Uniwersytet Wrocławski
Wydział Matematyki i Informatyki

Krzysztof Templin
nr albumu: 186850

Zwiększanie postrzeganej rozdzielczości obrazów w ruchu

Praca magisterska napisana pod kierunkiem
dra Andrzeja Łukaszewskiego

Wrocław, 2011

Spis treści

Przedmowa

W 2010 roku Piotr Didyk, Elmar Eisemann, Tobias Ritschel, Karol Myszkowski i Hans-Peter Seidel, naukowcy z Max-Planck-Institut für Informatik w Saarbrücken (Niemcy), opublikowali pracę pod tytułem „Apparent Display Resolution Enhancement for Moving Images". Przedstawiono w niej metodę, która wykorzystując pewne właściwości układu wzrokowego człowieka oraz możliwości nowoczesnych wyświetlaczy LCD umożliwia zwiększenie postrzeganej rozdzielczości poruszających się obrazów, co objawia się wyraźną poprawą w odwzorowaniu detali. Jak wskazują autorzy, niska rozdzielczość wyświetlaczy wciąż pozostaje czynnikiem ograniczającym w grafice komputerowej czy fotografii cyfrowej. Przedstawienie we właściwej skali szczegółów takich jak włosy czy metaliczna farba, wymaga większych rozdzielczości niż te, które oferują współczesne wyświetlacze. Również postępy miniaturyzacji i wzrost popularności urządzeń przenośnych zmuszają do poszukiwań sposobów czytelniejszego przedstawiania detali w warunkach mocno ograniczonej liczby dostępnych pikseli. Publikacja ta została zaprezentowana w Los Angeles na prestiżowej konferencji SIGGRAPH.

Od czerwca 2010 do stycznia 2011 miałem przyjemność współpracować z Karolem Myszkowskim i Piotrem Didykiem nad rozszerzeniem możliwości wspomnianej metody. Niniejsza praca magisterska powstała w oparciu o zebrany wówczas materiał. Opisuję w niej zarys publikacji Didyka i in., a następnie przedstawiam propozycję uogólnienia ich metody na inne typy animacji oraz dokumentuję przeprowadzone eksperymenty. Uzupełnienie pracy stanowi załączona implementacja w języku C++.

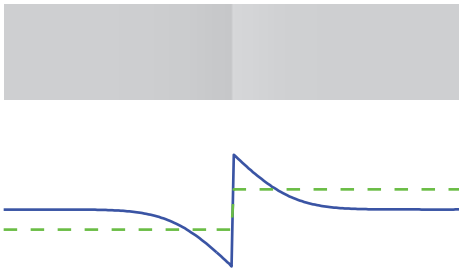
Chciałbym podziękować Karolowi Myszkowskiemu za umożliwienie mi prowadzenia badań nad tym zagadnieniem oraz Piotrowi Didykowi za długie godziny dyskusji, podczas których dzielił się ze mną swoją wiedzą. Udzielona przez nich pomoc w znacznym stopniu zaważyła na kształcie mojej pracy magisterskiej.

1. Informacje wprowadzające

Niniejsza praca ma następujący układ: w tej części zaznajamiamy czytelnika z paroma faktami odnośnie układu wzrokowego człowieka oraz jego własnościami, które leżą u podstaw opisywanej metody. Wspominamy także o mechanizmach działania wyświetlaczy komputerowych. Część 2 to streszczenie publikacji na której bazuje nasza praca [Didyk i in., 2010]. W części 3 przedstawiamy propozycję uogólnionej metody zwiększania postrzeganej rozdzielczości obrazu. Ostatecznie, w części 4 omawiamy naszą implementację, którą znaleźć można na dołączonej do pracy płycie kompaktowej.

Grafika a percepcja

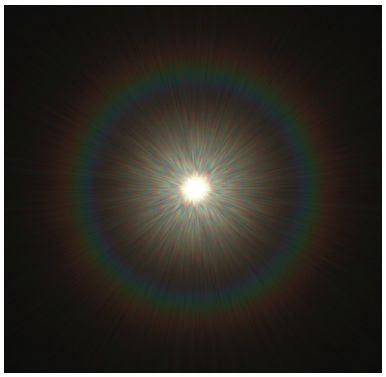
W dziedzinie grafiki komputerowej daje się zaobserwować w ostatnich latach starania, aby ograniczenia sprzętowe przezwyciężać przy pomocy technik uwzględniających właściwości układu wzrokowego człowieka. Przykładem takiego ograniczenia może być zakres dynamiczny współczesnych wyświetlaczy. Ich rozpiętość tonalna wciąż pozostaje dużo mniejsza od rozpiętości rzeczywistych scen, co często uniemożliwia ich realistyczne odwzorowanie. Z problemem tym możemy sobie częściowo poradzić stosując pewne „percepcyjne sztuczki".



Rysunek 1: Iluzja Craika-Cornsweeta. Przerywaną linią oznaczono jasność postrzeganą, linią ciągłą – jasność rzeczywistą. Źródło: adaptacja ilustracji Krawczyka i in. [2007].

W iluzji Craika-Cornsweeta, przedstawionej na rys. 1, mamy do czynienia z dwoma polami o różnych rozkładach jasności. Różnica w luminancji, dość znaczna na granicy między nimi, zmniejsza się stopniowo aż do zera wraz z odległością od środka rysunku. Wydaje się jednak, że jasność lewego pola jest mniejsza, nawet jeśli porównamy brzegi rysunku. Dopiero zasłonięcie krawędzi ujawnia, że mamy do czynienia z iluzją. Zestawiając odpowiednio kilka profili Craika-Cornsweeta możemy wywołać wrażenie szeregu obszarów o coraz większej jasności, mimo że początkowa i końcowa jasność jest taka sama. Pomysłowe wykorzystanie tej obserwacji pozwoliło Krawczykowi i in. [2007] na zwiększenie postrzeganego kontrastu fotografii poddanych kompresji zakresu dynamicznego.

Innym podejściem może być modelowanie układu optycznego oka i symulacja interakcji promieni świetlnych z jego poszczególnymi elementami. Spoglądanie wprost na silne źródło światła powoduje kilka interesujących efektów świetlnych, m. in. efekt *glare* przedstawiony na rys. 2. Wyświetlacze komputerowe nie dysponują tak dużą jasnością aby je wywołać, jednak wprowadzenie podobnych wzo-



Rysunek 2: Efekt *glare*. Źródło: Ritschel i in. [2009].

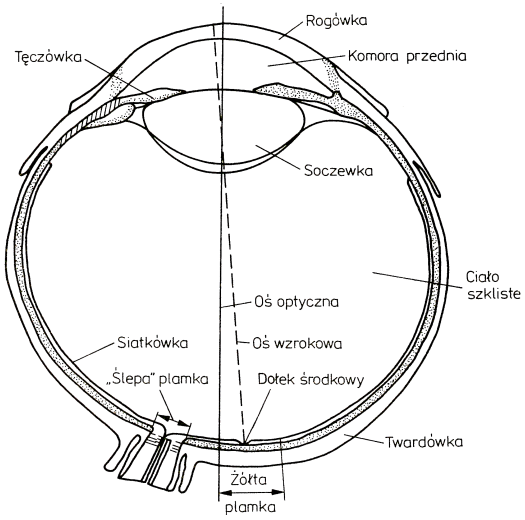
rów w odpowiednich miejscach obrazu może zwiększyć jego postrzeganą jasność [Yoshida i in., 2008]. Modelowanie może przebiegać z różnym stopniem szczegółowości: wiele osób zwraca uwagę na fakt, że rzeczywisty efekt *glare* nie jest statyczny, lecz ulega niewielkim fluktuacjom. Ritschel i in. [2009] pokazali, że uwzględnienie tego aspektu pozwala na dodatkowe wzmocnienie iluzji.

Publikacja Didyka i in. wpisuje się w ten trend, proponując sposób na zwiększenie postrzeganej rozdzielczości obrazu. Nie jest to pierwsza praca tego typu (por. Damera-Venkata i Chang [2009], Allen i Ulichney [2005]), jednak wyróżnia ją brak skomplikowanych rozwiązań technicznych oraz wykorzystanie zdolności człowieka do dokładnego podążania wzrokiem za poruszającymi się obiektami.

Budowa oka

Elementem oka o szczególnym znaczeniu dla ostrości widzienia jest region siatkówki zwany plamką żółtą (łac. *macula lutea*). W jej centrum leży dołek środkowy (łac. *fovea centralis*), obszar najostrzejszego widzenia, pokrywający 2° kąta wzrokowego [Ayoub, 2008]. To właśnie na dołek środkowy pada obraz obiektu, na którym fiksujemy wzrok. Dołek jest niezbędny przy czytaniu, oglądaniu filmów itp. Jest to miejsce szczególnej koncentracji czopków, komórek odpowiedzialnych za widzenie w dobrych warunkach oświetleniowych (widzenie fotopowe), natomiast praktycznie nie występują tam pręciki, czyli komórki umożliwiające widzenie nocne (skotopowe). Prawie każda komórka zwojowa przyjmuje sygnał z tylko jednego czopka [Fiok, 1991], tak więc ostrość widzenia w dołku ograniczona jest tylko przez gęstość fotoreceptorów.

