Systemy multimedialne
Zaawansowana edycja obrazów
Object Removal by Exemplar-Based Inpainting

Paweł Szołtysek

Spis treści

1 Wstęp

Usuwanie dużych obiektów z grafik rastrowych jest trudnym zadaniem, i polega na zapełnieniu części obrazu w sposób taki, aby był on przyjazny oku podczas przeglądania grafiki. W literaturze istnieje kilka pomysłów i gotowych sposobów na to, jak sobie z tym problemem poradzić. Jednym z nich jest połączenie idei syntezy tekstur (polegającej na zapełnianiu omawianej części spreparowanymi teksturami - dla dużych fragmentów) oraz zamalowywaniu (dobieraniu odpowiednich kolorów - dla małych fragmentów.
W niniejszej pracy podejście to zostanie przeanalizowane, oraz poddane weryfikacji poprzez przetestowanie dostępnej w sieci www implementacji.

2 Sposób działania algorytmu

Wybraną przez nas część docelową Ω mamy zapełnić wykorzystując pozostałą część obrazka Φ. Dobieramy wielkość próbki Ψ (domyślnie 9×9), i od tego momentu cały proces zamalowywania dokonuje się automatycznie.
Zakładamy, że każdy piksel obrazka posiada wartość *koloru* i *prawdopodobieństwa* (opisuje na ile jesteśmy pewni w kolor z którym piksel został zamalowany).
Algorytm polega na iteracji trzech kroków, i kończy się, gdy wszystkie piksele zostaną zamalowane.

2.1 Obliczanie priorytetów ścieżek

Ten krok wyznacza główne ścieżki, które muszą być zachowane przy wypełnianiu (chodzi tu m.in. o główne krawędzie itp), a także odpowiada za kolejność zamalowywania. Priorytet każdej ścieżki jest obliczany za pomocą iloczynu prawdopodobieństwa (będącego sumą prawdopodobieństw pikseli w ścieżce na próbkę Ψ) oraz danych (znormalizowanej wielkości iloczynu wektorowego kierunków krawędzi optycznej i prostopadłej do obrysu Ω). Wielkość ta może być traktowana jako miara pewności koloru zależna od otoczenia zamalowywanego piksela - chodzi tu przede wszystkim o zamalowanie tych pikseli, które mają w sąsiedztwie wiele pikseli już zamalowanych, z faworyzowaniem tych, które były tworzone wcześniej. Dzięki temu, nie ma określonej z góry funkcji kolejności zapełniania obszaru.

2.2 Propagacja informacji o teksturach i strukturze obrazu

W tym kroku, najpierw znajdujemy ścieżkę, która ma największy priorytet (obliczony w poprzednim kroku). Następnie wypełniamy go źródłem z Φ, pobierając z niej najbardziej podobny fragment. Kopiowanie odbywa się bezpośrednio, piksel po pikselu.

2.3 Odświeżanie wartości prawdopodobieństwa

Ostatnim krokiem jest nadanie nowej wartości prawdopodobieństwa pikselom które zostały zamalowane. Otrzymują one po prostu wartość równą wartości źródła.

3 Przygotowanie grafik do testowania algorytmu

Przedstawiony algorytm wydaje się być stosunkowo prosty i matematycznie poprawny. Pewne obawy o jego czas pracy wzbudza potrzeba częstego dokonywania pełnego przeglądu grafiki w celu znalezienia najlepszego źródła. Dla każdej grafiki, czas pracy zostanie zmierzony i podany.
Testy zostały przeprowadzone na komputerze dysponującym procesorem AMD Athlon 64 x2 6000+, 4GB pamięci ram, do zapisu wykorzystywany był dysk Western Digital - Caviar ST3250820A. Testy dla małych plików były przeprowadzane szeregowo, dla dużych plików - równolegle. We wszystkich przypadkach, procesy algorytmu wykorzystywały jeden rdzeń procesora.
Działanie algorytmu zostało przetestowane dla ośmiu różnych grafik, w sumie dla piętnastu obrazów.

4 Przedstawienie otrzymanych wyników

Do przeprowadzenia eksperymentu została użyta implementacja problemu dostępna w [2]. Jedną z jej cech charakterystycznych jest fakt, iż dla każdej iteracji generuje ona plik bmp, co powoduje wysokie wykorzystanie dysku twardego. Ma to też znaczny wpływ na czas przeznaczony na sumę czasu trwania algorytmu.
Przy omawianiu każdego pliku będą zwykle przedstawiane trzy obrazki: oryginał, zaznaczony obszar, wynik algorytmu.

4.1 Boisko i człowiek

W pierwszym przykładzie rozpatrzę plik, który został zamieszczony przez autora implementacji jako plik testowy.
Jego główne cechy to prostota w kolorze (dominacja trzech kolorów), mała rozdzielczość pliku oraz ogólna niska jakość - wysoki poziom rozmycia, niski kontrast, niski poziom szczegółów.

Rysunek 1: Usuwanie postaci z jednorodnej grafiki.

