algorytmem znajdowania i określania parametrów podstawowych figur w obrazie, jak linii czy okręgów. Charakteryzuje się optymalnymi lub suboptymalnymi rezultatami, biorąc pod uwagę precyzje wyników [1] oraz jest w dużym stopniu odporna na przysłanianie fragmentów szukanego obiektu czy zaszumienie obrazu. Do zdecydowanych wad algorytmu należy natomiast wymaganie dużej ilości pamięci do reprezentacji pośrednich wyników obliczeń (w formie przestrzeni parametrów) oraz duża złożoność obliczeniowa algorytmu.

Do dnia dzisiejszego zaproponowano wiele modyfikacji redukujących koszt obu wspomnianych wad [2, 3, 4]. Autor artykułu koncentruje się na jednej z nich, zrandomizowanej transformacie Hougha (RHT,[2]). Pomysł polega na uwzględnianiu w transformacie tylko niektórych, losowo dobranych, próbek, tak aby relatywnie szybko uzyskać przybliżony obraz rzeczywistej szukanej przestrzeni parametrów [2]. Parametry szukanych figur można odczytać z tak utworzonej przestrzeni, nie popełniając dużego błędu w stosunku do wykonania pełnej transformaty Hougha [5].

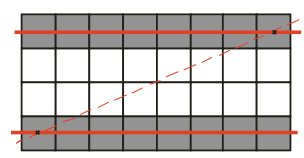
Autor w pracy koncentruje się na pewnej odmianie transformaty RHT. W 2009 roku na łamach konferencji RUC autor opublikował artykuł o modyfikacja transformaty Hougha, tak aby zaimplementować ją efektywnie w strukturze FPGA [6]. Oryginalna transformata RHT jest skuteczna dla architektury von Neumanna, gdzie bardzo łatwo implementuje się dynamiczne struktury pamięci, jak listy dynamiczne czy drzewa. Dodatkowo łatwe jest warunkowe wykonanie dużych fragmentów algorytmu. Dla statycznych struktur FPGA implementacja taka nie jest oczywista. Autor w artykule [6] zasugerował, że skuteczna mogłaby być taka modyfikacja, która ma wspólną pamięć punktów aktywnych i linii oraz dla każdej linii uwzględnia jej długość. Umożliwiło to sprowadzenie algorytmu do losowego wybierania adresów z jednej wspólnej liniowo adresowalnej struktura pamięciowej i zestawianiu tak wylosowanych linii w procedurze łączenia.

Wyżej wspomniane rozwiązanie jest skuteczne głownie dla obrazów czystych z niewielką liczbą obiektów. Dla obrazów mocno zaszumionych i o dużej liczbie detali skuteczniejsza okazywała się metoda zaproponowana przez zespół dr Linpeng [5]. Polega ona na wykonaniu oryginalnej RHT, jednak kończąc jej wykonanie na bardzo wczesnym etapie i sprawdzanie dalej obiektów poprzez zwykłe liczenie punktów na obrazie należących do figur wskazywanych w przestrzeni parametrów. W ramach głębszych badań nad tym problemem autor zaproponował dalszą modyfikację algorytmu, będącą rodzajem kompilacji pomysłu [6] z propozycją wczesnego kończenia przetwarzania punktu w przestrzeni parametrów, charakterystyczną dla [5].

Modyfikacja proponowana przez autora ma głównie taką zaletę, że nie wymaga implementacji dynamicznych struktur pamięciowych, nie ustępując przy tym efektywnością rozwiązaniom konwencjonalnym. Dzięki temu znacznie prostsza jest implementacja algorytmu w statycznych strukturach, jak FPGA. Propozycja takiego procesora, dedykowanego dla zmodyfikowanego algorytmu RHT, zaimplementowanego w FPGA i napisanego w języku Verilog przedstawiona jest w artykule.

2. Algorytm

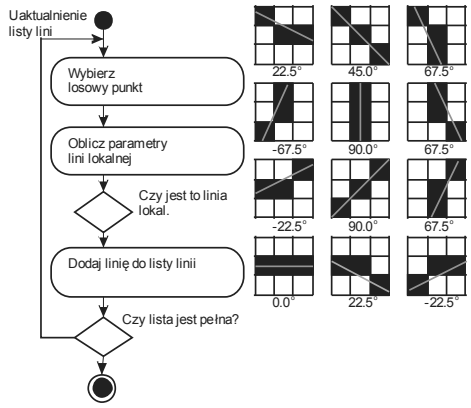
Algorytm rozpoczyna się od poszukiwania krótkich lokalnych linii. W tym celu losowo wybierane są punkty z obrazu i porównywane z zestawem 12 masek o wielkości 3x3 punkty (rys. 2). Gdy odpowiednia linia zostaje znaleziona, dodawana jest ona do listy linii. Z tej listy wybierane są linię do zestawiania ze sobą. Zestawianie polega na sprawdzeniu, czy dwie linie łączą się razem, tworząc jedną dłuższą linię. Jeśli tak, to obie są kasowane i wstawiana jest w ich miejsce linia dłuższa.