Pomiary Curcio i in. [1990] pokazują, że zagęszczenie receptorów w dołku środkowym dochodzić może do 28" (sekund kątowych). Piksel wyświetlacza full-HD o przekątnej 23 cali obserwowanego z odległości 60 cm ma szerokość około 1.5' (minuty kątowej). Oznacza to, że na jeden piksel może przypadać około 9 receptorów. Sugeruje to, że człowiek może rozróżnić detale o rozmiarach mniejszych niż te, które jest w stanie odwzorować wyświetlacz.



Rysunek 3: Schematyczny przekrój oka. Zaczerpnięto z: Fiok [1991], źródło: Felhorski i Stanioch [1973].

Ruch śledzący oka

Istnieją dwa sposoby wolicjonalnej zmiany miejsca skupienia wzroku: sakady – bardzo szybkie skokowe ruchy, występujące np. podczas czytania oraz ruchy śledzące (ang. *smooth pursuit eye movements*, SPEM). Ruch śledzący pojawia się, gdy chcemy skupić wzrok na poruszającym się obiekcie. W istocie, dość trudno wywołać ten ruch przy braku poruszającego się bodźca. Celem tego ruchu jest stabilizacja obrazu na siatkówce, co przekłada się na ostrość widzenia. Mechanizm ten działa szczególnie dobrze dla obiektów poruszających się jednostajnym ruchem prostoliniowym: śledzenie obiektu poruszającego się z prędkością kątową 0.625—2.5 st/s jest niemal doskonałe, a przy prędkościach poniżej 7 st/s pozostaje wciąż bardzo dobre [Laird i in., 2006]. Samo rozpoczęcie ruchu śledzącego jest procesem bardzo szybkim: przebiega w czasie krótszym niż 100 ms.

Uśrednianie sygnału w czasie

Inną ważna własnością układu wzrokowego jest uśrednianie w czasie. Mechanizm ten leży u podstaw działania wyświetlaczy CRT: każda plamka ekranu, zależnie od modelu i ustawień, rozbłyska kilkadziesiąt do stu kilkudziesięciu razy na sekundę, jednakże obraz wydaje się nie zmieniać. Jeśli rozbłyski są zbyt wolne, pojawia się charakterystyczne „pływanie" bądź migotanie obrazu (ang. *flickering*). Częstotliwość przy której się to dzieje nazywana jest częstotliwością zanikową migotania (*critical flickering frequency*, CFF).

Wartość CFF zależna jest od wielu czynników, takich jak średnia jasność pola obserwacji (ang. *adaptation luminance*), rozmiar obserwowanego obiektu czy region siatkówki, na który pada badany obraz. Prawo Ferry'ego-Portera mówi, że CFF wzrasta liniowo wraz z logarytmem jasności bodźca. Oznacza to, że im jaśniejszy bodziec, tym łatwiej zaobserwować migotanie. Czopki są bardziej czułe na migotanie niż pręciki, tak więc w warunkach widzenia fotopowego CFF będzie wyższa niż podczas

widzenia skotopowego. Zmiana regionu siatkówki pociąga za sobą zmianę proporcji czopków do pręcików, a zatem widzenie centralne wiąże się na ogół z wyższą wartością CFF. Badania wykazały, że dla centralnie obserwowanego bodźca o rozmiarach 0.3°, CFF może nieznacznie przekroczyc 40 Hz. Zwiększanie rozmiaru bodźca powoduje wzrost CFF, jednak do 19° pozostaje ona poniżej 60 Hz. Należy przy tym odnotować, że powyżej CFF efektywna jasność bodźca równa jest jego średniej jasności. Innymi słowy, przy odpowiednio krótkim okresie sygnału nie ma znaczenia dla postrzeganej jasności czy światło jest równomiernie rozłożone w czasie czy też nie. Własność ta znana jest jako prawo Talbota-Plateau [Kalloniatis i Luu, 2009].

Wyświetlacze

Obecnie wśród wyświetlaczy komputerowych wyróżnić można dwa najpopularniejsze typy: wyświetlacz kineskopowy (CRT) oraz wypierający go wyświetlacz ciekłokrystaliczny (LCD). Pomimo wielu niewątpliwych zalet wyświetlacze LCD mają tendencję do rozmywania poruszających się obiektów (ang. *motion blur*). Przyczyną tego zjawiska jest odmienny sposób prezentacji sygnału przez wyświetlacz: podczas cyklu odświeżania pojedynczy piksel monitora CRT rozbłyska przez krótką chwilę, podczas gdy w monitorze LCD świeci światłem ciągłym. Gdy obserwator zaczyna śledzić poruszający się na monitorze obiekt, fotoreceptory mieszają sygnał sąsiadujących pikseli. Ponadto, piksele wyświetlacza LCD nie zmieniają swojej jasności natychmiastowo, co również przyczynia się (aczkolwiek w mniejszym stopniu) do rozmycia obrazu [Pan i in., 2005].

Podczas swoich badań Didyk i in. używali 22-calowego wyświetlacza LCD firmy Samsung, model SyncMaster 2233 RZ, o częstotliwości odświeżania 120 Hz i rozdzielczości 1680×1050. W przyjętym modelu pominęli oni opóźnienia ekranu i przyjęli, że zmiana jasności pikseli jest natychmiastowa.

2. Zwiększanie postrzeganej rozdzielczości obrazu

Celem, do którego dążą Didyk i in. [2010] jest wyświetlenie statycznego obrazu I_H na wyświetlaczu o rozdzielczości mniejszej niż res(I_H). Osiągają to poprzez wyświetlenie serii obrazów mniejszej rozdzielczośći $I_L^t, t = 0, 1, 2, \ldots$ z prędkością 120 klatek na sekundę. Tak duża prędkość powoduje oczywiście ich uśrednienie, jednak ze względu na odpowiedni dobór klatek postrzegany obraz nie zawiera artefaktów i zdaje się mieć rozdzielczość wyższą niż rozdzielczość ekranu. Przypadkiem, któremu poświęcono szczególną uwagę była sytuacja, w której $\text{res}(I_H) = 3 \cdot \text{res}(I_L^t)$. Proporcja ta koresponduje z pomiarem gęstości receptorów w siatkówce oraz – jak przekonamy się później – z częstotliwością zanikową migotania.

Model

Didyk i in. zakładają następujący model: z pojedynczym fotoreceptorem przy danych warunkach obserwacji związany jest pewien okres całkowania T. Reakcja receptora r obserwującego pozycję p zmieniającego się w czasie obrazu I określona jest wzorem $r = \int_0^T I(p, t)\,\mathrm{d}t$. Zauważmy, że jeżeli oko pozostaje nieruchome względem wyświetlacza, wszystkie receptory obserwujące pojedynczy piksel, zareagują tak samo i nie będziemy w stanie zwiększyć postrzeganej rozdzielczośći obrazu. Jeśli pozwolimy, aby receptory zmieniały swoją pozycję, wzór ten uogólni się do postaci

$$r = \int_0^T I(p(t), t)\,\mathrm{d}t. \tag{1}$$

Dla różnych ścieżek $p(t)$ otrzymamy w ogólności różne wartości powyższej całki, co daje możliwość zróżnicowania reakcji pojedynczych receptorów.

Kolejnym założeniem modelu jest rozkład receptorów w siatkówce. Autorzy przyjęli, że obserwują one rozmieszczone kratowo pozycje, odpowiadające pikselom wyświetlanego obrazu o dużej rozdzielczości. Sytuacja nie zmienia się, gdy obraz zaczyna się przemieszczać: uznajemy, że dokładność SPEM jest na tyle wysoka, aby założyć doskonałe przypisanie pojedynczych receptorów do pikseli poruszającego się obrazu.

Metoda

Przedstawimy teraz metodę, za pomocą której Didyk i in. wyświetlają obraz I_H. Po pierwsze, należy wymusić ruch oka względem wyświetlacza. W tym celu obraz I_H jest z klatki na klatkę przemieszczany o pewien wektor. Gdy obserwator skupi się na jakimś detalu obrazu, pojawia się ruch śledzący oka, który stabilizuje obraz na siatkówce. Zgodnie z założeniami modelu jesteśmy w stanie dokładnie wyznaczyć ścieżki $p(t)$ związane z receptorami.

Zauważmy, że funkcja $I(p(t), t)$ jest schodkowa: zmiana wartości następuje po wyświetleniu następnej klatki lub po przejściu do innego piksela na ścieżce. Stąd całkę w równaniu 1 można zastąpić ważoną sumą wartości pikseli w kolejnych klatkach:

$$r = \sum_k w_k \cdot I(p(t_k), t_k). \tag{2}$$

Dla zilustrowania metody, rozważymy teraz prosty przykład, w którym obraz I_H jest jednowymiarowy, a jego rozdzielczość jest dwukrotnie większa od rozdzielczości wyświetlacza. W każdej klatce obraz przesuwa się o pół piksela wyświetlacza. Każdemu pikselowi $I_H(i)$ odpowiada śledzący go receptor r_i. Przyjmujemy okres integracji T równy czasowi wyświetlenia dwóch kolejnych klatek. Zauważmy, że po dwóch klatkach obraz I_H przesunie się dokładnie o jeden piksel wyświetlacza. Znajdziemy się zatem w sytuacji wyjściowej, więc będziemy mogli wyświetlić ponownie te same klatki, przesunięte o jeden piksel.