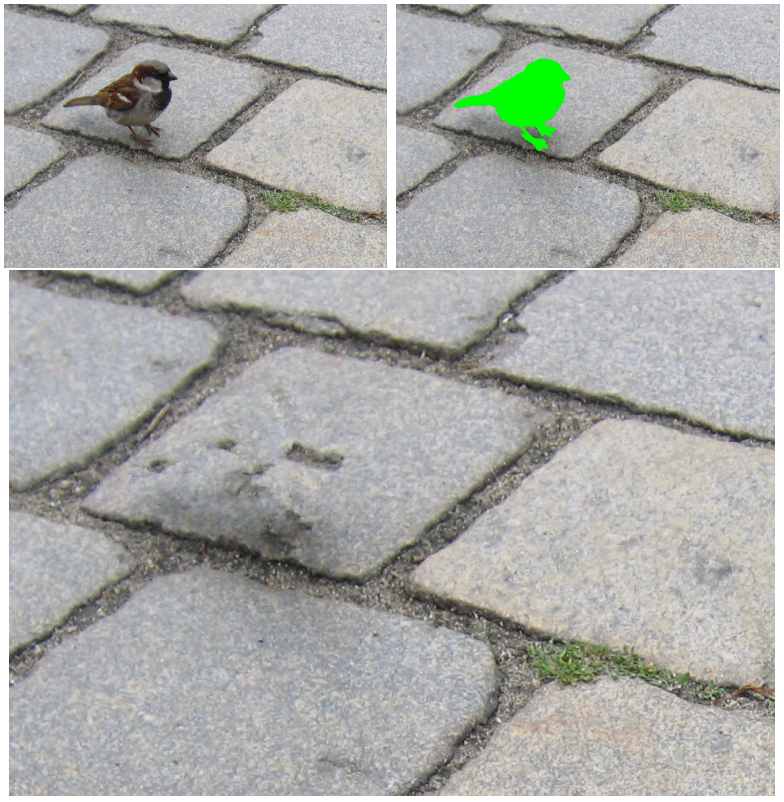
Wielkość pliku wejściowego: 275 678 B.
Rozdzielczość pliku: 350 × 262 px (91 700 px).
Wielkość wycinanego obrazu: 7 645 px.
Ratio (rozdzielczość do wycięcia): 12.
Ilość iteracji: 268.
Czas trwania algorytmu: 53 sekundy.
Średnia ilość iteracji: ok. 5 i/s.
Suma wielkości plików wyjściowych: 73 881 704 B.
Średnia wielkość przetwarzania [B/s]: ok. 1 393 994 B/s.
Średnia wielkość przetwarzania [px/s]: ok. 463 690 px/s.
Wykorzystany w tym przykładzie plik jest prawie jednorodny. Składa się w zasadzie trzech dominujących kolorów (cieniowany zielony, słoneczny zielony, błękitne niebo), więc spodziewany jest dobry wynik pracy tego algorytmu. I nie zawiedliśmy się - fotografia wynikowa wydaje się być naturalną.

4.2 Ptak na kostkach granitowych

Jako drugi zostanie przeanalizowane zdjęcie autorskie, które charakteryzuje się powtarzalną strukturą oraz elementem wyróżniającym się, które będziemy chcieli usunąć. Konkretnie, będzie to ułożona kostka brukowa, której fragment będzie zajmował wróbel.
Główne cechy to powtarzalność tekstury którą zapełniamy wyciętą część, podobny poziom kolorów na całej grafice oraz prosty do przewidzenia (dla człowieka) oczekiwany wygląd fotografii.

4.2.1 Fragment obrazka, nie uwzględnianie cienia.

Na początek przypatrzmy się na wynikom otrzymanym przy pracy na fragmencie obrazka, wycinając z niego tylko postać wróbla.



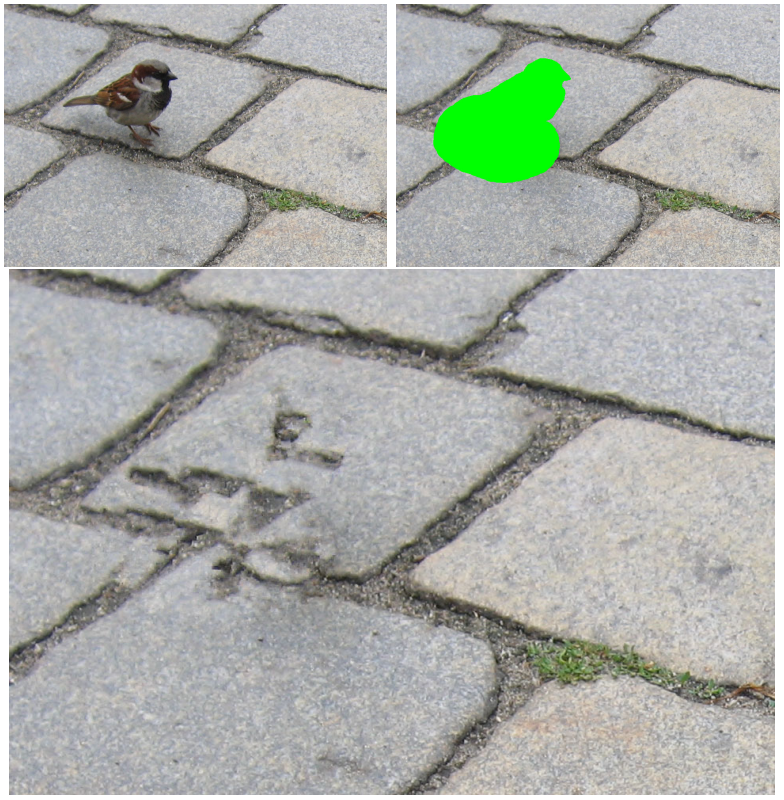
Rysunek 2: Usuwanie elementu z grafiki posiadającej cechy tekstury.

