ŠOLSKI CENTER CELJE

Srednja šola za kemijo, elektrotehniko in računalništvo

# Optimizacija fraktalnih algoritmov

**raziskovalna naloga**

Avtor:

Nino Gašparuš, R-3.b

Mentor:

mag. Boštjan Resinovič, prof.

Mestna občina Celje, Mladi za Celje

ŠOLSKI CENTER CELJE

Srednja šola za kemijo, elektrotehniko in računalništvo

# Optimizacija fraktalnih algoritmov

**raziskovalna naloga**

Avtor:

Nino Gašparuš, R-3.b

Mentor:

mag. Boštjan Resinovič, prof.

Mestna občina Celje, Mladi za Celje

**IZJAVA***

Mentor/-ica Boštjan Resinovič v skladu z 20. členom Pravilnika o organizaciji mladinske raziskovalne dejavnosti »Mladi za Celje« Mestne občine Celje, zagotavljam, da je v raziskovalni nalogi z naslovom Optimizacija fraktalnih algoritmov, katere avtor/-ica je Nino Gašparuš:

- besedilo v tiskani in elektronski obliki istovetno,
- pri raziskovanju uporabljeno gradivo navedeno v seznamu uporabljene literature,
- da je za objavo fotografij v nalogi pridobljeno avtorjevo dovoljenje in je hranjeno v šolskem arhivu,
- da sme Osrednja knjižnica Celje objaviti raziskovalno nalogo v polnem besedilu na knjižničnih portalih z navedbo, da je raziskovalna naloga nastala v okviru projekta Mladi za Celje,
- da je raziskovalno nalogo dovoljeno uporabiti za izobraževalne in raziskovalne namene s povzemanjem misli, idej, konceptov oziroma besedil iz naloge ob upoštevanju avtorstva in korektnem citiranju,
- da smo seznanjeni z razpisni pogoji projekta Mladi za Celje.

Celje, 8.4.2024

žig šole

Podpis mentorja

Podpis odgovorne osebe

ZA ANITA LAZNIK:

*

POJASNILO

V skladu z 20. členom Pravilnika raziskovalne dejavnosti »Mladi za Celje« Mestne občine Celje je potrebno podpisano izjavo mentorja (-ice) in odgovorne osebe šole vključiti v izvod za knjižnico, dovoljenje za objavo avtorja (-ice) fotografskega gradiva, katerega ni avtor (-ica) raziskovalne naloge, pa hrani šola v svojem arhivu.

# ZAHVALA

Zahvaljujem se vsem, ki so mi kakor koli pomagali pri izdelavi raziskovalne naloge.

Velika zahvala gre profesorici Helenci Klepej Viher, za razlage matematičnih tem, prispevanje raznih idej ter spodbudo med izdelavo naloge.

Zahvaljujem se tudi mag. Boštjanu Resinoviču za spodbudo vseskozi nastanek naloge in njen končni strokovni pregled.

// POVZETEK
# POVZETEK

Fraktali so predmeti neskončne kompleksnosti. Pojavljajo se tako v abstraktnem svetu matematike kot tudi v zapletenih vzorcih narave. Gotovo najslavnejši fraktal je Mandelbrotova množica. Gre za množico kompleksnih števil, pridobljeno z iterativnim izračunom fraktalne formule.

Izris fraktalnih oblik je računsko izjemno zahtevna naloga, zato so optimizacije ključnega pomena za hiter in odziven program. Za izris Mandelbrotovega fraktala je potrebno izvesi več miljard računskih operacij, ki terjajo svoj čas. Računski postopek lahko pospešimo z učinkovitejšo rabo računalniških virov, matematično poenostavitev problema ter uporabo lastnosti fraktalov.

Raziskovalna naloga se osredotoča na matematično in računalniško opimizacijo algoritmov, zadolženih za njihov izris. Cilj naloge je bilo ustvariti idealen algoritem, ki bi bil zmožen izračunati ter izrisati Mandelbrotov fraktal vsaj 30 -krat na sekundo ter z njim ustvariti interaktivni program, ki omogoča prosto raziskovanje Mandelbrotovega fraktala.

Rezultati kažejo, da končni program deluje pravilo. Sicer ni najprijaznejši, do uporabnika, uspešno izrisuje pri 30 ali več sličicah na sekundo.

**Ključne besede**: fraktal, iteracija, orbita, kompleksno število.

# ABSTRACT

Fractal - subject of infinite complexity, appearing both in the abstract world of mathematics as well as in intricate patterns of nature. Surely, the world's most famous fractal is the Mandelbrot set – a set of complex numbers obtained through the iterative computation of its fractal formula.

Drawing fractal shapes is a computationally expensive task, necessitating optimizations for a fast and responsive program. Rendering the Mandelbrot set involves several billion arithmetic operations, leading to significant computation times. The computation process can be optimized with more efficient utilization of computer resources, mathematical simplification of the task, and the application of fractal properties.

The paper focuses on the mathematical and computational optimization of algorithms responsible for rendering fractals. The research goal was to create an ideal algorithm capable of rendering the Mandelbrot set at least 30 times per second, using it to develop an interactive program that allows free exploration of the Mandelbrot fractal.

The results indicate that the developed program functions as intended. While it may not be the most user-friendly, it successfully renders at 30 frames per second or more.

**Keywords**: fractal, iteration,  orbit, complex number.

# KAZALO


PREDGOVOR....9
UVOD....10
STRUKTURA RAZISKAV....11
HIPOTEZE....11
FRAKTALI....12
  Samopodobe....13
  Ničelna ploščina....13
  Neskončen obseg....14
MANDELBROTOVA MNOŽICA....15
PROBLEM....16
MATEMATIČNE OPTIMIZACIJE....17
  POENOSTAVITEV IZRAČUNOV....17
  SIMETRIČNOST....19
SPLOŠNE OPTIMIZACIJE....21
  PREDČASNE PREKINITVE....21
    Konvergentne Točke....21
    Ciklične točke....24
  DELJENJE NA SEKCIJE....29
  DINAMIČNA LOČLJIVOST....32
  DELJENJE ITERACIJ....33
IZRAČUN KORAKA....35
RAČUNALNIŠKA OPTIMIZACIJA....37
  VEČNITNOST....37
  POSPEŠEVANJE Z GPE....40
  SIMD UKAZI....42
BARVNI ALGORITMI....44
  FRAGMENTNI SENČNIKI....48
ŠIRJENJE NA DRUGE FRAKTALE....48
UPORABLJENE TEHNOLOGIJE....49
PREDSTAVITEV APLIKACIJE....52
ANALIZA REZULTATOV....53
  ANALIZA HIPOTEZ....53
ZAKLJUČEK....55
VIRI IN LITERATURA....56
  Viri slik....57
PRILOGE....59
  Delovanje osnovnega programa....59
  Algoritem 1....61
  Delovanje končnega programa....61
  Sistemske specifikacije....62
    Sistem:....62
    Sistem 2:....62

# KAZALO SLIK

Slika 1: Sierpińskijev trikotnik....10
Slika 2: Sierpińskijeva preproga....10
Slika 3: Mandelbrotova množica....10
Slika 4: Fraktal goreča ladja....10
Slika 5: Praprot....10
Slika 6: Romanesco brokoli....10
Slika 7: Sierpińskijeva preproga pri 1, 2, 3 iteracijah....12
Slika 8: Mandelbrot, 1x povečava....13
Slika 9: Mandelbrot, 10800x povečava....13
Slika 10: Graf ploščine Sirpinskijeve preproge za a = 1....13
Slika 11: Kochova krivulja pri 1, 2, 3, 4, 5 iteracijah....14
Slika 12: Re[-2.8,1.2], Im[-1.125, 1.125] n=1....15
Slika 13: Re[-1.76, 0.16], Im[-0.54,0.54] n=5....15
Slika 14: Računski čas osnovnega algoritma, povprečje: 1538.14ms....16
Slika 15: Absolutna vrednost kompleksnega števila....17
Slika 16: Računski časi pooenostavljenega algoritma, povprečje: 1501.*1 ms*....18
Slika 17: Simetrala Mandelbrotovega fraktala....19
Slika 18: Računski čas s simetrijo poenostavljenega algoritma, povprečje: 825.06 ms....20
Slika 19: Orbtia c = 0.17+0.412i....21
Slika 20: Orbita c = -O.118-0.538i....21
Slika 21: Vizualizacija privlačnih točk....22
Slika 22: Orbita c = 0.2635 + 0.37195í, $It_{max}$ = 50....22
Slika 23: Y - Razdalja med dvema točkama, X – iteracija....22
Slika 24: Orbita c = -0.12561 + 0.47578í, $It_{max}$ = 50....23
Slika 25: Y - Razdalja med dvema točkama, X -iteracija....23
Slika 26: Orbita c = -0.31863 + 0.61319í, $It_{max}$ = 800....23
Slika 27: Y - Razdalja med dvema točkama, X -iteracija....23
Slika 28: Y - Razdalja med dvema točkama, X - iteracija....23
Slika 29: Orbita c = -0.20361 + 0.67105í, $It_{max}$ = 50....23
Slika 30: Primer ciklične orbite....24
Slika 31: Orbita prvega cikla....25
Slika 32: Orbita drugega cikla....25
Slika 33: Orbita tretjega cikla....26
Slika 34: Orbita četrtega cikla....26
Slika 35: Orbita petega cikla....26
Slika 36: Šesti cikel....27
Slika 37: Sedmi cikel....27
Slika 38: Osmi cikel....27
Slika 39: Idealne začetne vrednosti....27
Slika 40: Orbita primer 1....28
Slika 41: Orbita primer 2....28
Slika 42: Računski časi algoritma s predčasnimi prekinitvami, povprečje 514.12 ms....28
Slika 43: Točke z enakim $It_{esc}$....29

