Akademia Górniczo-Hutnicza w Krakowie

Wydział Fizyki i Informatyki Stosowanej

Bartosz Rogowski, II rok, 303793

Mateusz Barnacki, II rok, 303731

Wojciech Gomułka, II rok, 303747

Podstawy grafiki komputerowej

Dokumentacja projektu

Selektywna desaturacja obrazu

1. Opis projektu

Projekt dotyczy aplikacji, której zadaniem jest selektywna desaturacja obrazu w formacie JPG, który można wczytać na wejściu. Desaturacja ta to często spotykany efekt fotograficzny, mający na celu zamianę kolorów na czarnobiałe z pozostawieniem niektórych fragmentów w kolorze. Proces ten można zautomatyzować poprzez ustawienie wartości odpowiedniego kryterium. Użytkownik wybiera, powyżej których wartości należy zostawić dany parametr.

2. Założenia wstępne przyjęte podczas realizacji projektu

Program pozwala ustawić poziom saturacji – maksymalna wartość (100) odpowiada za zerową desaturację, minimalna (0) – maksymalną. Oprócz tego, użytkownik wybiera z listy parametr, który ma podlegać zmianie – i tak dla naszego projektu są to: Lightness, Hue, Red, Green, Blue, Cyan, Magenta, Yellow . Konieczne jest ustawienie także wartości (z zakresu 0 – 255 [1]), powyżej których ma zachodzić desaturacja. Dostępny jest także filtr gaussowski, który pozwala na rozmycie, co poprawia wtapianie się różnych fragmentów w obraz (z zakresu 0 – 30).

3. Analiza projektu

3.1. Specyfikacja danych wejściowych

Jedynym możliwym rodzajem danych wejściowych jest obraz w formacie JPG. Na obrazach o „żywych kolorach” najlepiej jest zaobserwować działanie programu (np. natura, owoce, itp.)

3.2. Definicja struktur danych

Oryginalny obraz oraz jego kopia znajdują się w zmiennych typu wxImage. Dostępna jest też seria parametrów typu int do przechowywania takich wielkości jak: próg desaturacji, wartość

[1] Wyjątek stanowi tu Hue, które może przyjmować wartość z zakresu 0 – 359.

rozmazania, procent saturacji czy zmienna odpowiadająca za wybrany parametr. Zmienna typu logicznego dostarcza informacji o tym, czy konieczne jest wykonanie funkcji refresh, a zmienna typu std::mutex zapobiega przed równoczesnym dostępem do wątków tablicy.

Oprócz metod klasy GUIMyFrame1, wiele niezbędnych funkcji zostało zaimplementowanych w oddzielnych plikach (General.h, General.cpp). Większość operacji wykonywana jest na tablicach.

3.3. Specyfikacja interfejsu użytkownika

Użytkownik ma do wyboru panel z doborem m.in. poziomu saturacji, parametru, którego wartość jest zmieniana za pomocą pola tekstowego, stopnia rozmycia, a także przycisku, którym akceptuje wprowadzone zmiany. Może wykorzystać również menu, w którym m.in. ma możliwość odczytu oraz zapisu obrazu, a także wyświetlenia krótkiej informacji o aplikacji.

3.4. Użyte narzędzia programistyczne

W projekcie postanowiliśmy użyć biblioteki *wxWidgets*, ze względu na przystępny interfejs oraz wachlarz dostępnych metod. Skorzystaliśmy także z biblioteki *CImg*, ponieważ zawiera niezwykle pomocne w naszym projekcie konwersje kolorów. Program został napisany w Visual Studio 2017.

4. Podział pracy i analiza czasowa

Po wspólnym omówieniu projektu, podzieliliśmy się zadaniami wedle umiejętności oraz zainteresowań tak, aby realizacja projektu przebiegła pomyślnie i sprawnie. Mimo, że staraliśmy się utrzymywać kontakt i wspólnie realizować projekt, wydzielone zostały pewne obowiązki dla danej osoby. Zestawione są one w poniższej tabeli.

Zagadnienie	Osoba odpowiedzialna	Analiza czasowa [w godzinach]
Omówienie projektu oraz zadań projektowych	wszyscy	$5-7$
Przygotowanie GUI (interfejsu użytkownika w programie *wxFormBuilder*) wraz z niektórymi metodami (wstępna walidacja oraz niektóre przyciski)	Bartosz	$4-5$
Research algorytmów	Mateusz	$2-3$
Implementacja głównego trzonu desaturacji oraz kilku niezbędnych metod	Mateusz	$11-12$
Implementacja filtrów	Mateusz	2
Poprawienie walidacji	Wojciech	2
Poprawa wydajności programu oraz kontrola algorytmów	Wojciech	6
Testowanie aplikacji	Bartosz, Wojciech	$2-3$
Przygotowanie dokumentacji	Bartosz	$6-7$
Pomoc w przygotowaniu dokumentacji (punkty 5 – 7)	Wojciech	4
Koordynacja zespołu oraz kontakt z Prowadzącym	Bartosz	1

Tabela 1. Analiza czasowa z podziałem obowiązków.