Receptor r_i zależnie od parzystości i przez dwie klatki obserwuje jeden bądź dwa sąsiadujące piksele. Jego reakcja wyraża się wzorem:

$$r_i = \frac{1}{2} \cdot \begin{cases} I_L^0(i/2) + I_L^1(i/2) & i \bmod 2 = 0, \\ I_L^0(i/2) + I_L^1(i/2+1) & i \bmod 2 = 1. \end{cases} \tag{3}$$

Aby postrzegany obraz odwzorowywał I_H, wartości r_i powinny być równe wartościom $I_H(i)$. Odpowiada to układowi równań liniowych:

$$Wx = I_H, \tag{4}$$

gdzie $x = (I_L^0 I_L^1)^T$, a macierz W koduje wagi zawarte w sumowaniu 2; w tym przypadku każdy wiersz zawiera dokładnie dwie niezerowe wartości, równe $\frac{1}{2}$.

Poza szczególnymi przypadkami, rząd macierzy W jest większy od liczby niewiadomych (jasności wyświetlanych pikseli), stąd dokładne rozwiązanie układu jest niemożliwe. Do wyznaczenia niewiadomych autorzy zastosowali metodę najmniejszych kwadratów, tj. minimalizację wyrażenia

$$\|Wx - I_H\|_2^2, \tag{5}$$

przy założeniu, że elementy x należą do zakresu $[0, 1]$. Ograniczenie to uzasadnione jest tym, że jasność piksela nie może być ujemna, ani przekraczać maksymalnej jasności wyświetlacza.

Aliasing i migotanie

Istotnym zagadnieniem jest występowanie aliasingu w postrzeganym obrazie. W poszczególnych klatkach otrzymanych w wyniku optymalizacji pojawia się aliasing, jednak ze względu na częstotliwość odświeżania ekranu kolejne klatki zlewają się ze sobą. Klatki są tak dobierane, aby po uśrednieniu przez układ wzrokowy jak najbardziej przypominały oryginalny obraz. Zatem o ile obraz o wysokiej rozdzielczości nie zawierał aliasingu, nie powinien go zawierać również obraz postrzegany. Własność tę co prawda trudno udowodnić formalnie, jednak została ona potwierdzona eksperymentalnie – żadna z badanych osób nie zaobserwowała występowania aliasingu.

Przedstawiony model fotoreceptora jest znacznym uproszczeniem. Prawdą jest, że reakcja receptora jest wypadkową zmian sygnału w czasie, jednak nie wyraża się ona jako prosty filtr prostokątny (por. Van Hateren [2005]). Dodatkowo należy pamiętać, że oprócz reakcji na poziomie neurofizycznym znaczenie mają też czynniki psychofizyczne. Jeśli jednak wyświetlać będziemy okresowy sygnał o dużej częstotliwości (powyżej CFF), obowiązywać zacznie prawo Talbota-Plateau i rozbieżność między średnim sygnałem na przestrzeni jednego okresu, a sygnałem postrzeganym stanie się niezauważalna: obraz

będzie miał stałą jasność. Jak wspomnieliśmy, CFF dla małych, obserwowanych centralnie bodźców tylko nieznacznie przekracza 40 Hz. Taką samą częstotliwość ma sygnał powstały w wyniku zapętlenia trzech klatek na wyświetlaczu o odświeżaniu 120 Hz. Didyk i in. co prawda starają się poprawić jedynie widoczność *detali*, ale na powierzchni *całego obrazu*, co może prowadzić do migoczących powierzchni o rozmiarach kilkunastu stopni: obraz o szerokości 600 pikseli daje około 15°. Stąd ich metoda w pewnym sensie balansuje na granicy, jednakże w toku eksperymentów migotanie nie było obserwowane. Zwiększenie liczby klatek cyklu do czterech wymagałoby już zastosowania omówionego dalej mechanizmu redukcji migotania. Oczywiście sytuacja zmieniłaby się, gdyby zastosować wyświetlacz o wyższej częstotliwości.

Eksperymenty

W celu zbadania skuteczności swojej metody Didyk i in. przeprowadzili dwie serie eksperymentów, do których zaprosili odpowiednio 14 i 5 osób. Pierwsza seria badała skuteczność metody w przypadku statycznych dwuwymiarowych obrazów i stanowiła trzon części eksperymentalnej. Druga natomiast zajmowała się przypadkiem animacji trójwymiarowych, jednakże w dość ograniczonym zakresie: zbadano jedynie animacje, w których pewne regiony miały odpowiednie prędkości i klatki animacji dobierano pod kątem tychże regionów, nie dbając o jakość pozostałych.

W pierwszej serii sprawdzono pięć różnych obrazów (fotografie, tekst oraz obrazy syntetyczne o wysokim stopniu szczegółowości), które przesuwano o całkowite wielokrotności piksela (prędkości składowe w zakresie 0–3) i zmniejszano trzykrotnie. Animacje wygenerowane metodą Didyka i in. porównywane były do obrazów powstałych na drodze standardowej procedury decymacji z zastosowaniem uprzedniego filtrowania antyaliasowego (okienko Lanczosa lub jeden z rodziny filtrów Mitchella-Netravali, dopasowany do preferencji badanej osoby). Badanym osobom przedstawiano do porównania nieoznaczone animacje w losowej kolejności i proszono o wskazanie tej, która lepiej odwzorowuje detale. Badanie wykazało przewagę metody Didyka i in. W badaniu sprawdzono również *czytelność* generowanych obrazów. W tym celu ręcznie zaprojektowano czcionkę rastrową o wymiarach 6×9 pikseli, którą następnie zmniejszano trzykrotnie (rozmiar pojedynczego znaku to jedynie 2×3 piksele!) i przesuwano w poziomie. Eksperyment wykazał lepszą czytelność 13 z 26 badanych znaków przy zastosowaniu metody Didyka i in. Po szczegółowe wyniki eksperymentów wraz z dokładnym opisem zastosowanych procedur zainteresowanego czytelnika odsyłamy do oryginalnej pracy.

Szczegóły implementacyjne

Algorytm wyznaczający optymalne podklatki zaimplementowano jako kod programu Matlab. Optymalizacja przebiegała w oparciu o funkcję `lsqlin`. Podejście to sprawdziło się, jeśli chodzi o jakość wyników, jednak jego efektywność nie była najlepsza. Przykładowo, wyznaczenie 3 klatek dla obrazu o rozdzielczości full-HD zajmowało ok. 5 minut. Skłoniło to autorów do poszukiwania szybszych metod optymalizacji. Metoda gradientu prostego zaimplementowana jako zestaw *fragment shaderów* GLSL pozwoliła uzyskać czasy poniżej 1 sekundy. Ograniczenie przestrzeni poszukiwań do przedziału $[0, 1]$ było wymuszane poprzez obcięcie po każdej iteracji.

Redukcja migotania

Wprowadzenie dłuższych okresów integracji, czy to ze względu na mniejszą częstotliwość odświeżania wyświetlacza czy też większą liczbę podklatek w okresie powoduje pojawienie się migotania obrazu. Bazując na dostępnych pomiarach czułości układu wzrokowego na migotanie [Mäkelä i in., 1994], autorzy wprowadzają dodatkową metodę pozwalającą na jego redukcję. Istota tej metody polega na mieszaniu (ang. *alpha blending*) wyznaczonych podklatek z niemigoczącym obrazem wyznaczonym w drodze standardowej decymacji. Proporcja w jakiej mieszane są obrazy jest indywidualnie wyznaczana dla każdego piksela i dobierana tak, aby fluktuacje w jasności pikseli utrzymywały się poniżej progu dostrzegalności. Oczywiście taki sposób redukcji migotania odbywa się kosztem zmniejszenia szczegółowości obrazu, jednak jak podają autorzy, dla 4-klatkowego okresu na 120-hercowym wyświetlaczu, poprawa postrzeganej rozdzielczości jest wciąż widoczna. Ponieważ widoczność migotania jest zależna od rozmiaru obserwowanego obszaru, metoda wykrywająca nadmierne fluktuacje nie może polegać tylko na wartościach pojedynczych pikseli, ale musi też brać pod uwagę ich sąsiedztwo. Dlatego budowana jest *piramida gaussianów* – ciąg obrazów $I_0, I_1, I_2, \ldots$ o coraz mniejszych rozmiarach, z których każdy powstaje z poprzedniego w drodze *downsamplingu*. Przez I_0 rozumiemy oryginalny obraz, natomiast żeby wyznaczyć I_{k+1} do obrazu I_k stosowany jest odpowiedni filtr gaussowski, po czym liczba próbek zmniejszana jest dwukrotnie. Na każdym poziomie piramidy obliczane są współczynniki redukcji migotania, które następnie są propagowane w dół piramidy, tak że pojedynczy piksel obrazu I_0 jest redukowany z maksymalnym współczynnikiem spośród przypisanych mu na wszystkich poziomach.

