

„Sortowanie obrazów”

I. Przedstawienie zadania

Celem projektu było napisanie programu sortującego obrazy według ich podobieństwa.

II. Opis programu - możliwości

a) Opis

- Program został napisany w języku C#, wymaga .NET Framework Version 2.0 i jest przeznaczony dla systemu operacyjnego Microsoft Windows.
- Zaimplementowane zostało 5 algorytmów porównywania obrazów zarówno dla tablicy pikseli tworzących bitmapę jak i dla histogramu, co łącznie daje 10 metod.
- Aplikacja posiada graficzny interfejs użytkownika (GUI)

b) Możliwości

- Wczytywanie obrazów z plików w formatach *.bmp, *.jpg
- Wczytanie kilku obrazów jednocześnie
- Dodawanie obrazów do aktualnie załadowanej listy
- Przerwanie porównywania (przez użytkownika) w przypadku zbyt długiego oczekiwania na wynik
- Jednoczesne porównywanie obrazów w obu obsługiwanych formatach
- Porównywanie obrazów bez względu na ich rozmiar
- Wybór jednej z 10 metod porównywania
- Podział obrazu na kilka części wzdłuż wybranej osi
- Podgląd wczytanych obrazów
- Pasek postępu
- Skalowanie obrazu podczas ładowania

III. Opis zaimplementowanych algorytmów

a) Maksimum (MAX)

Jest to najprostsza z metod, a co z tym idzie najmniej dokładna. Algorytm polega na znalezieniu różnicy pomiędzy największą i najmniejszą wartością koloru na danym obrazie i porównaniu jej z wartością uzyskaną dla drugiego obrazu.

$$M = \max_{(x,y)} (|a_R - b_R| + |a_G - b_G| + |a_B - b_B|)$$

b) Suma różnic bezwzględnych (SAD)

Metoda polega na zsumowaniu różnic wszystkich pikseli danego obrazu, a następnie podzieleniu uzyskanej liczby przez ilość pikseli. Różnice są liczone tak samo jak w metodzie Maksimum.

$$M = \frac{1}{N \cdot M} \sum_{(x,y)} (|a_R - b_R| + |a_G - b_G| + |a_B - b_B|)$$

c) Błąd średniokwadratowy (MSE)

MSE obliczane jest poprzez zsumowanie kwadratów różnic pikseli obrazu i podzieleniu jej przez liczbę pikseli.

$$M = \frac{1}{N \cdot M} \sum_{(x,y)} \left((a_R - b_R)^2 + (a_G - b_G)^2 + (a_B - b_B)^2 \right)$$

d) Znormalizowany błąd średniokwadratowy (NMSE)

Metoda jest dwufazowa, najpierw oba obrazy są normalizowane a następnie obliczany jest MSE. Normalizacja jest przeprowadzana osobno dla każdej składowej.

$$a'(x, y) = \frac{a(x, y)}{\mu_a} - 1$$

μ_a - średnia intensywność składowej obrazu

$$S_a' = \sqrt{\frac{\sum_{(x,y)} a'(x, y)^2}{N \cdot M}}$$

S_a' - odchylenie standardowe składowej obrazu a'

$$a''(x, y) = \frac{a'(x, y)}{S_a'}$$

a'' - obraz po normalizacji

e) Błąd średniokwadratowy-luminacja (MSE-Lum)

Metoda polega na upodobnieniu obrazu do krzywej odpowiedzi wzroku człowieka na bodźce. Algorytm składa się na dwa etapy:

- Obliczenie wartości luminacji na podstawie intensywności składowych RGB obrazu:

