AKADEMIA GÓRNICZO-HUTNICZA
im. St. Staszica w Krakowie
WEAIiE, Katedra Automatyki
Laboratorium Biocybernetyki

Przedmiot: **Systemy rekonfigurowalne.**

Sygnatura projektu: Pr12sr512

Temat projektu:

Równoległa interpolacja obrazu barwnego z kamery cyfrowej

Wykonali: Bartłomiej Bułat
Tomasz Drzewiecki

Rok i kierunek: Informatyka Stosowana rok I semestr zimowy

Opiekun Projektu: dr inż. Mirosław Jabłoński

V 1.1

Kraków, styczeń, 2013

1. Abstrakt

Celem projektu było napisanie biblioteki do interpolacji obrazu barwnego na podstawie zapisu z kamery z matrycą Bayera. Matryca ta charakteryzuje się specjalnym ułożeniem pikseli, które ma za zadanie odwzorowywać ludzką percepcje obrazu, gdyż znajduje się na niej ponad dwukrotnie więcej receptorów koloru zielonego od pozostałych barw. Biblioteka ma wykorzystywać wspomaganie obliczeń na GPU z wykorzystaniem technologii OpenCL. Interpolacja polega na wyliczeniu średnich wartości odpowiednich pikseli z trzech różnych masek z uwzględnieniem wzmocnienia kolorów. Aplikacja oparta o stworzoną bibliotekę ma w czasie rzeczywistym interpolować obraz z kamery HD. Wynikiem badań jest działająca biblioteka do pojedynczej i sekwencyjnej interpolacji obrazów, jak również aplikacja do interpolacji obrazów z kamery wysokiej rozdzielczości JAI BB-500GE.

Udało się pokazać znaczne przyśpieszenie algorytmu zaimplementowanego na karcie graficznej w porównaniu do referencyjnego algorytmu wykonywanego na CPU.

Słowa kluczowe: interpolacja barwna, matryca Bayera, OpenCL, GPU, przemysłowa kamera wysokiej rozdzielczości, JAI BB-500GE.

2. Wstęp

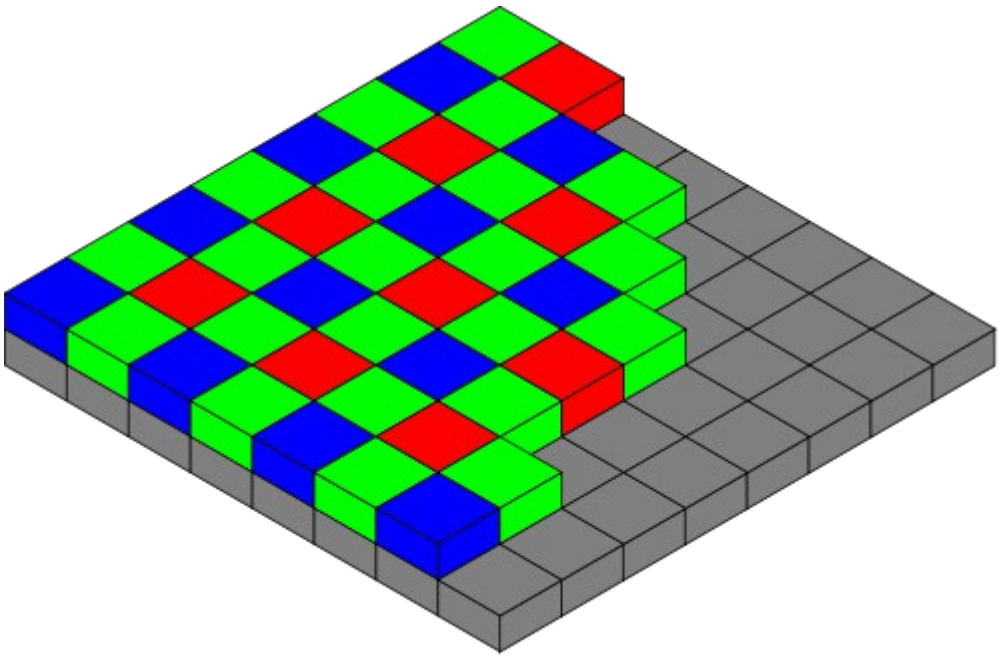
Badanym problemem była równoległa interpolacja obrazy barwnego otrzymanego z czujnika wizyjnego wyposażonego w matrycę CFA. Pracę nad problemem można podzielić na 3 etapy: opracownie algorytmu, implementacja algorytmu w OpenCL[1] oraz akwizycja obrazu z kamery HD. Obraz z kamery należało podać na wejście algorytmu, aby otrzymać poprawnie interpolowany obraz barwny.