Rys. 1. Przykład iluzorycznej linii stworzonej w wyniku zestawiania punktów
Fig. 1. Example of a virtual line resulting from line pairing

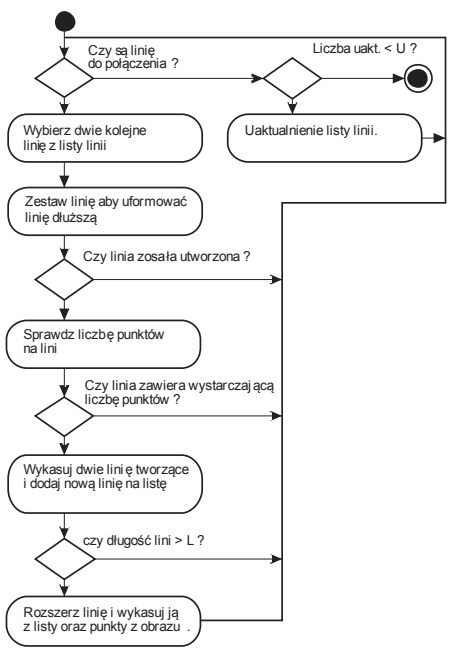
Przed połączeniem linii w wyniku zestawienia sprawdzane jest czy wzdłuż fragmentu linii łączącego obie istniejące linie występuje wystarczająca liczba punktów. Mechanizm został dodany

w celu redukcji znaczenia problemu łączenia linii nie łączących się w rzeczywistym obrazie (rys. 1). Linie krótkie na wejściu mają zazwyczaj dość dużą tolerancję nachylenia i niewłaściwe połączenie mogłoby relatywnie łatwo być wynikiem niewłaściwego połączenia dwóch linii znajdujących się obok siebie. Opisany mechanizm nazwany został sprawdzeniem (rys. 3).



Rys. 2. Algorytm uaktualniania listy linii
Fig. 2. The line update list algorithm

Zaimplementowano także mechanizm rozszerzania. Jest to mechanizm podobny do poszukiwania nowych linii, inaczej wybiera się natomiast punkty z obrazu do tworzenia linii. Punkt wybierany jest z przestrzeni możliwych punktów rozszerzających dana linię, na przykład ze strony lewej i prawej (rys. 4). Znajdujący się tam punkt zestawiany jest z linią w celu rozszerzenia tej pierwszej.

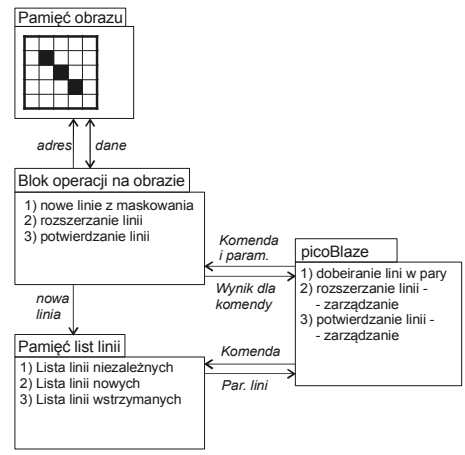


Rys. 3. Algoritym zmodyfikowanego RHT
Fig. 3. Modyfied RHT algorithm

Rys. 4. Przykład przestrzeni oraz możliwych linii przy rozszerzaniu
Fig. 4. Example of extension space and possible lines resulting from extension

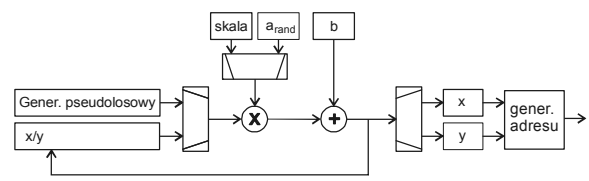
3. Implementacja algorytmu w FPGA

Algorytm zaimplementowano w strukturze FPGA. Struktura złożona jest z czterech głównych modułów: pamięci obrazu, bloku operacji na obrazie, modułu pamięci listy linii oraz modułu picoBlaze (rys. 5). Pamięć obrazu reprezentuje obraz po filtracji krawędziowej i progowaniu (w postaci 1-bitowych pikseli). Z takiej pamięci na raz wybierane jest 9 pikseli otaczających piksel o adresie podanym na szynie *adres*. Trafiają one do bloku operacji na obrazie, gdzie wykonywane są operacje podstawowe: znajdowanie linii lokalnych, sprawdzanie oraz rozszerzanie. Nowe linie lokalne przesyłane są do pamięci list linii. Z tej pamięci wybierane sa linie do zestawiania. Zestawianie odbywa się w sposób proceduralny na procesorze picoBlaze. Procesor także przesyła do bloku operacji na obrazie ewentualne rozkazy rozszerzenia lub potwierdzenia linii.