Slika 44: Sekcije....29
Slika 45: Slika s prenizkim številom $It_{max}$....30
Slika 46: Globina 5....30
Slika 47: Globina 1....30
Slika 48: Globina 8....31
Slika 49: Globina 11....31
Slika 50: Računski časi programa pospešenega z deljenjem na sekcije....31
Slika 51: Ločljivost 240 x 135....32
Slika 52: Ločljivost 320 x 180....32
Slika 53: Ločljivost 960 x 540....32
Slika 54: Ločljivost 480 x 270....32
Slika 55: Primer približka orbite....33
Slika 56: Računski časi programa z deljenjem iteracij, povprečje: 263.62 *ms*....34
Slika 57: Preslikavna funkcija....36
Slika 58: Računski časi algoritma z izračunanim korakom, povprečje: 1489.33 ms....36
Slika 59: Računski časi večnitnega programa....38
Slika 60: Računski časi algoritma z deljenim delom 1....39
Slika 61: Računski časi večnitnega algoritma z deljenim delom 2....39
Slika 62: Računski čas programa, izvedenega na GPE, povprečje: 57.3 ms....41
Slika 63: Vektorsko in skalarno procesiranje....42
Slika 64: Računski čas SIMD algoritma, povprečje: 229.2 ms....43
Slika 65: Primeri lepo obarvane Mandelbrotove množice; slike so posnete v programu [Xaos]....44
Slika 66: Barvna statik....45
Slika 67: Gladek barvni algoritem....45
Slika 68: $r = 2 \times t$, $g = (r/2) \times 0.22$, $b = (r/2) \times 0.33$....46
Slika 69: $r = 9 \times (1-x) \times x^3$ , $g = 15 \times (1-x)^2 \times x^2$ , $b = 8.5 \times (1-x)^3 \times x$....46
Slika 70: $r = 1/(-x) + 2$, $g = 2 \times x$, $b = (r+g) \times 0.5$....47
Slika 71: Računski časi barvnih algoritmov....47
Slika 72: KSF Osmica [Xaos]....48
Slika 73: KSF Feniks [Xaos]....48
Slika 74: Processing....49
Slika 75: Vim....49
Slika 76: C++....50
Slika 77: GCC....50
Slika 78: CUDA....50
Slika 79: SFML....51
Slika 80: Navodila za uporabo....52
Slika 81: Osnovni pogled....52

# UPORABLJENE KRATICE

**ALU –** arithmetics logics unit, sl. aritmetično logična enota.
**API –** **a**pplication programming interface. Programski vmesnik namenjen za olajšanje komunikacij med programsko in ali strojno opremo.
**CPE** – centralna procesna enota. Glavni del vsakega računalnika, osnovna komponenta, ki izvaja vse aritmetično/logične operacije ter komunicira z vsemi ostalimi komponentami rač. Sistema.
**FLOP** – floating point operation, merska enota za procesno moč računalniškega sistema ali posamezne komponente. Osnovna enota [TFLOP/s]. 1TFLOP/s = $10^{12}$ FLOP operacij na sekundo.
**GPE –** grafično procesna enota. Sestavni del računalnika, namenjen procesiranju vseh grafičnih nalog računalnika.
**$It_{max}$** – število iteracij, opravljenih, preden je točka definirana kot element M.
**$It_{esc}$** – iteracija, po kateri točka pobegne v neskončnost.
**KSF –** kompleksno številski fraktal. Množica kompleksnih števil, ki ob vizualizaciji ustvari fraktalno obliko.
**M** – mandelbrotova množica.
**RGB –** red, green, blue. Standardni barvni model, uporabljen pri programiranju ter v digitalnih zaslonih. Barve ustvarja z aditivnim mešanjem oddtenkov rdeče, modre in zelene barve.
Barve zapiše kot kombinacijo treh vrednosti, vsake v razponu od 0 do 255. 0 predstavlja odsotnost, 255 pa popolno koncentracijo specifične barvne komponente.
**SIMD –** single instruction multiple data, sl. en ukaz na več podatkih.

# PREDGOVOR

Ob začetku naloge je bil napisan osnovni algoritem [Alg1], ki bo v uporabi skozi celotno nalogo. V vsakem poglavju bo predstavljena po ena oblika optimizacije, ki bo nato implementirana na kopiji osnovnega algoritma.

Vsi grafi meritvenih časov bodo prikazovali čaz izračuna v milisekundah (Y os) ter zaporedno številko meritve (X os). Meritve bodo izvedene pri $It_{max}$ = 1000 ter ločljivosti 1080p.

Tehnične specifikacije osnovnega sistema in prevajalnika, na katerem je bil grajen program, ter izvedene meritve hitrosti algoritmov so na voljo v prilogi [Spec].

Za razumevanje te naloge je od bralca zaželjeno osnovno poznavanje kompleksnih števil (kaj so, osnovne aritmetične operacij v kartezijski obliki).

Poznavanje strokovne računalniške terminologije ter programskih jezikov je priporočljivo, ni pa zahtevano za razumevanje tem besedila.

### Razlaga uporabljenih strokovnih besed

**Programska nit** – najmanjše zaporedje programskih navodil, ki jih lahko samostojno nadzoruje organizator (scheduler), običajno del operacijskega sistema.

**Računalniški ukaz** (instruction) – je ukaz, podan procesorski enoti; je najnižja oblika komunikacije s procesorjem, saj jo predstavljata le sekvenci 0 in 1, ki procesorju povesta na katerih podatkih naj izvede neko operacijo. [1]

**Register** – komponenta izjemno hitrega spomina, vgrajena neposredno v procesorsko enoto (CPE), ki služi shranjevanju trenutno potrebnih podatkov za učinkovito izvajanje trenutnega procesa.

**Orbita** – pot, ki jo točka opravi ob iteraciji fraktalne formule. Črta, ki povezuje $z_0$ $z_1$ ... $z_{Itmax}$.

**Hausdorffova dimenzija**/fraktalna dimenzija – merska enota kompleksnosti oblik.
Ko neko obliko razpolovimo po vseh topoloških dimenzijah, pridobimo $2^D$ manjših kopij osnovne oblike (D je topološka dimenzija osnovne oblike).
Ko razpolovimo črto, dobimo 2 krajši črti ($2^1$),
Ko razpolovimo kvadrat, dobimo 4 manjše kvadrate ($2^2$),
Ko razpolovimo kocko, dobimo 8 manjših kock ($2^3$),
Ko razpolovimo Sierpińskijev trikotnik, dobimo 3 manjše trikotnike ($2$^$\log_2(3)$).
Večina fraktalnih oblik ima neceloštevilsko Hausdorffovo dimenzijo.

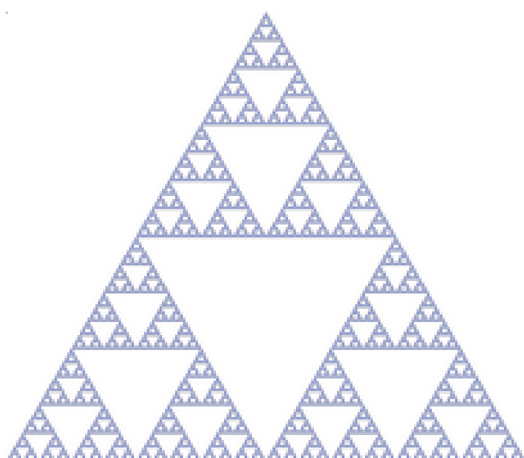
**Polje/**vektor – **s**premenljivka, ki lahko hrani več kot eno vrednost.

