Usuwanie tła z wykorzystaniem GPU – raport z projektu

Szymon Wąsik, czerwiec 2007

Założenia projektu

Cele i opis projektu

Celem projektu było stworzenie aplikacji, która z filmu wczytanego z pliku bądź kamery usunie z niego tło znajdujące się za poruszającymi się obiektami i zamieni je na inne. Dodatkowo jak najwięcej wykonywanych obliczeń miało zostać przystosowane do uruchamiania na procesorze karty graficznej (GPU). Wydajność kart graficznych coraz szybciej rośnie oraz wzrastają możliwości zrównoleglenia obliczeń z ich zastosowaniem, tak więc jest to ciekawa alternatywa dla klasycznych obliczeń na CPU, mogąca znacznie zwiększyć wydajność aplikacji.

W celu umożliwienia wydajnej implementacji na GPU konieczne było opracowanie algorytmu, który byłby jak najbardziej lokalny. Najlepiej, żeby operował tylko na pojedynczych pikselach. Jako, że w praktyce niemożliwe jest skonstruowanie jakiejkolwiek uniwersalnej bazy modeli służącej do wyodrębniania tła, pewne było, że zastosowane będzie wnioskowanie oparte na cechach. Początkowo rozważane było podejście oparte o macierze historii obrazu, jednak później zostało ono porzucone ze względu na kiepską jakość i trudności implementacyjne na GPU. Ostatecznie użyty został algorytm bazujący na liczeniu średniej i odchyleń standardowych kolejnych klatek.

Wykorzystane technologie i biblioteki

W programie wykorzystane zostały następujące technologie oraz biblioteki:

- Karta ATI Radeon X1400 Mobility – karta graficzna dla laptopów zgodna z OpenGL 2.0 oraz DirectX 9c ze 128MB pamięci oraz dwoma vertex i czterema pixel shaderami, w pełni zgodna z OpenGL 2.0,
- Biblioteka OpenGL 2.0 służąca do inicjalizacji i obsługi trybu graficznego karty graficznej,
- Biblioteka GLee 5.21 – biblioteka deklarująca adresy funkcji oraz wartości stałych potrzebnych do skorzystania z rozszerzeń OpenGL (takich jak Frame Buffer Object) – funkcjonalność ta nie jest częścią składową właściwej biblioteki OpenGL,
- Środowisko NVidia Cg 1.5 umożliwiające pisanie i kompilowanie programów uruchamianych na karcie graficznej,
- Biblioteka OpenCV 1.0 wykorzystywana do przechwytywania kolejnych klatek z kamery wideo.

Sposób przetwarzania danych na GPU

Architektura kart graficznych narzuca specyficzny sposób wykonywania na nich obliczeń. Po pierwsze dane reprezentowane są jako tekstury. Większość w miarę nowych kart oferuje możliwości zadeklarowania tekstur zmiennoprzecinkowych, w których każda współrzędna koloru każdego piksela może być opisana dowolną liczbą zmiennoprzecinkową. Te tekstury można następnie traktować jako tablice elementów typu float. Takie tablice można łatwo przekazać do pamięci karty graficznej. Można do

niej również łatwo przekazać pojedyncze liczby typu float, będące na przykład parametrami opisującymi algorytm.

Obliczenia na karcie graficznej wykonywane są podczas renderowania nowej tekstury, która zawiera wartości obliczanych elementów. Główny program wykonywany na CPU zleca karcie graficznej wyrenderowanie tekstury, która ma następnie pokryć pewien prostokąt. Karta graficzna dla każdego piksela generowanej tekstury uruchamia tak zwany program fragmentu, który oblicza kolor danego piksela. Przy przetwarzaniu danych na GPU wartość tego koloru powinna być wartością, która ma być obliczona. Programy fragmentów uruchamiane są na tzw. jednostkach teksturujących, których na jednej karcie graficznej może być od kilku do kilkudziesięciu, co daje duże możliwości zrównoleglenia wykonania programu. Niestety zanim przetwarzanie dla wszystkich pikseli nie zakończy się, nie można poznać wartości obliczonej dla tych pikseli, dla których przetwarzanie zakończyło się już wcześniej.

