UNIVERZITA SV. CYRILA A METODA V TRNAVE
FAKULTA PRÍRODNÝCH VIED

Mandelbrot Set

Semestrálna práca

2023 **Samuel Vaský**

Obsah

1. Mandelbrot Set

Mandelbrotov set je fraktál, ktorý bol objavený matematikom **Benoitom Mandelbrotom** v roku 1979. Je to zaujímavý a vizuálne pôsobivý fraktálny obrazec, ktorý sa skladá z nekonečného počtu detailov. Mandelbrotov set sa skladá z množiny bodov v komplexnej rovine, ktoré majú špeciálnu vlastnosť. Táto vlastnosť sa prejavuje tak, že ak vezmeme určitý bod z tejto množiny a aplikujeme naň určitú iteratívnu funkciu, tak táto funkcia sa bude približovať nekonečne blízko k nekonečnu. Ak funkcia dosiahne nekonečno, tak je tento bod súčasťou Mandelbrotovho setu. Ak funkcia ostane ohraničená, tak tento bod súčasťou Mandelbrotovho setu nie je.

Mandelbrotov set sa často zobrazuje v podobe farebnej fraktálnej grafiky, kde každá farba predstavuje rôzne iterácie. Táto grafika ukazuje neuveriteľný detail a krásu tohto fraktálu a je často používaná na výučbu matematiky a na výskum chaotických systémov.

1.1 Počítanie Mandelbrot Setu

Pre každý bod v mriežke (reprezentovaný ako komplexné číslo c) vypočítať iteratívny vzorec zvaný Mandelbrotova iterácia: z_{n+1} = z_n^2 + c, kde z_0 = 0 + 0i (komplexné číslo reprezentujúce bod v mriežke). Iteratívne vypočítať hodnoty z_{n+1} pre každú iteráciu až do dosiahnutia maximálneho počtu iterácií alebo kým veľkosť hodnoty |z_n| neprekročí určitú hranicu. Ak veľkosť |z_n| prekročí hranicu, potom sa považuje, že bod nie je súčasťou množiny. Ak sa počet iterácií pre daný bod nedosiahol maximálneho limitu a hodnota |z_n| neprekročila hranicu, potom sa považuje, že bod je súčasťou množiny.

Po dokončení výpočtu pre každý bod v mriežke zobraziť pomocou farebnej mapy, kde farby závisia na počte iterácií potrebných na určenie, či je bod súčasťou množiny alebo nie.

2. Implementácia

Na implementáciu sme použili programovací jazyk C++, grafické API OpenGL a kompilátor CLang. Projekt bol písaný na operačnom systéme MacOS ale keďže sme na vytvorenie projekte použili CMake, nemal by byť problém skompilovať projekt aj na inom operačnom systéme.

2.1 Setup

Pred začiatkom implementácie je potrebné nainštalovať nasledujúce knižnice.

- **CMake** – Štruktúra projektu, generovanie MakeFile. Požadovaná verzia 3.26.1. Inštalácia pomocou package manageru Brew → **brew install cmake**
- **ImGUI** – GUI knižnica pre OpenGL napísana v C++. Stiahnuť zdrojový kód z https://github.com/ocornut/imgui a linknúť cez CMake.
- **ImPlot** – Rozšírenie pre ImGUI, pridá funkcionalitu vykresľovať grafy. Zdrojový kód dostupný na https://github.com/epezent/implot .
- **GLFW3** – Knižnica na vytváranie okna pre OpenGL a registrovanie vstupu z klávesnice alebo myši. Inštálacia → **brew install glfw**
- **GLEW** - Načítavanie OpenGL rozšírení → **brew install glew**
- **OMP** – Open MP → **brew install libomp** . Brew neodporúča robiť symlink preto je potrebné do CMake zadať cestu ku lib aj include, v mojom prípade to bolo /opt/homebrew/opt/libomp/lib .
- **MPI** – Open MPI → **brew install open-mpi**
- **STB** – Knižnica na prácu s obrázkami. Dostupné na https://github.com/nothings/stb . Používame túto knižnicu na export OpenGL buffera to obrázka formátu PNG. Stačí stiahnuť „stb_image_write.h“ a includnúť do projektu.

Celý CMake projekt sa generuje pomocou súboru „CMakeLists.txt“ kde je potrebné všetky knižnice linknút. V prípade, že sa cesta ku knižniciam na vašom zariadení líši je potrebné tento súbor upraviť.

