Paralelno računanje Mandelbrot seta

Josip Delač Filip Zlopaša Fran Pećnjak Mijo Rajič Danijel Tumir

***Abstract*—Ovaj projekt rješava izazove računanja i vizualizacije povezane s Mandelbrotovim skupom, zanimljivim kompleksnim fraktalom koji nastaje iterativnim procesom temeljenim na jednostavnim matematičkim pravilima. Beskonačna složenost i ljepota Mandelbrotovog skupa leže u otkrivanju ponavljajućih detalja na svakoj razini povećanja. Naša predložena rješenja uključuju implementaciju računanja i iscrtavanja Mandelbrotovih fraktala u programskom jeziku C++, uz korištenje OpenMP API-ja za paralelizaciju i SDL2 (Simple DirectMedia Layer 2) biblioteke za renderiranje grafike i upravljanje prozorima. Paralelizacijom računanja fraktala cilj nam je poboljšati učinkovitost programa i ubrzati njegovo izvođenje. Pritom uspoređujemo brzinu izvođenja s neparaleliziranom verzijom istog programa.**

***Index Terms*—Mandelbrot set, paralelizirano računanje fraktala, grafička reprezentacija fraktala**

I. Uvod

Mandelbrotov skup, kao kompleksni fraktal, predstavlja izazov u računanju i vizualizaciji zbog svoje beskonačne složenosti i estetske privlačnosti. Ovaj projekt stavlja fokus na rješavanje problema računanja i iscrtavanja Mandelbrotovog skupa kroz implementaciju u programskom jeziku C++. Korištenjem OpenMP API-ja za paralelizaciju te SDL2 biblioteke za upravljanje grafikom i prozorima, cilj nam je postići učinkovito računanje fraktala i istražiti optimalnu strategiju za vizualno predstavljanje. Kroz eksperimentiranje s različitim pristupima želimo pridonijeti razumijevanju ravnoteže između računske efikasnosti i realno-vremenskog iscrtavanja, uz naglasak na prilagodljivosti programa različitim hardverskim konfiguracijama.

A. Pozadina problema

Mandelbrotov skup, otkriven od strane francuskog matematičara Benoita B. Mandelbrota, predstavlja jedinstvenu matematičku strukturu koja se formira iterativnim procesom prema jednostavnom pravilu. Ovaj kompleksni fraktal fascinira svojom beskonačnom složenosti, otkrivajući detalje koji se ponavljaju u beskonačnosti na svakoj razini povećanja.

Računanje i vizualizacija Mandelbrotovog skupa predstavljaju izazovne zadatke, osobito kada se želi postići visoka učinkovitost. U ovom kontekstu, programski jezik C++ pruža snažan okvir za numeričke izračune, dok se OpenMP API koristi za paralelizaciju, čime se ubrzava proces računanja fraktala. Dodatno, SDL2 biblioteka olakšava manipulaciju grafikom i upravljanje prozorima, što je ključno za stvaranje vizualno privlačnih reprezentacija Mandelbrotovog skupa.

Osnovna formula koja generira Mandelbrotov skup je iterativna relacija:

$$Z_{n+1} = Z_n^2 + C$$

gdje Z_n i C predstavljaju kompleksne brojeve, a n označava iteraciju. Skup se formira promatranjem ponašanja ovih iteracija, pri čemu se kompleksni broj C mijenja unutar određenog raspona vrijednosti.

B. Teorijski koncepti

Generiranje slike Mandelbrotovog skupa temelji se na matematičkim izračunima i konceptima fraktala. Osnovna formula $Z_{n+1} = Z_n^2 + C$ igra ključnu ulogu u procesu stvaranja vizualnih reprezentacija. Ovaj iterativni postupak prati ponašanje kompleksnih brojeva Z_n kroz niz koraka, pri čemu se konstanta C pridružuje svakom pikselu na ravnini. Ovisno o tome kako iteracije evoluiraju, određuje se pripadnost točke Mandelbrotovom skupu.

Tipično, postupak generiranja slike s Mandelbrotovim skupom uključuje iteriranje kroz svaki piksel na ravnini, dodjeljujući odgovarajuću konstantu C i prateći ponašanje iteracija kako bi se odredila boja piksela. Boje se često dodjeljuju prema broju iteracija potrebnih da bi se apsolutna vrijednost Z_n povećala izvan nekog unaprijed određenog praga. reprezentacija.

C. Povezani radovi

1. Mandelbrot, B. B. (1983). *The Fractal Geometry of Nature.* W. H. Freeman and Company.

2. Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press.

3. Roettger, S., & Karlsson, S. (2008). *Real-Time Rendering of the Mandelbrot Set.* Proceedings of the 2008 Conference on Graphics Interface, 23-30.