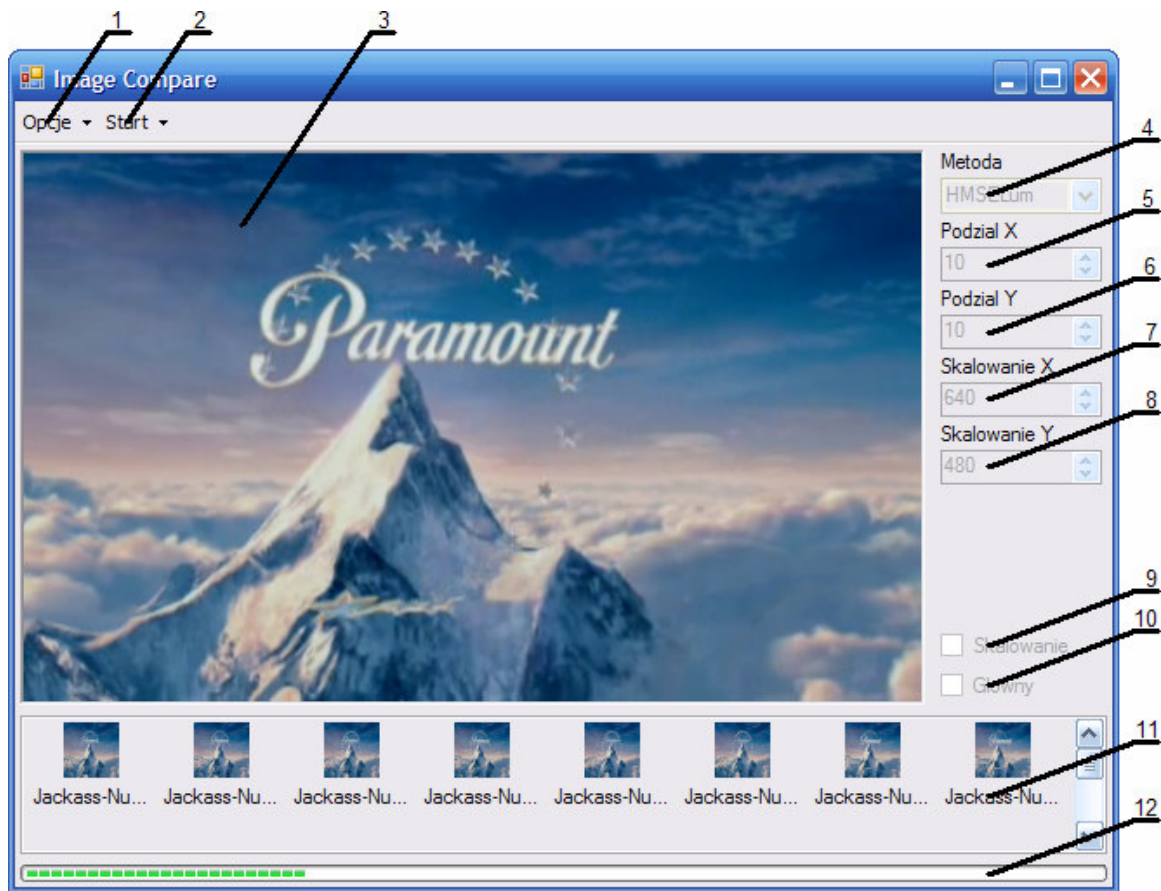
$$L(x, y) = 29.9\% \cdot I_R(x, y) + 58.7\% \cdot I_B(x, y) + 11.4\% \cdot I_G(x, y)$$

- Delinearyzacja intensywności:

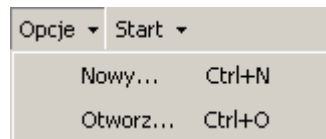
$$L'(x, y) = \frac{L(x, y)}{L(x, y) + 12.6 \cdot L(x, y)^{0.63}}$$

Zaprezentowane algorytmy zostały zaimplementowane zarówno do porównywania obrazów jak i ich histogramów, co zwiększyło możliwości programu. Główna zaleta zastosowania histogramu jest niewrażliwość porównywania na rotację obrazu (metody bez histogramu traktują obraz odwrócony jako inny obraz) oraz szybkość działania (operowanie na tablicy 255 odcieni koloru lub 3 tablicach dla każdej składowej zamiast na całym obrazie).

IV. Opis interfejsu użytkownika – instrukcja obsługi, przykłady



1. Opcje



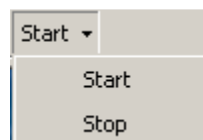
- **Nowy...**

Wyczyszczenie listy wczytanych obrazów, rozpoczęcie nowego porównywania.

- **Otwórz...**

Wczytanie obrazów do nowego porównywania lub dodanie nowych obrazów do aktualnego porównywania.

2. Start



- **Start**

Rozpoczęcie porównywania.

- **Stop**

Przerwanie porównywania.

3. Podgląd

Podgląd obrazu wskazanego na liście wczytanych obrazów.

4. Metoda

Wybór jednej z 10 metod porównywania. Metody porównujące histogram oznaczone są literą H na początku np. HMAX, HSAD itp.



5. Podział X

Podział obrazu na kilka części wzdłuż osi X-ów przed rozpoczęciem porównywania. Ma wpływ na jakość porównywania słabszymi metodami np. MAX, a jego wydajność ze względu na konieczność utworzenia wielu mniejszych obrazów nie jest zbyt duża. Obraz można maksymalnie podzielić na tyle fragmentów z ilu pikseli się składa, czyli jeden fragment odpowiada jednemu pikselowi.

