ALGORITMI PARALELI ȘI DISTRIBUIȚI
Tema #1 Desenarea paralelă de curbe contur folosind algoritmul Marching Squares

Responsabili: Radu-Ioan Ciobanu, Valentin Bercaru, Corina Mihăilă, Ioana Marin, Sergiu Toader, Diana Megelea, Cătălin Rîpanu, Alexandru Tudor, Alexandra Ion, Andreea Borbei, Elena Anghel, Taisia Coconu

Termen de predare: 12-11-2023 23:59 (soft), 19-11-2022 23:59 (hard)
Ultima modificare: 21-10-2023 18:45

Cuprins

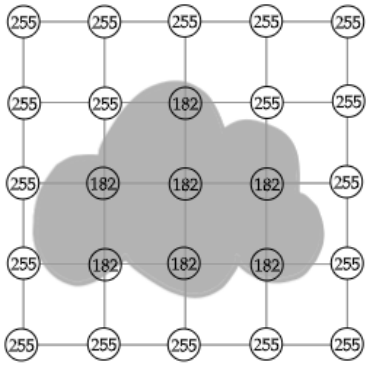
Cerință

Pornind de la implementarea secvențială, să se paralelizeze, cu ajutorul Pthreads în C/C++, un program care generează contururi pentru hărți topologice folosind algoritmul Marching Squares. Scopul final este de a se obține aceleași fișiere de ieșire ca la varianta secvențială, dar cu timpi de execuție îmbunătățiți. Tema are scopul de a vă obișnui cu ideea de a porni de la un cod existent și de a-l optimiza (în acest caz, folosind Pthreads).

Algoritmul Marching Squares

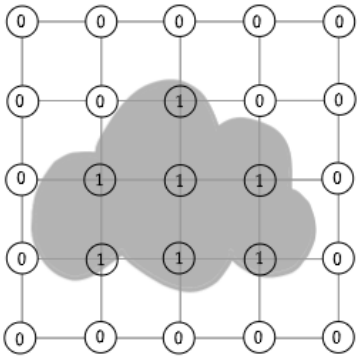
Marching Squares este un algoritm de grafică introdus în anii 1980 care poate fi folosit pentru delimitarea contururilor dintr-o imagine. El poate fi folosit pentru a desena linii de altitudine pe hărți topografice, temperaturi pe hărți termice, puncte de presiune pe hărți de câmp de presiune, etc. Algoritmul funcționează după pașii descriși în continuare.

În primă fază, întreg domeniul D al imaginii de intrare este împărțit în pătrate de dimensiune fixă (pe baza unei valori prestabilite), și apoi se creează un grid G format din colțurile acestor pătrate (astfel, dacă avem N^2 pătrate, atunci G va fi format din $(N+1)^2$ puncte). Acest lucru se poate observa în imaginile de mai jos.



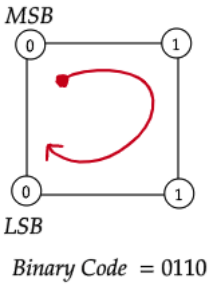
The Grid G

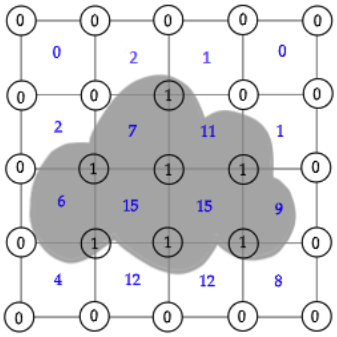
În continuare, se iterează prin grid pentru a se determina starea fiecărui punct raportată la o valoare de izolare σ. Pe baza valorii de izolare și a grid-ului, se creează mai departe un alt grid binar F, unde fiecare punct este 0 dacă valoarea punctului corespunzător din G este mai mare decât σ, sau 1 altfel. Un exemplu de astfel de grid binar (pentru $\sigma = 200$) se poate observa în imaginea de mai jos.