2.1. Barwny czujnik wizyjny

Większość kolorowych kamer posiada matryce czujnika zorganizowane w postaci mozaiki kolorowych filtrów (CFA, Color Filter Array). Jedna z najbardziej powszechnych układów filtrów jest topologia matrycy Bayer-a, w której filtry koloru czerwonego, niebieskiego i zielonego ułożone są naprzemiennie, z przewagą filtrów zielonego. Taki układ filtrów pozwala odwzorować sposób

[1] Biblioteka programistyczna pozwalająca wykorzystać zasoby karty graficznej lub wielordzeniowego procesowa do zrównoleglania obliczeń. Więcej na https://www.khronos.org/opencl/

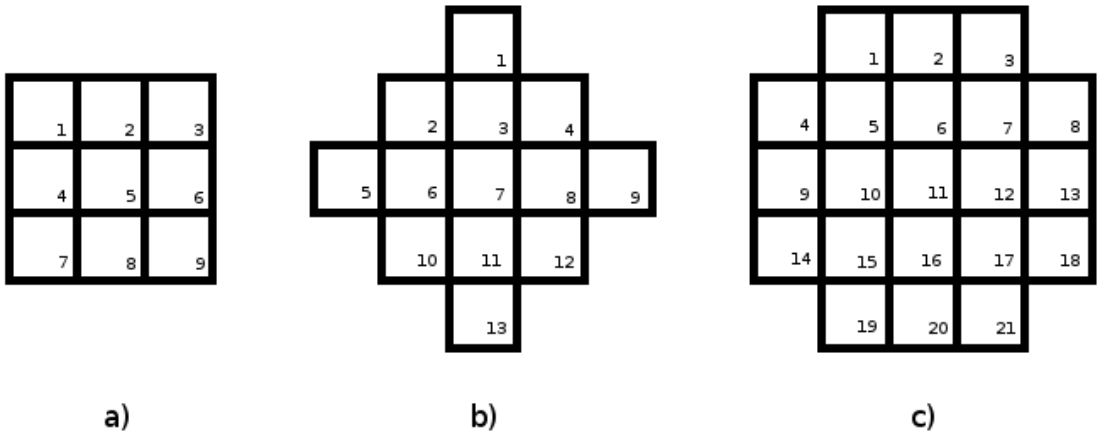
postrzegania kolorów przez ludzkie oko, które jest bardziej wrażliwe na światło widzialne w paśmie koloru zielonego. Układ filtrów na matrycy Bayer-a przedstawiono na rysunku 1:



Rysunek 1: Układ filtrów na matrycy Bayer-a (źródło Wikipedia.org)

Podstawowym formatem zapisu obrazu kolorowego jest format RGB, gdzie każdy piksel zapisany jest jako trzy wartości intensywności koloru czerwonego, zielonego i niebieskiego. Aby przedstawić obraz z kamery wyposażonej w filtr mozaikowy należy dla każdego piksela zastosować interpolację każdej składowej na podstawie wartości z otoczenia.

2.2 Interpolacja pikseli kolorowych

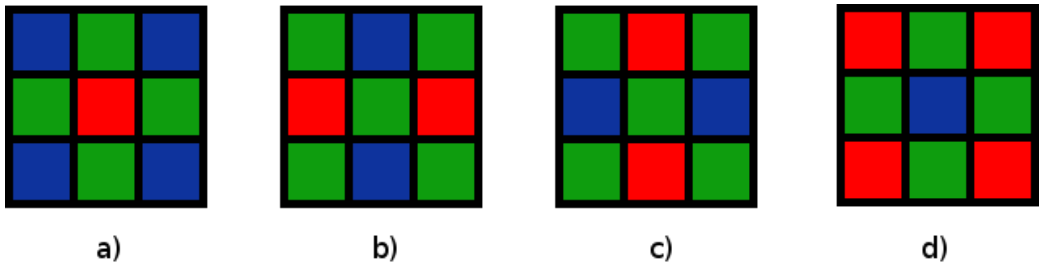


Rysunek 2: Konteksty użyte do interpolacji barwnej; a) maska prosta 3x3; b) maska okrągła o promieniu 5; c) maska "krzyż"

Interpolacja musi wykorzystywać informacje zawarte w sąsiednich pikselach. Według [1] maska interpolacji o rozmiarze 3 na 3 piksele daje wystarczające dobre rezultaty wizualne. W wymienionej

publikacji rozmiar maski był podyktowany wymaganiem budowy linii opóźniającej, w naszym przypadku nie ma to znaczenia, dlatego możemy zastosować dowolny kształt maski (ograniczony jednak kształtem piksela). Rysunek 2 przedstawia wykorzystane kształty masek.

Przy wykorzystaniu maski 2a), gdy pikselem środkowym (oznaczony numerem 5) jest czerwony lub niebieski, jego wartość jest brana tylko i wyłącznie z tego piksela. Przy użyciu maski 2b) dodatkowo brane są pod uwagę 4 piksele o barwie takiej samej jak piksel środkowy (tutaj, numer 7). Maska 2c) w porównaniu do maski drugiej używa większej liczby pikseli koloru czerwonego i niebieskiego przy interpolacji, gdy pikselem środkowym (w tym przypadku 11) jest punkt koloru zielonego.