6. Podział Y

Podział obrazu na kilka części wzdłuż osi Y-ów.

7. Skalowanie X

Zmiana rozmiaru obrazu wzdłuż osi X podczas wczytywania.

8. Skalowanie Y

Zmiana rozmiaru obrazu wzdłuż osi Y podczas wczytywania.

9. Skalowanie

Określa czy obraz ma zostać skalowany.

10. Główny

Obraz oznaczony jako główny jest wzorcem do porównywania obrazów tzn. obrazy będą sortowane według podobieństwa do wskazanego obrazu.

11. Lista wczytanych obrazów

Lista zawiera miniaturki wczytanych plików graficznych.

12. Pasek postępu

Pasek informuje o postępie porównywania. Nie pokazuje fazy podziału obrazu na podobrazy przy użyciu opcji Podział X i/lub Podział Y.

13. Normalizacja

Normalizuje wektory porównywanych obrazów przed porównaniem.

Szybki start:

- a) Uruchomienie programu *ImageCompare.exe*
- b) Wybranie opcji *Nowy...* z menu *Opcje*
- c) Wybranie opcji *Otwórz...* z menu *Opcje* i wczytanie kilku obrazów
- d) Wybranie jednego obrazu na *liście wczytanych obrazów*
- e) Ustawienie wybranego obrazu jako *Główny*
- f) Wybranie metody w polu *Metoda*
- g) Wybranie opcji *Start* z menu *Start*

V. Uwagi końcowe

Program był testowany pod względem poprawności działania oraz otrzymywanych wyników. Testy wykazały, że dla obrazów z większą ilością kolorów jak np. zdjęcia doskonale nadaje się metoda MSELum, zarówno z histogramem jak i bez niego. Do rozpoznawania mniej złożonych obrazów wystarczą prostsze metody porównywania z wyjątkiem metody MAX, która dla poprawnego działania wymaga podziału obrazu na kilka części. Metoda MAX może okazać się lepsza od metod teoretycznie bardziej zaawansowanych takich, jak np. SAD czy MSE zwłaszcza, jeżeli obraz zostanie podzielony na dużą liczbę podobrazów, wadą takiego podziału jest wspomniany wcześniej spadek wydajności. Dodanie metod operujących na histogramie umożliwiło rozpoznawanie obrazów obróconych, które są traktowane przez metody bez histogramu jako zupełnie inne obrazy. Kolejną zaletą metod z histogramem jest ich wydajność, obliczenia są wykonywane na tablicy 255 elementowej zawierającej rozkład kolorów lub na trzech tablicach dla każdej składowej RGB zamiast na całym obrazie (różnica jest widoczna zwłaszcza dla dużych obrazów). Dzięki zastosowaniu bezpośredniego dostępu do bufora, w którym jest przetrzymywana bitmapa możliwe stało się porównywanie znacznie większych obrazów w znacznie mniejszym czasie. Program posiada również możliwość skalowania obrazów, co umożliwia porównywanie bitmap o różnych rozmiarach (zastosowane algorytmy do porównania wymagają obrazów tej samej wielkości). Ze względu na brak możliwości sprawdzenia wszystkich przypadków oraz pomysłowość użytkowników program może posiadać błędy, których nie udało się wykryć podczas testów. Niemniej jednak program spełnia założenia projektu, a dzięki obiektowemu podejściu do zagadnienia może być w prosty sposób wzbogacany o nowe algorytmy porównywania, oraz dodatkowe funkcje zwiększające jego możliwości.

„Sortowanie obrazów: Dokumentacja kodu”

1. Opis klas

- **BITMAP**

Klasa zawiera funkcje i pola związane z obsługą bitmapy takie jak np.:

- Nazwa bitmapy
- Ścieżka do bitmapy
- Histogram
- Wynik obrazu dla poszczególnych metod

- **CDATA**

Przechowuje między innymi:

- Numer ostatnio wybranego obrazu z listy podglądu
- Listę obrazów
- Metody służące do:
 - + wyszukiwanie obrazu głównego
 - + pobieranie histogramu
 - + zmiana rozmiaru bitmapy
 - + podział bitmapy
 - + sortowanie bitmap

- **CMAX, CMSE, CMSELUM, CNMSE, CSAD**

Zbiór zaimplementowanych algorytmów podzielonych na klasy. Każda klasa zawiera dwie metody porównywania, z histogramem oraz bez histogramu.