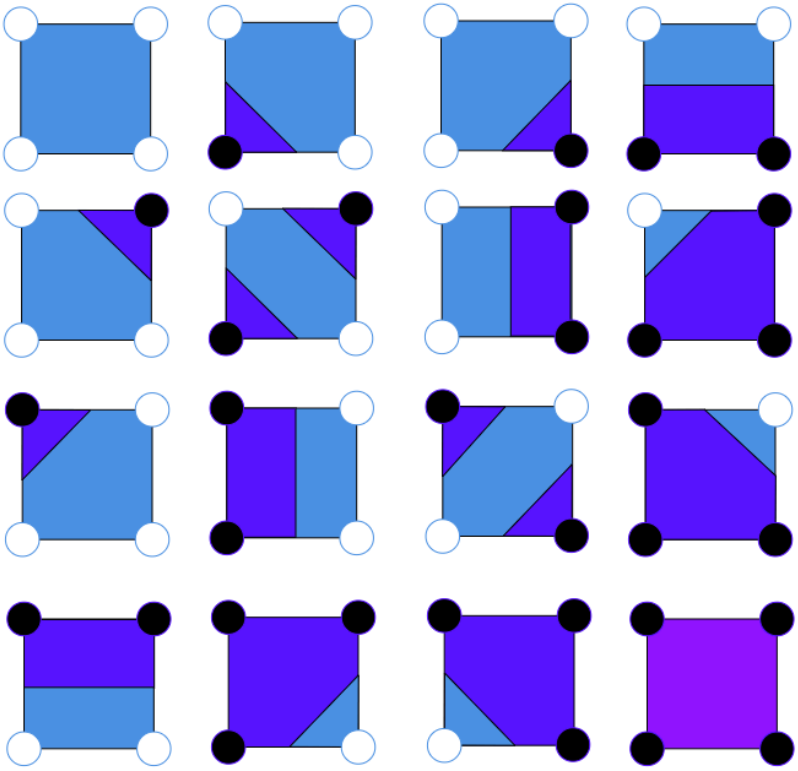
Fiecare pătrat component din F este mai departe asociat cu o configurație, formându-se câte un cod binar bazat pe valoarea fiecărui vârf al pătratului, prin parcurgerea în sensul acelor de ceasornic de la colțul din stânga sus la colțul din stânga jos. În partea stângă a imaginii de mai jos, se poate observa o astfel de

abordare, iar echivalentul decimal al codului binar specific unui pătrat reprezintă configurația sa, așa cum este prezentat în partea dreaptă a imaginii.

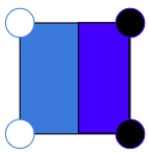




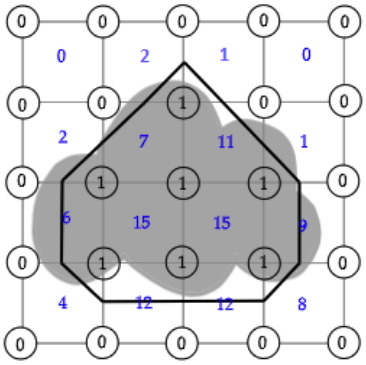
Configurația fiecărui pătrat din grid este potrivită cu o intrare în tabela de căutare a contururilor din imaginea de mai jos.



Pentru exemplul de mai sus în care rezultase codul binar 0110, conturul echivalent este cel din imaginea de mai jos.



Astfel, pe baza gridului binar calculat anterior și al tabelei de contururi, rezultatul final al aplicării algoritmului Marching Squares pe imaginea inițială este cel din imaginea de mai jos.



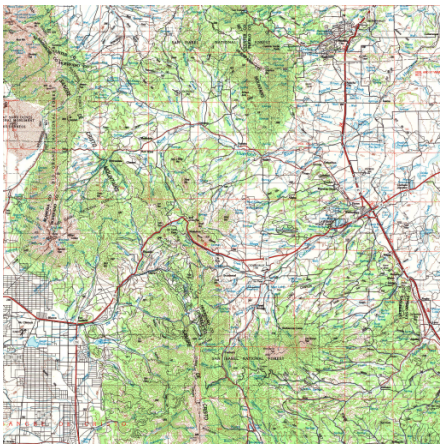
Detalii adiționale despre Marching Squares, pseudocodul, precum și imaginile prezentate mai sus, se pot găsi la această adresă.