**Kardioid** – krivulja, pridobljena s premikom točke na obodu kroga po obodu drugega kroga. Medtem ohranjamo enako razdaljo med njunima središčima.

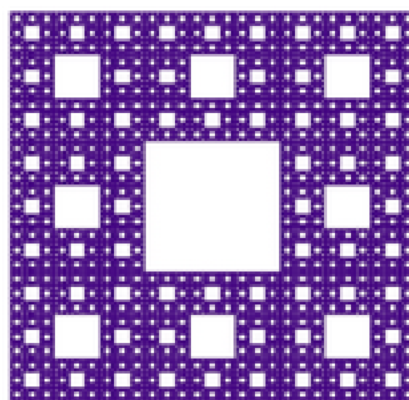
# UVOD

Beseda fraktal nima točne definicije. Pred odkritjem KSF so bili fraktali definirani kot "oblike rekurzivne samopodobe", kasneje se je le ta razvila v "oblike, katerih kompleksnost je neodvisna od povečave". Leta 1975 je Benoit Mandelbrot ustvaril besedo fraktal in jo definiral kot "množico s Hausdorffovo dimenzijo". Beseda je bila kasneje redefinirana na "množico, katere Hausdorffova dimenzija je večja od njene topološke dimenzije".

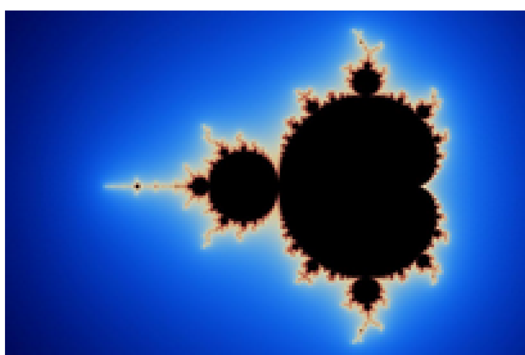
Še do danes definicija ni sprejeta kot formalna. Beseda fraktal je najbojlje opredeljena kot "zbirka lastnosti, skupnh vsem fraktalnim oblikam". [2]



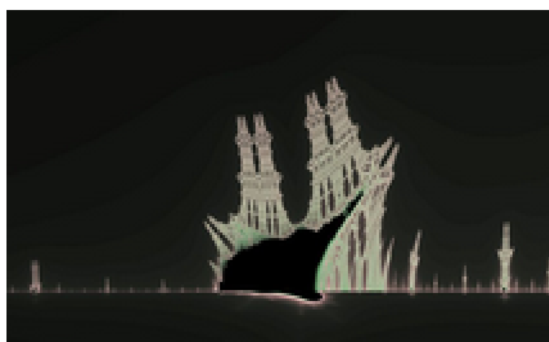
*Slika 1: Sierpińskijev trikotnik.*



*Slika 2: Sierpińskijeva preproga.*

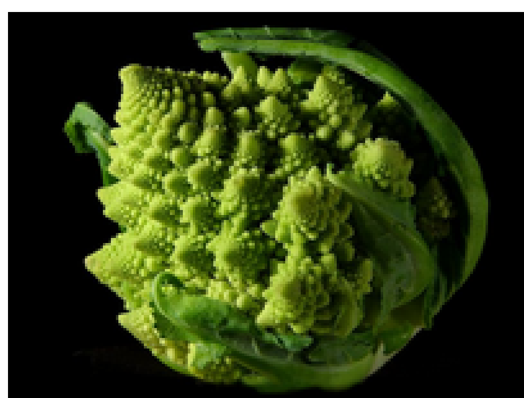


*Slika 3: Mandelbrotova množica.*



*Slika 4: Fraktal goreča ladja.*

*Slika 5: Praprot.*



*Slika 6: Romanesco brokoli.*

# STRUKTURA RAZISKAV

Med izdelavo naloge je delo potekalo v dveh delih: raziskovanje ter testiranje in implementacija. Iskanje podatkov o fraktalih, možnih optimizacijah in kakršnih koli že obstuječih del na to temo je potekalo skozi proces izdelave naloge.

Vsaka nova potencialna metoda je bila testno implementirana in preizkušena. V primeru, da ni bila preprosto združljiva z ostalimi ali pa je njeno delovanje onemogočilo funkcionalnost drugih pomembnejših algoritmov, ki prinesejo bolj znatne pospeške, je bila ta opuščea kot neuporabna.

# HIPOTEZE

Cilj raziskovalne naloge je optimizirati osvnoni algoritem [Alg1] in ustvariti aplikacijo za interaktivno raziskavo KSF, primarno Mandelbrotove množice. Osnovna prioriteta aplikacije je tekoče delovanje ter hiter izris slike fraktala.

Za nalogo sem postavil naslednje hipoteze:

1. Ustvarjen program bo zmožen izrisa slike Mandelbrotovega fraktala pri povečavi $10^{100}$ v manj kot eni sekundi.

2. Najhitrejši bo večniten program, ki deluje na GPE.

3. Optimizacije bodo preprosto razširljive med različnimi KSF.

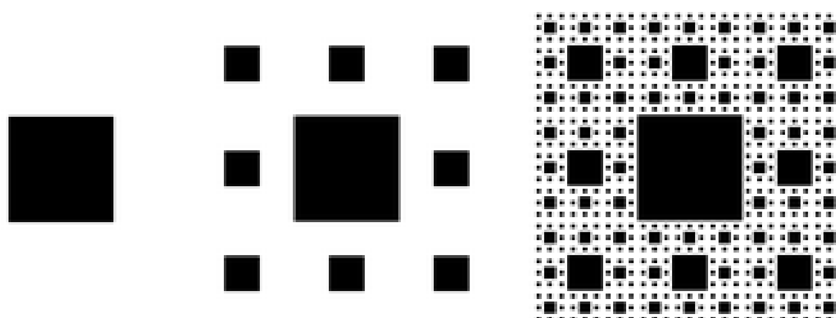
4. Računska natančnost "osnovnih" podatkovnih tipov bo predstavljala večjo oviro pri višjih povečavah.

# FRAKTALI

Kot relativno nova tema v matematiki fraktali nimajo formalno določene definicije. Vendar pa jih lahko opišemo kot skupino oblik, ki si delijo skupne "fraktalne lastnosti". Najpomembnejša med njimi je neskončna kompleksnost. Vizualizacija fraktalov poteka z neskončnim ponavljanjem (iteriranjem) fraktalne formule.

Za vizualizacijo je najpreprostejša Sierpińskijeva preproga ali neskončni kvadrat. Ustvarimo ga v petih preprostih korakih:

1) Začnemo s kvadratom na listu papirja.
2) Kvadrat razdelimo na 9 enako velikih kvadratkov.
3) Pobarvamo srednji kvadrat.
4) Vsakega od preostalih kvadratkov ponovno razdelimo na 9 manjših kvadratkov.
5) Postopek ponavljamo/iteriramo v neskončnost.



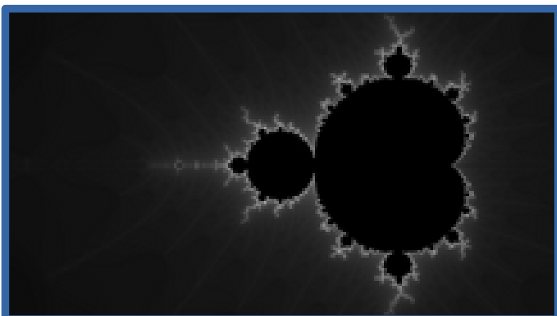
*Slika 7: Sierpińskijeva preproga pri 1, 2, 3 iteracijah.*

Vsi ostali, četudi veliko bolj kompleksni fraktali, nastanejo na podobne načine. Začnejo se z navodilom, funkcijo ali algoritmom, ki ga izpolnjujemo v neskončnost.

