FESB, SVEUČILIŠTE U SPLITU

SEMINARSKI RAD

Bilateralno filtriranje

Ivan Babić

Tomislav Boros

Ivan Delija

lipanj 2017.

# UVOD

Digitalne slike (fotografije) moguće je obrađivati nakon što su snimljene na mnoštvo različitih načina. Obrađivanjem (manipulacijom) slike želimo poboljšati kvalitetu slike i samim time tako dobivena slika pogodnija je za specifičnu primjenu, što ima za utjecaj da priroda, tj. tehnika obrade dosta ovisi o daljnjoj primjeni. Najčešće korišteni način obrade upravo je filtriranje slike, postupak kojim se iz originalne slike, primjenom željenog efekta, upotrebom odgovarajućeg matematičkog algoritma nad pikselima stare slike, stvara nova slika. Glavna svrha filtriranja je redukcija šuma. Istodobno se filtriranjem nastoji istaknuti željena svojstva slike kao što su rubovi ili boje, te potisnuti neželjena.

Filtriranje je proces gdje se na svaki piksel slike koristi filter. Filtriranje svakog pojedinog piksela je operacije neovisna o primjeni filtera nad ostalim, susjednim pikselima. Kako se slika sastoji od velikog broja piksela, što predstavlja mogućnost primjene paralelizma kod filtriranja svakog pojedinog piksela.

Gaussian Filter je filter zaglađivanja koji može biti primjenjen na slike kako bi zagladio šumove. Bilateralni filter zaglađuje površine kao i Gaussian filter ali uz zadržavanje oštrih rubova objekata na slici. Bilateralni filter daje impresivne rezultate, ali uz zahtjev velike količine resursa.

Stoga ćemo u ovom seminarskom radu pokazati kako se vrlo zahtjevne operacije mogu paralelizirati, tj. ubrzati. Predstaviti ćemo jedan algoritam implementiran CUDA tehnologijom koji paralelizacijom izvođenja operacija filtriranja dovodi do značajnih ubrzanja filtriranja.

# BILATERALNI FILTER

Bilateralni filtar je niskopropusni, dvodimenzionalni filtar vrlo sličan Gaussovom filtru, uz tu prednost što zadržava oštrinu bridova u slici. Vrijednost intenziteta svakog piksela slike mjeri kroz prosjek vrijednosti susjednih piksela. Te vrijednosti imaju dvije komponente, od kojih je prva korištena Gaussianovim filterom. Druga komponenta uzima u obzir razliku intenziteta susjednih piksela u usporedbi sa ispitivanim.

Bilateralni filter je implementiran korištenjem CUDA tehnologije koja značajno poboljšava vrijeme izvršavanja. Sa sekvencijalnog izvođenja ka paralelnom dovodi do ubrzanja do reda veličine 100x, čak i više daljnom optimizacijom koda.

Filtriranje treba reducirati šum, a da pritom ostanu sačuvana karakteristična svojstva slike. Zato se koriste filtri koji «izglađuju» oštre prijelaze u slici.

Bilateralno filtriranje slika koristi se u nizu komercijalnih programa za obradu slike: Adobe Photoshop („surface blur tool“), GIMP („Selective Gaussian Blur“). Jedna od primjena bilateralnog filtera u stvarnome svijetu je u medicini prilikom pregleda RTG i CT snimaka.

# Izvršavanje CPU i GPU koda i rezultati izvršavnja

Postoje 2 verzije koje ćemo prikazati: algoritam bilateralnog filtra koji koristi CPU i onaj na GPU. Program se pokreće uz unos parametara:

<input file> <width> <height> <channels>

<neighborhood radius> <spatial sigma> <range

sigma> <iterations>

gdje je ulazni file slika u .raw formatu gdje se širina, visina i channels označava njene parametre. Spatial sigma i range su floati, a broj iteracija označava koliko puta se test izvršava.

Kod je kompajliran korištenjem **Visual Studia 2013, CUDA 8.0**, a korišteni hardver je:

Intel Core i5-2410m, 2.90 GHz

6GB DDR3 ram, 1333 MHZ

GeForce GT540m, 1GB DDR3

Uvid u test se pokreće na file koji se nalazi u folderu Release.

Test je napravljen za 10 uzastopnih pokretanja oba algoritma, veličina slike 800x533, gdje je prosjek vremena izvršavanja po radiusu: 4:

-prosječno vrijeme **CPU: 3,8234**

-prosječno vrijeme **GPU: 0,602667**

Vidljiva je razlika u korištenju GPU i CPU u brzini izvršavanja zbog korištenja svojstva paralelnog izvršavanja threadova kod GPU. Razlika u brzini izvršavanja još dodatno dolazi do izražaja povećanjem razlučivosti slike ili videa.