3. Uogólniona metoda zwiększania postrzeganej rozdzielczości

W tej części postaramy się uogólnić przedstawioną metodę na inne typy animacji. Zajmiemy się kolejno trzema przypadkami o wzrastającym stopniu ogólności. Najpierw przyjrzymy się sytuacji, gdy statyczny obraz porusza się z dowolną prostoliniową prędkością (3.1). Następnie zaproponujemy metodę zwiększenia rozdzielczości wyrenderowanych animacji trójwymiarowych, dla których dysponujemy funkcją przepływu optycznego (3.2), po czym przejdziemy do najogólniejszego przypadku animacji, których przepływ optyczny jest nieznany (3.3).

3.1. Ruch statycznego obrazu

W przypadku ruchu statycznego obrazu Didyk i in. [2010] skupili się w swoich eksperymentach głównie na wyświetlaniu obrazu o 3-krotnie większej rozdzielczości przesuwającego się z całkowitoliczbową prędkością z zakresu $[0, 3]^2$. Wymusiło to pewną szczególną postać problemu optymalizacji klatek, gdyż animacja była zapętlona: wystarczyło wyznaczyć postać trzech klatek. Dodatkowo założyli oni uproszczoną postać układu równań: dokładne ścieżki receptorów nie były wyznaczane, lecz każdy receptor zależał w równym stopniu od wartości dokładnie 3 pikseli, a każdy piksel miał wpływ na dokładnie 9 receptorów.

W tej sytuacji, pierwszym narzucającym się uogólnieniem jest zbadanie ruchu prostoliniowego o dowolnym przesunięciu w obu osiach, niekoniecznie powodującego zapętlenie animacji. Zgodnie z zaproponowanym modelem, reakcja fotoreceptora w ustalonym przedziale całkowania T jest średnią ważoną kolorów zaobserwowanych pikseli, gdzie wagi są wprost proporcjonalne do czasu obserwacji danego piksela. Formalnie reakcję możemy wyrazić jako sumę $\sum_{i,j,k} w^k_{i,j} I^k_{i,j}$, gdzie wagi obliczane są według wzoru:

$$w^k_{i,j} = \frac{1}{|p(t)|} \int_0^T \chi_{i,j}(p(t)) \chi_k(t) \, \mathrm{d}t. \tag{6}$$

Przez $\chi_{i,j}$ rozumiemy funkcję charakterystyczną, równą 1 gdy argument leży wewnątrz piksela o współrzędnych (i, j). Analogicznie, χ_k jest równa 1 gdy argument, będący punktem w czasie, przynależy do k-tej klatki animacji.

Niestety nie możemy już polegać na zapętleniu animacji. Czas potrzebny na powrót obrazu wysokiej rozdzielczości do wyjściowego położenia (lub zbliżonego) względem siatki pikseli ekranu może być dużo dłuższy niż 3 klatki. W wyświetlanej animacji pojawią się zatem częstotliwości mniejsze niż 40 Hz, które będą się przyczyniać do niepożądanego zjawiska flickeringu. Efekt ten postaramy się zneutralizować poprzez wymuszanie właściwej reakcji receptorów w każdym podciągu trzech klatek animacji. Niestety powoduje to wzrost liczby niezerowych elementów macierzy W proporcjonalny do długości animacji.

Implementacja

Podczas swoich badań korzystaliśmy z Plexusa – wciąż rozwijanego frameworku i środowiska do badań nad grafiką komputerową autorstwa Tobiasa Ritschela. W systemie tym, inspirowanym przez programy

do *compositingu*, użytkownik-programista implementuje w postaci pojedynczych klas C++ tzw. urządzenia. Są to obiekty ze ściśle zdefiniowanym interfejsem wejścia-wyjścia, odpowiedzialne za wykonywanie pojedynczych operacji graficznych, które następnie wraz z innymi urządzeniami stanowią jednostki składowe acyklicznego grafu, reprezentującego całość badanego algorytmu. Graf budowany jest w wizualnym edytorze, a do swojej dyspozycji użytkownik ma bogatą bibliotekę urządzeń wykonujących podstawowe zadania. Na przepływ danych w grafie, którym zajmuje się sam system, użytkownik ma wpływ poprzez specjalne urządzenia logiczne, które modyfikują domyślny sposób propagacji danych. Podczas implementowania urządzeń programista dysponuje szeregiem *wrapperów*, reprezentujących obiekty takie jak obraz, tekstura OpenGL, shader GLSL itp.

Plexus posłużył nam do początkowych badań nad algorytmem optymalizacji (ostatecznie eksperymenty przeprowadziliśmy na niezależnej implementacji w C++, którą znaleźć można na dołączonej płycie) oraz do wyznaczenia przepływu optycznego dla animacji (por. 3.3).

W naszej implementacji użyliśmy do optymalizacji wariantu metody gradientu prostego, wzorowanej na metodzie użytej przez Damerę-Venkatę i Changa [2009]. Ogólny zarys algorytmu przedstawiał się następująco:

1. Wczytaj obraz.
2. Przejdź do przestrzeni liniowej odwracając gamma-korekcję.
3. Zainicjalizuj klatki wyjściowe jednolitym kolorem.
4. Wykonaj kilka iteracji:
   (a) Dla każdego podciągu trzech klatek zasymuluj reakcję receptorów.
   (b) Oblicz różnicę między oryginalnym obrazem a reakcją receptorów w każdym podciągu.
   (c) Dystrybuuj różnicę w podciągach na całą sekwencję.
5. Zastosuj gamma-korekcję i zwróć podklatki.

W sercu tych procedur leży funkcja do wyznaczania wag pikseli zgodnie z równaniem 6:

```
input
    off ▷ wektor oznaczający początek odcinka
    vel ▷ wektor oznaczający długość odcinka
output
    result ▷ lista trójek zawierająca współrzędne pikseli wraz z wagami
fun result = generate_mask(off, vel)
    result = [ ]

    ▷ factor to współczynnik, o jaki zmniejszamy rozmiar obrazka; na ogół 3
    off = off / factor
    vel = vel / factor

    ▷ całkowite współrzędne prostokąta ograniczającego odcinek
    sleft = ⌊min(off.x, off.x + vel.x)⌋
```

```
sright = ⌈max(off.x, off.x + vel.x)⌉ + 1
sbot = ⌊min(off.y, off.y + vel.y)⌋
stop = ⌈max(off.y, off.y + vel.y)⌉ + 1

▷ jeśli odcinek jest bardzo krótki
if |vel| < ϵ
    result = [(sleft, sbot, 1)]
    return
endif

▷ iteracja po wszystkich pikselach prostokąta ograniczającego
for r = sbot to stop − 1
    for c = sleft to sright − 1
        ▷ współrzędne końców odcinka
        (x1, y1) = off
        (x2, y2) = off + vel
        ▷ współrzędne rozważanego piksela
        left = c, right = c + 1, bot = r, top = r + 1

        ▷ te same operacje względem obu osi
        for axis = 1 to 2
            ▷ zamień miejscami końce odcinka jeśli pierwszy jest leksykograficznie większy
            if x1 > x2
                swap(x1, x2)
                swap(y1, y2)
            endif
            ▷ sprawdź czy rzuty na oś odcinka i piksela nie są rozłączne
            if x1 > right or x2 < left
                x1 = x2 = y1 = y2 = 0
                break
            endif
            ▷ odrzuć części odcinka, których rzut leży poza rzutem piksela
            if x1 < left
                y1 = y1 + (y2 − y1) · (left − x1) / (x2 − x1)
                x1 = left
            endif
            if x2 > right
                y2 = y2 − (y2 − y1) · (x2 − right) / (x2 − x1)
                x2 = right
            endif
            swap(x1, y1)
            swap(x2, y2)
```

```
            swap(left, bot)
            swap(right, top)
        endfor
        ▷ znormalizowana waga proporcjonalna do długości pozostałej części odcinka
        weight = |(x2 − x1, y2 − y1)| / |vel|
        ▷ odrzuć piksele o bardzo małych wagach (w szczególności o wadze zero)
        if weight > ϵ
            result = [result, (c, r, weight)]
        endif
    endfor
  endfor
endfun
```

W oparciu o powyższą funkcję możemy zdefiniować dwie dualne procedury zależne od numeru klatki. Pierwsza z nich dla zadanego piksela obrazu wysokiej rozdzielczości (receptora) zwraca listę współrzędnych wraz z wagami, odpowiadającą pikselom niskiej rozdzielczości, przez które „przejdzie" receptor:

```
input
  off ▷ współrzędne receptora
  num ▷ numer klatki
output
  pixels ▷ lista trójek zawierająca współrzędne pikseli wraz z wagami
fun pixels = receptor_to_pixels(off, num)
  pixels = generate_mask(off + num · vel, vel)
endfun
```

Druga procedura dla danych współrzędnych piksela i numeru klatki wyznacza listę receptorów na które piksel ma wpływ, wraz z wagami:

```
input
  off ▷ współrzędne piksela
  num ▷ numer klatki
output
  receptors ▷ lista trójek zawierająca współrzędne receptorów wraz z wagami
fun receptors = pixel_to_receptors(off, num)
  receptors = {(x, y, w) | (off.x, off.y, w) ∈ receptor_to_pixels((x, y), num)}
endfun
```