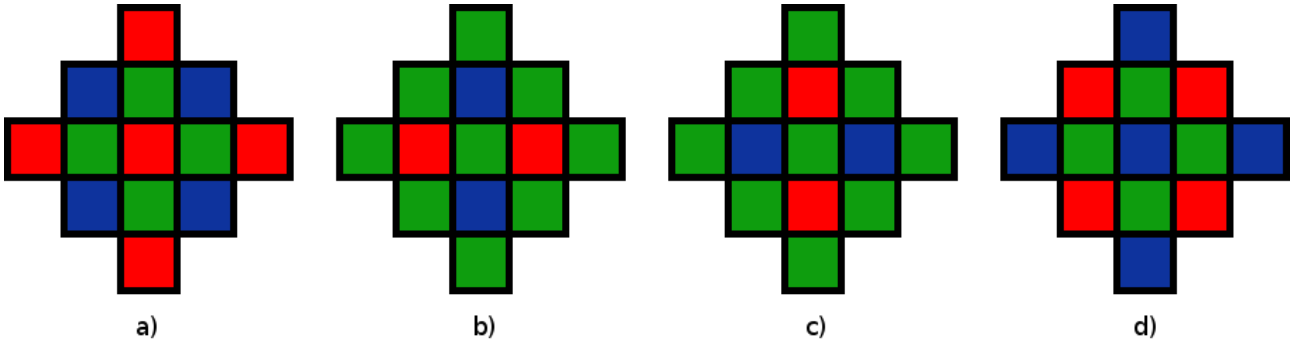


Rysunek 3: Możliwe konfiguracje pikseli dla maski 3x3.

Na rysunku 3 pokazano możliwe konfiguracje pikseli w kontekście maski 3x3. Wartość koloru w każdym przypadku liczona jest jako średnia arytmetyczna wartości danego koloru należącego do maski, na przykład dla konfiguracji 3a, z uwzględnieniem numeracji z rysunku 2:

$$\begin{cases} P_R = P_5 \\ P_G = \dfrac{P_2 + P_4 + P_6 + P_8}{4} \\ P_B = \dfrac{P_1 + P_3 + P_7 + P_9}{4} \end{cases}$$

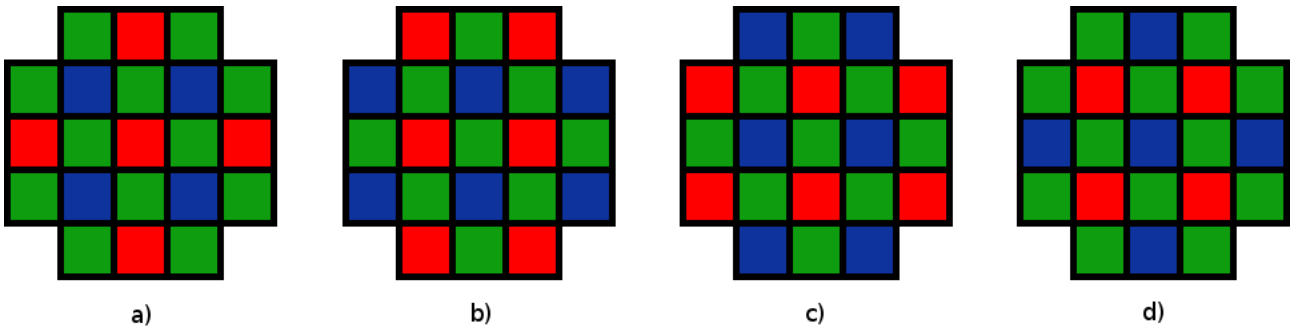
Analogicznie wyglądają wzory dla pozostałych 3 konfiguracji tego kontekstu.



Rysunek 4: Możliwe konfiguracje pikseli dla okrągłej maski.

Na rysunku 4 przedstawiono możliwe konfiguracje pikseli na matrycy w kontekście, dla wygody nazwany „okrągłym”. Jak zauważono wcześniej w przypadku 4a i 4d do wyliczenia wartości koloru czerwonego/niebieskiego używanych jest ich więcej wartości z sąsiedztwa. Analogicznie do maski

3x3 wartość każdego koloru liczona jest jako średnia arytmetyczna wartości odpowiednich kolorów pikseli należących do kontekstu.



Rysunek 5: Możliwe konfigurację pikseli w kontekście maski "krzyż".

Powyższy rysunek przedstawia możliwe konfiguracje pikseli na matrycy w kontekście maski, nazwanej dla ułatwienia przyszłych odwołań, maską „krzyż". W analogi do powyższych przykładów, wartość piksela jest liczona jako średnia arytmetyczna wartości odpowiednich kolorów należących do tego konteksu.

W rozdziale 5 zostanie pokazane porównanie obrazów wynikowych po interpolacji z wykorzystaniem różnych masek.

2.3 Zrównoleglanie algorytmów w OpenCL

Algorytmy splotowe (takie jak ten) na obrazach bardzo dobrze nadają się do zrównoleglania. Od kilku lat producenci sprzętu komputerowego udostępnią możliwość wykonywania obliczeń równoległych na kartach graficznych. Istnieją dwa wiodące rozwiązania: CUDA, pozwalające wykonywać obliczenia z wykorzystaniem procesora graficznego (GPU) nVidia oraz OpenCL, otwarty standard programowania współbieżnego na GPU jak i na wielordzeniowych CPU. Za wyborem drugiego rozwiązania przemawia jego dostępność i wieloplatformowość.