Rys. 5. Ogólna struktura organizacji procesora
Fig. 5. General structure of processor organization

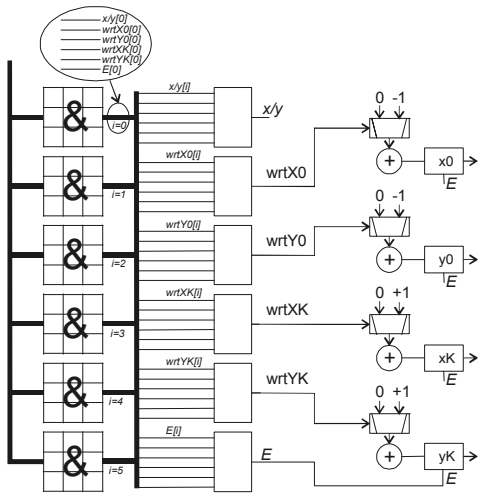
Krytyczny dla efektywności rozwiązania jest moduł bloku operacji na obrazie. Posiada on jeden generator pseudolosowy oraz układ mnożarki z sumatorem (rys. 6). Pozwala to na wybranie liczby losowej z zadanego przedziału jak również obliczenie wartości linii w danym punkcie. W dwóch cyklach uzyskuje się współrzędne x i y danego piksela. Trzeci cykl wymagany jest do określenia adresu do pamięci na podstawie tych współrzędnych.



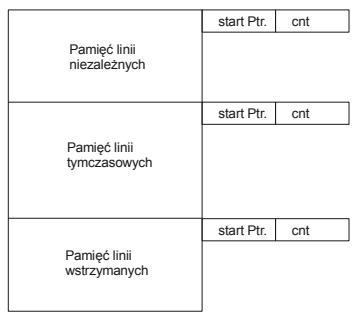
Rys. 6. Organizacja części wybierania współrzędnych i generowania adresu w bloku operacji na obrazie
Fig. 6. Organization of coefficient selection and address generation in the image processing block

Układ znajdowania linii lokalnych wymaga także 3 cykli (rys. 7). W pierwszym cyklu wykonane jest 6 operacji maskowania, pozostałe 6 w cyklu drugim. Cykl trzeci dotyczy określenia współrzędnych punktu początkowego (x_0, y_0) oraz końcowego (x_K, y_K) nowej linii.

Moduł pamięci linii zawiera pamięć dla linii w trzech możliwych stanach w systemie (rys. 8). Pamięć linii niezależnych to pamięć linii w ramach której wszystkie linie nie zestawiają się między sobą. Pamięć linii tymczasowych przechowuje linie, które należy zestawić z liniami niezależnymi i ze sobą. Jeśli linia nie zestawia się z żadną inną zostaje na jakiś czas wstrzymana. Jest na ten czas umieszczona w pamięci linii wstrzymanych.



Rys. 7. Organizacja części maskowania w bloku operacji na obrazie
Fig. 7. Organization of masking part in the image processing block

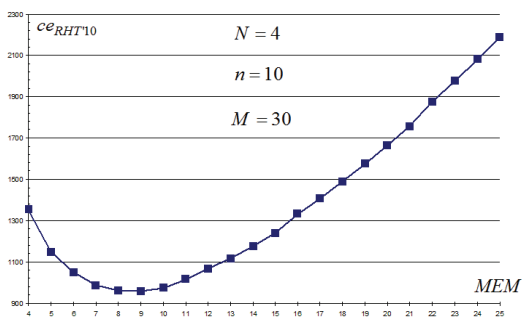


Rys. 8. Struktura organizacji pamięci linii
Fig. 8. Organization of the line list memory

Procedura zestawiania polega na *{1}* wybraniu najbardziej oddalonych od siebie punktów, z czterech tworzących dwie linie, *{2}* utworzeniu nowej linii z tych punktów oraz *{3}* sprawdzeniu czy punkty środkowe należą do nowo utworzonej linii. Procedura ta nie zrównolegla się dobrze i zdecydowano się na jej implementacje proceduralną na procesorze programowalnym picoBlaze.

