ALGORITMI PARALELI SI DISTRIBUITI: Tema #1
Micro Renderer si Resize folosind tehnici antialiasing

Termen de predare: 01 Noiembrie 2015

Titulari curs: *Valentin Cristea, Ciprian Dobre, Elena Apostol*

Responsabil Tema: **Cristian Chilipirea**

Obiectivele Temei

Implementarea unui software capabil sa redimensioneze o imagine micsorand pierderea de informatie folosind antialiasing de tip SSAA (numit si FSAA), super sampling antialiasing. A doua parte va fi implementarea unui micro motor de randare, capabil sa creeze o poza ce contine o linie.
Acest software va folosi thread-uri multiple si va fi implementat in limbajul C folosind OpenMP.
Un punct important al acestei teme va fi obtinerea scalabilitatii, cu privire la numarul de thread-uri, si demonstrarea acesteia.
O tema fara readme sau readme necorespunzator va primi automat 0 puncte.

Part 1. Super Sampling Anti Aliasing

In momentul in care o imagine este randata poate aparea efectul de aliasing. Acest efect este cel mai usor de vazut in cazul liniilor sau al fonturilor, acestea au un aspect pixelat. Pentru a ascunde acest efect se randeaza o imagine cu rezolutie mai mare si apoi se scade rezolutia la cea a ecranului folosind una din **multele** tehnici de anti aliasing.
In cazul SSAA (sau FSAA) fiecare pixel din imaginea finala reprezinta mai multi pixeli din imaginea originala (un patrat cu latura de $resizefactor$ pixeli). Aceste patrate nu se suprapun in nici un fel.
Prima parte a temei considera in completarea fisierelor homework.c si homework.h pentru a implementa micsorarea de imagine folosind SSAA in paralel. Vor exista doua moduri de a micsora o imagine: Folosind media aritmetica dintre valorile pixelilor aferenti (daca factorul de micsorare - $resizefactor$ este par); Folosind kernel gaussian (daca factorul de micsorare - $resizefactor$ este 3).

$$GaussianKernel = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Pentru a aplica kernelul gaussian fiecare matrice de 3x3 din imagine este inmultita **element cu element** cu kernelul. Valoarea pixelului rezultat este suma elementelor matricei rezultate impartita la 16 (suma elementelor kernelului gaussian).
Exemplu: Presupunem ca imaginea originala are 4 pixeli, 3 albi, unul negru. Pixelul din imaginea cu rezolutie scazuta, va avea un pixel ce corespunde celor 4 ce va avea o culoare gri (compusa din 75 la suta alb si 25 la suta negru). Acest lucru este evidentiat cel mai bine de imaginile 1a, cu o rezolutie de 4x6, si 1b, cu o rezolutie de 2x3. Figura 1c arata imaginea micsorata in marimea ei normala.

Part 2. Micro renderer

Un renderer este un software care converteste elemente matematice 2 sau 3 diminesionale, gen linii, cuburi, sfere, in pixeli ai unei imagini care le reprezinta.

(a) Inainte

(b) Dupa Anti Aliasing

(c) Dupa Anti Aliasing
Marime normala

Figure 1: Micsorare de imagine

Pentru aceasta parte a temei trebuie sa completati fisierele homework1.c si homework1.h pentru a scrie un renderer capabil sa creeze o imagine alba ce contine o linie neagra. Se presupune un spatiu patratic de 100 de metri pe 100 de metri. Acest spatiu contine o linie corespunzatoare formulei $-1*x+2*y+0=0$. Grosimea acestei linii este de 3 metri. $x=0$ si $y=0$ sunt in stanga jos a ecranului iar $x=max$ si $y=max$ sunt in coltul dreapta sus al ecranului. Imaginea finala poate avea orice rezolutie patratica si va contine intregul spatiu de 10.000 metri patrati.
Pentru a crea imaginea, se va lua centrul fiecarui pixel si se va calcula distanta de la acesta la linie, daca rezultatul este mai mic de 3m atunci este negru, altfel este alb.

