ALGORITMI PARALELI ȘI DISTRIBUIȚI
Tema #1 Antialiasing și micro renderer

Termen de predare: 28 Octombrie 2018

Responsabili Temă: **Cristian Chilipirea, Ion Dorinel Filip, Petru Guriță**

Obiectivele Temei

Implementarea unui program capabil să redimensioneze o imagine micșorând pierderea de informație folosind tehnica anti-aliasing de tip **super sampling antialiasing** (SSAA sau FSAA) și implementarea unui micro **motor de randare**, capabil să creeze o poză ce conține o linie.
Aceste programe vor fi implementate **obligatoriu** folosind thread-uri prin API-ul oferit de librăria **pthread**, în limbajul **C**. Rezolvarea va trebui să scaleze cu numărul de thread-uri, rămânând într-un interval de eficiență acceptat (determinat empiric prin limitarea timpului de execuție la testare).

Part 1. Super Sampling Anti Aliasing

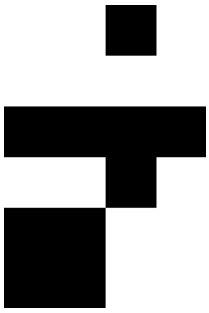
În momentul in care o imagine este randată poate apărea efectul de aliasing. Acest efect este cel mai ușor vizibil în cazul liniilor sau fonturilor, acestea având un aspect pixelat. O tehnică uzuală de a ascunde acest efect este de a randa imaginea la o rezoluție mare apoi a o micșora la rezoluția dorită folosind o aproximare. În cazul SSAA fiecare pixel din imaginea finală reprezintă mai mulți pixeli din imaginea originală (un pătrat cu latura de $resizefactor$ pixeli). Astfel imaginea micșorată va fi de $resizefactor$ ori mai mică decât cea originală. Va trebui să implementați două moduri de a micșora o imagine:

- Dacă $resizefactor$ are o valoare pară se va calcula valoare pixelului din imaginea micșorată ca fiind media aritmetică a $resizefactor^2$ pixeli din poza originală.
- Dacă $resizefactor$ este 3, se va calcula valoarea pixelului din imaginea micșorată înmulțind valorile din submatricea originală, de 9 pixeli, cu valorile Kernel-ului Gaussian (de asemenea o matrice de 3x3 elemente), făcând suma elementelor rezultat și împărțind rezultatul la 16 (suma elementelor Kernel-ului Gaussian).

În testarea temei nu vor fi folosiți factori de scalare impari diferiți de 3.

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

Imaginea poate fi color (RGB cu câte o valoare unsigned, pe un octet, pentru fiecare canal) sau grayscale (fiecare pixel dat ca o valoare între 0 și 255). Dacă imaginea este color calculele se vor face pentru fiecare culoare separat.
Exemplu: Presupunem că imaginea originală are patru pixeli, trei albi, unul negru. Pixelul din imaginea cu rezoluție scăzută, va avea o culoare ce corespunde celor 4, un gri compus din 75% alb și 25% negru. Un exemplu vizual ar fi imaginea. 1a, cu o rezoluție de 4x6, și 1b, cu o rezoluție de 2x3. Figura 1c arată imaginea micșorată în mărimea ei normală.
Pentru această parte a temei va trebui să porniți de la fișierele **homework.c** și **homework.h**.



(a) Înainte

(b) După Anti Aliasing

(c) După Anti Aliasing Mărime normala

Figure 1: Micșorare de imagine

Part 2. Micro renderer

Un renderer este un program care creează o imagine dintr-un set de elemente geometrice bi sau tri dimensionale (exemplu. linii, cuburi, sfere).
Se cere să implementați un motor de randare capabil să creeze o imagine albă ce conține o linie neagră. Se presupune un spațiu pătratic de 100cm x 100cm. Acest spațiu conține o linie corespunzătoare formulei $-1 * x + 2 * y + 0 = 0$. Grosimea acestei linii este de 3cm.
Coordonatele $x = 0$ si $y = 0$ sunt in stânga jos a ecranului, iar $x = max$ și $y = max$ în dreapta sus a ecranului. Imaginea finală poate avea orice rezoluție pătratică și va conține întregul spațiu 100cm x 100cm. Pentru a crea imaginea, se va lua centrul fiecărui pixel și se va calcula distanța de la acesta la linie, dacă rezultatul este mai mic de 3cm, atunci este negru, altfel este alb.
Pentru această parte a temei trebui să completați fișierele **homework1.c** si **homework1.h**.