5. Opracowanie i opis niezbędnych algorytmów

W projekcie użyto filtru Gaussa z maską 5x5. Algorytm ten polega na przetworzeniu obrazu w ten sposób, by do obliczenia nowej wartości składowych dla danego piksela wykorzystać jego otoczenie.

Piksel, dla którego wykonywane są obliczenia musi znajdować się w środku takiej maski, niektóre piksele przy brzegach obrazu muszą zostać pominięte.

Aby obliczyć nową wartość składowych dla danego piksela (np. w systemie RGB), musimy dla każdej składowej policzyć sumę ważoną składowej piksela i pikseli sąsiadujących. Niech dana będzie maska 5x5 (macierz):

m(-2, -2)	m(-2, -1)	m(-2, 0)	m(-2, -2)	m(-2, 2)
m(-1, -2)	m(-1, -1)	m(-1, 0)	m(-1, -1)	m(-1, 2)
m(0, -2)	m(0, -1)	m(0, 0)	m(0, 0)	m(0, 2)
m(1, -2)	m(1, -1)	m(1, 0)	m(1, 1)	m(1, 2)
m(2, -2)	m(2, -1)	m(1, 0)	m(2, 2)	m(2, 2)

Tabela 2. Maska – macierz 5x5.

Wówczas sumę policzymy jako:

$$S = \sum_{i=-2,j=-2}^{i=2,j=2} \left[m_{i,j} * wartość\_składowej_{i,j} \right]. \quad (1)$$

Podzieliwszy tę sumę przez sumę wszystkich elementów macierzy, otrzymujemy nową wartość danej składowej.

Desaturacja polega na stopniowym „poszarzaniu” pikseli obrazu o wartości parametru (np. składowej RGB) poniżej pewnego przyjętego poziomu.

6. Kodowanie

Napisany przez nas kod programu stanowi siedem plików: General.h, GUI.h, GUIMyFrame1.h, General.cpp, GUI.cpp, GUIMyFrame1.cpp, main.cpp.

W pliku General.h znajdują się deklaracje funkcji pomocniczych zaimplementowanych na potrzeby projektu, służących głównie do obliczeń umożliwiających przeprowadzenie desaturacji. Metody te zostały zdefiniowane w pliku General.cpp.

Pliki GUI.h oraz GUI.cpp definiują klasę MyFrame1 dostarczającą GUI programu. Lwia część tych plików została wygenerowana przy użyciu *wxFormBuilder*.

Pliki GUIMyFrame1.h oraz GUIMyFrame1.cpp definiują klasę GUIMyFrame1 dziedziczącą po MyFrame1. Przechowywane w niej są piksele obrazu oryginalnego i zmodyfikowanego. Przesłonięte zostały również funkcje wirtualne z MyFrame1 reagujące np. na kliknięcie konkretnego przycisku czy wpisanie wartości.

Budowa GUI została w dużym stopniu zrealizowana przy pomocy programu *wxFormBuilder*. Umożliwiło to w miarę szybkie przygotowanie menu użytkownika i panelów na których

wyświetlane zostaną obrazy przy napisaniu minimalnej ilości kodu (większość zostanie wygenerowana przez *wxFormBuilder*).

Konieczne okazało się jednak reagowanie na zmieniający się rozmiar okna, gdy wczytano już obraz oryginalny (kopia zostaje wyświetlona w panelu po prawej stronie). Zostało to zrealizowane przy pomocy dwóch metod klasy GUIMyFrame1: firstRefresh i secondRefresh:

```
127  void GUIMyFrame1::firstRefresh(wxClientDC &dc)
128  {
129      wxBufferedDC buffer(&dc);
130      wxImage *temp;
131      temp = new wxImage(originalPhoto);
132      if (temp->IsOk()) {
133          temp->Rescale(dc.GetSize().x, dc.GetSize().y);
134          wxBitmap bitmap(*temp);
135          buffer.DrawBitmap(bitmap, 0, 0);
136      }
137      delete temp;
138  }
139
140  void GUIMyFrame1::secondRefresh(wxClientDC &dc)
141  {
142      wxBufferedDC buffer(&dc);
143      wxImage *temp;
144      temp = new wxImage(copyPhoto);
145      if (temp->IsOk()) {
146          temp->Rescale(dc.GetSize().x, dc.GetSize().y);
147          wxBitmap bitmap(*temp);
148          buffer.DrawBitmap(bitmap, 0, 0);
149      }
150      delete temp;
151  }
```

Rysunek 1. Implementacja metod firstRefresh i secondRefresh.