Wielkość pliku wejściowego: 1 928 312 B.
Rozdzielczość pliku: 968 × 664 px (642 752 px).
Wielkość wycinanego obrazu: 28 141 px.
Ratio (rozdzielczość do wycięcia): 23.
Ilość iteracji: 966.

Czas trwania algorytmu: 1 754 sekundy.
Średnia ilość iteracji: ok. 0.55 i/s.
Suma wielkości plików wyjściowych: 1 862 747 460 B.
Średnia wielkość przetwarzania [B/s]: ok. 1 061 999 B/s.
Średnia wielkość przetwarzania [px/s]: ok. 353 989 px/s.
Dla osoby, która nie widziała pliku wstępnego, wynikowa fotografia jest akceptowalna, a zniekształcenia w wypełnianium miejscu są uznawane bardziej za niedoskonałość kostki, niż miejsce wykonywania jakichś skomplikowanych przekształceń graficznych. Bardziej zastanawiający jest cień, który nie został poddany działaniu algorytmu. Chociaż tutaj interpretacja semantyczna może też być racjonalna (w tym miejscu stał samochód, z którego nastąpił wyciek oleju), wartym sprawdzenia jest także to miejsce z powodu naruszenia podstawowej faktury obrazka.

4.2.2 Fragment obrazka, uwzględnienie cienia.

Przedstawione poniżej obrazki pokazują (ten sam) plik źródłowy, nowe zaznaczenie uwzględniające już cień oraz plik wynikowy.



Rysunek 3: Usuwanie elementu z grafiki posiadającej cechy tekstury.

Wielkość pliku wejściowego: 1 928 312 B.

Rozdzielczość pliku: 968 × 664 px (642 752 px).
Wielkość wycinanego obrazu: 71 137 px.
Ratio (rozdzielczość do wycięcia): 9.
Ilość iteracji: 2 360.
Czas trwania algorytmu: 3 902 sekundy.
Średnia ilość iteracji: ok. 1.65 i/s.
Suma wielkości plików wyjściowych: 4 550 811 600 B.
Średnia wielkość przetwarzania [B/s]: ok. 1 166 276 B/s.
Średnia wielkość przetwarzania [px/s]: ok. 388 748 px/s.
Przy tej próbie widać, że obsługa priorytetów ścieżek nie jest do końca poprawnie rozpatrywana przez ten algorytm, gdy owe ścieżki są większe niż założone wielkości próbek. Lokalnie na krawędziach optycznych są one zachowane, natomiast im dalej od nich, występują znaczne deformacje wizualne.
W porównaniu do zaznaczenia rozpatrywanego w 4.2.1 należy zauważyć, że także w obrębie w którym zaznaczenia się pokrywają uzyskano różne wyniki. Wynika to z kolejności przeprowadzania zamalowywania - dokładnie ilustruje to załączone video.

4.2.3 Cały obrazek, nie uwzględnienie cienia.

Dla całego pliku źródłowego, algorytm raz jeszcze został wykonany, tym razem jednak obszar Φ był znacznie większy.
Wielkość pliku wejściowego: 15 116 600 B.
Rozdzielczość pliku: 2 592 × 1 944 px (5 038 848 px).
Wielkość wycinanego obrazu: 28 726 px.
Ratio (rozdzielczość do wycięcia): 175.
Ilość iteracji: 989.
Czas trwania algorytmu: 18 952 sekundy.
Średnia ilość iteracji: ok. 0.05 i/s.
Suma wielkości plików wyjściowych: 14 950 315 422 B.
Średnia wielkość przetwarzania [B/s]: ok. 788 851 B/s.
Średnia wielkość przetwarzania [px/s]: ok. 262 949 px/s.
Obrazek ma podobne cechy do obrazka uzyskanego w 4.2.1 - jest akceptowalny na takich samych zasadach. Z racji tego, że wycięty fragment zajmuje zaledwie pół procenta wielkości obrazka, uzyskany uszczerbek kostki brukowej wtapia się w ogólną wizję obrazka, dzięki czemu jeszcze trudniej powiedzieć, czy zdjęcie podlegało edycji tego typu.

