Óbudai Egyetem, Neumann János Informatikai kar, Számítógépes képfeldolgozás és grafika

Mini PhotoShop

Jegyzet a kódhoz, kivitelezés C# -ban

Burian Sándor, AWXYHE

2020-2021, második félév

Tartalom

Megjegyzés

Ez a dokumentum nem tekinthető dokumentációnak, csupán jegyzet a kódhoz, magyarázat némelyik eljáráshoz, és az ihletet adó algoritmusok, leírások forrásmegjelölése!

Feladat leírás

A betöltött, vagy valamely kiválasztott képen lefuttatva az adott funkciót, az eredmény külön jelenjen meg.

A funkciók helyes megvalósítása mellett elvárt a gyorsításra vonatkozó optimalizálás. Mérjük a futási időt.

Funkciók:
Negálás
Gamma transzformáció
Logaritmikus transzformáció
Szürkítés
Hisztogram készítés
Hisztogram kiegyenlítés
Átlagoló szűrő (Box szűrő)
Gauss szűrő
Sobel éldetektor
Laplace éldetektor
Jellemzőpontok detektálása

Megoldás kulcselemei

A folyamatok sebességének növelése érdekében éddemes *Int* típusok helyett *byte* típust használni ahol csak lehet[1].

Az algoitmusok alappját a következő algoritmus rész képzi:

```
1. Bitmap image = new Bitmap(pictureBox.Image);
2. BitmapData bitmapData = image.LockBits(new Rectangle(0, 0, image.Width,
   image.Height), ImageLockMode.ReadWrite, image.PixelFormat);
   //lefoglalunk egy kép méretével megegyező részt a képen bitmap
   formátumra konvertálva írásra és olvasására, és bekérjük a
   csatornakiosztás módját
3. int height = bitmapData.Height;
4. int bPP = System.Drawing.Bitmap.GetPixelFormatSize(image.PixelFormat) /
   8; //pixelenkénti bájtok száma, az iteráláshoz
5. int width = bitmapData.Width*bPP;
6.
7. unsafe //mert a c# nem támogatja a pointereket bitmapen
8. {
9.          byte* firstPixel = (byte*)bitmapData.Scan0; //a lefoglalt
   terület első pixele
10.          Parallel.For(0, height, j =>
11.          {
12.                  byte* line = firstPixel + (j * bitmapData.Stride);
   //hogy soronként menjünk végig az leső pixeltől
13.                  for (int i = 0; i < width; i= i + bPP)//egy pixel hátom
   elem a három csatorna a sorban, BGR sorrendben a csatornák
14.                  {
15.                          int oldB = line[i];
16.                          int oldG = line[i+1];
17.                          int oldR = line[i+2];
18.
19.                          line[i] = (byte) oldB;
20.                          line[i + 1] = (byte) oldG;
21.                          line[i + 2] = (byte) oldR;
22.                  }
23.          });
24.          image.UnlockBits(bitmapData);
25.  }
26.
27.  pictureBox.Image = image;
```

Negálás

A képpontokat kivonjuk 255-ből, mindegyik csatornát, tehát: 255-pixel.r; 255-pixel.g; 255-pixel.b.[2]

Gamma transzformáció

Gamma transzformációhoz a következő képletet[3] használjuk: $S = c * R^{\gamma}$ ahol *S* az utput pixel, *c* egy konstans, többnyire 1, *R* a bementi pixel, γ a transzformáció mértéke. Ez a valóságban a következő képletet jelenti[4] a pixel minden csatornájára külön-külön: $255 \cdot c \cdot \left(\frac{pixelcsatorna}{255.0}\right)^{\gamma}$ ahol a $pixelcsatorna$ egy *pixel.r* vagy *pixel.g* vagy *pixel.b* érték.

Logaritmikus transzformáció

Logaritmikus transzformáció lényege, hogy megcseréljük a kép intenzitását, ahol eddig magas volt ott alacsonyabbra vesszük, ahol alacsony volt ott magasebbra, ehhez a következő képletet[5] használhatjuk: $s = c \cdot \log_{10}(1 + r)$ ahol *s* a pixel transzformáció utáni értéke, *c* egy konstans, amit mi szabunk meg, ehhez viszonyul a transzformáció maga, és *r* a pixel eredeti értéke. Ezt is, mint az eddigiekben is a pixel külön-külön csatornáján értelmezzük. A *c* értéket ha színes képekre szeretnénk alkalmazni[6] az eljárást érdemes az adott színcsatorna maximum értékéhez kötnünk: $\frac{255}{\log_{10}(1+maximum\_csatornaérték)}$ formában. Így az új érték nem fog „kilógni" a csatornatartományokból (0-255) és nem is ad szélsőséges értékeket, valamint ha előzőleg szürkített képen használjuk akkor is jól fog működni. Tehát alkalmazva a következő módon[7] járhatunk el: $\frac{255}{\log_{10}(1+maximum\_csatornaérték)} \cdot \log_{10}(1 + pixel\_csatornaértéke)$ amihez tudnunk kell, a *maximum_csatornaérték*-et ami mondjuk a piros csatornán a piros hisztogram maximuma, és a *pixel_csatornaértéke*t ami mondjuk a pixel piros intenzitása. A +1 azért szükséges mert log 0 végtelen, tehát ha valahol 0 lenne az érték akkor inkább vegyük 1-nek és számolhassunk tovább. Ezt természetesen mind az *R,G,B* csatornákon használjuk külön-külön, majd viszahelyezzük a már módosított pixelt a képbe.