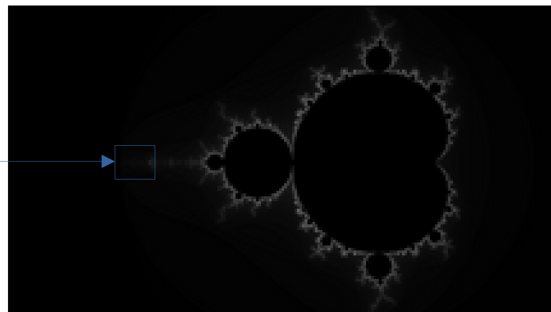
## Samopodobe

Temelj fraktalne geometrije, samopodobe so vzorci, ki se ponavljajo pri različnih velikostih in oblikah.

V Mandelbrotovi množici lahko pri večmilijonkratni povečavi naletimo na osnovni "fraktal".



*Slika 9: Mandelbrot, 10800x povečava.*



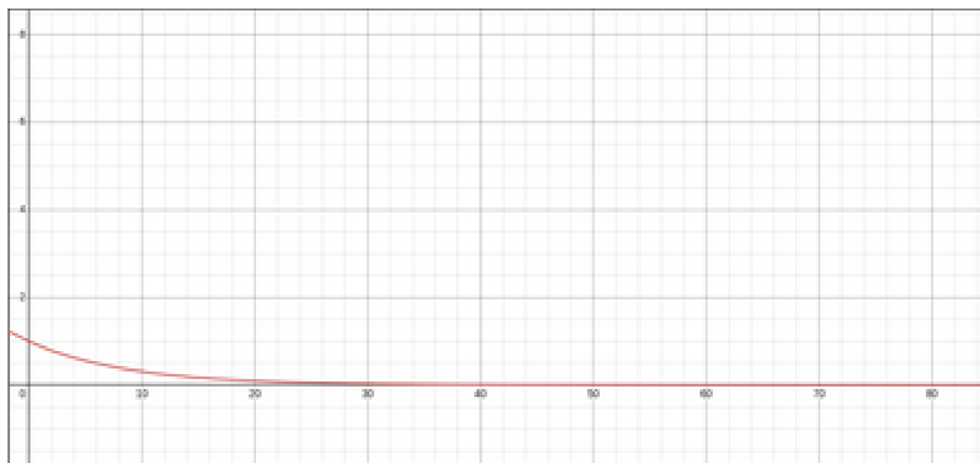
*Slika 8: Mandelbrot, 1x povečava.*

## Ničelna ploščina

Nekateri fraktali, kot na primer Sierpińskijeva preproga, nimajo ploščine. Z vsako iteracijo od preostanka odvzamemo 1/9 ploščine, kar ponavljamo v neskončnost. Ploščino Sierpińskijeve preproge lahko izrazimo kot:

$$s = \left(\frac{8}{9}\right)^{x} \times a$$

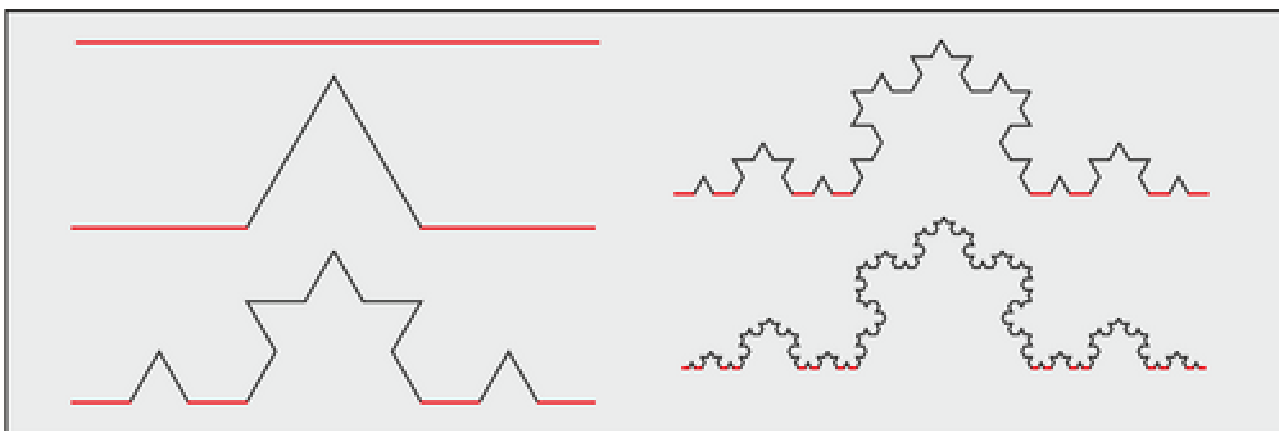
kjer je a ploščina osnovnega kvadrata.



*Slika 10:* Graf ploščine Sirpinskijeve preproge *za a = 1.*

## Neskončen obseg

Čeprav se zdi neskončen obseg marsikomu nemogoč in protiintuitiven koncept, je marsikateri fraktal kot geometrijska oblika neskončen in temu primerno omejen s krivuljo neskončne dolžine. Kochova krivulja je lomljena črta, ki omejuje podobnoimenski fraktal Kochove snežinke. Z vsako iteracijo se dolžina mejne krivulje podaljša. Izračunamo jo po formuli
$l_n = l_0 \times (4/3)^n$, kjer je $l_n$ dolžina krivulje pri n-ti iteracij, $l_0$ pa dolžina osnovne črte.



*Slika 11: Kochova krivulja pri 1, 2, 3, 4, 5 iteracijah.*

# MANDELBROTOVA MNOŽICA

Mandelbrotova množica je množica kompleksnih števil, za katera velja, da po iterativnem izračunu formule:

$$z_{n+1} = z_n^2 + c$$

absolutna vrednost n-te iteracije ($z_n$) ne presega 2.
Izračun pripadnosti množice poteka tako, da:

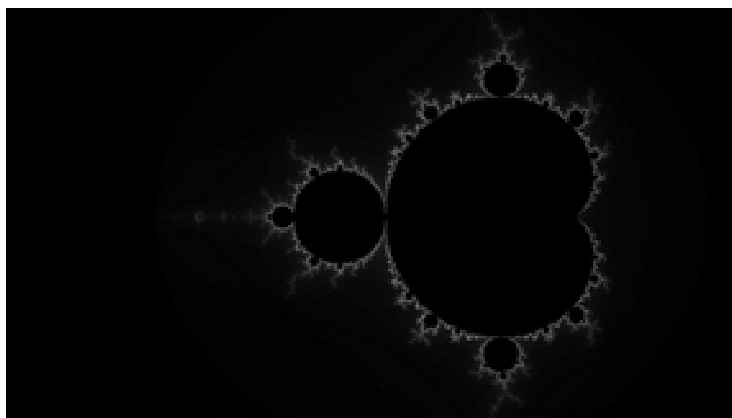
1) izberemo poljubno kompleksno število c ter določimo $z_0 = 0$
2) iterativno ovrednotimo formulo do vrednosti $It_{max}$

$$z_1 = z_0{}^2 + c$$
$$z_2 = z_1{}^2 + c = (z_0{}^2 + c)^2 + c$$
$$z_3 = z_2{}^2 + c = ((z_0{}^2 + c)^2 + c)^2 + c$$
$$\ldots$$
$$z_{Itmax} = z_{Itmax-1}{}^2 + c$$

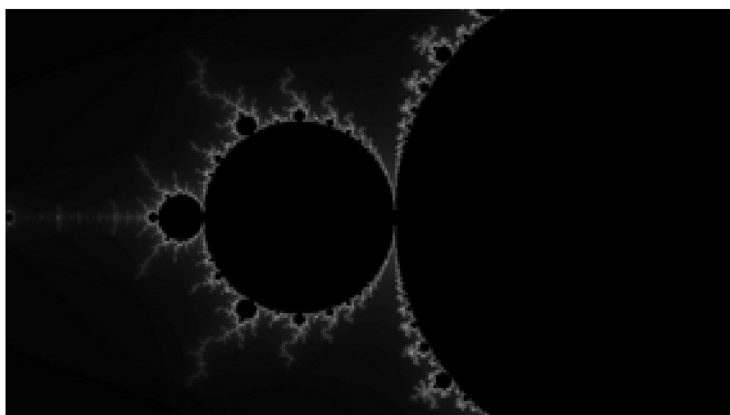
3) Preverimo absolutno vrednost $z_{Itmax}$. Če je ta manjša od 2, iz tega sledi, da je točka c del Mandelbrotove množice.