Jak widać do metod należy dostarczyć wxClientDC, które rysują nowe obrazy o zmienionych rozmiarach.

Oryginał oraz kopia zdjęcia są przechowywane od momentu, kiedy zostanie wczytany oryginał przy pomocy funkcji LoadFile:

```
void GUIMyFrame1::LoadFile(wxCommandEvent& event)
{
    wxClientDC dc(OriginalImage); // Tworzymy obiekt DC
    //Poniżej dodanie handlerów dla kopii i orginału
    originalPhoto.AddHandler(new wxJPEGHandler);
    copyPhoto.AddHandler(new wxJPEGHandler);
    //Wywołanie okna dialogowego
    std::shared_ptr<wxFileDialog> window(new wxFileDialog(this, _("Choose a file"), _(""), _(""), "JPEG files (*.jpg)|*.jpg", wxFD_OPEN));
    //Warunek czy okienko się poprawnie wyświetliło
    if (window->ShowModal() == wxID_OK) {
        //Jeżeli wystąpiły problemy z wywołaniem zdjęcia wyświetlamy error
        if (!originalPhoto.LoadFile(window->GetPath(), wxBITMAP_TYPE_JPEG)) {
            wxLogError(_("Can't open a file!")); //w pliku GUI.h jest specjalny include
        }
        else {
            OriginalImage->Enable(true); //zabezpieczenie przed zapisaniem pustego zdjęcia
            copyPhoto = originalPhoto; //Kopiujemy orginał
            should_refresh = true; //zmienna pomocnicza, określająca czy powinniśmy rozpocząć odświeżanie (umożliwiające zmianę wielkości zdjęcia przy rozciąganiu
            //okienka)
        }
    }
    firstRefresh(dc); //Odświeżamy panel z orginałem
}
```

Rysunek 2. Implementacja metody, która wczytuje plik do aplikacji.

Dodane zostają stosowne handlery, a samo działanie zostało skomentowane w kodzie.

Istnieje również możliwość zapisu zmodyfikowanego obrazu do pliku. Realizuje ją funkcja SaveFile, zapewniająca podstawową obsługę błędów:

```
90    //Zapisanie pliku
91    void GUIMyFrame1::SaveFile(wxCommandEvent& event)
92    {
93        //Poniżej sprawdzamy warunek, czy cokolwiek znajduje się w panelu zdjęcia wynikowego. Panel jest odblokowany jeżeli coś tam jest
94        if (ResultImage->IsEnabled()) {
95            //Ponownie tworzymy odpowiednie okno dialogowe
96            std::shared_ptr<wxFileDialog> window(new wxFileDialog(this, _("Save file"), _(""), _(""), "JPEG files (*.jpg)|*.jpg", wxFD_SAVE));
97            if (window->ShowModal() == wxID_OK) {
98                //Jak się nie uda zapisać wyskakuje error
99                if (!copyPhoto.SaveFile(window->GetPath(), wxBITMAP_TYPE_JPEG)) {
100                   wxLogError(_("Can't save an image!"));
101               }
102           }
103       }
104       //Wypisujemy error, że panel jest pusty
105       else {
106           wxLogError(_("Can't save an empty image!"));
107       }
108   }
```

Rysunek 3. Implementacja metody zapisującej obraz do pliku.

Podobnie jak przy odczytywaniu plików, ewentualne komentarze umieszczono w kodzie.

Funkcje reagujące na poszczególne zdarzenia zdefiniowane są MyFrame1.h, w pliku MyFrame1.cpp następuje zbindowanie ich do poszczególnych zdarzeń. Funkcje te zostają przesłonięte w klasie pochodnej GUIMyFrame1, która dostarcza ich szczegółową implementację. Nie ma sensu szczegółowo pochylać się nad większością z nich. Nadmieńmy jedynie, że metoda GUIMyFrame1::ResultImage_OnUpdateUI odpowiada m.in. za wywołanie funkcji GUIMyFrame1::value_check, która sprawdza czy w polu z wartością znajduje się odpowiednia liczba (przy uwzględnieniu co zostało wybrane z listy rozwijanej). Metoda value_check zapewnia podstawową obsługę błędów i reaguje stosownie na wadliwe wejście uniemożliwiając wciśnięcie przycisku zatwierdzającego zmiany.

Metoda jest zbyt długa by wkleić ją do tego opracowania w całości. Istotne jest, że konwertuje ona zawartość pola tekstowego do liczby całkowitej, o ile jest to możliwe. Dostarczono możliwość łapania wyjątków klas std::invalid_argument, std::out_of_range lub po prostu std::exception dla niezdefiniowanego wyjątku (w naszym programie jest to raczej nadmiarowe). Jeżeli uda się skonwertować zawartość pola tekstowego do stringa, to metoda sprawdza, czy z listy rozwijanej nie wybrano opcji („HUE”). Wówczas wartość powinna mieścić się w zakresie [0, 360), dla pozostałych parametrów powinna się ona mieścić w zakresie [0, 255].