Te dwie procedury posłużą odpowiednio do wyznaczenia błędu (punkt 4b) oraz jego dystrybucji (punkt 4c). Całość optymalizacji ilustruje następujący pseudokod:

```
input
    image ▷ obraz wysokiej rozdzielczości
    factor ▷ współczynnik o jaki zmniejszamy rozmiar obrazu
    vel ▷ prędkość z jaką poruszać się ma obraz
    len ▷ długość sekwencji do wygenerowania
    iterations ▷ liczba iteracji optymalizacji
output
    subframes ▷ optymalna lista klatek niskiej rozdzielczości
fun subframes = optimize(image, factor, vel, len, iterations)
    (width, height) = size(image)
    (swidth, sheight) = size(image) / factor
    ▷ odwróć gamma-korekcję
    image = gamma^{-1}(image)
    ▷ inicjalizuj klatki; new_image() zwraca czarny obraz
    for i = 0 to len − 1
        subframes[i] = new_image(swidth, sheight)
        corrections[i] = new_image(swidth, sheight)
        weightsums[i] = new_image(swidth, sheight)
    endfor
    for iter = 1 to iterations
        for i = 0 to len − 3
            response = new_image(width, height)
            ▷ symuluj reakcję receptorów
            for x, y, s in {0, ..., width − 1} × {0, ..., height − 1} × {0, 1, 2}
                L = receptor_to_pixels((x, y), i + s)
                for sx, sy, w in L
                    response[x, y] = response[x, y] + w ·(1/3)· subframes[i + s][sx, sy]
                endfor
            endfor
            ▷ błąd reakcji receptorów
            error = image − response
            ▷ dystrybuuj błąd reakcji na piksele klatek
            for sx, sy, s in {0, ..., swidth − 1} × {0, ..., sheight − 1} × {0, 1, 2}
                L = pixel_to_receptors((sx, sy), i + s)
                ▷ poprawka piksela to średnia ważona błędów odpowiednich receptorów
                for x, y, w in L
                    corrections[i + s][sx, sy] = corrections[i + s][sx, sy] + w · error[x, y]
                    weightsums[i + s][sx, sy] = weightsums[i + s][sx, sy] + w
                endfor
            endfor
```

```
        endfor
        ▷ uaktualnij podklatki, zeruj poprawki i współczynniki normalizacji
        for i = 0 to len − 1
            subframes[i] = subframes[i] + corrections[i] / weightsums[i]
            ▷ obetnij wartości pikseli do zakresu [0, 1]
            subframes[i] = clamp(subframes[i], 0, 1)
            corrections[i] = new_image(swidth, sheight)
            weightsums[i] = new_image(swidth, sheight)
        endfor
    endfor
    for i = 0 to len − 1
        subframes[i] = gamma(subframes[i])
    endfor
endfun
```

Klasyczny resampling

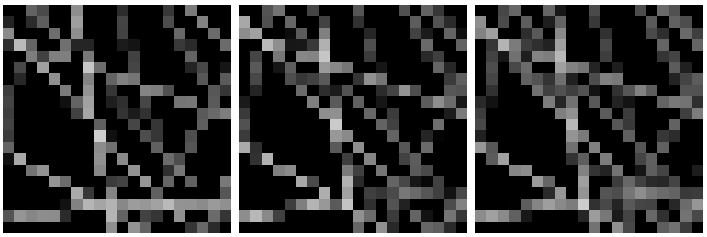
W ocenie skuteczności metody ważne jest wybranie właściwego punktu odniesienia do porównań. Didyk i in. swoje wyniki porównywali z animacją, w której każda klatka była wynikiem *downsamplingu* oryginalnego obrazu. W procesie tym używamy próbek bitmapy o wysokiej rozdzielczości do rekonstrukcji oryginalnej sceny przedstawionej na obrazie. Następnie, z obrazu usuwamy częstotliwości powyżej granicy Nyquista aby uniknąć aliasingu, po czym sygnał ponownie próbkujemy. W wyidealizowanym przypadku oba te kroki możemy przeprowadzić przy użyciu splotu z przeskalowaną funkcją sinc. W dziedzinie fourierowskiej odpowiada to dokładnej selekcji częstotliwości poniżej pewnej częstotliwości granicznej zależnej od współczynnika skalowania (im „szersze" jądro splotu, tym więcej częstotliwości odetniemy). Niestety jest to proces stratny, a dodatkowo w praktyce zmuszeni jesteśmy stosować jego aproksymacje, stąd ocena jakości wyniku jest w znacznej mierze subiektywna. Didyk i in. używali okienka Lanczosa:

$$L(x) = \begin{cases} \operatorname{sinc}(x)\operatorname{sinc}(x/2) & |x| \leqslant 2, x \neq 0, \\ 1 & x = 0, \\ 0 & 2 < |x| \end{cases}$$

lub jednego z filtrów Mitchella-Netravali:

$$M_{b,c}(x) = \frac{1}{6} \cdot \begin{cases} (12 - 9b - 6c)|x|^3 + (-18 + 12b + 6c)|x|^2 + (6 - 2b) & |x| \leqslant 1, \\ (-b - 6c)|x|^3 + (6b + 30c)|x|^2 + (-12b - 48c)|x| + (8b + 24c) & 1 < |x| \leqslant 2, \\ 0 & 2 < |x|. \end{cases}$$

Oba filtry przeskalowano tak, aby ich promień miał długość 6. Promień ten był dobrany tak, aby pojedyncze klatki animacji nie zawierały aliasingu. Podczas naszych badań zaobserwowaliśmy jednak



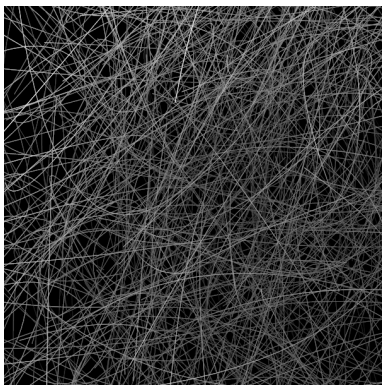
4.1 4.2 4.3

Rysunek 4: Wycinek z klatki uzyskanej za pomocą optymalizacji (4.1) oraz filtrowania Lanczosa w przestrzeni liniowej z promieniem 3 (4.2) i 4 (4.3).

następującą własność: jeśli zmniejszymy promień do 3-4 jednostek, tym samym pozwalając na pojawienie się aliasingu w pojedynczych klatkach, otrzymamy rezultaty bardzo podobne do tych powstałych w wyniku optymalizacji. Dodatkowo, odwrócenie gamma-korekcji przed filtrowaniem może spowodować dalsze zmniejszenie tej różnicy[1]. Porównanie przedstawiono na rys. 4. Powstaje zatem pytanie, czy optymalizacji nie można by całkowicie wyeliminować, podobnie jak w pracy Damery-Venkaty i Changa [2009] zastępując ją odpowiednio dobranymi filtrami. Pierwsze próby Didyka i in. w tym kierunku zakończyły się jednak niepowodzeniem i problem ten pozostaje otwarty.

Eksperyment

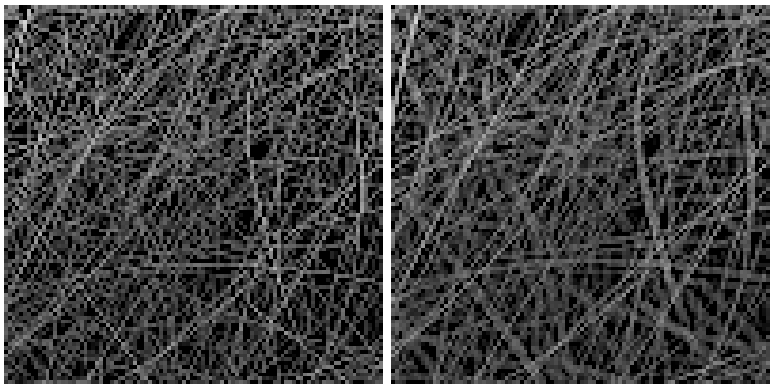
Eksperyment polegał na wygenerowaniu animacji odpowiadających ruchom prostoliniowym o prędkościach z zakresu $[0, 3]^2$. Przestrzeń tę pokryliśmy równomiernie stosując krok 0.1 piksela. Ograniczyliśmy się do prędkości o składowej x niewiększej niż składowa y, ze względu na symetrię pozostałych przypadków. W wyniku otrzymaliśmy 495 animacji.



Rysunek 5: Obraz testowy dla ruchu o dowolnej prędkości. Źródło: Didyk i in. [2010].

Jako obrazu testowego użyliśmy zdesaturowanego wycinka z oryginalnego obrazu stosowanego w

[1] Obserwacja ta przyczyniła się do decyzji, aby we wszystkich eksperymentach omawianych w tej pracy, przed zastosowaniem filtrowania Lanczosa przechodzić do przestrzeni liniowej odwracając gamma-korekcję.



6.1 6.2

Rysunek 6: Porównanie działania optymalizacji dla prędkości (1, 1) i prędkości (6, 6).

eksperymentach przez Didyka i in. (rys. 5). Wycinek miał wymiary 600×600 pikseli, długość sekwencji 120 klatek.