2.2 Štruktúra projektu :

- /bin → binárka
- /src → zdrojový kód
- /include → hlavičkové súbory
- /shaders → shadre pre OpenGL
- /imgui → knižnica ImGUI
- /implot → knižnica ImPlot
- /mpi_src → zdrojový kód pre MPI workera

2.3 Kompilácia

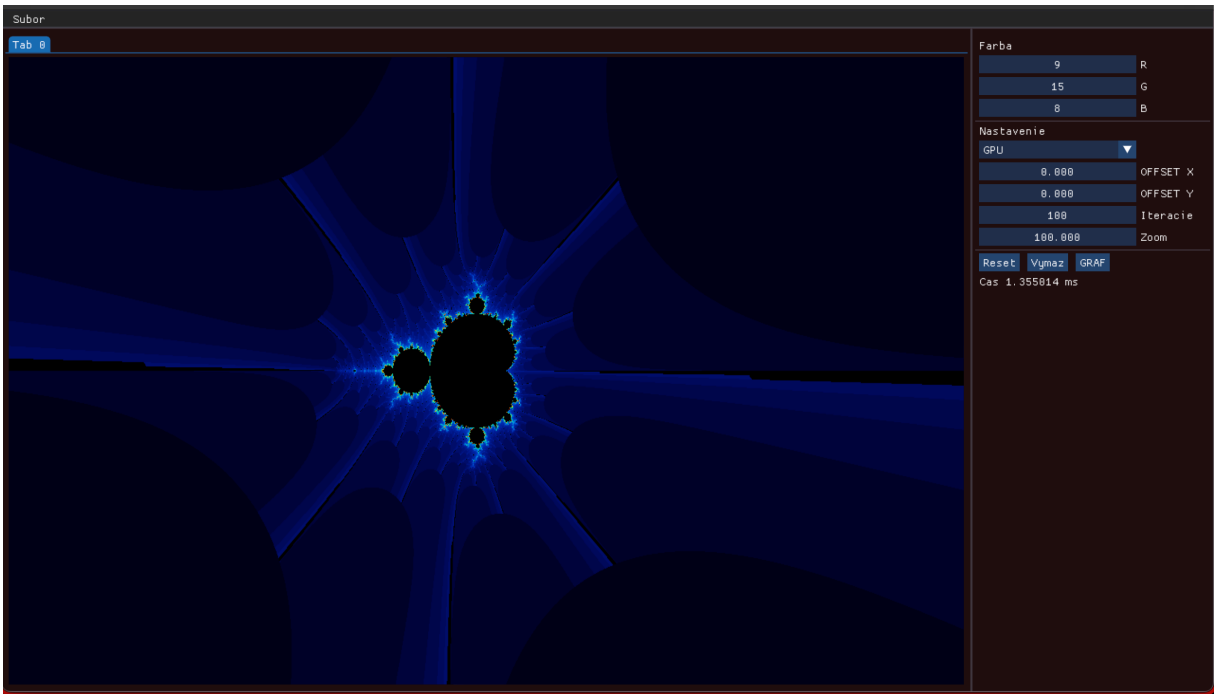
- cmake . (V roote projektu kde sa nachádza súbor „CMakeLists.txt“)
- mpicxx mpi_src/main.cpp -o main.o (Kompilácia MPI workera)
- make
- ./bin/fraktal (Spustenie aplikácie)

2.4 Core

Pomocou glfw3 a ImGUi sme vytvorili okno s Gui pomocou ktorého je možné nastavovať parametre pre Mandelbrot Set. V programe je možné vytvárať nové taby. Každý tab reprezentuje nezávislý MandelBrot set. Vytvorenie tabu cez hlavné menu Súbor -> Nový Tab. Taktiež je možné exportovať render aktuálne zvoleného tabu Súbor -> Export -> zadať názov.

Ovládanie/Nastavovanie Mandelbrot Setu je možné cez Input Boxy na pravej strane obrazovky. Okrem toho sa dá program ovládať aj pomocou klávesnice a myši.