Metoda reaguje stosownie zależnie od tego czy podane dane są poprawne. Jeżeli tak jest, to umożliwia kliknięcie przycisku powodującego zmiany w kopii obrazu. W przeciwnym wypadku przycisk staje się nieaktywny. Dodatkowo zostają wyświetlone jako wxStaticText pomocnicze informacje dla użytkownika, określające czy wejście jest poprawne lub co ewentualnie należy poprawić.

Reakcję na kliknięcie obsługuje metoda GUIMyFrame1::ApplyChanges_OnButtonClick:

```
41  void GUIMyFrame1::ApplyChanges_OnButtonClick(wxCommandEvent& event)
42  {
43      wxClientDC applyDc(ResultImage);
44      //Poniżej przypisujemy do zmiennych w klasie parametry z wszystkich pozostałych narzędzi
45      selectedOption = Choice->GetSelection();
46      blurParameter = SetBlur->GetValue();
47      tresholdParameter = wxAtoi(Value_Text->GetValue());
48      saturationParameter = SetSaturation->GetValue();
49      //Wywołanie funkcji modyfikującej obraz
50      modifyImage();
51      //Odświeżamy panel przeznaczony dla kopii obrazu
52      secondRefresh(applyDc);
```

Rysunek 4. Implementacja metody uruchamianej po naciśnięciu przycisku „Apply changes”.

Jak widać w ciele powyższej funkcji zostaje wywołana metoda GUIMyFrame1::modifyImage. To w niej koncentrują się operacje wykonywane na kopii obrazu.

Metoda ta sprawdza najpierw według którego z parametrów przeprowadzamy desaturację.

```
298                 //Przypisujemy wyznaczoną wartość dla odpowiedniej opcji
299                 if (selectedOption == 0)
300                     temp = calculateLighteness(image[iter], image[iter + 1], image[iter + 2]);
301                 else if (selectedOption == 1)
302                     temp = calculateHue(image[iter], image[iter + 1], image[iter + 2]);
303                 else if (selectedOption == 2)
304                     temp = static_cast<int>(image[iter]);
305                 else if (selectedOption == 3)
306                     temp = static_cast<int>(image[iter + 1]);
307                 else if (selectedOption == 4)
308                     temp = static_cast<int>(image[iter + 2]);
```

Rysunek 5. Fragment metody GUIMyFrame1::modifyImage.

Implementacja w razie wyboru CMY, wymaga konwersji na RGB, co zostało zaimplementowane osobno. W razie wyboru opcji „lightness” lub „hue”, wywołane zostaną stosowne metody zdefiniowane w General.h. W przeciwnym wypadku za wartość zmiennej tymczasowej, co do której będziemy przeprowadzać porównanie, przyjmiemy wartość stosownej składowej z systemu RGB.

Następnie przy użyciu parametru temp i metod z pliku General.h dokonujemy desaturacji.

```
309                 //Sprawdzamy czy wartość jest mniejsza niż obszar 'przejścia'
310                 if (temp < (tresholdParameter - blurParameter)) {
311                     //Wyznaczamy max z RGB
312                     max = findMax(image[iter], image[iter + 1], image[iter + 2]);
313                     if (max < 0) {
314                         max = 0;
315                     }
316                     //Dokonujemy częściowej desaturacji zależnie od parametru saturationParameter
317                     partlyDesaturation(copy[iter], copy[iter + 1], copy[iter + 2], max, saturationParameter);
318                 }
319                 //Jeżeli piksel znajduje się w obszarze przejściowym
320                 else if ((temp > tresholdParameter - blurParameter) && (temp < tresholdParameter)) {
321                     delta = tresholdParameter - temp; // Obliczamy o ile różni się od progu
322                     max = findMax(image[iter], image[iter + 1], image[iter + 2]);
323                     //Funkcja która ma imitować rozmycie
324                     blurDesaturation(copy[iter], copy[iter + 1], copy[iter + 2], max, saturationParameter, blurParameter, delta);
325                 }
```

Rysunek 6. Fragment metody GUIMyFrame1::modifyImage.

Postępowanie dla systemu CMY jest podobne, skorzystano jednak z konwersji do RGB i odwrotnie, którą zapewniła biblioteka CImg.

W ramach realizacji dodatkowych wymagań do programu, podjęto próbę zastosowania filtru Gaussa. Został on bardziej szczegółowo opisany w punkcie 6. W naszym programie realizuje go metoda GUIMyFrame1::useGaussian2. Aby przyspieszyć działanie programu postarano się o skorzystanie z std::thread przy realizacji filtru Gaussa.