4.2.4 Cały obrazek, uwzględnienie cienia.

W tym kroku ponownie zastosowano pełną wersję obrazka, zwiększono jednak obszar zaznaczenia, który teraz obejmuje cień wróbla.
Wielkość pliku wejściowego: 15 116 600 B.

Rysunek 4: Usuwanie elementu z grafiki posiadającej cechy tekstury.

Rozdzielczość pliku: 2 592 × 1 944 px (5 038 848 px).
Wielkość wycinanego obrazu: 78 570 px.
Ratio (rozdzielczość do wycięcia): 64.
Ilość iteracji: 2 626.
Czas trwania algorytmu: 47 133 sekundy.
Średnia ilość iteracji: ok. 0.05 i/s.
Suma wielkości plików wyjściowych: 39 696 186 348 B.
Średnia wielkość przetwarzania [B/s]: ok. 842 216 B/s.

Rysunek 5: Usuwanie elementu z grafiki posiadającej cechy tekstury.

Średnia wielkość przetwarzania [px/s]: ok. 280 737 px/s.
Takie podejście do tej grafiki okazało się najgorszym ze wszystkich czterech, chociaż zdrowy rozsądek podpowiadałby, że najlepiej przydzielić właśnie największy zbiór źródłowy.
Z całego rozpatrywanego przypadku widać, że algorytm nie poradził sobie nie tyle z usunięciem cienia, co z zachowaniem ścieżki krytycznej. Wynikło to ze... zbyt wysokiej rozdzielczości pliku źródłowego.

4.2.5 Cały obrazek w niskiej rozdzielczości, uwzględnienie cienia.

Kończąc ten przypadek, weźmiemy obrazek z przykładu 4.2.4 i zmniejszymy go dwunastokrotnie.

Rysunek 6: Usuwanie elementu z grafiki posiadającej cechy tekstury.

Wielkość pliku wejściowego: 90 056 B.
Rozdzielczość pliku: 200 × 150 px (30000 px).
Wielkość wycinanego obrazu: 528 px.
Ratio (rozdzielczość do wycięcia): 56.
Ilość iteracji: 22.
Czas trwania algorytmu: 2 sekundy.
Średnia ilość iteracji: ok. 11 i/s.
Suma wielkości plików wyjściowych: 1 981 188 B.
Średnia wielkość przetwarzania [B/s]: ok. 990 594 B/s.
Średnia wielkość przetwarzania [px/s]: ok. 330 000 px/s.
W tym przypadku algorytm poradził sobie prawie doskonale. Rozmiary próbki tym razem obejmowały całą ścieżkę krytyczną, co pozwoliło na poprawne przedstawienie treści znajdującej się pomiędzy kostkami brukowymi. Ten podprzykład potwierdza, że dla prezentowanego algorytmu niska rozdzielczość grafiki wejściowej pozwala na poprawną interpretację. Nie jest to zaskakujący wniosek.

4.3 Grupa osób w nocy

Zostanie teraz przedstawiony przykład, w którym z grupy czterech osób zostanie wycięta jedna. Zdjęcie to charakteryzuje się wysoką rozpiętością tonalną i kontrastem.
Wielkość pliku wejściowego: 1 276 856 B.
Rozdzielczość pliku: 800 × 533 px (426 400 px).
Wielkość wycinanego obrazu: 44 045 px.
Ratio (rozdzielczość do wycięcia): 9.
Ilość iteracji: 1 518.
Czas trwania algorytmu: 1 497 sekundy.
Średnia ilość iteracji: ok. 1 i/s.

Rysunek 7: Usuwanie postaci ze średniej jakości fotografii dynamicznej.

Suma wielkości plików wyjściowych: 1 938 267 408 B.
Średnia wielkość przetwarzania [B/s]: ok. 1 294 767 B/s.
Średnia wielkość przetwarzania [px/s]: ok. 432 381 px/s.
W tym przypadku, patrząc się na zdjęcie po algorytmie, na pierwszy rzut oka, wszystko wydaje się być w porządku. Na prawdę jednak po dokładnym przyjrzeniu się widać wiele niedoróbek i rzeczy kompletnie nie pasujących do sytuacji. Dla tego przypadku, także znaczne zmniejszenie rozdzielczości nie daje satysfakcjonujących wyników. Wynika to z faktu, że zdjęcie samo w sobie jest wymagające i brakuje poprawnego źródła (tła), którym brakująca postać mogłaby zostać efektywnie wypełniona (można nie przymierzając powiedzieć, że zajmuje ono w takim przypadku mniej niż jedną trzecią Ω, czyli zaledwie dwa razy więcej niż obszar, który chcemy wypełnić.