Dla żadnego ruchu o prędkości niecałkowitej nie otrzymaliśmy w pełni zadowalających rezulatatów, aczkolwiek dla kilkunastu z nich efekty były bardzo dobre, np. ruch $(1.2, 1.2)$ lub $(1.8, 1.8)$. Właściwie dla każdego przypadku prędkości niecałkowitej dało się zaobserwować artefakty, które podzielić można na trzy grupy:

1. Flickering o niskiej częstotliwości. Jak wspomnieliśmy wcześniej, klatki dla prędkości całkowitoliczbowych zawierały aliasing przestrzenny niwelowany przez uśrednianie w czasie. Dla innych prędkości bardzo często pojawiał się jednak efekt „pływających schodków", t.j. widocznego aliasingu przestrzennego przemieszczającego się wzdłuż krawędzi. Ruch ten wynika z fluktuacji o niskiej częstotliwości.

2. Flickering wysokiej częstotliwości. Gdy fluktuacje zwiększały częstotliwość, aliasing przestrzenny w tym miejscu przestawał być widoczny, jeśli jednak częstotliwość nie była dostatecznie wysoka (powyżej CFF), powstawał efekt migotania.

3. Statyczny aliasing przestrzenny. Nawet gdy nie występował żaden z dwóch poprzednich artefaktów, widoczny był aliasing przestrzenny („schodki" na krawędziach). Prawdopodobnie tego artefaktu można by uniknąć stosując wstępne filtrowanie obrazu.

Należy jednak odnotować, że testowane prędkości są problematyczne również dla tradycyjnego resamplingu. Migotanie występowało (aczkolwiek w trochę mniejszym stopniu) nawet gdy stosowaliśmy okienko Lanczosa o promieniu 6. Żeby pozbyć się flickeringu trzeba by zwiększyć promień filtru, co w jeszcze większym stopniu zredukowałoby ostrość obrazu.

Zbadaliśmy również 30 prędkości postaci (x, x) dla $x \in (3, 6]$. Duża część tych animacji nie zawierała flickeringu, jednak efekt wyostrzenia, zwłaszcza dla większych prędkości, był dyskusyjny. Przy takich prędkościach wciąż można zaobserwować pewną różnicę, jednak według naszej oceny nie jest ona już tak oczywista. Przede wszystkim proces optymalizacji rozmywał pojedyncze klatki, co ilustruje rys. 6.

Ponadto, wśród potencjalnych przyczyn wskazać można fakt, że wraz ze wzrostem prędkości nasila się efekt rozmycia (*motion blur*, patrz część 1), dodatkowo wyniki eksperymentalne [Laird i in., 2006] sugerują spadek możliwości układu wzrokowego w zakresie precyzyjnego ruchu śledzącego.

3.2. Animacje ze znaną funkcją przepływu optycznego

Uogólnienie metody Didyka i in. na animacje wymaga wprowadzenia pojęcia przepływu optycznego (ang. *optical flow*). Przez obraz rozumiemy dwuwymiarowy układ wzorów świetlnych będący odwzorowaniem różnic jasności obserwowanej sceny. Jak podają Horn i Schunck [1981], przepływ optyczny jest to rozkład postrzeganych prędkości wzorów w obrazie i może być wynikiem ruchu obiektów w scenie względem obserwatora. Jednakże związek między ruchem obiektów a przepływem optycznym niekoniecznie jest oczywisty. Przykładowo, kula o jednorodnej barwie obracająca się wokół własnej osi nie tworzy poruszających się wzorów. Z drugiej strony zmiany oświetlenia statycznej sceny mogą powodować ruch refleksów świetlnych na powierzchni obiektów. Niemniej dla wygody ograniczymy się do przypadków, w których przepływ optyczny może być wprost utożsamiony z ruchem powierzchni obiektów.

Animacje trójwymiarowe są cyfrowymi obrazami wirtualnej, zmiennej w czasie trójwymiarowej sceny. W pewnym uproszczeniu, każdy piksel ustalonej klatki jest obrazem punktu przestrzeni wirtualnej, znajdującego się na powierzchni jakiegoś obiektu, podlegającemu przekształceniom, takim jak ruch czy deformacja. Położenie tego punktu możemy śledzić w przestrzeni wirtualnej, a zatem również jego rzut na powierzchnię ekranu w kolejnych klatkach. Możliwe jest więc zdefiniowanie funkcji przepływu optycznego $f\colon N^3 \to R^2$, której argumentami są współrzędne (x, y) piksela ekranu oraz numer klatki i, a wartością wektor przesunięcia $(\Delta x, \Delta y)$ na powierzchni ekranu. Funkcję f otrzymać można w łatwy sposób podczas generowania animacji, gdyż jest ona elementem opisu sceny lub daje się z niego łatwo wyznaczyć.

Śledzenie punktów

Stoimy teraz przed następującym problemem: dla danej animacji o prędkości 120 kl/s wraz z przepływem optycznym wygenerować animację o trzykrotnie mniejszej rozdzielczości, zachowując przy tym jak największą rozdzielczość postrzeganą. W przypadku statycznych obrazów wszystkie punkty przemieszczały się w ten sam, dobrze zdefiniowany sposób, jednak teraz nie dysponujemy tak dokładną informacją. W każdej klatce pojawia się nowy zestaw punktów o znanej prędkości zastępując poprzedni. Pojawia się zatem pytanie, co tak naprawdę jest śledzone i jak przebiegają ścieżki receptorów na ekranie. Żeby zaradzić tej trudności, zastosujemy uproszczoną strategię: zamiast wyznaczać długie ścieżki, skupimy się na lokalnej optymalizacji krótkich ścieżek, które niekoniecznie łączą się ze sobą. Dla każdej klatki wprowadzać będziemy nowy zbiór wirtualnych receptorów, poruszających się ruchem prostoliniowym o długości życia trzech klatek:

$$r^i_{x,y} = \int_0^3 I(p^i_{x,y}(t), i + t)\,\mathrm{d}t. \tag{7}$$

Ścieżka $p^i_{x,y}$ jest odcinkiem wyznaczonym przez przepływ optyczny:

$$p^i_{x,y}(t) = (x, y) + t \cdot f(x, y, i). \tag{8}$$

Innymi słowy zakładamy, że ścieżki receptorów są lokalnie (w czasie i przestrzeni) stałe. Powyższe wzory podobnie jak poprzednio definiują układ równań liniowych, z pikselami animacji niskiej rozdzielczości jako niewiadomymi, do którego stosujemy tę samą metodę optymalizacji. Zauważmy, że dla całkowitoliczbowego ruchu statycznego obrazu rozważanego przez Didyka i in. mamy $f \equiv (k_1, k_2)$ i problem redukuje się do tego samego co poprzednio układu równań.

Adaptacja pseudokodu przedstawionego w punkcie 3.1 do przypadku ogólnej animacji wymaga jedynie drobnych zmian – musimy zmodyfikować funkcje receptor_to_pixels i pixel_to_receptors, tak aby uwzględniały funkcję przepływu optycznego oraz wprowadzone do modelu receptory wirtualne:

```
input
    off ▷ współrzędne receptora
    rec ▷ numer receptora
    num ▷ numer klatki
output
    pixels ▷ lista trójek zawierająca współrzędne pikseli wraz z wagami
fun pixels = receptor_to_pixels(off, rec, num)
    vel = optical_flow(off, rec)
    t = num − rec
    pixels = generate_mask(off + t · vel, vel)
endfun
```

```
input
    off ▷ współrzędne piksela
    rec ▷ numer receptora
    num ▷ numer klatki
output
    receptors ▷ lista trójek zawierająca współrzędne receptorów wraz z wagami
fun receptors = pixel_to_receptors(off, rec, num)
    receptors = {(x, y, w) | (off.x, off.y, w) ∈ receptor_to_pixels((x, y), rec, num)}
endfun
```

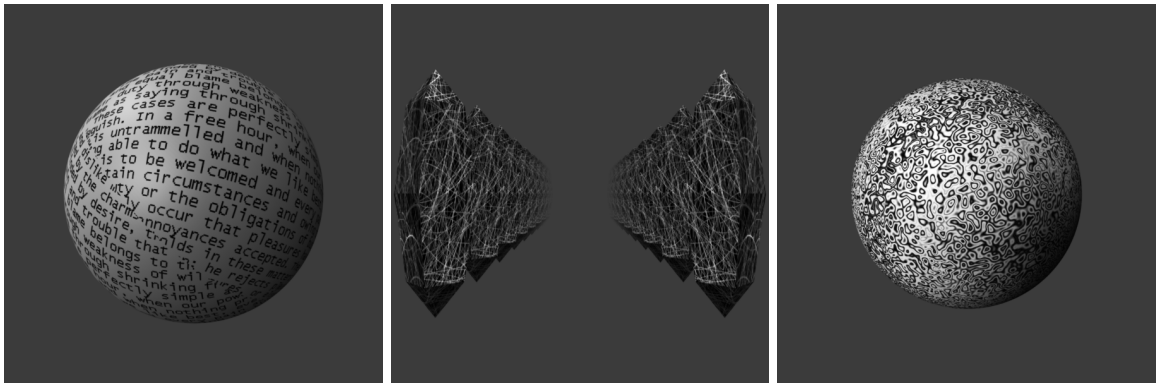
oraz uaktualnić ich wywołania w głównej procedurze:

```
fun subframes = optimize(image, factor, vel, len, iterations)
    ...
    receptor_to_pixels((x, y), i, i + s)
    ...
    pixel_to_receptors((sx, sy), i, i + s)
    ...
endfun
```

Ze względu na efektywność obliczeń w naszej implementacji z optymalizacji wyłączyliśmy receptory, dla których funkcja przepływu optycznego była większa niż 6 w kierunku którejś z osi. W takich przypadkach zakładaliśmy, że f jest równa $(0, 0)$, przyjmując, że znacząca poprawa ostrości w tym miejscu animacji nie jest możliwa.