2. Rozwiązania projektowe

- **Zmiana rozmiaru bitmapy**

```
public static Bitmap ResizeBitmap(Bitmap b, int x, int y)
{
    Bitmap tmp = new Bitmap(x, y);
    using (Graphics g = Graphics.FromImage((Image)tmp))
        g.DrawImage(b, 0, 0, x, y);
    return tmp;
}
```

Funkcja tworzy nową bitmapę o rządany rozmiarze do której zapisana zostaje oryginalna bitmpa. Dzięki przekształceniu obiektu typu Bitmap do Graphics bitmapa zostaje automatycznie przeskalowana.

- **Szybki odczyt pikseli**

```
BitmapData bd3 = ModelBitmapClusters[j].LockBits(...); // (0)*
BitmapData bd4 = clasterBitmaps[j].LockBits(...);      // (0)*
```

```

Unsafe
{
    byte* r3 = (byte*)bd3.Scan0; // (1)*
    byte* r4 = (byte*)bd4.Scan0; // (1)*
    int h = ModelBitmapClusters[j].Height;
    int w = ModelBitmapClusters[j].Width;
    for (int k = 0; k < h; k++)
        for (int l = 0; l < w; l++)
        {
            // (2)*
            double tmp = Math.Abs((double)r3[2] -
                                   (double)r4[2]) +
                        Math.Abs((double)r3[1] -
                                   (double)r4[1]) +
                        Math.Abs((double)r3[0] -
                                   (double)r4[0]);

            // (3)*
            r3 += 3;
            r4 += 3;

            if (tmp > tmpmax)
                tmpmax = tmp;
        }
    ModelBitmapClusters[j].UnlockBits(bd3); // (4)*
    clusterBitmaps[j].UnlockBits(bd4); // (4)*
}

```

(0)* - Zablokowanie dostępu do bitmapy.

(1)* - Czytanie bitmapy po bajcie.

(2)* - Dostęp do składowych koloru R – r3[2] i r4[2], G – r3[1] i r4[1], B – r3[0] i r4[0]

(3)* - Skok do następnego piksela.

(4)* - Zwolnienie bitmapy.

Rozwiązanie polega na bezpośrednim dostępie do pamięci, w której są przechowywane piksele. Użycie standardowych funkcji dostępu do pikseli np. `getPixel()`, `setPixel()` okazało się zbyt wolne (ok. 50 razy wolniejsze od bezpośredniego dostępu do pamięci) dla obrazów o wymiarach 5Mpix.

3. Uwagi

- Obliczenia są wykonywane w osobnym wątku, aby nie blokować aplikacji.
- Pasek postępu jest aktualizowany poprzez wysłanie delegata z wątku odpowiadającego za obliczenia do wątku, w którym działa główna forma aplikacji.
- Konieczne było (ze względu na wielowątkowość aplikacji) sprawdzenie czy wątki poszczególnych kontrolek i zdarzeń wywoływane są z właściwego dla siebie wątku (zmienna: *InvokeRequired*).

„Sortowanie obrazów: Testy”

1. Obrazy testowe

Obrazy testowe zawiera plik TESTY.zip z wyjątkiem testu przedmiotów (18MB)
Na testy składają się obrazki podzielone na kategorie:

- Barwy – Dwa rodzaje obrazów o zmienionej barwie, kontraście i natężeniu koloru
- Figury – Stożek z coraz bardziej obcięty wierzchołkiem
- Gry – Plansz do gry w kółko i krzyżyk
- Film – Kilka klatek z filmu
- Przedmioty – Kilka przedmiotów (kubek, komórka etc.)
- Natura – Kilka zdjęć

2. Wyniki testów

- Barwy
 - MAX – obrazy są podzielone prawidłowo na dwie kategorie
 - MSELum 15x7 – prawidłowy podział na kategorie, symetryczne rozmieszczenie barw
- Figury
 - MAX 8x8

Pozostałe metody sortują obrazy dla ustawień podstawowych.
- Gry
 - MAX 2x1

Pozostałe metody sortują obrazy dla ustawień podstawowych.
- Film
 - Jedynie metoda MAX nie poradziła sobie z tym testem.
- Przedmioty
 - Dość dobre wyniki uzyskuje metoda HSAD 20x20.
- Natura
 - NMSE 2x5
 - HSAD 2x5
 - MSELum 9x9

Największe problemy algorytmom porównywania sprawiają obrazy przedmiotów znajdujące się na tle w kolorze innych przedmiotów. Takie obrazy są zazwyczaj umieszczane w miejscach na pierwszy rzut oka losowych np.:



XIMGP3484... XIMGP3485... XIMGP3481... XIMGP3483... XIMGP3466... XIMGP3482

Niebieska komórka znajduje się pomiędzy komórkami czarnymi, ponieważ jej jasne klawisze są mylone z tłem, na jakim się znajdują komórki ciemne.