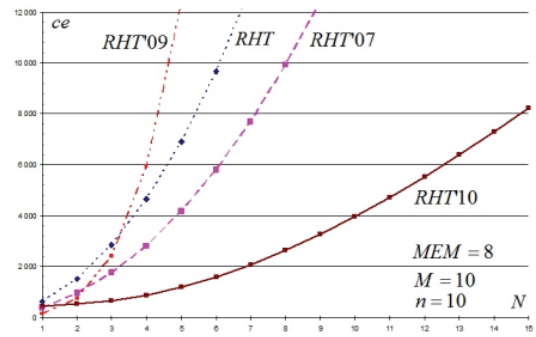
4. Wyniki testów i dyskusja

Część uzyskanych wyników badań dało, zdaniem autora, intrygujące wyniki. W szczególności za interesujące autor uznał dwie zależności. Pierwsza przedstawia zależność pomiędzy wielkością pamięci a uzyskaną efektywnością obliczeniową (rys. 9). Wynik wydaje się wskazywać na istnienie optymalnej wielkości pamięci listy linii i dalsze powiększanie tej pamięci może skutkować pogarszaniem parametrów rozwiązania. Wielkość optymalna ma wartość umiarkowaną, mieszczącą całą strukturę pamięciową w środku struktury FPGA.



Rys. 9. Złożoność obliczeniowa jako funkcja wielkości pamięci listy
Fig. 9. Computational effort as a function of algorithm memory size

Druga zależność pokazuje efektywność obliczeniową różnych algorytmów RHT. Badanie przeprowadzono dla sztucznie stworzonej sceny posiadającej zmienną liczbę N linii o długości n pikseli oraz M punktów szumu. Algorytm prezentowany w pracy okazuje się mieć lepsze parametry (mniejszą złożoność obliczeniową) dla dużej liczby niezbyt długich linii w obrazie (rys. 10). Dla jednej albo niewielkiej liczby linii lepsze wyniki daje algorytm [5].



Rys. 10. Złożoność obliczeniowa jako funkcja liczby linii na obrazie
Fig. 10. Computational effort for different algorithms as a function of number of lines

5. Dyskusja i podsumowanie

Główną zaletą proponowanego rozwiązania, z punktu widzenia implementacji w FPGA, jest statyczna organizacja pamięci dla algorytmu. Zrezygnowano z reprezentacji wprost przestrzeni parametrów i zastąpiono ją liniowo adresowalną listą linii. Takie rozwiązanie sprawuje się bardzo dobrze dla niewielkiej liczby długich linii. O ile nie zostanie wsparte dodatkowymi mechanizmami efektywność bardzo pogarsza się jednak przy wzroście liczby linii lub szumu. Badania tego efektu skłoniły autora do wprowadzenia trzech dodatkowych mechanizmów: sprawdzania, rozszerzania i wprowadzania krótkich linii lokalnych zamiast punktów na wejściu algorytmu. Takie rozwiązanie implementuje się efektywnie w FPGA oraz daje również lepsze wyniki dla większości typowych przypadków liczby linii i szumu w obrazie. Klasyczna podmiana przestrzeni parametrów na listę linii musi być kompensowana dużym stopniem równoległości uzyskiwanym przy implementacji w FPGA. Proponowane rozwiązanie nie wymaga takiego zabiegu.

6. Literatura

[1] Davis E.R.: Machine Vision, Theory, Algorithms and Practicalities. Morgan Kaufmann, Elsevier 2005.
[2] Xu L., Oja E., Kultanen P.: A New curie detection metod: Randomized Hough Transform (RHT), Pattern Recognition Letters 11, 1990, pp 331-338.
[3] Illingworth H., Kittler J.: The adaptive Hough transform, IEEE Transactions on Pattern Analysis and Machine Intelligence 1987.
[4] Delalandre M., Simon B., Guillas S., Ogier J.M., Bertet K.: Stright Line Detection based on the Hough Transform a System and its Performance Evaluation, Draft, 2006.
[5] Linpeng Ch., Guoliang Z., Guangming J., Qi T.: A New Algorithm for Line Detection Based on the Randomized Hough Transform. The Eighth International Conference on Electronic Measurement and Instruments, ICEMI'2007.
[6] Kapruziak, M.: Randomized Hough Transform in FPGA. Pomiary Automatyka Kontrola 2009, R. 55, nr 8, pp. 624-626.

otrzymano / received: 13.05.2011
przyjęto do druku / accepted: 04.07.2011 *artykuł recenzowany*