Z powyższego schematu wynika, że do implementacji na GPU dobrze nadają się głównie algorytmy lokalne, w których wartość każdego obliczanego elementu tablicy nie zależy od wyniku obliczeń dla innych elementów.

Wykorzystane algorytmy i techniki

Podejście początkowe

Początkowo przeprowadzona została próba wykorzystania algorytmu opierającego się na macierzy historii (Motion History Image – MHI). W podejściu tym każdy piksel zaznaczany jest jako poruszający się, jeżeli jego wartość zmieniła się w stosunku do jego wartości kilka klatek wstecz o więcej niż pewna wartość progowa δ. Piksel jest zaznaczony jako poruszający się tak długo, aż nie upłynie pewien okres czasu τ, mierzony liczbą klatek. Po jego upływie piksel zaznaczany jest jako tło. Na podstawie macierzy MHI można następnie obliczyć macierz gradientu na podstawie której można przeprowadzić segmentację obrazu, w wyniku której znajdziemy przemieszczające się obiekty. Niestety takie podejście okazało się mieć dwie wady:

- Ciężko zaimplementować je z wykorzystaniem GPU, ponieważ w celu dokonania segmentacji potrzebne jest wykonanie przeszukania całej macierzy (np. algorytmem DFS), które jest operacją mało lokalną.
- Otrzymane z próbnej implementacji na CPU wyniki nie były najlepsze – algorytm nie znajdował sporych części przemieszczających się obiektów i bardzo szybko przestawał wykrywać obiekty, które zatrzymały się na planie.

Podejście ostateczne

Ze względu na powyższe wady wykorzystany został inny algorytm. Opiera się on o wartości dwóch macierzy obliczane następująco:

$$\mu_{i+1} = \alpha * F_i + (1 - \alpha) * \mu_i$$
$$\sigma_{i+1}^2 = \alpha * (F_i - \mu_{i+1})^2 + (1 - \alpha) * \sigma_i^2$$

gdzie macierz F zawiera wartości kolorów pikseli kolejnych klatek, a indeks i numeruje kolejne klatki.

Macierz μ jest to średnia ważona klatek, przy czym wagi zmieniają się zgodnie z postępem geometrycznym zależnym od współczynnika α, tak że ostatnio pobrana klatka ma zawsze największą wagę. Średnia taka jest dobrym przybliżeniem tła, ponieważ gdy wartość współczynnika α jest mała, poru-

szające się obiekty, które widoczne są przez stosunkowo krótki okres czasu, mają niewielki wpływ na średnią, natomiast pozostałe fragmenty obrazów, które stanowią tło, powodują uśrednianie macierzy μ do aktualnego wyglądu tła. Dzięki temu tło może się z czasem trochę zmieniać i zmiany te zostaną uwzględnione.

Macierz σ^2 to macierz odchyleń standardowych kolejnych klatek. Dzięki niej, w przypadku, gdy na obraz nakładają się szumy, można określić jaka zmiana koloru jest już ruchem, a jaka tylko szumem. W celu określenia, które piksele znajdują się w ruchu, stosowany jest warunek:

$$| \mu_i - F_{i+1} | > k * \sigma_i$$

W powyższym wzorze współczynnik k jest mała liczbą, najczęściej z przedziału (2.0, 3.0) – zgodnie z zasadą trzech sigm dla rozkładu normalnego.