- Scroll Left/Right → Nastavovanie Offsetu
- Shift + Scroll → Zoom



Obrázok 1. Obrazovka Aplikácie

2.5 Mandelbrot Set na GPU

Kalkulácia MandelBrot setu na GPU beží na OpenGL fragment shadery. Rendrovanie prebieha do framebuffera ktorý sa následne zobrazí na obrazovke v GUI okne. Implementácia vyzerá následovne :

```
float mandelbrot(vec2 c) {
    vec2 z = vec2(0.0,0.0);
    for(int i = 0; i < iteracie; i++ ) {
        vec2 znew;
        znew.x = (z.x * z.x ) - (z.y * z.y ) + c.x;
        znew.y = ( 2.0 * z.x * z.y) + c.y;
        z = znew;
        if((z.x * z.y) + (z.y * z.y) > threshold) {
            break;
```

```
        }
        n++;
    }
    return n/float(iteracie);
}

vec4 farby(float t) {
    float r = color_r * (1.0 - t) * t * t * t;
    float g = color_g * (1.0 - t) * (1.0 - t) * t * t;
    float b = color_b * (1.0 - t) * (1.0 - t) * (1.0 - t) * t;
    return vec4(r, g, b, 1.0);
}

void main()
{
    vec2 coord = vec2(gl_FragCoord.xy);
    coord = vec2(coord.x - screen_w/2, coord.y - screen_h/2);
    coord = coord/zoom;
    coord = vec2(coord.x - offset_x, coord.y-offset_y);

    float mb = mandelbrot(coord);
    FragColor = farby(mb);
}
```

Funkcia „float mandelbrot“ kalkujue MandelBrot set pre súradnicu c a následne jej funkcia farby pridelí farbu. FragColor je výstup zo shadera.

2.6 Mandelbrot cez OMP

Paralelizácia MandelBrotu Setu cez OMP. OMP vlákna si rozdelia foor loop. Výsledky sa zapíšu do OpenGL bufferu na vykreslenie.

```
#pragma omp parallel for num_threads(pocet)
    for(int i=0; i < g_get_screen_h(); i++) {
        for( int j=0; j< g_get_screen_w(); j++) {
            float cy = i - (float)g_get_screen_h()/2;
            float cx = j - (float)g_get_screen_w()/2;
            cx = cx / this->zoom;
            cy = cy / this->zoom;
            cx = cx - this->offset_x;
            cy = cy - this->offset_y;

            float mb = mandelbrot(cx, cy, this->iter);
            float r = farby_r(mb, this->get_r());
            float g = farby_g(mb, this->get_g());
            float b = farby_b(mb, this->get_b());

            int index_jedna = (i*g_get_screen_w() + j) * 3;
            int index_dva = (i*g_get_screen_w() + j) * 3 + 1;
            int index_tri = (i*g_get_screen_w() + j) * 3 + 2;
```

```
            image_data[index_jedna] = r;
            image_data[index_dva] = g;
            image_data[index_tri] = b;
        }
    }
```

2.7 Mandelbrot cez MPI

Rozdelenie na mastera a workerov. Master spúšta/vytvára X workerov, rozdelia si buffer a spracujú ho. Nakoniec cez MPI_Gather sa všetky data spoja a pošlú sa do OpenGL bufferu na rendrovanie. Worker funguje ako samostatný program. Jeho implementácia sa nachádza v mpi_src/main.cpp

Master:

```
MPI_Comm child;
int spawnError[255];
MPI_Comm_spawn("mpi_src/main.o", MPI_ARGV_NULL, this->get_omp_threads(),
MPI_INFO_NULL, 0, MPI_COMM_SELF, &child, spawnError);

int parent_id;
MPI_Comm_rank(MPI_COMM_WORLD, &parent_id);

MPI_Bcast(&parent_id,1,MPI_INT,MPI_ROOT,child);
MPI_Bcast(&w,1,MPI_INT,MPI_ROOT,child);
MPI_Bcast(&h,1,MPI_INT,MPI_ROOT,child);
MPI_Bcast(&iter,1,MPI_INT,MPI_ROOT,child);
MPI_Bcast(&zoom,1,MPI_FLOAT,MPI_ROOT,child);
MPI_Bcast(&off_x,1,MPI_FLOAT,MPI_ROOT,child);
MPI_Bcast(&off_y,1,MPI_FLOAT,MPI_ROOT,child);
MPI_Bcast(&c_r,1,MPI_INT,MPI_ROOT,child);
MPI_Bcast(&c_g,1,MPI_INT,MPI_ROOT,child);
MPI_Bcast(&c_b,1,MPI_INT,MPI_ROOT,child);
MPI_Bcast(&pocet,1,MPI_INT,MPI_ROOT,child);


float *image_data_sub;
MPI_Gather(image_data_sub, size/pocet, MPI_FLOAT, image_data, size,
MPI_FLOAT,MPI_ROOT,child);
```