4.4 Zbędny element na fotografii

Przykład czwarty dotyczy zbędnego obiektu, który został uwieczniony na fotografii. Otoczenie tego obiektu jest w wysokim stopniu rozmyte, oczekujemy więc wysokiej jakości wypełnienia.
Wielkość pliku wejściowego: 1 280 054 B.
Rozdzielczość pliku: 800 × 533 px (426 400 px).
Wielkość wycinanego obrazu: 2 032 px.
Ratio (rozdzielczość do wycięcia): 210.
Ilość iteracji: 79.
Czas trwania algorytmu: 89 sekundy.
Średnia ilość iteracji: ok. 1 i/s.
Suma wielkości plików wyjściowych: 101 124 266 B.
Średnia wielkość przetwarzania [B/s]: ok. 1 136 227 B/s.
Średnia wielkość przetwarzania [px/s]: ok. 378 489 px/s.
Zdecydowana większość obrazka otrzymanego w wyniku działania algorytmu

Rysunek 8: Usuwanie elementu ze średniej jakości rozmytego otoczenia.

spełnia postawione oczekiwania. Można mieć uwagi odnośnie niepoprawnej analizy najbardziej wysuniętej na prawo części zamalowania, jednak jest możliwe proste poprawienie manualne tego fragmentu obrazu. Względnie, zaznaczenie nieco większego obszaru będzie owocowało także korzystymi wynikami.

4.5 Autostrada

Przykład piąty to fotografia autostady i próba wycięcia samochodów. Jest to więc próba wyjęcia z fotografii kilkunastu obiektów które są ze sobą rozłączne, których otoczenie różni się między sobą, i które zasłaniają różne elementy.

4.5.1 Średnia rozdzielczość grafiki

Najpierw weźmy pod uwagę fotografię w niższej rozdzielczości.
Wielkość pliku wejściowego: 1 279 254 B.
Rozdzielczość pliku: 800 × 533 px (426 400 px).
Wielkość wycinanego obrazu: 6 890 px.
Ratio (rozdzielczość do wycięcia): 62.
Ilość iteracji: 277.
Czas trwania algorytmu: 313 sekundy.
Średnia ilość iteracji: ok. 0.88 i/s.
Suma wielkości plików wyjściowych: 354 353 358 B.
Średnia wielkość przetwarzania [B/s]: ok. 1 132 119 B/s.
Średnia wielkość przetwarzania [px/s]: ok. 377 357 px/s.
Wyniki, które zostały otrzymane przechodzą nasze najśmielsze oczekiwania. Wprawdzie występuje kilka miejsc, o których można jednoznacznie powiedzieć, że znajdował się tam samochód przed wykonaniem algorytmu, jednak

Rysunek 9: Usuwanie wielu rozłącznych elementów z różnymi otoczeniami.

żadna z sondowanych osób nie przypuszczała, że na tej fotografii było aż 18 samochodów!
To, co w innych przykładach było słabością prezentowanego algorytmu, w tym przypadku okazuje się jego siłą - na horyzoncie pomimo prawie pełnego zasłonienia przez samochód ciężarowy instalacji na górze, została ona poprawnie odtworzona. Podobnie jak w przypadku 4.4, niedociągnięcia związane ze strukturą asfaltu oraz pomalowanych pasów można łatwo manualnie poprawić korzystając chociażby z narzędzia stempel w aplikacji Adobe Photoshop.

4.5.2 Wysoka rozdzielczość grafiki

Będąc pozytywnie zaskoczony tym, jak sobie w przypadku przedstawionym powyżej poradził algorytm, postanowiłem przebadać to samo zdjęcie w wyższej rozdzielczości.
Wielkość pliku wejściowego: 18 874 424 B.
Rozdzielczość pliku: 3 072 × 2 048 px (6 291 456 px).
Wielkość wycinanego obrazu: 23 116 px.
Ratio (rozdzielczość do wycięcia): 272.

Rysunek 10: Usuwanie wielu rozłącznych elementów z różnymi otoczeniami.

Ilość iteracji: 3 190.
Czas trwania algorytmu: 65 606 sekundy.
Średnia ilość iteracji: ok. 0.05 i/s.
Suma wielkości plików wyjściowych: 60 209 412 560 B.
Średnia wielkość przetwarzania [B/s]: ok. 917 742 B/s.
Średnia wielkość przetwarzania [px/s]: ok. 305 913 px/s.
Ponownie, dla wysokiej rozdzielczości obrazu algorytm poradził sobie zdecydowanie gorzej. Chociaż odwzorowanie części związanej ze światłami na horyzoncie jest akceptowalne, to artefakty które powstały wokół tego miejsca są nie do przyjęcia. Podobnie jest w innych miejscach, szczególnie na pierwszym planie.

4.6 Tekst (reklama) na fotografii

Kolejny przykład dotyczy tekstu naniesionego na fotografię przed (na przedmiot) lub po (na grafikę) wykonaniu zdjęcia. Został do tego celu wykorzystany samochód rajdowy, który ma w jednym kolorze karoserię oraz plandekę, a reklamy różnokolorowe.
Teksty te są również rozrzucone po całej grafice, jednak ich otoczenie jest

podobne.

Rysunek 11: Usuwanie tekstu.