W pliku General.h zadeklarowano kilka pomocniczych metod zdefiniowanych w General.cpp. Większość z nich realizuje proste operacje (poszukiwanie minimalnej, maksymalnej składowej, odczytanie kąta na kole barw itp.). Znajdują się tam również metody odpowiedzialne za desaturację stanowiące podstawę działania algorytmu:

```
178  void partlyDesaturation(unsigned char &r, unsigned char &g, unsigned char &b, int &max, int &saturationParameter) {
179
180      double deltaR, deltaG, deltaB;
181      //Sprawdzamy różnicę między wartością maksymalną a danym pikselem
182      deltaR = static_cast<double>(max) - static_cast<double>(r);
183      deltaG = static_cast<double>(max) - static_cast<double>(g);
184      deltaB = static_cast<double>(max) - static_cast<double>(b);
185
186      //Mnożymy daną różnicę przez odpowiedni procent
187      deltaR = deltaR * (1.0 - (static_cast<double>(saturationParameter) / 100.0));
188      deltaG = deltaG * (1.0 - (static_cast<double>(saturationParameter) / 100.0));
189      deltaB = deltaB * (1.0 - (static_cast<double>(saturationParameter) / 100.0));
190
191      //Aktualizujemy przesłane wartości
192      r += static_cast<int>(deltaR);
193      g += static_cast<int>(deltaG);
194      b += static_cast<int>(deltaB);
195
196  }
```

Rysunek 7. Implementacja funkcji partlyDesaturation.

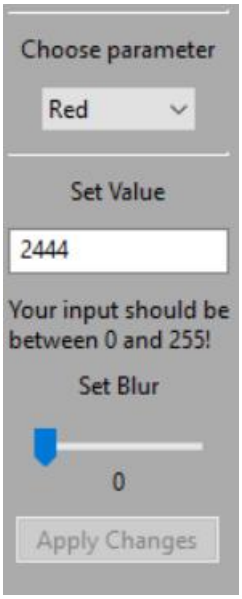
```
199  void blurDesaturation(unsigned char &r, unsigned char &g, unsigned char &b, int &max, int &saturationParameter, int &blurParameter, int &delta) {
200      double deltaR, deltaG, deltaB;
201      deltaR = static_cast<double>(max) - static_cast<double>(r);
202      deltaG = static_cast<double>(max) - static_cast<double>(g);
203      deltaB = static_cast<double>(max) - static_cast<double>(b);
204
205      deltaR = deltaR * (1.0 - (static_cast<double>(saturationParameter) / 100.0));
206      deltaG = deltaG * (1.0 - (static_cast<double>(saturationParameter) / 100.0));
207      deltaB = deltaB * (1.0 - (static_cast<double>(saturationParameter) / 100.0));
208
209      if(deltaR > 0.){
210          deltaR = deltaR * (1.0 - (static_cast<double>(delta)) / static_cast<double>(blurParameter));
211      }
212      if (deltaG > 0.) {
213          deltaG = deltaG * (1.0 - (static_cast<double>(delta)) / static_cast<double>(blurParameter));
214      }
215      if (deltaB > 0.) {
216          deltaB = deltaB * (1.0 - (static_cast<double>(delta)) / static_cast<double>(blurParameter));
217      }
218      r += static_cast<int>(deltaR);
219      g += static_cast<int>(deltaG);
220      b += static_cast<int>(deltaB);
221  }
```

Rysunek 8. Implementacja funkcji blurDesaturation.

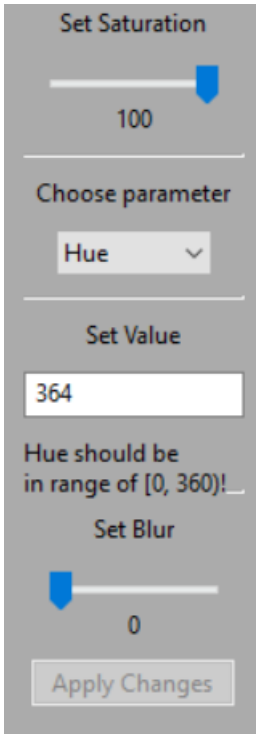
Obie funkcje przemnażają składowe w systemie RGB przez odpowiedni czynnik.