4. Grewal, M. S., & Andrews, A. P. (2015). *Kalman Filtering: Theory and Practice Using MATLAB.* John Wiley & Sons.

5. OpenMP Architecture Review Board. (2018). *OpenMP Application Program Interface Version 5.0.*

6. Peitgen, H. O., Saupe, D., & Hege, H. C. (1986). *The Science of Fractal Images.* Springer.

7. Bourke, P. (1991). *The Mandelbrot Set.* Available at: http://paulbourke.net/fractals/mandelbrot/

8. Hart, J. C. (1997). *Computer Recreations: A Computer Scientist's View of the Mandelbrot Set.* Scientific American, 276(4), 102-107.

9. Werner, A. (2006). *GPU-Based Interactive Visualization of Massive Fractals.* IEEE Transactions on Visualization and Computer Graphics, 12(5), 1067-1074.

10. Karperien, A., & Barenbrug, B. (2015). *GPU-Based Rendering of Fractal Terrains.* Computers & Graphics, 47, 15-26.

II. Implementacija

Računanje Mandelbrot seta

```
#include "fractals.hpp"

int mandelbrot_set(long double x, long
    ↪ double y, int max_iter) {
    int iterations = 0;
    long double cur_x = 0;
    long double cur_y = 0;

    while (cur_x*cur_x + cur_y*cur_y < 4 &&
        ↪  iterations < max_iter) {
        long double xtemp = cur_x*cur_x -
            ↪ cur_y*cur_y + x;
        cur_y = 2*cur_x*cur_y + y;
        cur_x = xtemp;
        iterations++;
    }

    return iterations;
}

std::pair<long double, long double>
    ↪ get_geometry_pos_from_pixels(int x,
    ↪ int y, Camera* camera) {
    long double xg = camera->zoom * (long
        ↪ double)x + camera->pos.first;
    long double yg = camera->zoom * (long
        ↪ double)y + camera->pos.second;
    return {xg, yg};
}
```

Opis: Ovaj kod pruža implementaciju računanja Mandelbrotovog skupa, klasičnog fraktala u matematici i računalnoj grafici. Funkcija `mandelbrot_set` prima tri parametra: `x` i `y` predstavljaju koordinate na kompleksnoj ravnini, dok `max_iter` označava maksimalni broj iteracija.

Unutar `while` petlje, funkcija izračunava iteracije pomoću Mandelbrotovog iterativnog postupka. Koristi kompleksne brojeve `cur_x` i `cur_y` koji se iterativno ažuriraju prema Mandelbrotovoj formuli. Ako kvadrat zbroja `cur_x` i `cur_y` prelazi 4 ili broj iteracija dosegne maksimalni broj (`max_iter`), petlja se prekida, a funkcija vraća broj iteracija.

Osim toga, kod uključuje funkciju `get_geometry_pos_from_pixels`, koja prima koordinate piksela (`x` i `y`) i objekt kamere (`Camera*`). Ova funkcija mapira piksele na geometrijske pozicije u kompleksnoj ravnini prema postavkama kamere, uključujući i faktor zumiranja. Rezultat je par vrijednosti koje predstavljaju stvarne geometrijske koordinate na Mandelbrotovoj ravnini.

A. Dodjela boja pri iscrtavanju fraktala

```
std::vector<int> get_color(int iteration,
    ↪ int max_iterations){
    // Normaliziramo broj iteracija u
        ↪ rasponu [0, 1]
    float p = (float)iteration / (float)
        ↪ max_iterations;

    // Inicijalizacija boja i postotka
    std::vector<int> color1;
    std::vector<int> color2;
    float percentage = 1.0f;

    // Odabir boje ovisno o normaliziranom
        ↪ broju iteracija
    if (p < 0.16)
    {
        color1 = {0, 7, 100};
        color2 = {32, 107, 203};
        percentage = p / 0.16;
    } else if (p < 0.42)
    {
        color1 = {32, 107, 203};
        color2 = {237, 255, 255};
        percentage = p / 0.42;
    } else if (p < 0.6425)
    {
        color1 = {237, 255, 255};
        color2 = {255, 170, 0};
        percentage = p / 0.6425;
    } else if (p < 0.8575)
    {
        color1 = {255, 170, 0};
        color2 = {0, 2, 0};
        percentage = p / 0.8575;
    } else if (p <= 1.0) {
        color1 = {0, 2, 0};
        color2 = {0, 7, 100};
        percentage = p / 1.0;
    }

    // Linearna interpolacija izmeu dvije
        ↪ boje ovisno o postotku
    return {(int)(color1[0] * percentage +
        ↪ color2[0] * (1.0 - percentage)),
            (int)(color1[1] * percentage +
                ↪ color2[1] * (1.0 -
                ↪ percentage)),
            (int)(color1[2] * percentage +
                ↪ color2[2] * (1.0 -
                ↪ percentage))};
}
```

Funkcija `get_color` koristi se za dodjelu boja pikselima ovisno o broju iteracija u Mandelbrotovom skupu. Parametri funkcije su `iteration`, koji označava trenutni broj iteracija, i `max_iterations`, koji predstavlja maksimalni broj iteracija.