Wielkość pliku wejściowego: 1 276 854 B.
Rozdzielczość pliku: 800 × 533 px (426 400 px).
Wielkość wycinanego obrazu: 6 041 px.
Ratio (rozdzielczość do wycięcia): 70.
Ilość iteracji: 286.
Czas trwania algorytmu: 345 sekundy.
Średnia ilość iteracji: ok. 0.82 i/s.
Suma wielkości plików wyjściowych: 365 180 244 B.
Średnia wielkość przetwarzania [B/s]: ok. 1 058 493 B/s.
Średnia wielkość przetwarzania [px/s]: ok. 353 479 px/s.
Wynik algorytmu jest zadowalający. Uwagi można mieć do przedniego spoilera, który został wypełniony niepoprawnie, ale jest ono akceptowalne. Z pozostałymi napisami, artefakty można zauważyć dopiero na trójkrotnym powiększeniu zdjęcia, a i ich forma wskazuje raczej na to, że raczej niedokładnie zostało wykonane zaznaczenie w niektórych miejscach, niż algorytm miałby pewny stopień nieskuteczności w takich przypadkach.

4.7 Usuwanie dwuelementowe

Ostatni przykład dotyczący manipulacji fotografii dotyczy fotografii której tło jest wyraźnie podzielone na dwie podobne struktury, a pierwszy plan dzieli się na dwa różne obiekty.

Rysunek 12: Usuwanie dwuelementowe.

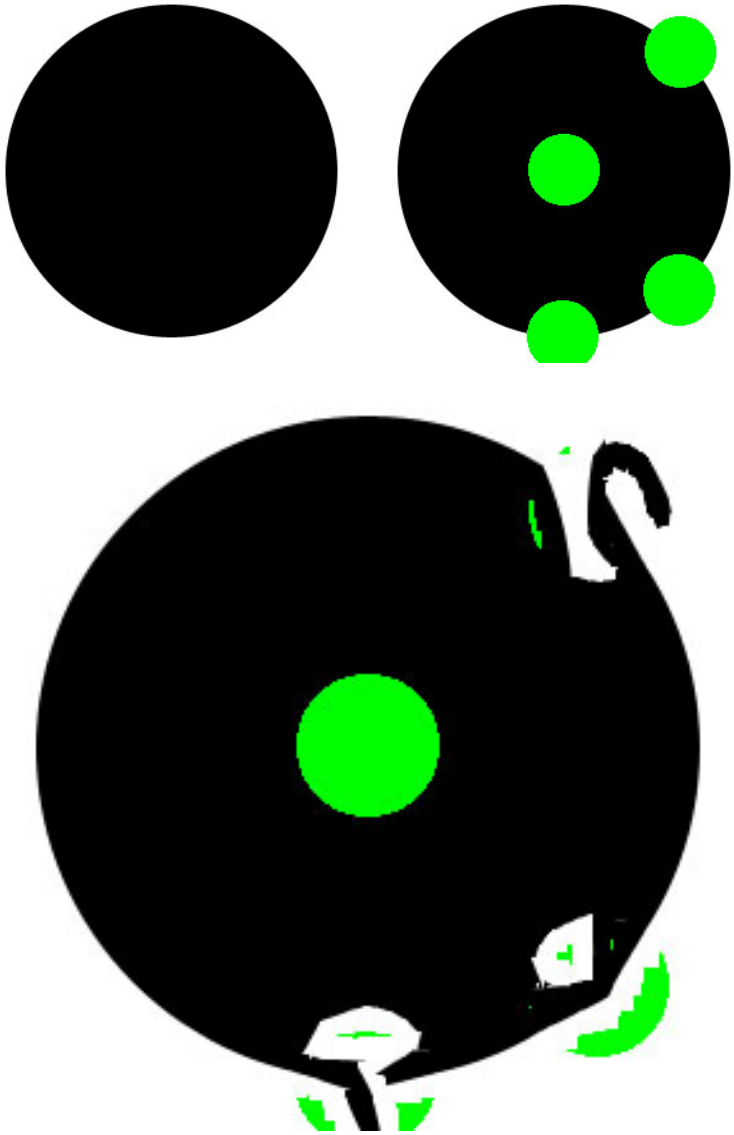
Ten przykład jest bardzo ciekawy, i jego analiza jest ważna przy chęci efektywnego wykorzystywania tego algorytmu w przyszłości. Usunięcie pojedynczego elementu pierwszego planu, połączone z pozostawieniem drugiego na fotografii oznacza pojawienie się zdecydowanych i bezpośrednio widocznych artefaktów. Dopiero usunięcie obu obiektów naraz pozwala na uzyskanie bardzo dobrego wyniku działania algorytmu, w który dopiero możemy techniką fotomontażu wkleić drugi obiekt - i w ten sposób uzyskać największą możliwą w tym przypadku jakość usuwania obiektów.
Wartym zauważenia jest fakt, iż wszystkie z osobna zamalowane szprychy potwierdzają tezę która się pojawiła przy przykładzie 4.6 - algorytm ten doskonale sobie radzi z zamalowywaniem małych fragmentów obrazu.

4.8 Figury geometryczne

Ten przykład dotyczy usuwania fragmentów figur geometrycznych. Pod uwagę zostało wzięte koło, w dwóch różnych przypadkach - samo, z wymaganymi do zamalowania fragmentami, oraz takie koło z idealnym kołem obok. Mamy pewną intuicję, która nam podpowiada, że algorytm zupełnie nie sprawdzi się w pierwszym przypadku, a w drugim da doskonałe wyniki, ale sprawdźmy to na przykładzie.