Z enakim postopkom ovrednotimo vse točke kompleksne ravnine, omejene na Re[-2.8,1.2], Im[-1.125, 1.125]. Mandelbrotova množica je v celoti znotraj tega intervala, zato večji niso potrebni. V primeru, da oba intervala delimo z enakim številom n, s tem pomanjšamo vidno polje in posledično približamo sliko.

Točke, ki so del množice, so obarvane črno, vse ostale pa pridobijo barvo glede na hitrost divergence proti nesnončnosti; hitreje kot pobegne, temnejša je točka.



*Slika 12: Re[-2.8,1.2], Im[-1.125, 1.125] n=1.*



*Slika 13: Re[-1.76, 0.16], Im[-0.54,0.54] n=5.*

# PROBLEM

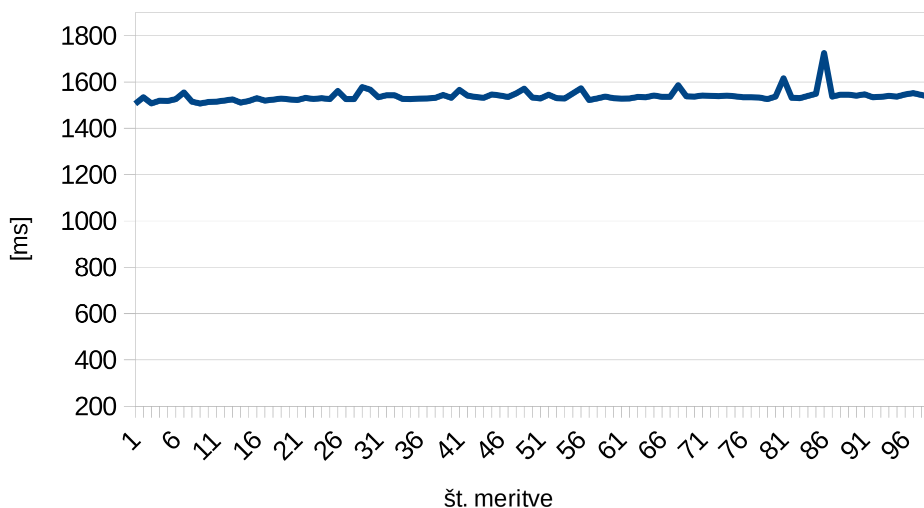
Izris KSF je računsko izjemno kompleksna naloga. Ne s strani težavnosti potrebnih izračunov, temveč zaradi njihove količine.

Za izris slike Mandelbrotove množice pri standardni ločljivosti 1080p s številom $It_{max}$ = 1000 je potrebno izvesti okoli 15 milijard aritmetičnih operacij. Algoritem [Alg1] to opravi v približno eni sekundi.

Osnovni program [Dop] začne svoje delovanje z izračunom intervalov vidnega polja glede na parametre, kot so povečava ter premik vidnega polja. Nato ustvari polje za hrambo vrednosti $It_{esc}$ vsake točke v vidnem polju. Sledi izračun vseh točk. Najprej algoritem [Alg1] vsako točko na zaslonu preslika v kompleksno ravnino z uporabo [preslikavne funkcije] nato nad njimi iterira fraktalno funkcijo. Pridobljeno vrednost $It_{esc}$ shrani v polje. Ob izračunu vseh točk pokliče metodo, ki polje s podatki izriše na zaslon.

Za prijetno uporabniško izkušnjo ter da je izris fraktala realnočasoven, se ta mora izvesti v 33 milisekundah ali manj[1]. Višji računski časi privedejo do neodzivnosti programa med izračuni, kar bi negativno vplivalo na interaktivnost pri raziskovanju fraktala.

Z višanjem povečave in izrisom slik globje v fraktalu program operira na vse manjših in manjših številih. V nekem trenutku pa ta postanejo premajhna tudi za računalnik in standardne podatkovne tipe, kot so float in double. Za premaganje te ovire (v jeziku C++) obstaja knjižnica GNU MPFR, ki implementira podatkovne tipe arbitrarne natančnosti omejeno le s količino RAM-a, vendar so aritmetične operacije nad njimi veliko počasnejše. Dobro optimiziran algoritem je zato ključnega pomena za ohranitev nizkih računskih časov.




*Slika 14: Računski čas osnovnega algoritma, povprečje:* 1538.14ms

[1] 33 ms na sličico ≃ 30 sličic na sekundo

# MATEMATIČNE OPTIMIZACIJE

Vsak KFS je pridobljen z iterativnim izračunom fraktalne formule nad vsako točko kompleksne ravnine. Za izris slike ločljivosti 1080p je potrebno izvesti med nekaj sto milijonov ter več milijard računskih operacij.

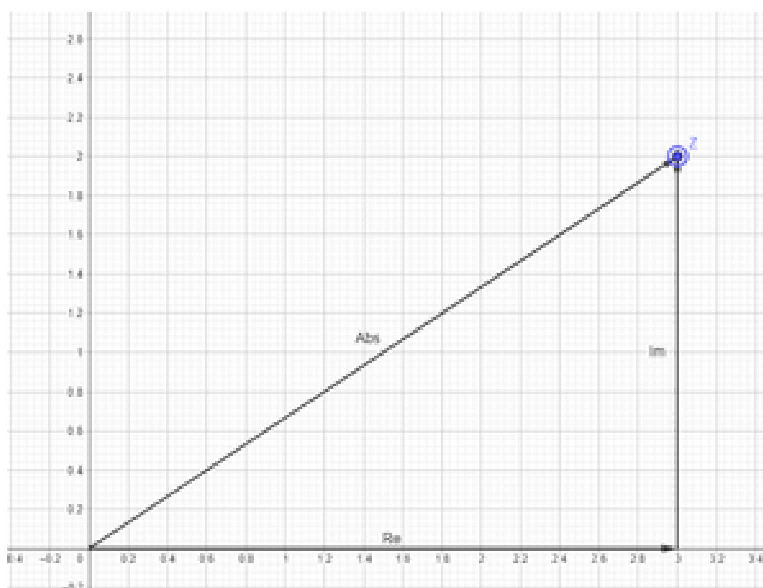
Cilj tega dela naloge bo zmanjšati število opearcij, potrebnih za izris ene slike.

## POENOSTAVITEV IZRAČUNOV

Vsak KSF je sestavljen iz vseh števil, ki po $It_{max}$ iteracijah fraktalne formule ne pobegnejo v neskončnost. Točka se smatra za pobeglo, ko njena absolutna vrednost preseže mejno vrednost, v primeru Mandelbrotove množice je to 2. Samo preverjanje, ali je |z| > 2, v teoriji ni problematično, v praksi pa lahko izračun absolutne vrednosti poenostavimo.

Ker je absolutna vrednost le oddaljenost od številskega izhodišča, lahko absolutno vrednost kompleksnega števila pridobimo z uporabo Pitagorovega izreka.

Formula absolutne vrednosti: $|z|=\sqrt{z_{\Re}^2+z_{\Im}^2}$

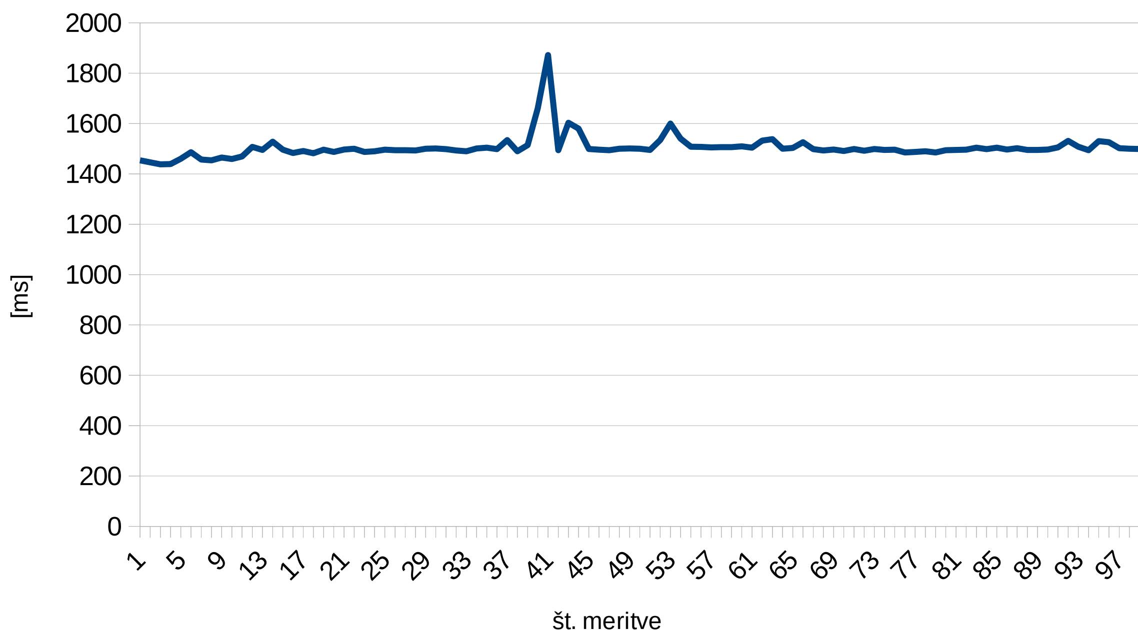



*Slika 15: Absolutna vrednost kompleksnega števila.*

Med iteracijo se absolutna vrednost primerja z 2, kar lahko zapišemo kot preprosto neenačbo $|z| > 2$, ki jo lahko poenostavimo.