Programowanie w OpenCL polega na przygotowaniu jednostki algorytmu (kernela) i uruchomieniu go na wcześniej załadowanych danych w środowisku uruchomieniowym dostarczonym przez producenta sprzętu.

Ta biblioteka posiada bardzo dobrą dokumentację [2], która w nieoceniony sposób przydała się przy implementacji algorytmu. Pomogła dobrze zrozumieć ideę programowania równoległego realizowanego na karcie graficznej.

2.4 Pobieranie obrazu z kamery HD JAI BB-500GE

Kamera udostępnia interfejs poprzez kabel typu Ethernet. Został on użyty podczas implementacji. Dzięki zastosowaniu dołączonej do kamery biblioteki ten punkt był łatwy do zrealizowania. Głównym celem podczas implementacji pobierania obrazu z kamery było nie dopuszczenie

do sytuacji, by ta część blokowała wykonanie innych. Zadanie to okazało się proste, ponieważ architektura biblioteki dołączonej do kamery opierała się na rozwiązaniu wielowątkowym, nieblokującym wykonania innych części programu. Wykorzystywana jest kolejka, do której zapisywane są obrazy przychodzące z kamery. Z tej samej kolejki w dowolnym momencie użytkownik może pobrać obraz.
Do realizacji zadania użyto biblioteki dołączonej do kamery [4].

3. Koncepcja proponowanego rozwiązania

Biblioteka została podzielona na moduły: moduł OpenCL oraz moduł obsługi kamery. Oba moduły są od siebie niezależne, kompilują się do plików DLL i mogą zostać użyte w dowolnych innych projektach.