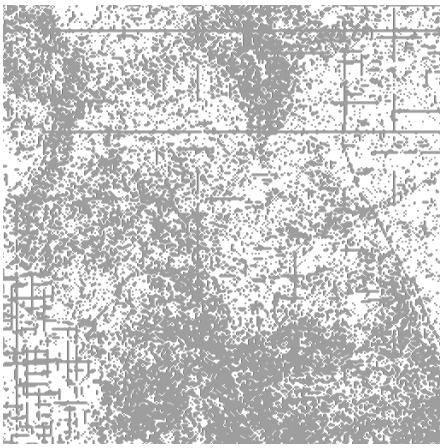
Detalii tehnice

În cadrul acestei teme, aveți deja implementat algoritmul Marching Squares într-o variantă secvențială, scopul fiind de a-l paraleliza folosind Pthreads în C/C++. Implementarea secvențială o găsiți în repository-ul temei, în directorul **src** (fișierele **tema1_par.c**, **helpers.c** și **helpers.h**). Cei doi pași ai algoritmului (crearea grid-ului, respectiv identificarea contururilor din imaginea finală) se găsesc în funcțiile *sample_grid()* și *march()*.

În implementarea secvențială pe care o aveți la dispoziție, pentru uniformitate și un sampling adecvat, toate imaginile cu dimensiuni mari sunt scalate la 2048x2048 pixeli. Acest lucru se realizează prin intermediul funcției *rescale_image()* din fișierul **tema1_par.c**. Pentru redimensionarea imaginii, se folosește interpolare bicubică, despre care puteți afla mai multe detalii aici. Un alt element important de menționat este faptul că imaginile cu care se lucrează sunt în format PPM.

Exemple de câte o imagine de intrare (stânga) și de ieșire (dreapta) se pot observa în imaginea de mai jos.





Implementarea paralelă

Scopul vostru este de a implementa varianta paralelă a algoritmului Marching Squares folosind Pthreads în C/C++, pornind de la implementarea secvențială, pe baza scheletului aflat în fișierul **tema1_par.c** din directorul **src** al repository-ului temei. În același director, găsiți și cele două fișiere cu funcții ajutătoare, pe

care le puteți folosi, dar nu le puteți modifica (ele vor fi suprascrise automat de checker cu variantele originale).

Programul paralel se va rula în felul următor:

```
./tema1_par <fisier_de_intrare> <fisier_de_iesire> <nr_threaduri>
```

Atenție! Trebuie să porniți toate thread-urile într-o singură iterație a thread-ului principal. Nu se acceptă mai multe perechi de *pthread_create()/pthread_join()* în cod, acest lucru ducând la un punctaj de **0** pe întreaga temă.

Notare

Tema se poate testa local sau pe VMChecker, după cum se explică mai jos. Tema se va încărca pe Moodle sub forma unei arhive Zip care, pe lângă fișierele sursă C/C++, va trebui să conțină următoarele două fișiere **în rădăcina arhivei**:

- *Makefile* - cu directiva *build* care compilează tema voastră (**fără flag-uri de optimizare**) și generează un executabil numit *tema1_par* aflat în rădăcina arhivei, și directiva *clean* care șterge executabilul
- *README* - fișier text în care să se descrie pe scurt implementarea temei.

Punctajul este divizat după cum urmează:

- **64p** - scalabilitatea soluției
- **56p** - corectitudinea rezultatelor (la rulări multiple pe aceleași date de intrare, trebuie obținute aceleași rezultate)[1]
- **30p** - claritatea codului și a explicațiilor din README.

Nerespectarea următoarelor cerințe va duce la depunctări:

- **-150p** - pseudo-sincronizarea firelor de execuție prin funcții cum ar fi `sleep()`
- **-150p** - utilizarea altor implementări de thread-uri în afară de Pthreads (cum ar fi *std::thread* din C++11) sau neutilizarea deloc a Pthreads
- **-150p** - crearea și oprirea de thread-uri în mod repetat (pentru a nu lua această depunctare, trebuie să creați thread-urile o singură dată la început)
- **-150p** - pornirea a mai multe/mai puține thread-uri decât se cer
- **-150p** - utilizarea de flag-uri de optimizare în Makefile
- **-30p** - utilizarea variabilelor globale (soluția pentru a evita variabile globale este să trimiteți variabile și referințe la variabile prin argumentele funcției pe care o dați la crearea firelor de execuție).

Atenție! Checker-ul vă dă cele 120 de puncte pentru scalabilitate și corectitudinea rezultatelor, dar acela nu este punctajul final (dacă, de exemplu, temă vă scalează și dă rezultate corecte, dar ați utilizat altă implementare de thread-uri pe lângă Pthreads, veți avea nota finală 0).