7. Testowanie

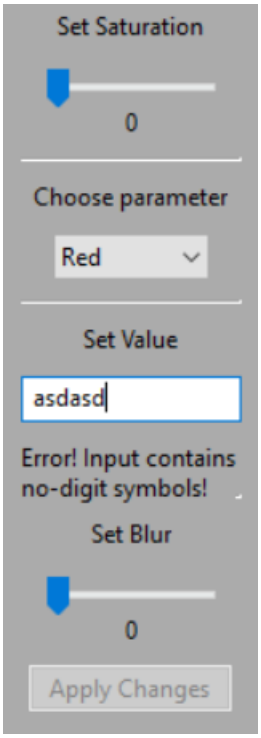
Nasze testy ograniczyły się do manualnego sprawdzania zachowania programu. Program przetestowano pod kątem zmiany wielkości okna, reakcji na nieprawidłowe dane w polu tekstowym, odczytu/zapisu plików i przede wszystkim modyfikacji dokonywanych na kopii obrazu. Poniżej przedstawiono serię przykładowych reakcji na „niepożądane wartości”.



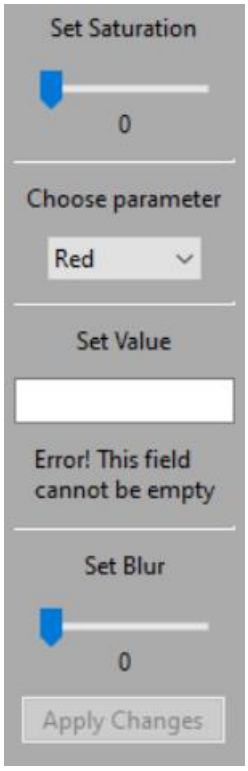
Rysunek 9. Reakcja na zbyt dużą wartość w polu Set Value.



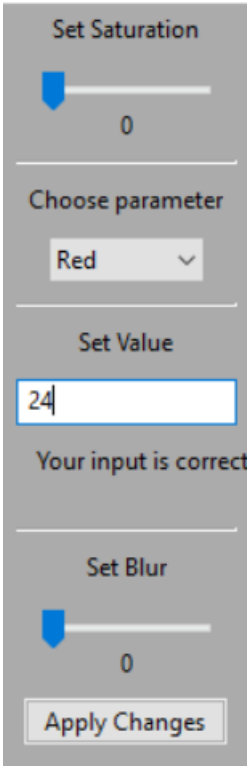
Rysunek 10. Reakcja na zbyt dużą wartość w polu Set Value w przypadku parametru Hue.



Rysunek 11. Reakcja na znaki niebędące cyframi w polu Set Value.



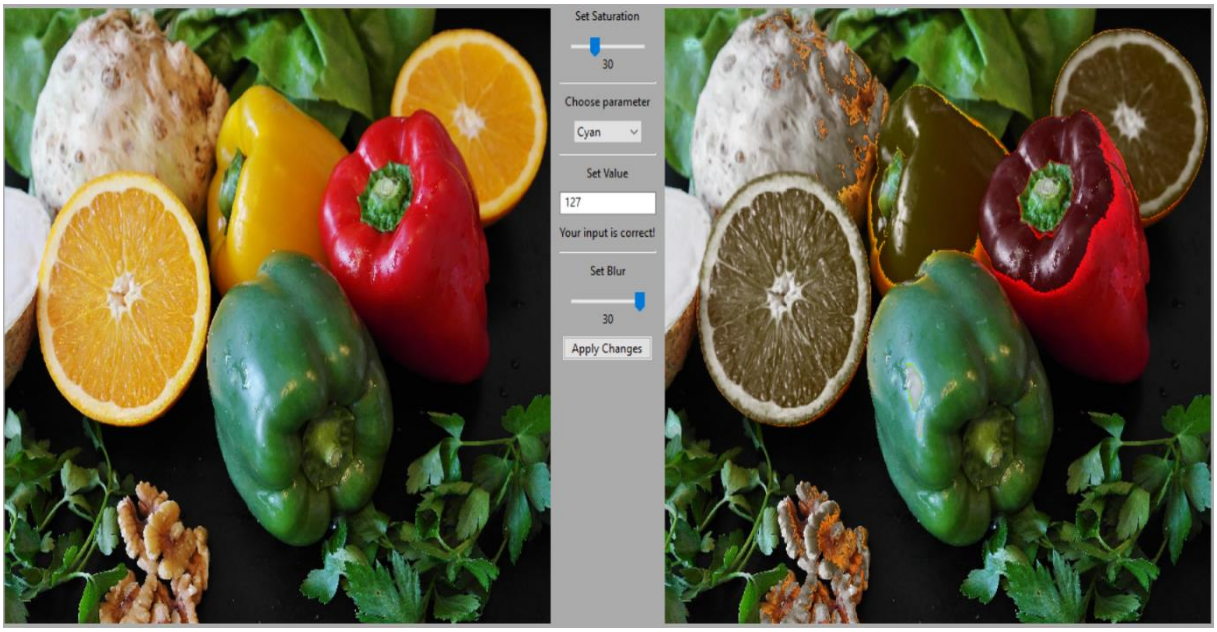
Rysunek 12. Reakcja na brak wartości w polu Set Value.