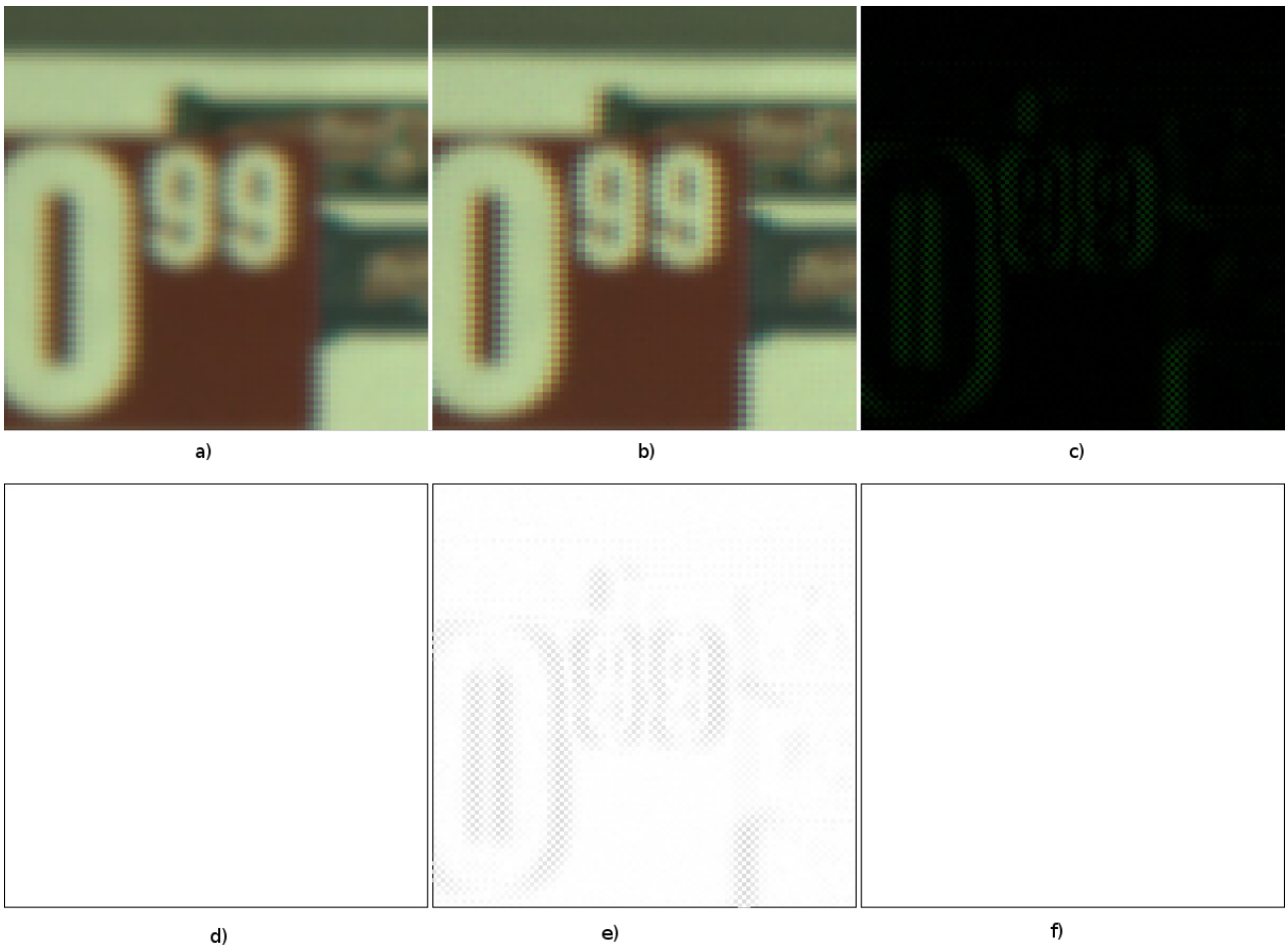
Moduł algorytmów OpenCL

Moduł ten oparty jest o abstrakcyjną klasę algorytmu wykonywanego w OpenCL. Na bazie tej klasy zaimplementowane zostały algorytmy konwersji typów danych ze całkowitoliczbowych na zmiennoprzecinkowe oraz algorytm interpolacji matrycy Bayera z wykorzystaniem struktury płaskiej (jednowymiarowej tablicy wartości) oraz struktury obrazu 2D, którą udostępnia OpenCL. Struktura ta pozwala na automatyczną interpolację piksela poza obszarem obrazu (lustrzane odbicie, piksel ramki, etc). Dodatkowo istnieją klasy pomocnicze definiujące urządzenie OpenCL, zapewniające obsługę wyjątków oraz funkcje pozwalające uzyskać dodatkowe informacje o implementacji OpenCL na komputerze. Do przetwarzania obrazu z kamery została użyta klasa strumienia algorytmów, który definiuje listę algorytmów, którymi kolejno przetwarza zadany obraz.

Moduł obsługi kamery

Moduł obsługi kamery składa się z kamery oraz klasy imitującej kamerę. Obsługa kamery została maksymalnie uproszczona i zaimplementowana z użyciem biblioteki dołączonej do kamery. Klasa imitująca kamerę pobiera kolejno obrazy z określonego folderu udostępniając je, dzięki czemu można było przeprowadzać testy bez konieczności posiadania kamery.
Kamera wraz z biblioteką do jej obsługi posiada szerokie możliwości. Oferuje do wyboru wiele formatów przesyłanego obrazu. W tym przypadku wykorzystano obraz w formacie matrycy Bayera oraz o rozdzielczościach piksela: 8, 10, 12 i 16 bitów.

4. Symulacja i Testowanie



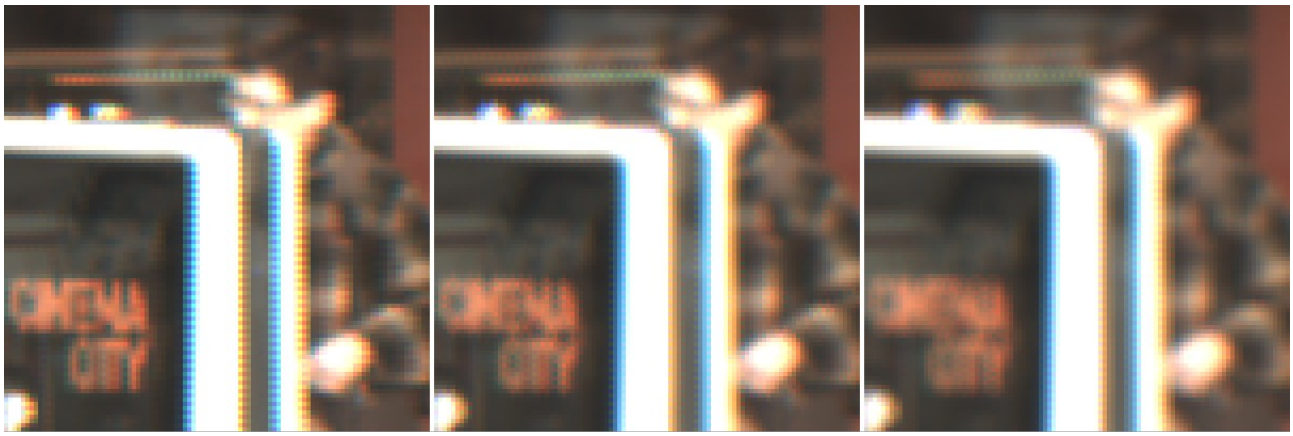
Rysunek 6: Fragmentu wynikowego obrazu z implementacji w OpenCL (a), implementacji w OpenCV (b) oraz róznicy obrazów (c). Obraz (d) przedstawia różnice koloru czerwonego, (e) zielonego, a (f) niebieskiego.

Testowanie i weryfikacja poprawności odbyło się za pomocą porównania stworzonej implementacji do już istniejącej. Do porównania wybrano realizację interpolacji z biblioteki OpenCV. Na rysunku 3 znajdują się (kolejno od lewej) fragmenty obrazu wynikowego z algorytmu zaimplementowanego w OpenCL, algorytmu z OpenCV oraz różnica wartości tych obrazów.

