Képfeldolgozás

- dgedit –

Felhasználói dokumentáció:

A program használata és működése

A felhasználónak egy parancssorból kell meghívni a programot, a program neve után első argumentumként a módosítandó fájl nevét kell megadni, majd utána a rajta végzendő műveletet '-' felvezető karakterrel, a művelet módosítási értékét (nem mindegyiknek van), és a kimeneti fájl nevét (opcionális, elhagyás esetén fajlnev_edited.ppm fájl kerül mentésre):

```
Program hívása

dgedit {input_img} {option} [<option_value>] [output_image]

options:

      -br   Kép fényességének beállítása a kép maximum fényerejének
            <option_value> százalékára. Pl. 10%-kal fényesítés: -br 110

      -c    Kép kontrasztjának beállítása a kép <option_value>
            százalékára. Pl. 20%-kal kisebb kontraszt beállítása:
            -c 80

      -gr   Kép szürkeárnyalatossá tétele itt nem tartozik külön
            <option_value>

      -bl   Képhez elmosási effekt hozzáadása, az elmosás intenzitását
            az <option_value> -vel lehet kontrollálni. Pl teljes
            elmosás applikálása: -bl 100

      -e    képen található szélek kiemelése, itt nem tartozik külön
            <option_value>

Például

   -  a mario.ppm nevű kép 30% való világosítása:

      dgedit mario.ppm -br 130 mario_2.ppm

   -  A mario.ppm nevű fájl éleinek kiemelése, majd automatikusan
      generált névvel fájlba írása:

      dgedit mario.ppm -e
```

A program csak .ppm képekkel képes dolgozni, ha más fájlformátum kerül megadásra, akkor a program hibakóddal tér vissza, valamint egy hibaüzenetet ír ki. A forráskép elhagyása esetén is egy egyedi hibakód térül vissza, valamint üzenet jelenik meg. A fájl formátumának továbbá meg kell felelnie a .ppm fájlformátum leírásának (http://netpbm.sourceforge.net/doc/ppm.html). Ha nem megfelelő a fájl formázása, akkor egy erre vonatkozó hibakóddal- valamint üzenettel tér vissza.

Fejlesztői dokumentáció:

Főbb funkciók:

1. ppmio.h:

ppm_open(PPM * img, char * filename)

Kép megnyitására használatos, átvesz egy PPM típusú struktúra pointert, ami majd a megnyitott képre mutat majd, amelynek elérési útját a *filename adja meg. Megnyitott képet fel is kell szabadítani, ezt végzi el a ppm_close() függvény.

ppm_fprinf(PPM * img, char * path)

PPM fájl szabványainak megfelelően kiírja a módosított képet a path változóban megadott fájlba.

ppm_close(PPM * img)

ppm_open() -al létrehozott képeket zárhatjuk be vele (felszabadítja a memóriából).

2. ppmmod.h

ppm_changeBr(PPM * img , double pr)

Kép világosságának a megváltoztatása, beolvasott képnek állíthatjuk a fényerejét. A megadott pr érték: 0-200, azt mondja meg, hogy a kép maximum fényességének hány százalékával növelje/csökkentse.

Pl: 80 → kép fényerejének 20%-kal csökkentése, 140 → kép fényerejének 40%-kal növelése.

ppm_changeCr(PPM * img, double pr)

Kép kontrasztjának a megváltoztatása, beolvasott képnek állíthatjuk a kontrasztját. A megadott pr érték: 0-200, azt mondja meg, hogy a kontrasztot a kép maximum fényességének (legnagyobb érték, amit egy pixel felvehet) hány százalékával növelje/csökkentse.

Pl: 80 → kép kontrasztjának 20%-kal csökkentése, 140 → kép kontrasztjának 40%-kal növelése.

ppm_toGrayscale(PPM * img)

A képet szürkeárnyalatossá alakítja, azonban nem korrigál a különböző színek emberi érzékelésére, csupán átlagolja a Vörös, Zöld és Kék értékeket.

ppm_blur(PPM * img, double pr)

Gauss elmosás szűrőt helyez a fényképre, a pr az elmosás intenzitását állítja, értéke 0-100 között (nagyobb vagy kisebb érték esetén az intervallumba kényszeríti a bevitt értéket)

ppm_edge(PPM * img)

Élkiemelés effektust rak a képre, mielőtt magát az éliemelést elvégzi, végrehajta a kép szürkeárnyalatosra konvertálását, hiszen ez a művelet csak egy 2D mátrixon értelmezhető, valamint egy csak egy színértéket figyelembe vevő elmosást is alkalmaz, hogy a kép zaját, valamint a hasonló nem éleket ne emelje ki.