Detalii input/output Formatul pnm

.pnm este unul din cele mai simple formate pentru imagini. Pentru prima parte a temei input va fi o imagine .pnm. Aceasta poate fi color sau grayscale. In urmatoarele linkuri gasiti detalii despre formatul pnm, despre formatul corespunzator color si corespunzator grayscale. .pnm poate avea inauntru o imgine color sau grayscale. Diferentierea se face prin prima linie (P5 sau P6). Pentru a obtine poze sau a extrage poze din aceste formate se recomanda utilitatiile linux:

- jpegtopnm - converteste de la o poza jpeg la una pnm
- pngtopnm - converteste de la o poza png la una pnm
- ppmtopgm - converteste de la o poza color la una grayscale
- pgmtoppm - converteste de la o poza grayscale la una color
- pnmtojpeg - converteste de la o poza pnm la una jpeg
- pnmtopng - converteste de la o poza pnm la una png

Folosire: ./jpegtopnm inputImage.jpg >outputImage.pnm
O simplificare este legata de formatul de intrare. Nu trebuie respectat cu exactitate formatul .pnm. Dar va trebui respectat cel putin urmatorul, pentru a putea citi si scrie fisiere compatibile cu executabilele de mai sus.

P(5 sau 6)\n
width height\n
maxval(maxim 255)\n
$height*width*numcolors$ bytes reprezentand poza

Prima parte a temei primeste o imagine color sau grayscale (se va diferentia dupa prima linie din imagine) si va avea ca output o alta imagine de tip corespunzator cu o rezolutie scazuta de $resizefactor$ ori.
A doua parte a temei va avea ca output o imagine de tip pnm grayscale.

Scalabilitate si readme

O parte importanta a acestei teme este obtinere scalabilitatii. Desigur pentru a obtine scalabilitate trebuie intai scris programului serial, si acesta va avea o parte mare din punctaj.
Scheletul de cod oferit apeleaza 3 functii care trebuie completate. Acestea trebuie sa citeasca fisierul de intrare, sau sa intializeze zona de memorie dupa caz, sa ruleze algoritmul de resize sau de randare, si in final sa scrie rezultatele. Timpul de executie al functiei din mijloc este masurat si afisat la stdout.
Pentru a demonstra scalabilitatea trebuie sa masurati timpul de executie al programului vostru sub aceleasi configuratii schimband numarul de threaduri.
Readme-ul trebuie sa contina, descrierea solutiei voastre, metodologia de testare si descrierea sistemelor pe care ati efectuat testele alaturi de rezultatele prin care demonstrati (sau nu) scalabilitatea.
Testele trebuie efectuate pe coada ibm-nehalem.q a clusterului din facultate.
qsub -cwd -q ibm-nehalem.q scriptName.sh
scriptul va rula testele.

Informatii rulare - parametri - upload

Prima parte se va rula conform:

./homework inputImageName outputImageName resizeFactor numThreads

outputImageName va fi inputImageName de acelasi tip (color sau grayscale) cu rezolutia scazuta de resizeFactor ori.
Daca rezolutia imaginii initiale nu se imparte perfect la resizeFactor atunci se va pierde informatia din partea dreapta sau de jos a imaginii. (folositi exemplele oferite cu codul de start pentru clarificare)
A doua parte se va rula conform:

./homework1 outputImageName width numThreads

Tema se va uploada folosind vmchecker. Se va uploada o arhiva zip ce contine readme, homework1.c, homework.c, homework1.h, homework.h. Este extrem de important ca .zip-ul sa nu contina alte foldere sau fisiere.

Orice incercare de frauda, de a falsifica rezultatele (inclusiv scalabilitatea), se va penaliza printr-o metoda cat mai extrema (overkill) conform regulamentelor in viigoare..

Este interzis sa afisati orice la stdout sau stderr

Punctaj:
36 puncte - corectitudine resize
40 puncte - scalabilitate resize
14 puncte - corectitudine renderer
10 puncte - scalabilitate renderer