Eksperymenty

W celu przetestowania naszej metody wygenerowaliśmy w programie Blender 2.49 pięć animacji prostych obiektów wraz z przepływem optycznym i porównaliśmy wynik optymalizacji klatek z filtrami Lanczosa o promieniach 3, 4, 5 oraz 6. Do zapisu użyliśmy formatu EXR, który umożliwia zapis kanałów jako liczb zmiennoprzecinkowych ze znakiem. Kanał czerwony przechowywał jasność, natomiast kanały zielony i niebieski posłużyły do zapisu funkcji przepływu optycznego (w Blenderze wartość *vector render layer*).



7.1 7.2 7.3

Rysunek 7: Obiekty użyte w testach 1–4.

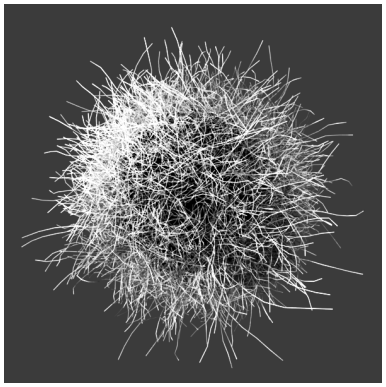
Pierwsza animacja przedstawiała obracającą się kulę z nałożonym tekstem jako teksturą (rys. 7.1). W tym, jak i w następnych testach zawierających obracający się obiekt oś obrotu była równoległa do przekątnej obrazu. Nasza implementacja dała wyniki lepsze od filtrowania z promieniem 3 (lekko widoczny aliasing) oraz 5, 6 (obraz mniej ostry). Filtrowanie z promieniem 4 dało porównywalny wynik. Warto odnotować, że dla obu metod litery na poruszającej się kuli wydawały się grubsze w porównaniu z pojedynczą klatką animacji, co nabierze pewnego znaczenia w teście 3.

Druga animacja przedstawiała dwa szeregi brył oteksturowanych bitmapą używaną do testowania ruchu statycznego obrazu (rys. 7.2). Obserwator przemieszczał się pomiędzy szeregami i równolegle do nich. Wektory prędkości skierowane były od środka obrazu na zewnątrz, a ich długość zwiększała się wraz z odległością od środka. W tym przypadku nasza implementacja dała lepsze wyniki od wszystkich sprawdzonych filtrów Lanczosa. Większy rozmiar filtru powodował rozmycie obrazu na jego brzegach, mniejszy natomiast skutkował aliasingiem w centrum obrazu. Aby uzyskać porównywalny efekt należałoby użyć filtrowania o zmiennym promieniu, zależnym od prędkości w danym miejscu obrazu. Warto zwrócić uwagę na fakt, że taki typ ruchu w obrazie, tj. wynikający z przemieszczania się kamery w nie-

ruchomym otoczeniu, ma spore znaczenie praktyczne w wielu dziedzinach, takich jak kinematografia, wizualizacje architektoniczne czy gry komputerowe.

Trzecia animacja przedstawiała tę samą co w pierwszym teście kulę (rys. 7.1), jednak tym razem ruch polegał na naprzemiennym zwiększaniu i zmniejszaniu obiektu. Zoptymalizowana animacja wykazywała interesującą właśność: w momencie gdy wektory prędkości zerowały się (tj. gdy kula osiągała minimalny bądź maksymalny rozmiar) tekstura traciła odrobinę kontrastu. Związane jest to ze wspomnianym przy okazji pierwszego testu efektem pogrubienia liter podczas animacji. Aby wyrównać jasność liter należałoby w jakiś sposób zróżnicować działanie algorytmu ze względu na obecność ruchu. Filtr o promieniu 6 dawał obraz bardziej rozmyty, mniejsze promienie zwiększały ilość aliasingu. W tym teście wedle naszej oceny, nie można przesądzić, która metoda dała lepsze wyniki.

W czwartej animacji badanym obiektem była znów obracająca się kula, jednak ze zmienioną teksturą (rys. 7.3). Ponownie filtry o promieniach 5 i 6 dały obraz bardziej rozmyty, promień 4 dał obraz porównywalny, a przy promieniu 3 zaobserwować można było zwiększony aliasing.



Rysunek 8: Obiekt użyty w teście 5.

W ostatnim teście obracaliśmy obiekt zilustrowany na rys. 8. Dla rejonów w centrum obrazu nasze obserwacje są analogiczne do tych w teście 1 i 4: filtrowanie Lanczosa o promieniu 4 dało wyniki porównywalne do optymalizacji. Jednak dla włókien bliżej brzegów występował aliasing, który wymagał zwiększenia promienia filtru. Jednocześnie w tych samych rejonach dla zoptymalizowanych klatek zaobserwować można było sporych rozmiarów czarne obwódki wokół niektórych włókien. Artefakt ten wynika prawdopodobnie ze znacznych różnic prędkości – cienkie włókna o dużej prędkości przemieszczają się na statycznym tle. Podobnie jak w teście 3 ciężko jest wskazać zwycięzcę, jednak zoptymalizowane klatki dały lepszy efekt na większym obszarze niż którykolwiek z filtrów.

Przeprowadzone testy wskazują, że zaimplementowana optymalizacja dla animacji może dać wyniki lepsze niż proste filtrowanie. Wedle naszej oceny, żeby uzyskać porównywalny efekt, należałoby uzależnić rozmiar filtru od rozkładu prędkości w obrazie. Jednocześnie nasz algorytm wymaga poprawek usuwających efekt zmiany ogólnego wyglądu odnotowany w teście 3, jak również artefakty widoczne w teście 5.

3.3. Animacje z nieznaną funkcją przepływu optycznego

Ostatnim rozważanym przez nas uogólnieniem metody był przypadek dowolnej animacji, dla której nie dysponujemy dokładnym przepływem optycznym. Pierwszym krokiem jest zatem jego estymacja. W tym celu zastosowaliśmy algorytm oparty na funkcjonale TV-L_1 zaproponowany przez Zacha i in. [2007]. Użyliśmy implementacji opublikowanej w ramach projektu GPU4Vision[2], zaadaptowanej do użycia w Plexusie (patrz rozdział 3.1).

Eksperymenty

Działanie naszego algorytmu w połączeniu z wyznaczaniem funkcji przepływu optycznego sprawdziliśmy na trzech testach. Pierwszy test zawierał tę samą co poprzednio obracającą się kulę z nałożonym tekstem, jednak tym razem przepływ optyczny był nieznany. Wygenerowane klatki na pierwszy rzut oka wydawały się nieodróżnialne. Dokładniejsze oględziny pozwoliły stwierdzić obecność bardzo małych artefaktów na krawędzi obracającego się obiektu, ale według nas pozostawały one bez wpływu na jakość wyniku.

9.1

9.2

Rysunek 9: Fragment animacji *Big Buck Bunny* (9.1, źródło: *http://www.bigbuckbunny.org/*) oraz testowa sekwencja video (9.2).

Następnie przetestowaliśmy nasz program na materiale zaczerpniętym z animacji *Big Buck Bunny*, której nieskompresowane klatki w rozdzielczości full-HD można pobrać z oficjalnej strony filmu[3]. Ponieważ film ten tworzony był z myślą o odtwarzaniu z prędkością 24 klatek na sekundę, do testów wybraliśmy fragment początkowej sceny (klatki 554–733), gdzie ruch kamery jest na tyle powolny, że pięciokrotne przyspieszenie wciąż pozwala na w miarę swobodną obserwację. Z klatek tych wyodrębniliśmy obszar w prawej części kadru o rozmiarze 900×900 (rys. 9.1). Filtry o promieniu 3 i 4 zawierały nieakceptowalną ilość aliasingu. Filtr o promieniu 6 byl mniej ostry niż zoptymalizowane klatki. Promień 5 pozwolił uzyskać podobny poziom szczegółowości, jednak porównanie jest utrudnione ze względu na to, że filtr Lanczosa zwiększył kontrast sceny. Stąd prawdopodobnie wynika również

[2] *OFlib*. Institute for Computer Graphics and Vision (Graz University of Technology) *http://gpu4vision.icg.tugraz.at/*
[3] *Big Buck Bunny*. ©2008 Blender Foundation, *http://www.bigbuckbunny.org/*

większa widoczność aliasingu niż w zoptymalizowanych klatkach. Zaskakująco niska okazała się widoczność artefaktów związana z niedokładnościami algorytmu wyznaczania przepływu optycznego.