Porównując wyniki obu implementacji można zauważyć różnicę w wartości koloru zielonego (6c i 6e). Kolory czerwony i niebieski nie różnią się w ogóle (moc obrazu różnicowego jest równa zero). Maksymalne różnice koloru zielonego wynoszą 30 i choć to spora wartość, trudno zauważyć je gołym okiem. Te rozbieżności wynikają z innego sposobu obliczania wartości koloru zielonego w przypadku kontekstu przedstawionego na rysunku 3b i 3c, w zaimplementowanym algorytmie wartość jest średnią arytmetyczną wszystkich 5 pikseli w algorytmie z OpenCV wartość jest kopiowana ze środkowego piksela.

5. Analiza jakościowa algorytmów

Zostały zaimplementowane 3 różne maski do interpolacji kolorów w przestrzeni RGB z obrazu pobranego z matrycy Bayera. Im rozmiar maski jest większy tym mniej poszarpane są krawędzie na obrazach i zmniejsza się efekt szumu RGB. Im rozmiar maski jest mniejszy tym ostrzejszy jest obraz wynikowy, operacja interpolacji przebiega szybciej. Na rysunku 4 przedstawiono (kolejno od lewej) fragmenty obrazów interpolowanych z maską kwadratową 3x3, maskę okrągłą i maską w kształcie krzyża. Widać na nich jak wraz ze zwiększeniem rozmiaru maski znikają poszarpane krawędzie, ale również zmniejsza się czytelność obrazu (np. napis „Cinema City" na ostatnim obrazie 7c) jest prawie nieczytelny).



Rysunek 7: Fragmenty obrazów interpolowanych z różnymi maskami. a) maska 3x3, b) maska okrągła, c) maska w kształcie krzyża.

6. Analiza czasowa algorytmu

Porównano czas wykonania obu implementacji: stworzonej oraz dostępnej w bibliotece OpenCV. Dla własnej realizacji porównano wynik wszystkich trzech użytych masek. W OpenCV dostępna jest tylko maska kwadratowa 3x3 piksele. Test był przeprowadzany przy użyciu 79 obrazów o rozmiarze 2456 x 2058 pobranych z kamery. Osiągnięte wyniki przedstawiono w tabeli 1. Czas obliczeń na karcie graficznej jest to czas wykonania algorytmu na karcie graficznej, bez liczenia czasu kopiowania pamięci oraz przeprowadzania koniecznych operacji pomocniczych. Testy zostały przeprowadzone z użyciem kart graficznych nVidia GeForce GT 555M, nVidia GeForce GeForce 9800GT i procesorze Intel Core i7-2670QM (dla implementacji w OpenCV). W tabeli maski podane są zgodnie z rysunkiem 2. Przedstawiono czasy:

- czas całości obliczeń – czas wywołania funkcji obliczeniowej (bez czasu odczytywania pliku i wyświetlania obrazu na ekranie). Zawiera obliczenia oraz konieczne transmisje danych do i z karty graficznej.
- czas obliczeń na karcie graficznej - czas wykonania kerneli.

Tabela 1: Tabela porównawcza czasów wykonania przetwarzania obrazów w zaimplementowanych algorytmach

	OpenCV	Maska a)	Maska b)	Maska c)	Maska a)	Maska b)	Maska c)
		GeForce 555M			GeForce 9800GT		
Czas całości obliczeń	2,548s	2,139s	3,107s	2,309s	4,914s	6,162s	8,947s
Średni czas na jeden obraz	0,032s	0,027s	0,039s	0,029s	0,062s	0,078s	0,113
Czas obliczeń na karcie graficznej		1,467s	2,462s	1,571s	4,301s	4,499s	7,837s
Średni czas obliczeń na karcie graficzne na jeden		0,018s	0,031s	0,020s	0,054	0,057s	0,099s

7. Rezultaty i wnioski

Na podstawie wyników testów można zauważyć, że różnice pomiędzy OpenCL a nowszą kartką graficzną w czasie całego przetwarzania obrazów są niewielkie,. Jednak jeśli porówna się czas samych obliczeń (przy realizacji OpenCV nie ma potrzeby kopiowania danych pomiędzy pamięcią a kartą graficzną) to różnica wychodzi znacząca. Widać także, że użycie starszej karty graficznej znacznie wydłuża czas obliczeń. Wydaje się, że głównym czynnikiem określającym szybkość działania algorytmu na karcie graficznej jest częstotliwość procesora graficznego oraz częstotliwość pamięci. Dla karty 555M jest to 1250MHz dla rdzenia i 1800MHz dla pamięci. Dla karty 9800GT jest to 600MHz dla rdzenia i 900MHz dla pamięci.