Szürkítés

Szürkítéshez két képlet is könnyen implementálható, én mindkettőt megtettem:

1. az *r, g ,b* értékek átlaga[8]: $\frac{pixel.r + pixel.g + pixel.b}{3}$
2. 0.299 * piros + 0.587 * zöld + 0.114 * kék[9]

Hisztogram készítés

Hisztoggram készítéséhez három tömböt foglalunk le a három színcsatornának és egyet az átlagnak. A tömbök annyiadik elemét növeljük egyel amennyi az adott pixel adott színcsatornán mért értéke.

Hisztogram kiegyenlítés

A hisztogram kiegyenlítéséhez először szürkítjük a képet, majd kiszámoljuk a hisztogramját. A kapott értéken végigiterálva kiszámljuk mindegyik értékhez az addigi $\frac{\text{hisztogram\_sorösszeg} \cdot 255}{\text{képméret}}$ arányt és ezt eltároljuk. Majd újból végigiterálunk a képen, de most kicseréljük a pixelek érétkeit, úgy, ahogy a hisztogrammot képeztük, csak visszafele, azaz, az újonnan létrehozott aránytömb annyiadik elemét vesszük amennyi az adott képcsatorna értéke.[10]

Simító Szűrők

Átlagoló szűrő (Box szűrő)

A lényeg az elmosás, a használt mátrix méretétől függő mértékben. Egy pixel körüli pixelek értékeit vesszük és elosztjuk azok számával, azaz 9-el:[11] $\frac{1}{9} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ ahol a $mátrix_{2,2}$ eleme az a pixel amit épp tárgyalunk. Ezt szintén minden csatornára külön-külön alkalamazhatjuk, vagy ha előtte szürkítettünk úgy is jó. Fontos, hogy mivel átlagolunk nem mehetünk ki a kép széléig az eljárással, hanem még előtte megállunk, hogy legyen mit átlagoljunk.

***Fontos**, hogy hány biten ábrázoljuk a képet a párhuzmaosításhoz, hogy ne csússzanak el a színtartományok[12] ezért, ha 24 bitre optimalizáljuk a kódot akkor olyan bemenetet is kell adnunk, tehát, konvertálnunk kell* `PixelFormat.Format24bppRgb` *– re[13].*

Gauss szűrő

Az átlagoló szűrőhőz hasonló eljárás, annyi lényegi különbséggel, hogy más formulát használunk[14]:

$\frac{1}{16} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$. A mátrix elemeit az azoknak megfelelő helyeken levő pixelek csatornaértékével szorozzuk. Tehát nem hagyományos értelemben végzünk mátrix szorzást: $\frac{1}{16} \cdot \begin{bmatrix} 1 \cdot előző\_sor\_egyel\_korábbi\_pixele & 2 \cdot előző\_sor\_pixele & 1 \cdot előző\_sor\_rákövetkező\_pixele \\ 2 \cdot egyel\_korábbi\_pixel & 4 \cdot vizsgált\_pixel & 2 \cdot rákövetkező\_pixel \\ 1 \cdot következő\_sor\_egyel\_korábbi\_pixele & 2 \cdot következő\_sor\_pixele & 1 \cdot következő\_sor\_rákövetkező\_pixele \end{bmatrix}$

***Fontos**, hogy hány biten ábrázoljuk a képet a párhuzmaosításhoz, hogy ne csússzanak el a színtartományok* ezért, ha 24 bitre optimalizáljuk a kódot akkor olyan bemenetet is kell adnunk, tehát, konvertálnunk kell* `PixelFormat.Format24bppRgb` *– re**.*

Éldetektorok

Robert éldetektor

A robert éldetektor[15] azért érdekes számunkra mert az egyik legegyszerűbb éldetektor eljárást használja. Két darab 2×2 $- es\ mátrixot$ használ. Két gradiens értéket számolunk ki minden pixelhez, egyet x egyet y irányban. $G_x = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$ és $G_y = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$. Azaz, vesszük az első sor értékeit és kivonjuk a második sor értékeit a pixelnek, úgy nézve, hogy a vizsgált pixelünk a bal felső sarokban van azaz a $G_{x_{(1,1)}}$ és a $G_{y_{(1,1)}}$ elemek, tehát a kapott képletek: $\mathrm{G_x} = \mathrm{p(i,j)} - \mathrm{p(i+1,j+1)}$ és $\mathrm{G_y} = \mathrm{p(i,j)} - \mathrm{p(i+1,j+1)}$ minden pixelre, külön- külön, ha a pixel a $\mathrm{p(i,j)}$ koordinátán van. Ezek után vesszük a gradiensét a két értéknek: $\nabla \mathrm{F} = \sqrt{\mathrm{G_x^2} + \mathrm{G_y^2}}$. Ez lesz a pixelünk új értéke.