$$|z| > 2$$
$$\sqrt{z_{\Re}^2 + z_{\Im}^2} > 2$$
$$z_{\Re}^2 + z_{\Im}^2 > 4$$

S kvadriranjem obeh strani neenačbe se znebimo korena. Korenjenje je računsko veliko kompleksnejše v primerjavi z osnovnimi operacijami, kot sta seštevanje ali množenje. Moderni računalniki sicer korenijo brez kakršnih koli težav (operacija kvadratnega korena std::sqrt() se izvede v okoli 40 ns). Čeravno gre za nepredstavljivo majhne časovne enote v velikih številih prinesejo dokajšnjo upočasnitev. Slika ločljivosti 1080p vsebuje okoli 2 milijona točk, pri $It_{max}$ = 1000 to pripelje do okoli 2 milijard korenjen, ki so popolnoma nepotrebna.

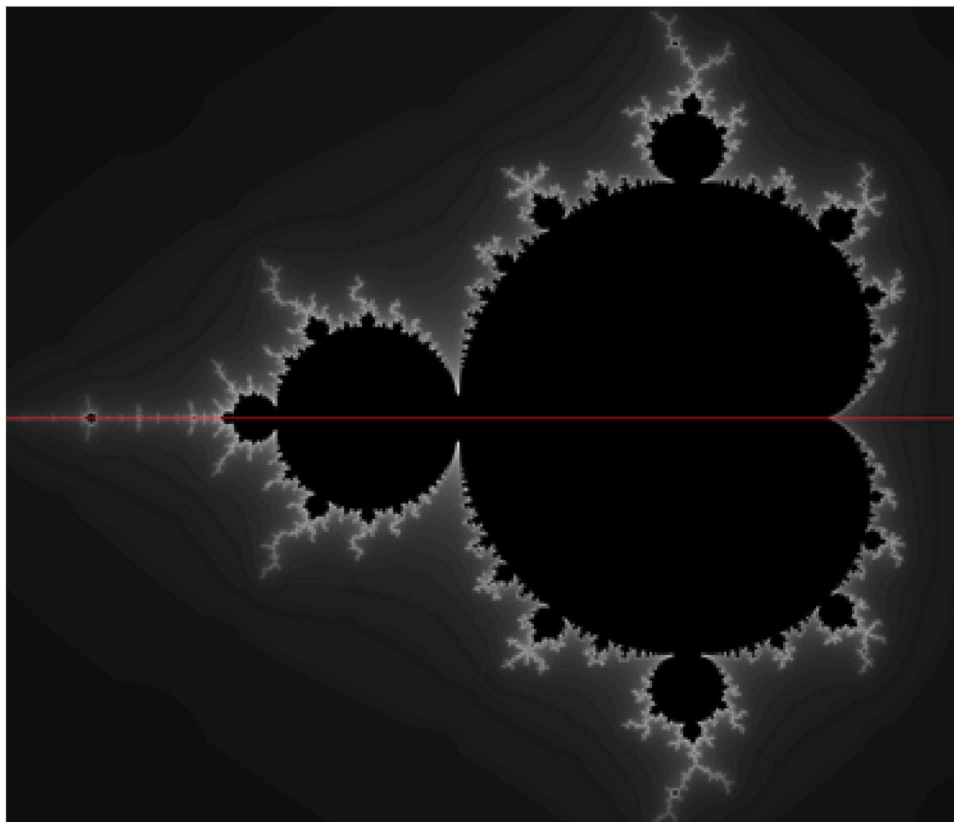



*Slika 16: Računski časi pooenostavljenega algoritma, povprečje: 1501.1 ms.*

Vidne so manjše spremembe, v povprečju okoli 30 ms hitreje od osnovnega algoritma.

## SIMETRIČNOST

Mandelbrotov fraktal že na prvi pogled izgleda simetrično. Simetričnost neskončno kompleksne oblike se morda zdi protiintuitivna, vendar jo je mogoče matematično dokazati.



*Slika 17: Simetrala Mandelbrotovega fraktala.*

Za dokaz simetričnosti najprej vzamemo 2 med seboj komplamantarni kompleksni števili nad katerima nato izvedemo iteracijo Mandelbrotove formule in opazujemo rezultate.

$$\overline{z_1} = a - bi$$
$$z_2 = \overline{z_1}^2 + \overline{z_1}$$
$$z_2 = (a-bi)\times(a-bi)+(a-bi)$$
$$z_2 = a^2 - 2abi + b^2 i^2 + a - bi$$
$$z_2 = a^2 - b^2 + a - 2abi - bi$$
$$z_2 = (a^2 + a - b^2) - (2abi + bi)$$

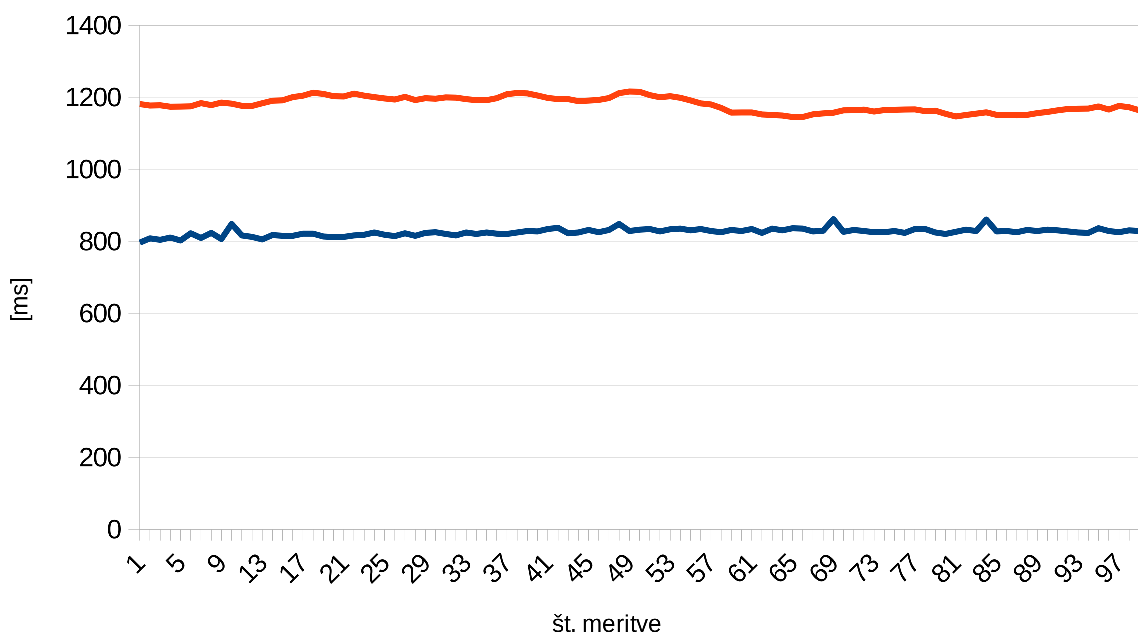
$$z_1 = a + bi$$
$$z_2 = z_1^2 + z_1$$
$$z_2 = (a+bi)\times(a+bi)+(a+bi)$$
$$z_2 = a^2 + 2abi + b^2 i^2 + a + bi$$
$$z_2 = a^2 - b^2 + a + 2abi + bi$$
$$z_2 = (a^2 + a - b^2) + (2abi + bi)$$

Ob iteraciji se številu $z_1$ in njegovi komplementarni vrednosti spremenita, obe realna in kompleksna komponenta. Od teh bosta realni vedno enaki ne glede na to, ali sta pozitivni ali negativni, saj $(-a)^2 = (a)^2 = a^2$. Kompleksni vrednosti pa si bosta vedno nasprotni, saj v primeru, da je drugi člen negativen, to obrne predznak srednjega člena (2ab), ki je edina kompleksna komponenta po množenju.

$$(a+b)^2 = a^2 + ab + ab + b^2 \qquad (a-b)^2 = a^2 - ab - ab + b^2$$
$$(a+b)^2 = a^2 + 2ab + b^2 \qquad (a-b)^2 = a^2 - 2ab + b^2$$

Ker se dve med seboj komplementarni točki vedno gibljeta simetrično glede na realno os, z izračunom $It_{esc}$ točke z prav tako pridobimo $It_{esc}$ njene komplementarne vrednosti. V primeru, da je fraktal v celoti viden in da realna os seka vidno polje točno čez polovico, to pomeni, da z izračunom $It_{esc}$ vseh točk prvega in drugega kvadranta prav tako izračunamo $It_{esc}$ njihovih komplementarnih vrednosti v tretjem in četrtem kvadrantu.