Łatwo zauważyć, że w powyższych wzorach piksele należące do poruszających się obiektów wpływają na średnią tak samo jak piksele nie należące do nich, co może powodować zniekształcenie wartości macierzy uśredniającej tło. Dlatego też w programie zastosowano poprawkę. Dla pikseli, które zgodnie z przedstawionym powyżej warunkiem zostały zaklasyfikowane jako poruszające się, macierze μ oraz σ^2 modyfikowane są zgodnie z następującymi wzorami:

$$\mu_{i+1} = \alpha_m * F_i + (1 - \alpha_m) * \mu_i$$
$$\sigma_{i+1}^2 = \sigma_i^2$$

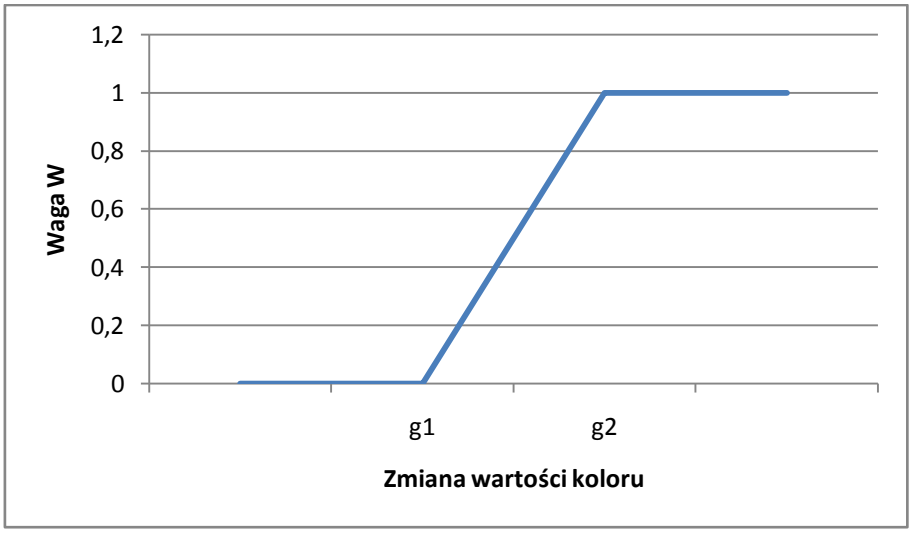
Średnia zmienia się zgodnie ze współczynnikiem α_m, który jest kilkukrotnie mniejszy od współczynnika α. Powoduje to dużo wolniejsze zmiany średniej pod wpływem przemieszczających się obiektów, pozwalając cały czas na lekkie modyfikacje tła w momencie, gdy z różnych powodów tło ulegnie zmianie (takich jak dostawienie lub zniknięcie przedmiotu, zmiana barwy lub natężenia oświetlenia, lekkie poruszenie przedmiotu znajdującego się w tle, poruszenie kamery i inne).

Odchylenie standardowe przemieszczających się pikseli nie jest modyfikowane, ponieważ zbyt mocno zawyżyłoby to wartości, które mają opisywać jedynie szumy występujące w filmie.

Zastosowane parametry

Razem z programem opracowane zostały eksperymentalnie dwa zbiory parametrów – dla scen dynamicznych oraz statycznych. W scenach dynamicznych następuje dużo ruchu, tło zmienia się często, a nawet w tle znajdują się lekko przemieszczające się obiekty. Tak więc algorytm musi nadążyć za dużą ilością zmian. W scenach statycznych tło nie ulega za dużym zmianom, a na samym początku algorytm ma szansę sfilmować samo tło, aby ustalić wartość początkową macierzy μ. W scenach dynamicznych przyjęto α i α_m równe odpowiednio 0.20 oraz 0.10, natomiast w scenach statycznych 0.09 oraz 0.001.

Warunek sprawdzający, czy piksel należy do przemieszczającego się obiektu został lekko zmodyfikowany. Na jego podstawie obliczana jest waga W zmieniająca się od 0.0 do 1.0 określająca w jakim stopniu piksel znajduje się w ruchu. Jeżeli $| \mu_i - F_{i+1} | < g_1$ to waga wynosi 0.0 i piksel uznawany jest za tło, jeżeli $| \mu_i - F_{i+1} | > g_2$ to piksel uznawany jest za przemieszczający się i waga wynosi 1.0. Dla wartości pośrednich waga zmienia się liniowo, tak jak jest to pokazane na poniższym wykresie.