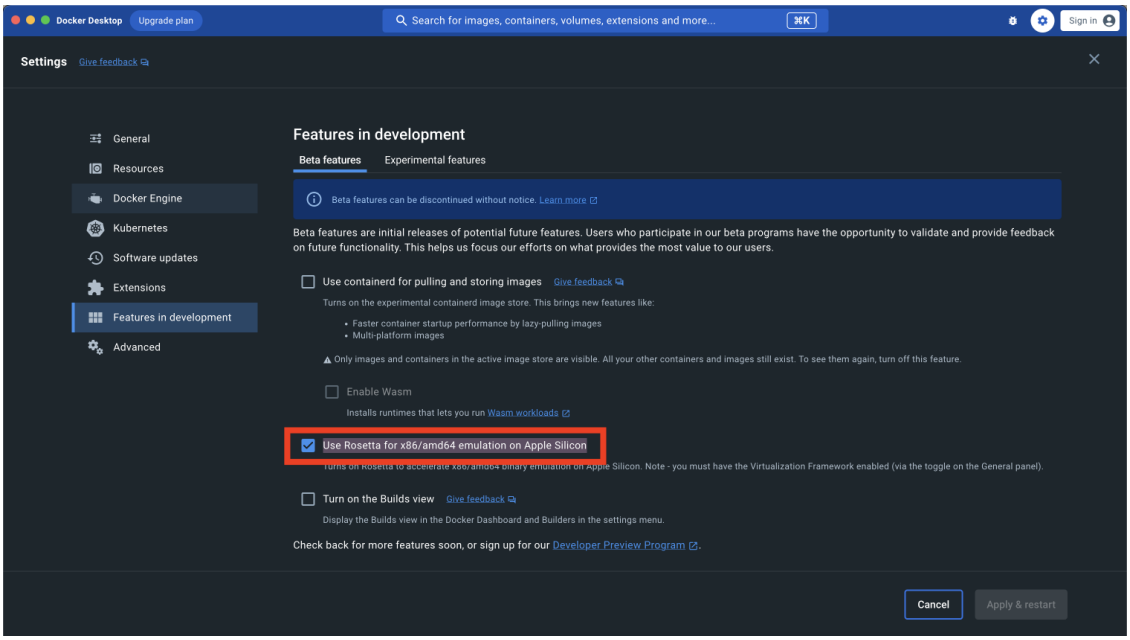
[1] Acest punctaj este condiționat de scalabilitate. O soluție secvențială, deși funcționează corect și dă rezultate bune, nu se va puncta.

Testare

Pentru a vă putea testa tema local, găsiți în repository-ul temei un set de fișiere de intrare de test, precum și un script Bash (numit *local.sh*) pe care îl puteți rula pentru a vă verifica implementarea. Rularea locală în acest mod necesită existența Docker pe sistemul vostru. Script-ul din repository va fi folosit și pentru testarea automată pe VMChecker Next, în fix aceleași condiții[2]. Scriptul de testare locală se rulează astfel:

```
$ ./local.sh checker
```

Atenție! Dacă aveți un calculator cu cipul M1 și vreți să rulați local, va trebui să activați opțiunea *Use Rosetta for x86/amd64 emulation on Apple Silicon* care se găsește în meniul *Settings* din Docker Desktop. Mai multe detalii puteți găsi aici.



La rulare, scriptul execută următorii pași:

1. compilează programul
2. pentru 7 teste, execută următoarele operații:
   (a) rulează implementarea secvențială ca etalon
   (b) rulează implementarea paralelă cu 2/4 thread-uri și verifică dacă rezultatele sunt corecte
   (c) pentru ultimele două teste, calculează accelerația și verifică dacă este peste niște praguri minime date
3. se calculează punctajul final din cele 120 de puncte alocate testelor automate (30 de puncte fiind rezervate pentru claritatea codului și a explicațiilor, așa cum se specifică mai sus).

Atenție! Dacă aveți un calculator cu două core-uri fizice (cu sau fără hyper-threading) și vreți să rulați local, va trebui să modificați măsurarea accelerației să nu ia în considerare și testele cu 4 thread-uri, pentru că implementarea paralelă nu va scala.

Pentru testarea folosind VMChecker Next, găsiți detalii într-un document intitulat "Tutorial VMChecker Next" aflat pe pagina de Moodle a cursului de APD.

[2]Nota obținută în urma rulării automate poate fi scăzută pe baza elementelor de depunctare descrise mai sus.

Link-uri utile

1. Drawing Shapes with Marching Squares
2. Marching squares
3. Image Processing – Bicubic Interpolation
4. PPM file format