4.8.1 Pojedyncze koło

Ten przykład rozpatruje przypadek w którym mamy pojedyncze koło.



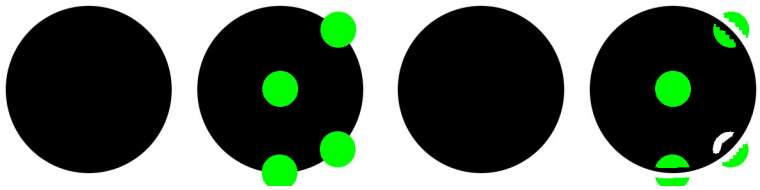
Rysunek 13: Figury geometryczne.

W zasadzie żaden komentarz do tych obrazków nie jest potrzebny. Zgodnie z przypuszczeniami, algorytm dobrał najbardziej pasujące fragmenty z pozostałych fragmentów koła i je po kolei wklejał. Z tego powoduy uzyskaliśmy w punktach na brzegach zdecydowane chropowatości.
Jednocześnie przy tym i następnym przypadku ukazała się niedoskonałość implementacji: jeśli zaznaczony obszar sąsiadował tylko z jednym kolorem,

to algorytm omijał taki przypadek i kontynuował prace - a gdy pozostały już tylko takie elementy, zapętlał się.

4.8.2 Dwa koła

Ten przykład rozpatruje przypadek w którym mamy koło idealne, oraz koło które wymaga wypełnienia.



Rysunek 14: Figury geometryczne.

I jeszcze raz, zgodnie z naszą intuicją - koło zostało idealnie odwzorowane, jako kopia lewego (ponownie - z dokładnością do precyzji implementacji).

5 Analiza efektów algorytmu

5.1 Ogólna analiza wyników.

Na początku zostaną przeanalizowane syntetycznie dane uzyskane podczas testowania algorytmu.
Najpierw przyglądnijmy się zestawieniu przetwarzania każdego przypadku.

Przykład	**Przetwarzanie** [B/s]	**Przetwarzanie** [px/s]
4.1	1 393 994	463 690
4.2.1	1 061 999	353 989
4.2.2	1 166 276	388 748
4.2.3	788 851	262 949
4.2.4	842 216	280 737
4.2.5	990 594	330 000
4.3	1 294 767	432 381
4.4	1 136 227	378 489
4.5.1	1 132 119	377 357
4.5.2	917 742	305 913
4.6	1 058 493	353 479

Tabela ta przedstawia wyraźnie, że tego samego typu grafiki były przetwarzane z podobną prędkością. Przykłady: 1, 2.5, były w niskiej rozdzielczości; 2.1, 2.2, 3, 4, 5.1, 6 w średniej; 2.3, 2.4, 5.2 w wysokiej - i właśnie w ten sposób rozkłada się wydajność wykorzystanej implementacji tego algorytmu. Jednocześnie należy zauważyć, że podczas zapisywania dysk nie stanowił

wąskiego gardła - według aplikacji HD Tune w wersji 2.55 osiąga on średnią prędkość zapisu rzędu 90 MB/s, co znacznie przekracza właśnie wyznaczone prędkości przetwarzania przez ten algorytm.

Poniższa tabela przedstawia zebrane dane na temat działania algorytmu.

Przykład	**Wielkość** [B]	**Rozdzielczość** [px]	**Ratio**	**Iteracji**	**Czas** [s]
4.1	275 678	91 700	12	268	53
4.2.1	1 928 312	642 752	23	966	1 754
4.2.2	1 928 312	642 752	9	2 360	3 902
4.2.3	15 116 600	5 038 848	175	989	18 952
4.2.4	15 116 600	5 038 848	64	2 626	47 133
4.2.5	90 056	30 000	56	22	2
4.3	1 276 856	426 400	9	1 518	1 497
4.4	1 280 054	426 400	210	79	89
4.5.1	1 279 254	426 400	62	277	313
4.5.2	18 874 424	6 291 456	272	3190	65 606
4.6	1 276 854	426 400	70	286	345

Wartości wielkości i rozdzielczości powtarzają się w niektórych przypadkach. Spowodowane jest to użyciem standardowych wielkości plików graficznych, głównie 800 × 533.
Czas pracy zgodnie z przypuszczeniami jest zależny od rozdzielczości pliku oraz od wielkości obszaru do zamalowania. W obrębie tych, ilość iteracji i czas trwania są wprost proporcjonalne do siebie.