Powyższy zabieg pozwolił na wygładzenie konturów przemieszczających się obiektów i pozwolił uniknąć pojawiania się dziur w tle bądź obiektach, złożonych z pikseli, które błędnie nie zostały do nich zaliczone. Takie piksele zostaną uwzględnione, jednak z lekko zmodyfikowaną wagą. Dodatkowo, po wyliczeniu wag, macierz W przetwarzana jest filtrem gaussowskim, w celu rozmycia ewentualnie występujących w niej ostrych konturów oraz zmniejszeniu widoczności szumu, który nie został obcięty przez macierz odchyleń standardowych. Ponieważ w programie wartości W były liczone osobno dla każdego kanału RGB, ostatecznie jako wartość określająca w jakim stopniu piksel się porusza brana była maksymalna z tych trzech wartości. Takie podejście eksperymentalnie okazało się dużo lepsze niż uśrednianie wartości składowych koloru.

W programie przyjęte zostało:

$g_1 = 2.0 * \sigma + \delta_1$
$g_2 = 3.0 * \sigma + \delta_2$

Dodawane progi δ_1 oraz δ_2 to małe wartości pozwalające na dokładniejsze odseparowanie ruchu. W programie wynosiły one odpowiednio 2.0 i 7.0 dla scen statycznych oraz 3.0 i 10.0 dla scen dynamicznych w skali w której opisane były składowe kolorów, czyli od 0.0 do 255.0.

Ostatnim współczynnikiem użytym w programie był współczynnik h, który określał od jakiej wartości wagi w piksel jest uznawany za poruszający się przy aktualizacji macierzy średnich i odchyleń. Tak więc dla wartości w > h używany był współczynnik α_m, dla pozostałych α. Dla scen dynamicznych współczynnik ten wynosił 0.2, dla statycznych 0.7.

Implementacja na GPU

Program został przystosowany do przetwarzania danych na procesorze karty graficznej z wykorzystaniem języka i kompilatora Cg firmy NVidia. Zostały w tym celu wykorzystane następujące technologie:

- Frame buffer object (FBO) – umożliwia zapisywanie wyników przetwarzanych obliczeń bezpośrednio do powiązanej z obiektem FBO tekstury, znajdującej się w pamięci karty graficznej – tzw. Render to Texture. Dzięki temu nie trzeba dokonywać kosztownego czasowo kopiowania narysowanego obrazu do tekstury dopiero po zakończeniu obliczeń.
- Multi Texture Rendering (MTR) – umożliwia podłączenie do obiektu FBO więcej niż jednej tekstury i zapisywanie danych jednocześnie do wszystkich z nich. W ten sposób można za

pomocą jednego przebiegu wygenerować do czterech tablic danych. W tym projekcie generowane były macierze średniej, odchyleń, wag oraz wynik powstały po nałożeniu nowego tła.

Test programu

Jakość działania

Dla scen w których nie występują duże zmiany tła jakość działania algorytmu jest bardzo dobra. Poprzez ustawienie parametrów α na niskim poziomie można uzyskać bardzo dobre wyniki, takie jak przedstawiony na poniższym obrazku. Przy dobrym, silnym oświetleniu, gdy na początku działania damy algorytmowi trochę czasu na wyznaczenie uśrednionego tła jakość rozpoznawania ruchu i usuwania tła jest wręcz idealna.

Największy problem pojawia się z obiektami, które w momencie startu algorytmu znajdują się w obszarze rejestrowanym przez kamerę. W momencie gdy obiekty te zaczną się poruszać i ukaże się znajdujące się pod nimi tło, algorytm rozpozna również to tło jako poruszający się obiekt (ponieważ wartości pikseli mocno się zmieniły), przez co będą one bardzo wolno uwzględniane w średniej tła. Można temu zaradzić zwiększając wartość współczynnika α_m, jednak takie podejście spowoduje również dużo szybsze zlewanie się z tłem obiektów, które rzeczywiście poruszają się. Opisany w tym akapicie problem demonstruje poniższy zrzut ekranu.