Worker:

```
MPI_Bcast(&parent_id, 1, MPI_INT,0, parent);
MPI_Bcast(&screen_w, 1, MPI_INT,0, parent);
MPI_Bcast(&screen_h, 1, MPI_INT,0, parent);
MPI_Bcast(&iter, 1, MPI_INT,0, parent);
MPI_Bcast(&zoom, 1, MPI_FLOAT,0, parent);
MPI_Bcast(&offset_x, 1, MPI_FLOAT,0, parent);
```

```
MPI_Bcast(&offset_y, 1, MPI_FLOAT,0, parent);
MPI_Bcast(&r_color, 1, MPI_INT,0, parent);
MPI_Bcast(&g_color, 1, MPI_INT,0, parent);
MPI_Bcast(&b_color, 1, MPI_INT,0, parent);
MPI_Bcast(&pocet, 1, MPI_INT,0, parent);

const long size = screen_w * screen_h * 3;
float *image_data_sub = new float[size/pocet];
float *image_data;

int index =myid;
for(int i=0; i < (size/pocet); i+=3) {
     int e = (i  + index  * (size/pocet)) / 3  ;
    int x = (e ) % screen_w;
    int y = ((e ) / screen_w ) % screen_h  ;

    float cy = y - (float)screen_h/2;
    float cx = x - (float)screen_w/2;

    cx = cx / zoom;
    cy = cy / zoom;
    cx = cx - offset_x;
    cy = cy - offset_y;

   float mb = mandelbrot(cx, cy, iter);
   float r = farby_r(mb, r_color);
   float g = farby_g(mb, g_color);
   float b = farby_b(mb, b_color);

   image_data_sub[i] = r;
   image_data_sub[i+1] = g;
   image_data_sub[i+2] = b;
 }

MPI_Gather(image_data_sub, size/pocet, MPI_FLOAT, image_data, size, MPI_FLOAT,
parent_id, parent);
```

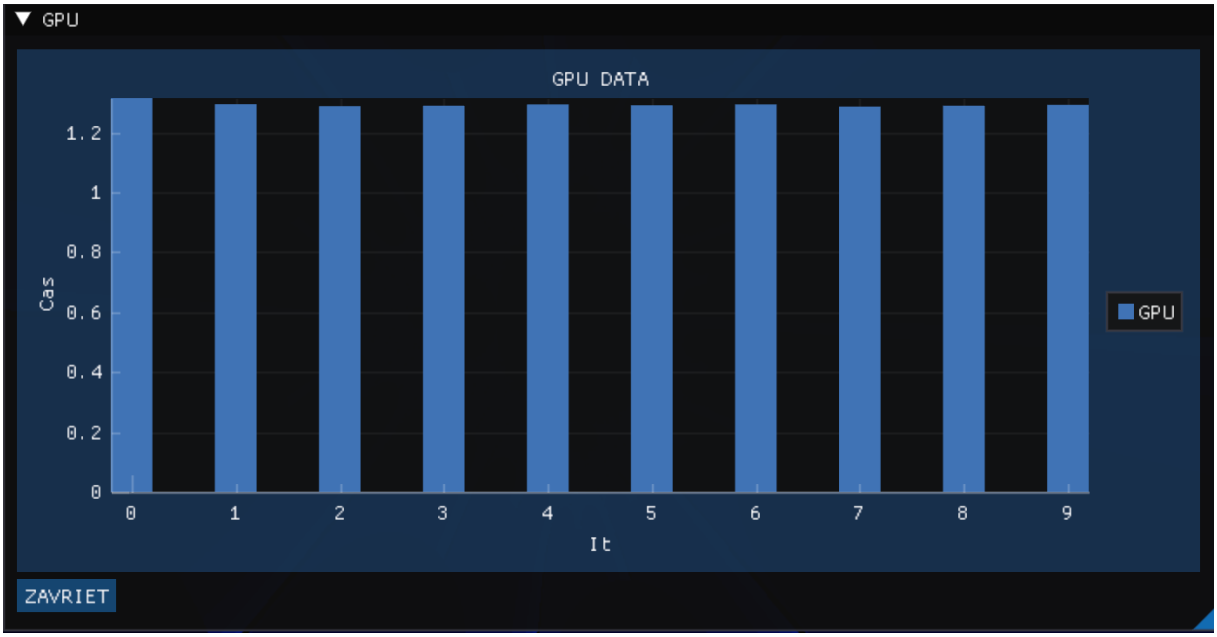
3. Meranie

Použité zariadenie : Macbook M1 Pro 14

Na meranie času sme použili C++ funkciu „high_resolution_clock“. Zistíme tak čas v ms na jednu iteráciu. Pre OMP sme merali rýchlosť iterácie pre 1,2,4,8,16,32,64,128 vlákien. Pre MPI 1,2,3 Procesov (Viac mi moje zariadenie nedovolilo vytvoriť) .