Razlikujemo Host(CPU i resurse) te Device(procesor grafičke kartice sa memorijskim resursima):

Funkcija za izračun dvodimenzionalnog Gaussiana:

float\* generateGaussianKernel(int radius, float sigma)

{

int area = (2\*radius+1)\*(2\*radius+1);

float\* res = new float[area];

for(int x = -radius; x <= radius; x++)

for(int y = -radius; y <= radius; y++)

{

int position = (x+radius)\*(radius\*2+1) + y+radius;

res[position] = gaussian2d(x,y,sigma);

}

return res;

}

Implementacija Gaussiana:

inline \_\_device\_\_

float gaussian1d\_gpu\_reg(float x, float variance, float sqrt\_pi\_variance)

{

float gaussian1d = -(x\*x)/(2\*variance);

gaussian1d = \_\_expf(gaussian1d);

gaussian1d /= sqrt\_pi\_variance;

return gaussian1d;

}

\_\_global\_\_

void bilateralFilterGPU\_v(float\* output, uint2 dims, int radius, float\* kernel, float variance, float sqrt\_sigma)

{

const unsigned int idx = blockIdx.x\*blockDim.x + threadIdx.x;

uint2 pos = idx\_to\_co(idx,dims);

if(pos.x >= dims.x || pos.y >= dims.y) return;

float currentColor = tex1Dfetch(tex, idx);

float res = 0.0f;

float normalization = 0.0f;

for(int i = -radius; i <= radius; i++) {

for(int j = -radius; j <= radius; j++) {

int x\_sample = pos.x+i;

int y\_sample = pos.y+j;

if( x\_sample < 0) x\_sample = -x\_sample;

if( y\_sample < 0) y\_sample = -y\_sample;

if( x\_sample > dims.x - 1) x\_sample = dims.x - 1 - i;

if( y\_sample > dims.y - 1) y\_sample = dims.y - 1 - j;

float tmpColor = tex1Dfetch(tex, y\_sample\*dims.x + x\_sample);

float gaussian1d = -((currentColor - tmpColor)\*(currentColor - tmpColor))/(2\*variance);

gaussian1d = \_\_expf(gaussian1d);

gaussian1d /= sqrt\_sigma;

float gauss\_spatial = kernel[co\_to\_idx(make\_uint2(i+radius,j+radius),make\_uint2(radius\*2+1,radius\*2+1))];

float weight = gauss\_spatial \* gaussian1d;

normalization = normalization + weight;

res = res + (tmpColor \* weight);

}

}

output[idx] = res / normalization;

}

Prethodno prikazani algoritam je korišten na GPU, vidljivo je da se koristi predefinirani kernel te jednodimenzionalni Gaussian filter. Mjeri se intenzitet susjednih filtera, te se sa dodjelom težinskih vrijednosti piksela slike u kombinaciji dobije bilateralni filter.

Sljedeći korišteni algoritam koristi RGB anotaciju umjesto tekstura koje su korištene u prethodnom. Ovaj algoritam koristi tri jednodimenzionalna filtera koje implementira na svaki pojedinačni piksel sekvencijalno.

void CPUbilateralFiltering(RGB\* data, int width, int height,int radius, float sigma\_spatial, float sigma\_range)

{

int numElements = width\*height;

RGB\* res\_data = new RGB[numElements];

for(int x = 0; x < width; x++)

{

for(int y = 0; y < height; y++)

{

int array\_idx = y \* width + x;

RGB currentColor = data[array\_idx]; //idx

RGB res = makeColor(0.0f,0.0f,0.0f);

RGB normalization = makeColor(0.0f,0.0f,0.0f);

for(int i = -radius; i <= radius; i++) {

for(int j = -radius; j <= radius; j++) {

int x\_sample = x+i;

int y\_sample = y+j;

//mirror edges

if( (x\_sample < 0) || (x\_sample >= width ) ) {

x\_sample = x-i;

}

if( (y\_sample < 0) || (y\_sample >= height) ) {

y\_sample = y-j;

}

RGB tmpColor = data[y\_sample \* width + x\_sample];

float gauss\_spatial = gaussian2d(i,j,sigma\_spatial); //gaussian1d(i,sigma\_spatial)\*gaussian1d(j,sigma\_spatial);//

RGB gauss\_range;

gauss\_range.R = gaussian1d(currentColor.R - tmpColor.R, sigma\_range);

gauss\_range.G = gaussian1d(currentColor.G - tmpColor.G, sigma\_range);

gauss\_range.B = gaussian1d(currentColor.B - tmpColor.B, sigma\_range);

RGB weight;

weight.R = gauss\_spatial \* gauss\_range.R;

weight.G = gauss\_spatial \* gauss\_range.G;

weight.B = gauss\_spatial \* gauss\_range.B;

normalization = normalization + weight;

res = res + (tmpColor \* weight);

}

}

res\_data[array\_idx] = res / normalization;

}

}

for(int i = 0; i < numElements; i++)

{

data[i] = res\_data[i];

}

delete[] res\_data;

}

Slika prikazuje rezultat filtriranja koristeći bilateralni algoritam. Vidljivo je da su rubovi ostali oštri, tj. očuvani.



Original:



# LITERATURA

* <https://en.wikipedia.org/wiki/Bilateral_filter>
* <https://bib.irb.hr/datoteka/625214.Doktorska_disertacija_-_Damir_Filko.pdf>
* <http://www.zemris.fer.hr/predmeti/ra/Magisterij/10_Sajko/azps.html>
* <https://groups.csail.mit.edu/graphics/bilagrid/bilagrid_web.pdf>