Mniejszy problem stanowią obiekty, które przy inicjalizacji algorytmu znajdowały się poza sceną, a dopiero później pojawiły się na niej i zatrzymały w miejscu. Ustawiając bardzo małą wartość parametru α_m oraz małą wartość α można otrzymać algorytm, który przez bardzo długi czas będzie potrafił odróżnić taki obiekt od tła.

Wydajność

Ze względu na dużą lokalność wykonywanych obliczeń – poza filtrem gaussowskim nie było konieczności pobierania wartości sąsiednich pikseli – algorytm doskonale nadaje się do implementacji na GPU. Dla każdego piksela algorytm musi wykonać około 200 operacji arytmetycznych (mnożenia, dzielenia, odejmowania, dodawania i pierwiastkowania) co dla rozdzielczości VGA (640 x 480) oraz 15 klatek na sekundę daje około 700 milionów operacji na sekundę, nie licząc wielu czasochłonnych odwołań do pamięci. Jest to tak duża liczba instrukcji, że algorytm zaimplementowany w wersji na CPU, na procesorze Intel Core Duo 1.83 GHz z 2 GB RAM w cztery razy mniejszej rozdzielczości QVGA i z pominięciem filtru gaussowskiego działał w tempie niecałe 2 klatki na sekundę. Ten sam algorytm zaimplementowany z wykorzystaniem GPU potrafił w czasie rzeczywistym przetwarzać film z 15 klatkami na sekundę w pełnej rozdzielczości VGA. Niestety jest to koniec jego możliwości, aby uzyskać większe przyspieszenie trzeba by zoptymalizować jego kod, na co cały czas jest miejsce lub użyć karty graficznej z większą liczbą jednostek teksturujących.

Używanie napisanego programu

Program można uruchomić za pomocą polecenia ROMove.exe. Program uruchomiony bez parametrów wczytuje film z zewnętrznego pliku ustawionego w kodzie programu. Jeżeli przekaże mu się jeden parametr „cam” obraz będzie wczytywany z kamery i przetwarzany jako scena dynamiczna. Po przekazaniu parametrów „cam static” obraz będzie wczytywany z kamery, ale użyte zostaną parametry dla sceny statycznej. Aby zmodyfikować wartości używanych parametrów lub nazwę pliku wejściowego trzeba niestety zmienić odpowiadające im wartości w pliku main.cpp. Niestety w bieżącej wersji nie ma możliwości podania ich z linii poleceń.

Możliwe przyszłe modyfikacje

Mimo, że program osiąga dość dobre wyniki, w przyszłości można by rozważyć przetestowanie następujących modyfikacji, które mogłyby poprawić jakość:

- Przy sprawdzaniu które piksele zmieniły swoją wartość nie kierować się kanałami RGB, ale kanałem Hue, co pozwoliłoby uniezależnić się od drobnych zmian oświetlenia.

- Postarać się odfiltrować pojawiające się dziury i niespójności, tak aby uniknąć usuwania fragmentów obiektów lub dodawania fragmentów tła.
- Zamiast modyfikować macierz średniej za pomocą skokowej zmiany parametru α rozważyć jego liniową zmianę w pewnym przedziale (tak jak jest to aktualnie wykonywane dla macierzy W).
- Wyznaczyć za pomocą jakiegoś algorytmu, na przykład piramidalnego Lucas-Kanade, przesunięcia pikseli i na tej podstawie wykrywać rzeczywiste ścieżki obiektów. Na tej podstawie można by wykryć obiekty, które długo znajdowały się w spoczynku i nagle zaczęły się poruszać.
- Wzbogacić możliwości definiowania parametrów z linii poleceń lub dorobić GUI.