Ostatni test zawierał materiał nakręcony przez nas kamerą. Sekwencja ta zawierała przesuwający się po przekątnej obraz zadrukowanej kartki papieru (rys. 9.2). Ponieważ ruch wymuszony był ręcznie, nie był on płynny i widoczne były wstrząsy. Filtr o promieniu 3 zawierał znaczny aliasing. Dla promienia 4 aliasing był mniejszy, jednak wciąż większy niż w przypadku zoptymalizowanych klatek. Promień 6 dał obraz bardziej rozmyty. Promień 5 obraz porównywalnej jakości, być może z lekkim wskazaniem na korzyść naszego algorytmu. Ponownie zaskakuje niska zawartość artefaktów związanych z wyznaczaniem przepływu optycznego: w tym teście nie byliśmy w stanie wskazać jednoznacznie ani jednego.

4. Implementacja

Implementacja naszego algorytmu to pojedynczy plik C++ (plik mgr.cpp). Do kompilacji wymagane jest API OpenGL w wersji 2.0 z obsługą rozszerzenia framebuffer object oraz biblioteki freeglut 2.6.0.0, GLEW 1.5.7.0, DevIL 0.1.7.8. Używaliśmy kompilatora MinGW 3.4.2. Binaria biblioteki DevIL na potrzeby MinGW konwertowaliśmy przy użyciu narzędzia reimp z pakietu mingw-utils w wersji 0.3 (nie udało nam się poprawnie ich skonwertować przy użyciu wersji 0.4-1). Program używa *fragment shaderów* do gamma-korekcji oraz filtrowania Lanczosa.

Jedynym parametrem wywołania programu, jest względna ścieżka do katalogu zawierającego pojedynczy test. Struktura katalogu z testem jest następująca:

```
test/
    lanczos/
        3/
        4/
        5/
        6/
    originals/
    subs/
    config.txt
    0001.exr
    0002.exr
    0003.exr
    ...
```

Pliki EXR to klatki wysokiej rozdzielczości, w których kanał czerwony oznacza luminancję, natomiast kanały niebieski i zielony kodują funkcję przepływu optycznego. W katalogach lanczos, originals oraz subs zapisane zostaną odpowiednio obrazy przefiltrowane okienkiem Lanczosa (liczba w nazwie podkatalogu oznacza promień filtru), oryginalne kanały luminancji o wysokiej rozdzielczości oraz zoptymalizowane podklatki niskiej rozdzielczości. Kluczowym plikiem jest config.txt, w którym zawarta jest konfiguracja testu. Struktura tego pliku jest następująca: w pierwszej linii liczba klatek animacji oraz flaga (1 lub 0) czy optymalizujemy animację czy statyczny obraz. W tym drugim przypadku program zignoruje wszystkie pliki EXR poza pierwszym. Druga i trzecia linia kodują format nazewniczy plików EXR: odpowiednio długość nazwy oraz numer pierwszej klatki. Linie czwarta i piąta to odpowiednio szerokość i wysokość pojedynczej klatki. Linia szósta zawiera dwie liczby rzeczywiste, oznaczające prędkość statycznego obrazu wyrażoną w pikselach wysokiej rozdzielczości na klatkę. Siódma linia zawiera współczynnik redukcji rozdzielczości (w naszych testach zawsze 3). Ósma linia oznacza liczbę iteracji optymalizacji. Linie 9-11 wskazują katalogi lanczos, originals oraz subs.

Na płycie zamieściliśmy również wygenerowane przez nas klatki dla testów z części 3.2 i 3.3 oraz program umożliwiający wyświetlenie i porównanie klatek (plik player.cpp). Oczywiście, użytkowanie tego programu ma sens tylko w przypadku monitora o częstotliwości odświeżania 120 Hz.

Czytelnik może wygenerować samodzielnie sceny testowe przy użyciu pliku scenes.blend.

Podsumowanie

W niniejszej pracy omówiliśmy metodę zwiększania postrzeganej rozdzielczości poruszających się obrazów na wyświetlaczach ciekłokrystalicznych o wysokiej częstotliwości odświeżania. Natępnie zaproponowaliśmy uogólnienie metody na przypadek ruchu o dowolnej prędkości i animacji ze znaną bądź nieznaną funkcją przepływu optycznego oraz przedstawiliśmy wyniki przeprowadzonych badań eksperymentalnych, w których użyliśmy załączonej implementacji algorytmu.

Przeprowadzone eksperymenty pokazały, że w przypadku ruchu statycznego obrazu o dowolnej prędkości z zakresu $(0, 3]^2$ wyostrzenie obrazu jest możliwe, niemniej prawie zawsze odbywa się to kosztem pojawienia się aliasingu. Z drugiej strony niecałkowite prędkości są podatne na występowanie aliasingu również przy zastosowaniu tradycyjnych metod resamplingu. Wzraz ze wzrostem prędkości, metoda daje gorsze wyniki i dla prędkości $(6, 6)$ jej stosowanie jest już nieopłacalne. Podobne wnioski wyciągnąć można dla animacji z daną funkcją przepływu optycznego: nasza metoda powoduje wzrost postrzeganej rozdzielczości obrazu, ale zależnie od rozkładu prędkości w obrazie, w niektórych miejscach może pojawić się aliasing. Wskazaliśmy przykłady, dla których filtrowania Lanczosa (nawet gdy dopuścimy zbyt mały w skali pojedynczej klatki promień filtru) nie można uznać ze bezsprzecznie lepsze, gdyż zależnie od rozmiaru filtru uzyskany przy jego użyciu obraz miał więcej aliasingu lub był mniej ostry. Pokazaliśmy, że gdy przepływ optyczny animacji nie jest znany, wciąż można zastosować naszą metodę, używając algorytmu jego estymacji. W takich wypadkach interesująca jest dosyć niska czułość metody na niedokładności w oszacowaniu przepływu.

Jako kierunki dalszych badań wskazalibyśmy rozważenie możliwości jakie niesie z sobą użycie wyświetlaczy o częstotliwościach wyższych niż 120 Hz, jak również sprawdzenie stosowalności metody do wyświetlaczy CRT. Wydaje się również, że zastosowanie okulografii (ang. *eye tracking*) mogłoby stworzyć dalsze możliwości rozwoju metody.

Literatura

Allen, W. i Ulichney, R. (2005). Wobulation: Doubling the addressed resolution of projection displays. *Proceedings of the Symposium Digest of Technical Papers (SID)*, tom 47.4 *The Society for Information Display*, str. 1514–1517.

Ayoub, G. (2008). An organ of exquisite perfection. *Visual Transduction and Non-Visual Light Perception.*

Curcio, C. A., Sloan, K. R., Kalina, R. E. i Hendrickson, A. E. (1990). Human photoreceptor topography. *The Journal of Comparative Neurology*, 292(4):497–523.

Damera-Venkata, N. i Chang, N. L. (2009). Display supersampling. *ACM Trans. Graph.*, 28(1):9:1–9:19.

Didyk, P., Eisemann, E., Ritschel, T., Myszkowski, K. i Seidel, H.-P. (2010). Apparent display resolution enhancement for moving images. *ACM Transactions on Graphics (Proceedings SIGGRAPH 2010, Los Angeles)*, 29(3).

Felhorski, W. i Stanioch, W. (1973). *Kolorymetria trójchromatyczna.* WNT, Warszawa.

Fiok, A. (1991). *Podstawy ogólne.* Telewizja. WKŁ, Warszawa.

Horn, B. K. P. i Schunck, B. G. (1981). Determining optical flow. *Artifical Intelligence*, 17:185–203.

Kalloniatis, M. i Luu, C. (2009). Temporal resolution. *http://webvision.med.utah.edu/temporal.html.*

Krawczyk, G., Myszkowski, K. i Seidel, H.-P. (2007). Contrast restoration by adaptive countershading. *The European Association for Computer Graphics Annual Conference EUROGRAPHICS 2007*, tom 26 *Computer Graphics Forum.* Blackwell.

Laird, J., Rosen, M., Pelz, J., Montag, E. i Daly, S. (2006). Spatio-velocity CSF as a function of retinal velocity using unstabilized stimuli. *Human Vision and Electronic Imaging XI*, tom 6057 *SPIE Proceedings Series*, str. 32–43.

Mäkelä, P., Rovamo, J. i Whitaker, D. (1994). Effects of luminance and external temporal noise on flicker sensitivity as a function of stimulus size at various eccentricities. *Vision Research*, 34(15):1981–91.

Pan, H., Feng, X.-F. i Daly, S. (2005). LCD motion blur modeling and analysis. *Proc. ICIP 2005*, str. 21–24.

Ritschel, T., Ihrke, M., Frisvad, J. R., Coppens, J., Myszkowski, K. i Seidel, H.-P. (2009). Temporal glare: Real-time dynamic simulation of the scattering in the human eye. *The European Association for Computer Graphics 30th Annual Conference : EUROGRAPHICS 2009*, tom 28 *Computers Graphics Forum*, str. 183–192, Monachium, Niemcy.

Van Hateren, J. H. (2005). A cellular and molecular model of response kinetics and adaptation in primate cones and horizontal cells. *J. Vision*, 5(4):331–347.

Yoshida, A., Ihrke, M., Mantiuk, R. i Seidel, H.-P. (2008). Brightness of the glare illusion. *Proceedings of ACM Symposium on Applied Perception in Graphics and Visualization*, str. 83–90, Los Angeles, CA, USA.

Zach, C., Pock, T. i Bischof, H. (2007). A duality based approach for realtime TV-L_1 optical flow. *Pattern Recognition (Proc. DAGM)*, str. 214–223, Heidelberg, Niemcy.