8. Podsumowanie

Udało się zrealizować osiągnięte cele. Zaimplementowany algorytm poprawnie interpoluje obraz barwny z kamery korzystającej z matrycy Bayera. Mimo istniejących różnic w obrazach wynikowych w porównaniu z metodą referencyjną z biblioteki OpenCV, spowodowanych inną definicją użytą w algorytmie, można przyjąć poprawność działania przygotowanej implementacji, ze względu na nierozróżnialność obrazów gołym okiem.

Uzyskane wyniki mogą nie być w pełni satysfakcjonujące, zwłaszcza dla karty 9800GT. Poza obliczeniami do realizacji algorytmu na karcie graficznej konieczny jest transfer danych do karty. Dla karty 9800GT, który ma większy interfejs pamięci (256bit) ten czas jest nieznacznie krótszy niż czas dla karty 555M (128bit). Dodatkowo można zaobserwować, że dla karty 555M czas transferu wynosi 30%-40% całej długości obliczeń.

W kolejnych modyfikacjach można spróbować zidentyfikować główne punkty opóźnień i zoptymalizować kernele, które zostały stworzone. Dodatkowo można rozwinąć bibliotekę i dodać kolejne algorytmy, które wykorzystywałyby interpolowany obraz. Dzięki strukturze strumienia nakład prac nad kolejnymi algorytmami byłby mniejszy niż realizacja od początku. Również przy dodaniu nowych algorytmów do jednego strumienia ograniczona będzie liczba operacji kopiowania pomiędzy pamięciami procesora i karty graficznej.

9. Literatura

1: Mirosław Jabłoński, Metodyka zrównoleglania algorytmówprzetwarzania i analizy obrazóww systemach przepływowych, 2009

2: Khronos, Dokumentacja OpenCL, 2012, http://www.khronos.org/registry/cl/sdk/1.1/docs/man/xhtml/

4: , Dokumentacja interfejsu do kamery: JAI SDK,

3: Google, GTest, 2012, http://code.google.com/p/googletest/

10. DODATEK A: Szczegółowy opis zadania

Specyfikacja projektu

Typowa kamera kolorowa wyposażona w czujnik z filtrem mozaikowym typu Bayer dostarcza strumień wizyjny w postaci poziomów szarości. Celem projektu jest implementacja algorytmu interpolacji składowych R, G i B obrazu z kamery kolorowej wysokiej rozdzielczości w czasie rzeczywistym z wykorzystaniem równoległych zasobów karty graficznej. Algorytm należy zakodować w postaci programu OpenCL (ang. kernel). W ramach projektu należy przeprowadzić testy wydajności poprzez uruchomienie algorytmu na GPU oraz CPU oraz wyznaczyć przyspieszenie względem wersji programowej (np. OpenCV). Oczekiwanym rezultatem jest biblioteka DLL realizująca interpolację z możliwością parametryzacji początkowych współrzędnych matrycy oraz współczynników wzmocnienia kanałów R i B. Oprogramowanie powinno pracę w różnych formatach danych: 8, 10 i 12 bitów. Testy powinny obejmować dwa rodzaje kerneli: wykorzystujące 1-wymiarową strukturę danych oraz 2-wymiarowe obrazy.

11. DODATEK B: Dokumentacja techniczna

Do tworzenia implementacji w systemie Windows użyto programu Visual Studio 2012. Wersja 2012 jest konieczna do kompilacji w systemie Windows z uwagi na użycie niektórych funkcji standardu C++11, które zostały wprowadzone w tej wersji.
W systemie Linux można użyć kompilatora GCC w wersji 4.7.2.
Stworzone testy jednostkowe są zrealizowane w oparciu o GTest [3].

Instrukcja wykonania programu:

Wraz z biblioteką dostarczony jest przykładowy program wykorzystujący możliwości biblioteki (BayerFilter.exe). Pozwala on na przeprowadzenie interpolacji barwnej z obrazów, które są zapisane w odpowiedniej postaci. Dostępne opcje:

- -c – pobieranie obrazów z kamery JAI. Jest to również opcja wybierana w przypadku nie podania żadnej.
- d katalog prefiks – pobieranie obrazów z katalogu. Obrazy muszą mieć wspólny prefiks, następnie trzycyfrowy kolejny numer (zaczynając od 000), oraz rozszerzenie bmp. Dodatkowa opcja to --openCV, która zmienia realizację z karty graficznej na OpenCV.
- -f plik_wejsciowy –o plik_wyjsciowy – przetworzenie jednego obrazu i zapisanie wyniku w podanym miejscu. Dodatkowa opcja to --openCV, która zmienia realizację z karty graficznej na OpenCV.

- -h – wyświetla pomoc.

Dodatkowe opcje:

- --wb r g b – pozwala ustalić balans bieli. Należy podać kolejno liczby odpowiadające kolorowi czerwonemu, zielonemu i niebieskiemu. Zalecane wartości to od 0.0 do 1.0. Ta opcja nie ma wpływu na wynik przy użyciu opcji –openCV.
- --device nvidia – umożliwia określenie urządzenia, które będzie wykorzystane do obliczeń. Wybierane jest pierwsze urządzenie z listy.