5.2 Czas pracy algorytmu

Omawiany algorytm okazał się być stosunkowo wolny. Dla średniej wielkości grafiki i wyciętych zaledwie 2% obrazu algorytm pracował około 5 - 6 minut na wykorzystywanym komputerze, i czas ten rósł zdecydowanie dla zwiększającego się ratio (blisko czterokrotnie dla dwukrotnego zwiększenia obszaru do zamalowania). Związane jest to przede wszystkim z koniecznością wykonywania podwójnego przeglądu zupełnego grafiki w każdym kroku. Aby poprawić ten wynik, możnaby wykorzystać indeksowanie poszczególnych próbek (jeśli tylko byłyby w odpowiedni sposób podzielone), lub ich generalną klasteryzację ze względu na dominantę danej części grafiki. Umożliwienie pracy algorytmu wielowątkowo sprawiłoby też, że implementacja algorytmu pracowałaby znacznie wydajniej.

5.3 Efekt motyla

Niestety, w przedstawionej i zaimplementowanej formie algorytm okazał się być wrażliwy na *efekt motyla*. Widać to doskonale na przykładzie w rozdziale 4.2.4, który z ludzkiego punktu widzenia powinien zostać wykonany

wzorcowo - algorytm jednak zawodzi zdecydowanie. Spowodowane jest to wielkościami próbek które są porównywane do miejsc w które chcemy je zamalować. Takie próbki są dobierane ze względu na lokalne dopasowanie, a nie globalne istnienie ścieżek krytycznych (którymi są we wspomnianym przykładzie wypełnienia między kostkami brukowymi), w skutek czego niewielkie zawirowanie w dobranej i zamalowanej próbce przekładaja się na zdecydowane zmiany w dalszych iteracjach (nawet dalekich) algorytmu, powodując w zaprezentowanym przypadku całkowite zachwianie się ścieżek krytycznych.
Podsumowując, to, co miało być zbawienne dla małych grafik (tworzenie płynnych przejść między kolejnymi próbkami podczas zamalowywania, bazując na dostępnym obszarze Ω), okazało się zabójcze dla większych obrazów. Stało się tak poprzez niewłaściwą obsługę (a w zasadzie jej brak) tak wspomnianych ścieżek krytycznych, jak i tekstur, które są większe niż zdefiniowana próbka.

5.4 Predykcja przydatności procesu

Wspomniane czasy pracy algorytmu w bieżącej postaci są zdecydowanie zbyt duże, aby można było o nim myśleć jako o algorytmie do codziennego zastosowania. Byłyby one jednak akceptowalne, gdyby przed wykonaniem procesu zamalowywania można było otrzymać choćby przybliżony wygląd obrazu po przeprowadzeniu działania filtru. Jest to jednak niemożliwe przez fakt istnienia efektu motyla - w przypadku krytycznym, zmiana jednej wartości koloru jednego piksela o najmniejszą możliwą wartość może się przełożyć na całkiem inne pokolorowanie obszaru - nie wspominając już o zdecydowanie bardziej złożonymi mechanizmami redukcji rozdzielczości.
Co więcej (!), ponieważ w implementacji która była wykorzystana wielkość próbki była ustalana nadążnie, nie można określić przewidywanego czasu trwania konwersji - ani przybliżonej ilości iteracji - inaczej niż odgórnie, zakładając, że każdy piksel będziemy kolorowali osobno. (Chociaż w przeprowadzonych testach wielkości te okazały się zależne przede wszystkim od rozdzielczości, to zdecydowanie nie można na tej podstawie określać takiego czasu.) Dla dużej wielkości pliku, implementacja algorytmu była w stanie wykonać zaledwie 0.05 iteracji na sekundę, czyli 3 iteracje na minutę. Dla ratio wynoszącego 1000 (co oznacza, że ingerujemy tylko w 0.1% obrazu) oznacza to maksymalny czas równy prawie półtorej doby.

6 Podsumowanie

Algorytm, pomimo wadliwej implementacji oraz kilku niedoróbek technicznych, na pierwszy rzut oka generuje poprawne obrazki, które dopiero po przyjrzeniu się pokazują, że była przeprowadzana w nie ingerencja. Radzi

on sobie dobrze z małymi i prostymi obrazkami, jednak poprzez niewłaściwą obsługę ścieżek krytycznych i tekstur przy zwiększaniu rozdzielczości lub stopnia skomplikowania grafiki zdecydowanie spada prawdopodobieństwo poprawnego zamalowania wskazanego obszaru.
Na podobnej podstawie jak przedstawiony algorytm oparty jest pomysł pokazany w [3], który w zasadzie różni się od powyższego znacznym rozszerzeniem zbioru Ω o inne grafiki, dostępne za darmo w sieci internet.
Pomimo przedstawionych minusów, algorytm w przetestowanej postaci jest warty polecenia do automatycznego usuwania małych fragmentów grafik, lub tych, które są w niskiej rozdzielczości.

Literatura

[1] A.Criminisi, P. Pérez, K. Toyama. Object Removal by Exemplar-Based Inpainting. In Conf. Computer Vision and Pattern Recog, CVPR'03, Volume 2, Pages 721-728, Madison, WI, June 2003. (http://research.microsoft.com/vision/cambridge/papers/Criminisi_cvpr03.pdf)

[2] http://www.cc.gatech.edu/grads/q/qszhang/project/inpainting.htm

[3] http://news.bbc.co.uk/2/hi/technology/6936444.stm