Sobel éldetektor

A Sobel éldetektorhoz két $3 \times 3 - as$ mátrixot használunk[16], egyet a horizontális, egyet a vertikálsi élek miatt. Ez a két mátrix: $G_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$ és : $G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$ Tehát minden egyes pixelhez, ami pl a $G_{x_{(2,2)}}$ és a $G_{y_{(2,2)}}$ helyén áll kiszámoljuk a kép pixeleinek érétékeit szorozva a G_x első és utolsó

oszlopával majd összeadjuk az értkeket és G_y első és utolsó sorával és összeadjuk az értékeket. A pixel végső értéke a pitagoraszi azonosság alapján: $G = \sqrt{G_x^2 + G_y^2}$ lesz[17]. Ehhez a gyakorlatban[18] viszont még el kell osztanunk 1141 -el, mert megközelítőleg ez lenne az lejárás elgnagyobb érétke, ami nyilván nem fér bele a 255-ös színtartományba, tehát normalizálnunk kell, így a használt képlet: $G = \frac{\sqrt{G_x^2+G_y^2}}{1141}$. Szintén számolhatunk egy szöget is, hogy az adott él milyen szögben áll: $\tan^{-1}\left(\frac{G_x}{G_y}\right) + \pi$ képletet kell használnunk. Ezt később az élek színezéséhez is felhasználhatjuk.

Laplace éldetektor

A Laplace féle eljárás csak egy $3 \times 3 - as$ mátrixot használ[19]: $\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$. Mivel itt csak egy mátrixot használunk, ezért gyorsabb a folyamat, de mivel a kétszeres deriváltra van szükségünk ezért sokkal érzékenyebb is, bármilyen zajra.

Jellemzőpontok detektálása

Ha a Harris féle módszert használjuk akkor gyakorlatilag csak egyesítetnünk kell az eddigi lépéseket egyetlen algoritmusba. Annyi különbséggel, hogy míg a Sobel éldetektorunk eddig vízszintes és függőleges éleket keresett, ha ezt egyszerre nézzük akkor megkapjuk a kereszt pontokat. Nekünk pedig pont ez kell!

Források

[1] Forrás: https://stackoverflow.com/a/34801225/6274697
[2] Forrás: https://www.codeproject.com/Articles/1989/Image-Processing-for-Dummies-with-C-and-GDI-Part-1
[3] Forrás: https://epochabuse.com/csharp-gamma-correction/
[4] Forrás: https://stackoverflow.com/questions/11211260/gamma-correction-power-law-transformation
[5] Forrás: https://epochabuse.com/log-transformation-on-images/
[6] Forrás: https://www.youtube.com/watch?v=4ZUNAqoOGWs&t=416s
[7] Forrás: https://www.codeproject.com/Tips/1062126/Image-Processing-2
[8] Forrás: https://dyclassroom.com/csharp-project/how-to-convert-a-color-image-into-grayscale-image-in-csharp-using-visual-studio
[9] Forrás: https://www.codeproject.com/Articles/1989/Image-Processing-for-Dummies-with-C-and-GDI-Part-1
[10] Forrás: https://www.codeproject.com/Tips/870536/How-To-Make-A-Clear-Gray-Image-Histogram-Equalizat
[11] Forrás: https://en.wikipedia.org/wiki/Box_blur
[12], * Forrás: https://stackoverflow.com/questions/67186696/implementing-box-blur-gaussian-blur-in-c-sharp#comment118771923_67186696
[13], ** Forrás: https://stackoverflow.com/questions/2016406/converting-bitmap-pixelformats-in-c-sharp
[14] Forrás: https://en.wikipedia.org/wiki/Kernel_(image_processing)
[15] Forrás: https://csharpvault.com/image-edge-detection/
[16] Forrás: https://www.cs.auckland.ac.nz/compsci373s1c/PatricesLectures/Edge%20detection-Sobel_2up.pdf
[17] Forrás: https://www.youtube.com/watch?v=uihBwtPIBxM
[18] Forrás: https://www.cpib.ac.uk/files/archives/convolution.zip
[19] Forrás: https://aishack.in/tutorials/sobel-laplacian-edge-detectors/