Funkcija normalizira broj iteracija u rasponu od 0 do 1 kako bi odredila postotak završene iteracije. Zatim koristi taj postotak za odabir boje piksela na temelju definiranih pragova.

Svaki prag definira raspon normaliziranog broja iteracija gdje se koristi određeni skup boja.

Boje se linearno interpoliraju između dvije definirane boje ovisno o postotku završene iteracije. Rezultat funkcije je vektor s trostrukim vrijednostima koje predstavljaju RGB komponente boje piksela.

Ova funkcija doprinosi vizualnom prikazu Mandelbrotovog skupa dodjeljujući pikselima boje koje odražavaju broj iteracija i time pridonose raznolikosti i estetici generiranog fraktala.

B. Paralelizacija rješenja

```
#define n_of_iterations 255

int* pixel_colors = nullptr;

void draw_screen(int width, int height,
    ↪ Camera* camera, SDL_Renderer*
    ↪ gRenderer){

   if (pixel_colors == nullptr)
   {
      pixel_colors = new int[width *
          ↪ height * 3];
   }

   #pragma omp parallel for num_threads(
       ↪ width)
   for (int x = 0; x < width; x++)
   {
      for (int y = 0; y < height; y++)
      {
         std::pair<long double, long
             ↪ double> pos =
             ↪ get_geometry_pos_from_pixels
             ↪ (x - width/2, y - height/2,
             ↪  camera);
         int iterations = mandelbrot_set(
             ↪ pos.first, pos.second,
             ↪ n_of_iterations);
         std::vector<int> color =
             ↪ get_color(iterations,
             ↪ n_of_iterations);
         pixel_colors[(x * height + y) * 3
             ↪  + 0] = color[0];
         pixel_colors[(x * height + y) * 3
             ↪  + 1] = color[1];
         pixel_colors[(x * height + y) * 3
             ↪  + 2] = color[2];
      }
   }

   for (int x = 0; x < width; x++)
   {
      for (int y = 0; y < height; y++)
      {
         draw_pixel(x, y, &pixel_colors[(x
             ↪  * height + y) * 3],
             ↪ gRenderer);
      }
   }
}
```

Funkcija `draw_screen` odgovorna je za iscrtavanje ekrana prikaza Mandelbrotovog skupa. Koristi se paralelizacijom pomoću OpenMP-a kako bi poboljšala performanse.

Najprije provjerava je li polje `pixel_colors` već alocirano. Ako nije, alocira ga.

Unutar paralelizirane petlje, funkcija prolazi kroz sve piksele na ekranu (`width` x `height`) i za svaki piksel izračunava boju na temelju Mandelbrotovog skupa. Paralelizacija se postiže pomoću OpenMP-a, što omogućuje ubrzanje izračuna boja za svaki piksel.

Nakon dobivanja boje, rezultati se spremaju u polje `pixel_colors` na odgovarajućem indeksu. Konačno, funkcija prolazi kroz sve piksele i poziva funkciju `draw_pixel` za svaki kako bi se boje prikazale na ekranu.

Ova funkcija optimizira izračun boja pomoću paralelizacije, pridonoseći učinkovitosti pri generiranju Mandelbrotovog skupa.

III. Evaluacija rezultata

A. Načini evaluacije

Proučavamo performanse razvijenog sustava za generiranje i prikazivanje Mandelbrotovog skupa kroz različite metrike. Analiziramo vrijeme izvođenja, memorijsku potrošnju te kvalitetu i estetiku generiranih slika kako bismo stekli dublje razumijevanje efikasnosti sustava.

B. Usporedba brzine izvođenja

Fokusiramo se na utjecaj paralelizma na brzinu iscrtavanja fraktala. Nakon pažljivog promatranja, primijetili smo značajan porast brzine iscrtavanja, čak do četiri puta, nakon implementacije paralelizma u algoritam. Ova optimizacija značajno poboljšava performanse sustava, omogućujući brže generiranje fraktala i unapređujući ukupno korisničko iskustvo.

Detaljnim zapažanjem došli smo do zaključka kako je uvođenje paralelizma značajno ubrzalo iscrtavanje fraktala, pridonoseći ukupnoj učinkovitosti razvijenog sustava.