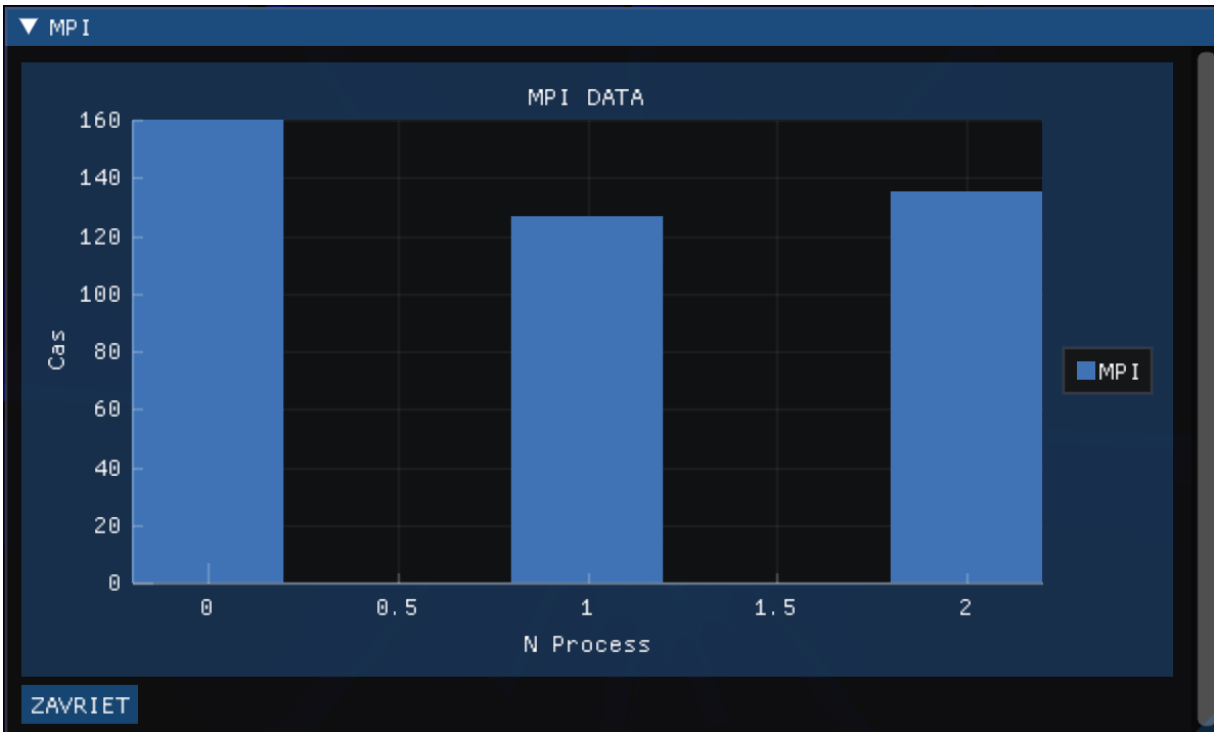
Pre GPU je takéto meranie nepresné, pretože sice nám OpenGL vratilo nejakú hodnotu, neznamená to, že vykresľovanie sa reálne uskutočnilo. Vždy keď pošleme grafickej karte príkaz aby niečo vykreslila sa tento príkaz dostane len do FIFO buffera a na radu sa dostane až za nejaký čas. Preto je pre grafickú kartu lepšie merať čas na vykreslenie jednej snímky spriemerovaný na jednu sekundu (https://www.khronos.org/opengl/wiki/Performance) . Práve z tohto dôvodu získanie grafu merania pre GPU trvá 10 sekúnd (10 iterácií) .

```
delta_time = glfwGetTime() * 1000 - prev_time;
frames++;

if (delta_time >= 1000.0) {
   if(g_get_active_mb_obj()->get_type() == 0) {
      g_get_active_mb_obj()->cas = delta_time/double(frames);
      if(g_get_active_mb_obj()->gpu_data.size() < 10 ) {
         g_get_active_mb_obj()->gpu_data.push_back(g_get_active_mb_obj()->cas);
      }
   }

   frames = 0;
   prev_time = glfwGetTime() * 1000;
}
```



Obrázok 2 - GPU Data

Obrázok 3 - OMP Data



Obrázok 4 - MPI Data

Záver

Z odmerných dát môžeme vidieť, že kalkulácia Mandelbrot setu na GPU je najrýchlejšia. Nasleduje OMP kde s vláknami čas klesá. Na poslednom mieste je MPI. Očakávane hodnoty pre MPI mali byť podobné OMP, pravdepodobné je za to zodpovedný broadcast pomocou ktorého posielame workerom data z mastera.