Teoretični maksimalni pospešek je 2x, v primeru, da realna os seka vidno polje točno čez polovico. S translacijo vidnega polja stran od realne osi manjšamo delež točk, ki imajo svojo komplementarno vrednost v vidnem območju. S tem višamo število točk, ki jih moramo izračunati.




*Slika 18: Računski čas s simetrijo poenostavljenega algoritma, povprečje: 825.06 ms.*

Dvakratni pospešek je lepo viden kot temno modri graf. Oranžen graf predstavlja računski čas v primeru, da vidno polje pomaknemo za +30 % po y osi.

# SPLOŠNE OPTIMIZACIJE

Nlednje optimizacije bodo poskušale pospešiti izračun z uporabo lastnosti fraktalov, selektivnim izračunom določenih točk ter ostalimi optimizacijami, ki ne sodijo nujno med matematične ali računalniške.

## PREDČASNE PREKINITVE

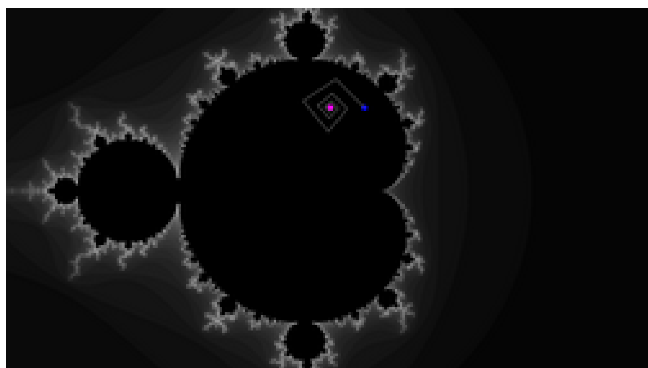
Ob iteriranju formule nad točko se ta "pomika" po ravnini. Lahko določimo tri načine gibanja; kaotično, ciklično in konvergento. Kaotične točke se gibljejo naključno in nimajo predvidljivih gibalnih vzorcev. Orbite cikličnih in konvergentih točk pa imajo nekaj lastnosti, po katerih jih lahko prepoznamo. Prepoznavanje cikličnih in konvergentih točk je ključnega pomena, saj so te vedno del KSF.

V primeru, da začnemo iterirati fraktalno formulo nad kaotično točko, bo ta v večini primerov precej hitro pobegnila v neskončnost in v tistem trenutku prekinila izračun (izvede se le manjši del vseh iteracij). Ko pa enak postopek izvedemo nad ciklično točko, npr -1 + 0i, bo ta (v primeru Mandelbrotovega fraktala) vedno privedla iteracijski postopek do konca, nikoli ne bo pobegnila (izvede se $It_{max}$ iteracij). Zato so točke, ki so del fraktala, vedno računsko "najnevarnejše" in moramo z njimi tako tudi ravnati.

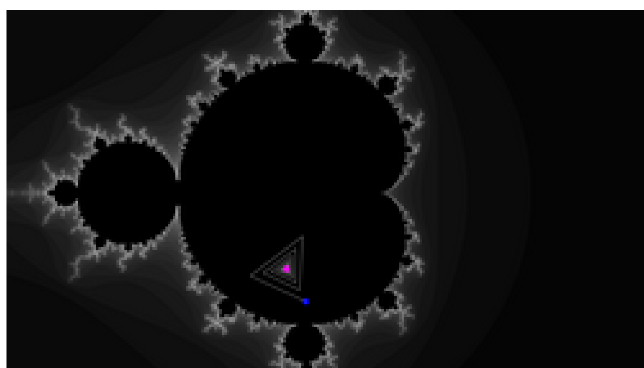
### Konvergentne točke

Za vse točke, ležeče znotraj kardioida s centrom na S = (-1/4, 0) in radijem 1/4, velja, da konvergirajo proti eni sami tako imenovani "privlačni točki". Vse točke znotraj privlačnega radija privlačne točke ob iteraciji Mandelbrotove formule proti njej konvergirajo. Posledično so vse, ki ležijo znotraj teh privlačnih območij, del M. [3]

Primeri konvergenc proti privlačni točki. Modra točka predstavlja $z_0$, rdeča pa $z_{50}$. Bele črte povezujejo dve zaporedni iteraciji za lažjo vizualizacijo.

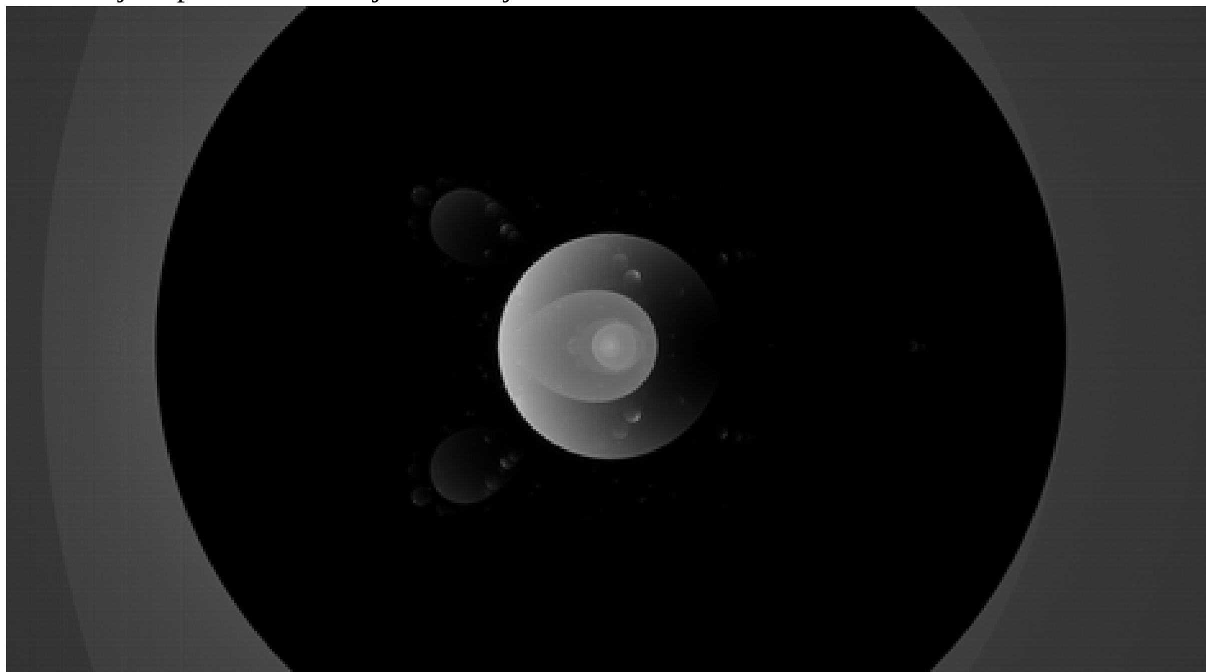


*Slika 19: Orbtia c = 0.17+0.412i.*



*Slika 20: Orbita c = -O.118-0.538i.*

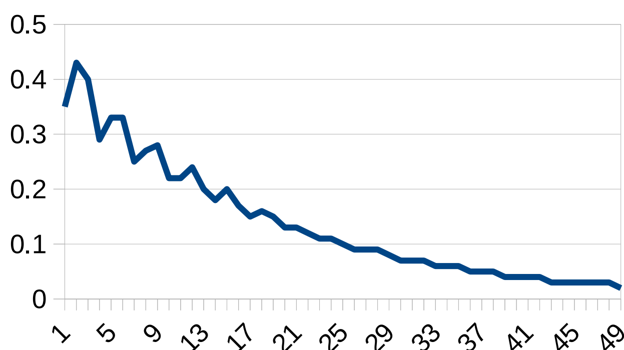
Proti privlačnim točkam konvergirajo vse točke znotraj nekega radija; večji, kot je radij, več točk tam tudi pristane. Naslednja slika poskuša vizualizirati z iteracijo Mandelbrotove formule nad vsemi točkami ravnine do prvih tisoč iteracij. Lokacijo, kjer $z_{1000}$ pristane, osvetlimo za nekaj odstotkov. V primeru, da več točk pristane na enaki lokacij, bo ta svetlejša, torej bodo privlačne točke z večjim privlačnim radijem svetlejše.