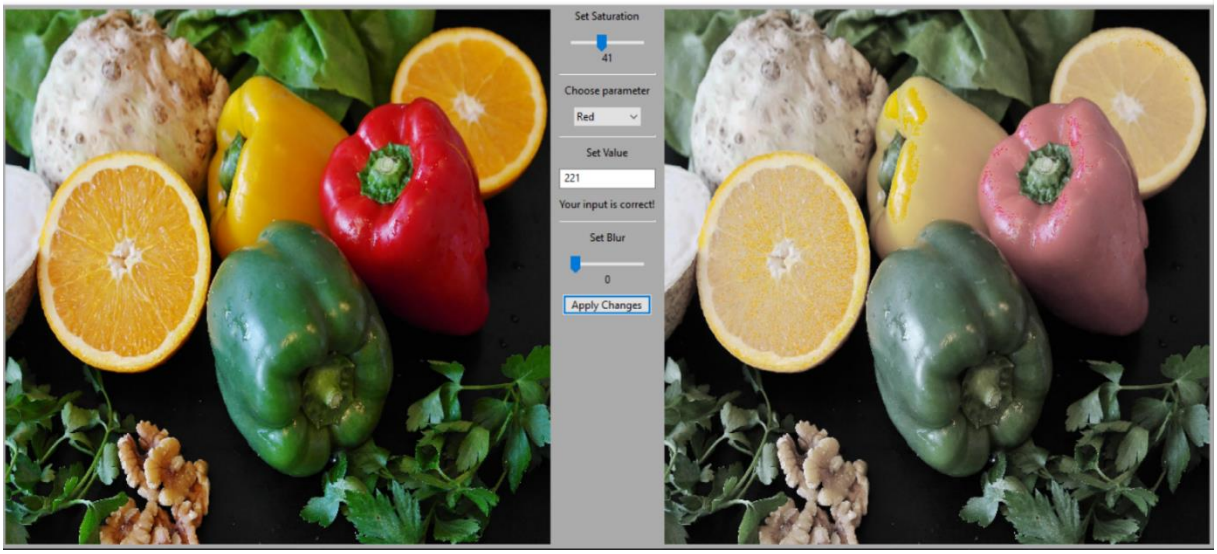
Rysunek 13. Reakcja na poprawną wartość w polu Set Value.

Warto zwrócić uwagę na wygląd przycisku Apply Changes.

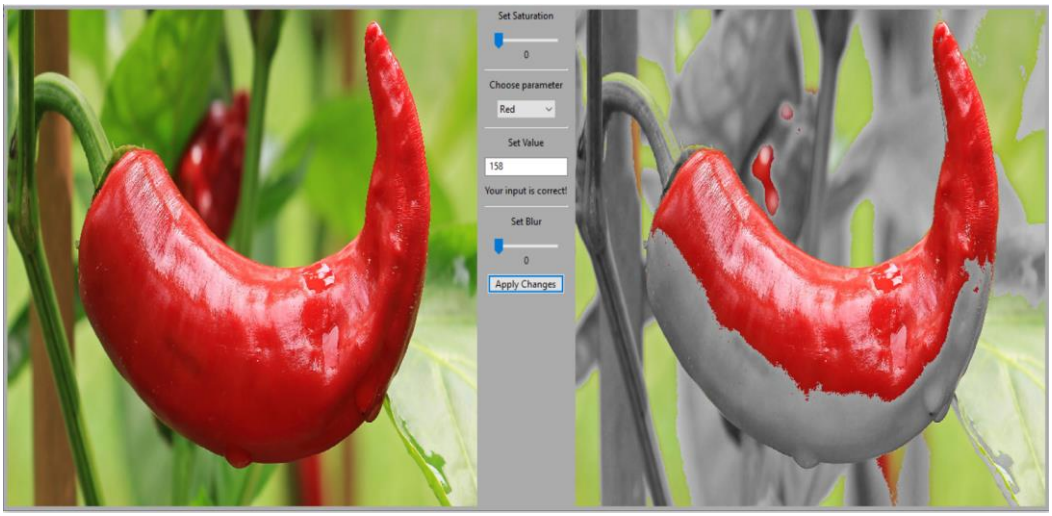
Przeprowadzono również działanie aplikacji, a właściwie jej sedna – częściowej desaturacji. Efekty na serii przykładowych zdjęć ukazano poniżej.