A blur és edge működése:

Mind a blur, mind az edge művelet felhasznál egy úgy nevezett kernelt, ez egy nxn-es mátrix, (n páratlan szám), amellyel végig pásztázunk az eredeti kép minden pixelén, úgy, hogy a kernel közpső mezőjét az adott pixelre illesztjük. Az így egymásra kerülő kép és kernel pixeleivel pedig műveletekt

végzünk, blur esetén súlyozott átlagot számolunk, edge esetén pedig csak összeadjuk a kernel és a kép pixeleinek szorzatát. Az így megkapott eredményeket egy új ppm kép adott pixelére kiírjuk. Azért nem az eredeti képet módosítjuk, mivel így ha a szomszédos pixelt néznénk, akkor nem az eredeti kép adataival számolnánk, hanem már a módosítottal, hiszen a kernelünk „rálógna” a módosított pixelre is. Az általam írt algoritmusok az érvénytelen kernel mezőket (amelyek lelógnak a képről) figyelmen kívül hagyja, így kihagyja a súlyozott átlagból, nem veszi hozzá az összeghez.
Mivel ezeket a műveleteket akár milliószor is el kell végezni egy nagyobb kép esetén, az átlagoló, valamint összeadó algoritmusokat optimalizálni próbáltam, így nem ellenőrzi le minden kernel pixelnél, hogy lelóg-e a képről, vagy sem, hanem még a művelet megkezdése előtt kiszámítja, hogy éppen hogyan helyezkedik el a képhez képest és így ad egy kezdő és egy vég oszlopot és sort, amin belül iterálnak a ciklusok

Összeadás kódja:

```
double kernel_add(PPM * img, Kernel * kernel, int pxRow, int pxCol, int rgb) {
    double sum = 0;

    int globRow = pxRow - kernel->size /2;
    int globCol = pxCol - kernel->size /2;

    int startRow, endRow;
    int startCol, endCol;

    startRow = globRow < 0 ? kernel->size - globRow : 0;
    startCol = globCol < 0 ? kernel->size - globCol : 0;

    endRow = globRow + kernel->size > img->height ?
                (globRow - kernel->size) - img->height
                : kernel->size - 1;
    endCol = globCol + kernel->size > img->width ?
                (globCol - kernel->size) - img->width
                : kernel->size - 1;
    for(int row = startRow; row <= endRow; row++)
        for(int col = startCol; col <= endCol; col++)
                sum += img->data[globRow + row][globCol + col][rgb] * kernel-
>grid[row][col];

    return sum;
}
```

Átlagolás kódja:

```
double kernel_avg(PPM * img, Kernel * kernel, int pxRow, int pxCol, int rgb) {
    int countf = 0; //On the edges of the picture not all of the kernel's fields will
count
    double avg = 0;

    int globRow = pxRow - kernel->size /2;
    int globCol = pxCol - kernel->size /2;

    int startRow, endRow;
    int startCol, endCol;

    startRow = globRow < 0 ? kernel->size - pxRow -1 : 0;
    startCol = globCol < 0 ? kernel->size - pxCol -1 : 0;

    //A kernel azon részéig megy, ami még bent van a kép keretein bleül -> a képen
kieső kernel részét ignorálja
    endRow = pxRow + kernel->size / 2 >= img->height ?
                kernel->size - ((pxRow + kernel->size / 2 + 1) - img->height)
                : kernel->size;
    endCol = pxCol + kernel->size / 2 >= img->width ?
                kernel->size - ((pxCol + kernel->size / 2 + 1) - img->width)
                : kernel->size;

    for(int row = startRow; row < endRow; row++) {
        for (int col = startCol; col < endCol; col++) {
         countf += kernel->grid[row][col];
         avg += img->data[globRow + row][globCol + col][rgb] * kernel-
>grid[row][col];
        }
    }

    avg = capval (avg /= countf, 0 , img->maxval);

    return avg;
}
```