Detalii input/output Formatul pnm

.pnm este unul din cele mai simple formate pentru imagini. Aceasta poate fi color sau grayscale. În următoarele link-uri găsiți detalii despre formatul pnm, despre formatul corespunzător color și corespunzător grayscale. .pnm poate avea înăuntru o imagine color sau grayscale. Diferențierea se face prin prima linie (P5 sau P6). Pentru a obține poze sau a extrage poze din aceste formate se recomandă utilitarele linux:

- jpegtopnm - transformă o poză jpeg la una pnm
- pngtopnm - transformă o poză png la una pnm
- ppmtopgm - transformă o poză color la una grayscale
- pgmtoppm - transformă o poză grayscale la una color
- pnmtojpeg - transformă o poză pnm la una jpeg
- pnmtopng - transformă o poză pnm la una png

Folosire: ./jpegtopnm inputImage.jpg >outputImage.pnm
În realizarea temei, se va folosi o simplificare legată de formatul de intrare/ieșire. Nu trebuie respectat cu exactitate formatul .pnm, dar va trebui respectat cel puțin următorul șablon pentru a putea citi și scrie fișiere compatibile cu executabilele de mai sus.

```
P(5 sau 6)\n
width height\n
maxval (maxim 255)\n
height*width*numcolors bytes binari reprezentÃnd poza
```

Prima parte a temei primește o imagine color sau grayscale (se va diferenția după prima linie din fișierul imaginii) și va avea ca output o altă imagine de tip corespunzător cu o rezoluție mai mică de $resizefactor$ ori. A doua parte a temei va avea ca output o imagine de tip pnm grayscale.

Scalabilitate

O parte importantă a acestei teme este obținerea de scalabilitate.
Scheletul de cod oferit apelează, pentru fiecare dintre probleme, 3 funcții care trebuise completate. Acestea trebuie să citească fișierul de intrare (sau să inițializeze zona de memorie pentru renderer), să ruleze algoritmul de resize (sau de randare), și respectiv să scrie rezultatele în fișierul ieșire. Timpul de execuție al funcției din mijloc este măsurat și afișat la stdout.
Este obligatoriu să păstrați neschimbat rolul celor 3 funcții. Astfel, nu este permis să faceți operații de I/O în cadrul funcției mediane, respectiv nu este permisă utilizarea de fire de execuție în funcțiile de citire/scriere. Mai mult, codul voastre trebuie să poată fi compilat cu Makefile-ul oferit în schelet.
Pentru a demonstra scalabilitatea trebuie să măsurați timpul de execuție al programului vostru sub aceleași configurații (fișiere si $resizefactor$) cu număr diferit de thread-uri.
Pentru determinarea timpilor de execuție se vor folosi mașini din coada $ibm - nehalem.q$ a clusterului din facultate. Doar acestia vor fi luați în considerare pentru a determinarea scalabilitatea soluției. Vă încurajăm să testați temele pe cluster în mai multe etape ale rezolvării.

Informații rulare - parametri - upload

Programele se vor rula conform:

```
./homework inputImageName outputImageName resizeFactor numThreads
./homework1 outputImageName width numThreads
```

$outputImageName$ va fi $inputImageName$ de acelasi tip (color sau grayscale) cu rezoluția scăzută de $resizefactor$ ori.
Dacă rezoluția imaginii inițiale nu se împarte perfect la $resizefactor$, atunci se va pierde informația din partea dreapta sau de jos a imaginii. Folosiți exemplele oferite cu codul de start pentru clarificare.
Se va uploada o arhiva zip ce conține $Readme.txt$, $homework1.c$, $homework.c$, $homework1.h$, $homework.h$.
Este extrem de important ca .zip-ul să nu conțină alte directoare sau fișiere.

Orice încercare de fraudă, de a falsifica rezultatele (inclusiv scalabilitatea), va duce la un punctaj de 0 pentru toate temele..

Este interzis să afișați orice la stdout sau stderr. Se vor păstra doar afișările din scheletul de cod.

Punctaj:
36 puncte - corectitudine resize
40 puncte - scalabilitate resize
14 puncte - corectitudine renderer
10 puncte - scalabilitate renderer
O temă care nu compilează sau nu trece niciun test va fi punctată automat cu 0 puncte.
Imaginiile finale de test vor fi foarte mari, dar vor încăpea în RAM. Vă recomandăm să construiți singuri teste foarte mari.