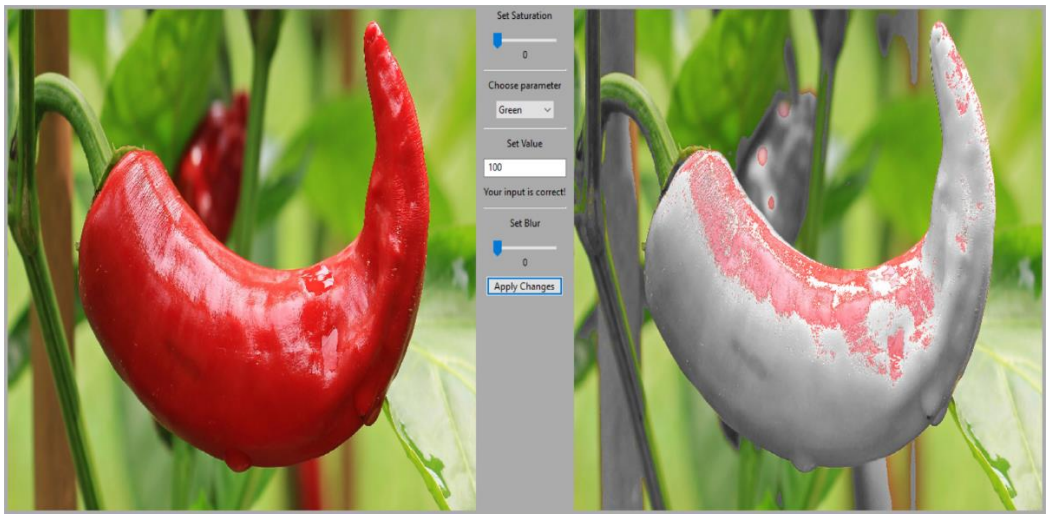
Rysunek 14. Test działania aplikacji (saturacja: 30, cyjan: 127, maksymalne rozmycie).



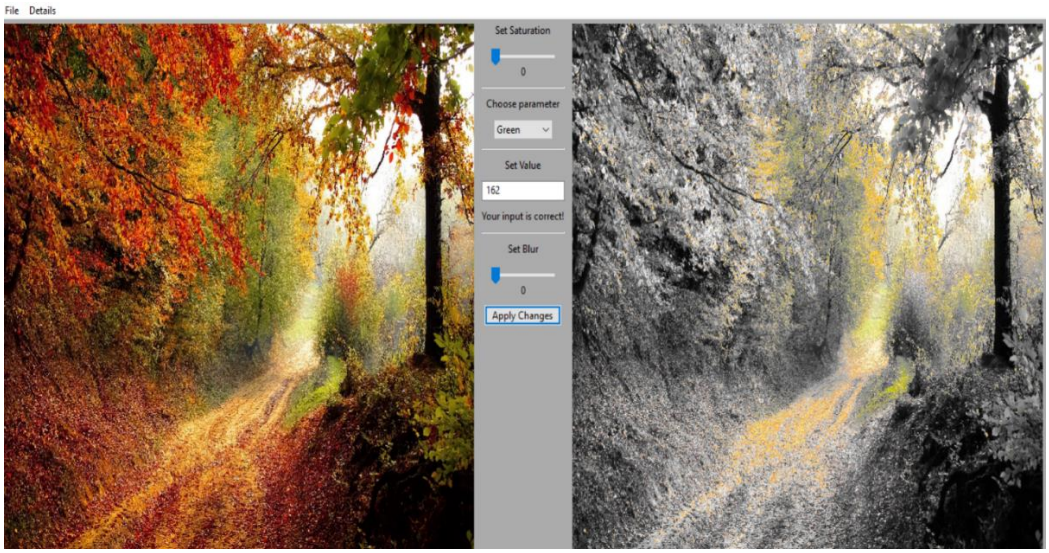
Rysunek 15. Test działania aplikacji (saturacja: 41, czerwony: 221, brak rozmycia).



Rysunek 16. Test działania aplikacji (saturacja: 0, czerwony: 158, brak rozmycia).



Rysunek 17. Test działania aplikacji (saturacja: 0, zielony: 100, brak rozmycia).



Rysunek 18. Test działania aplikacji (saturacja: 0, zielony: 162, brak rozmycia).

W czasie testowania programu zauważono zwiększenie się czasu dokonywania modyfikacji.

8. Wdrożenie, raport, wnioski

W projekcie – naszym zdaniem – udało się zrealizować wszystkie wymagania podstawowe zadania. Użytkownik może manipulować parametrami oraz ich wartościami tak, by dobrać odpowiednie wartości do obrazu, który może zostać zapisany do pliku JPG. Również część wymagań rozszerzonych została spełniona – dostępność składowych CMY oraz obecność filtru Gaussowskiego 5x5.

Wśród rzeczy, które mogłyby zostać poprawione, możemy wymienić m. in.: wydajność programu, płynność wtapiania, przejrzystość i optymalizację kodu.

9. Bibliografia i źródła

- http://www.algorytm.org/przetwarzanie-obrazow/filtrowanie-obrazow.html - opis filtru Gaussa
- http://www.fis.agh.edu.pl/~malinowski/GFK/files/46.pdf – treść zadania