Podczas uruchomienia bez opcji --device, jeśli jest potrzeba (bez opcji --openCV) program pyta o wybór urządzenia, które należy zastosować do obliczeń z użyciem OpenCL.

Dokumentacja klas

Przygotowany kod był dokumentowany w komentarzach, na podstawie których został wygenerowany podręcznik klas i funkcji który znajduje się w pliku `reference_manual.pdf`. Dokumentacja ta została wygenerowana w programie doxygen.

Dokumentacja kerneli realizujących interpolację

```
__kernel void  bayer_image(__read_only image2d_t input, __write_only image2d_t output, __constant uchar * params, __global float* balance)
```

```
__kernel void  bayer_image_circle_mask(__read_only image2d_t input, __write_only image2d_t output, __constant uchar * params, __global float* balance)
```

```
__kernel void  bayer_image_cross_mask(__read_only image2d_t input, __write_only image2d_t output, __constant uchar * params, __global float* balance)
```

Kernele reprezentujące algorytm interpolacji dla trzech kontekstów wymienionych w rozdziale 2.
Parametry:

- `__read_only image2d_t input` – wejściowa struktura obrazu przygotowana przez odpowiednie funkcje OpenCL
- `__write_only image2d_t output` – wyjściowa struktura obrazu przygotowana i zaalokowana wcześniej przez odpowiednie funkcje OpenCL.
- `__constant uchar * params` – wartość odpowiadająca ułożeniu pikseli w topologi Bayera dla pierwszego piksela. Wartość 0 dla RGR, 1 dla GRG, 2 dla GBG lub wartość 3 dla BGB.
- `__global float* balance` – trzyelementowa tablica wartości współczynników wzmocnienia odpowiednio dla wzmocnienia kolorów R, G i B. Wartości powinny być z zakresu 0 – 1.

Dokumentacja kerneli realizujących konwersję z float do int i odwrotnie

```
__kernel void intToFloat8bit(__read_only image2d_t input, __write_only image2d_t output)
__kernel void intToFloat10bit(__read_only image2d_t input, __write_only image2d_t output)
__kernel void intToFloat12bit(__read_only image2d_t input, __write_only image2d_t output)
__kernel void intToFloat16bit(__read_only image2d_t input, __write_only image2d_t output)
__kernel void floatToUInt8ThreeChannels(__read_only image2d_t input, __write_only image2d_t output)
__kernel void floatToUInt16ThreeChannels(__read_only image2d_t input, __write_only image2d_t output)
__kernel void floatToUInt32ThreeChannels(__read_only image2d_t input, __write_only image2d_t output)
```

Parametry:

- `__read_only image2d_t input` – wejściowa struktura obrazu w formacie zgodnym z openCV
- `__write_only image2d_t output` – wyjściowa struktura obrazu

12. DODATEK D: Spis zawartości dołączonego nośnika (płyta CD ROM)

W poszczególnych folderach nośnika, w zależności od rodzaju projektu, znajdują się:

- **SRC** - Kompletne źródła projektu wraz z plikami VisualStudio 2012 i plikami Cmake:
  - `BayerFilter` – źródła aplikacji integrującej napisane biblioteki, dające pokaz możliwości algorytmu.
  - `JAICameraInterface` – źródła biblioteki opakowującej interfejs kamery.
  - `OpenCLInterface` – źródła biblioteki opakowującej interfejs OpenCL oraz dostarczającej klas algorytmów.
  - `OpenCLAlgorithmTest` – źródła testów GTest do sprawdzania poprawności implementacji algorytmów OpenCL.
  - `OpenCLDeviceTest` – źródła testów GTest do sprawdzania poprawności implementacji klas opakowujących interfejsy OpenCL.
  -
- **EXE** - postać binarna skompilowana na systemie Windows 7 64bit:
  - `BayerFilter.exe` – plik wykonywalny
  - `OpenCLInterface.dll` – biblioteka dll zawierająca klasy implementujące algorytmy w OpenCL, m.in. algorytm BayerFilter
  - `JAICameraInterface.dll` – biblioteka dll opakowująca interfejs kamery JAI, oraz dodatkowo klasę „sztucznej" kamery, która dostarcza serię obrazów z dysku, jak normalna kamera.
  - Dodatkowe biblioteki potrzebne do uruchomienia programu.
- **TEST -** folder zawierający serię przykładowych obrazów:
  - data – pojedyncze pliki obrazów o rozmiarze 1024x1024,
  - data2 – seria obrazów o rozmiarze 2456x2058 z kamery JAI.
- **DOC** - ten dokument w wersji PDF i MS WORD i OpenOffice. Dokumentacja klas w wersij PDF.

13. DODATEK E: Historia zmian

Tabela $1 Historia zmian.

Autor	Data	Opis zmian	Wersja
Bartłomiej Bułat, Tomasz Drzewiecki	2013-01-09	Pierwsza wersja dokumentu	1.0
Bartłomiej Bułat, Tomasz Drzewiecki	2013-01-21	Poprawki, dokładny opis algorytmu, dokumentacja kerneli, szczegółowy opis zawartości płyty, rozszerzona część testów	1.1