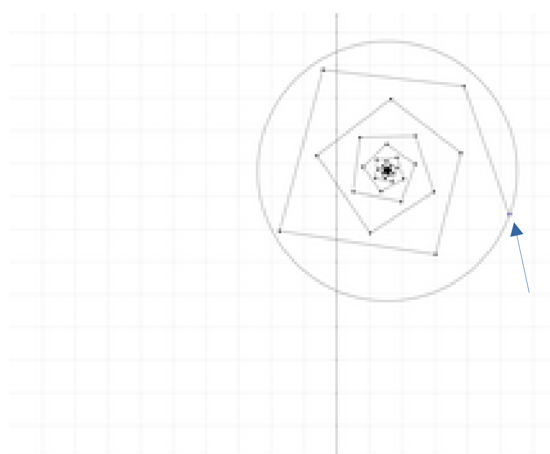
*Slika 21: Vizualizacija privlačnih točk.*

Konverzija proti privlačni točki je dokaj preprosto zaznana. Upadanje razdalje med dvema zaporednima točkama $z_n$ in $z_{n+1}$ je najboljši indikator.

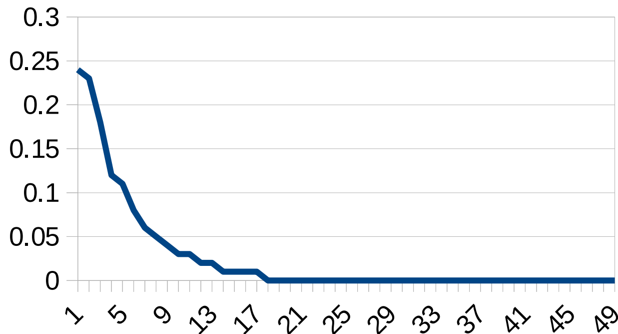
Naslednji grafi prikazujejo razdaljo med dvema zaporednima iteracijama, slika pa prikazuje orbito iteracij ($z_0$ nakazana z puščico).



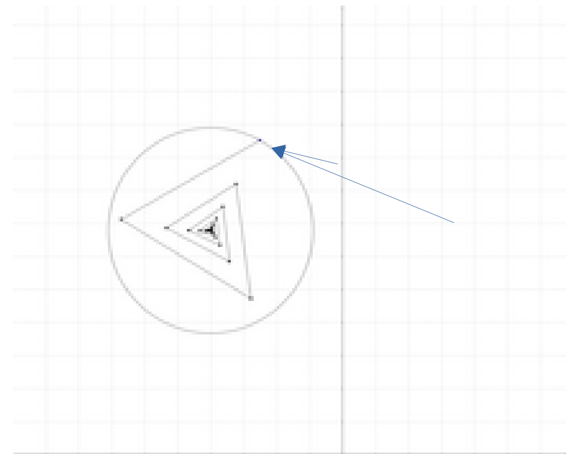

*Slika 23: Y - Razdalja med dvema točkama, X – iteracija.*



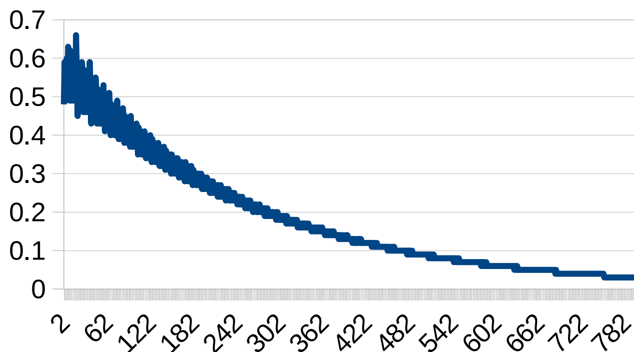
*Slika 22: Orbita c = 0.2635 + 0.37195í, $It_{max}$ = 50.*



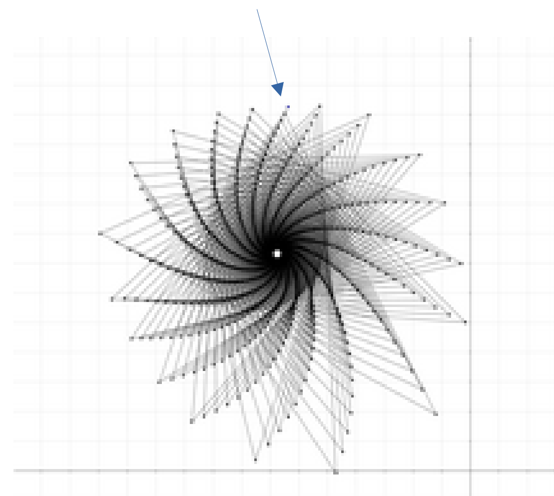

*Slika 25: Y - Razdalja med dvema točkama, X -iteracija.*



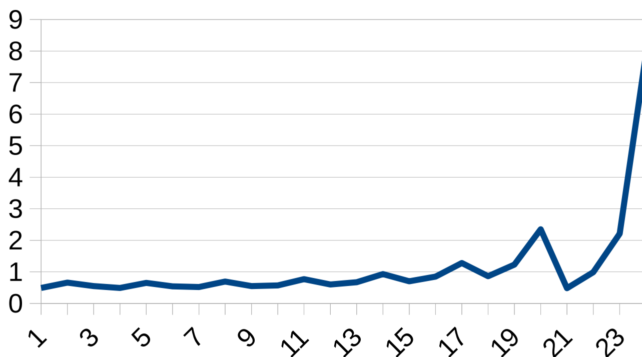
*Slika 24: Orbita c = -0.12561 + 0.47578í, $It_{max}$ = 50.*



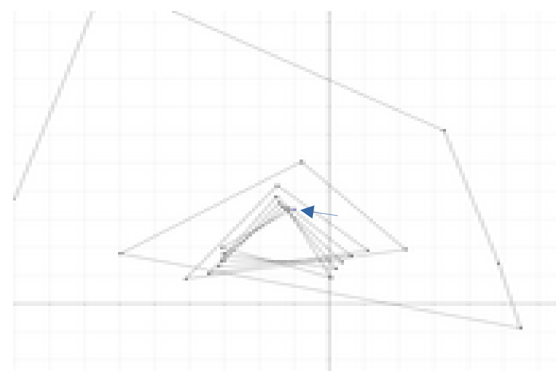

*Slika 27: Y - Razdalja med dvema točkama, X -iteracija.*



*Slika 26: Orbita c = -0.31863 + 0.61319í, $It_{max}$ = 800.*




*Slika 28: Y - Razdalja med dvema točkama, X - iteracija.*



*Slika 29: Orbita c = -0.20361 + 0.67105í, $It_{max}$ = 50.*

Za gotovo detekcijo konverzije proti privlačni točki je potrebno izračunati razdalje med n zaporednimi točkami ter analizirati graf; če ta upada, miruje ali raste. V primeru mirovanja ali padanja točko opredelimo kot del M (algoritem vrne 0 in jo obarva črno).

Število n je predvsem odvisno od kompleksnotsti orbite in števila $It_{max}$. V primeru slike 25 je upadanje vidno že po dveh zaporednih mertivah, ne pa tudi v primeru slike 26, kjer razdalja začne vidno upadati šele po okoli 50 iteracijah. Prej se giblje nepredvidljivo.

Dovolj veliko število meritev je pomembno, saj se razdalje gibljejo nepredvidljivo. V primeru premajhnega št. meritev lahko večje število točk napačno opredelimo, kar privede do popačene končne slike.

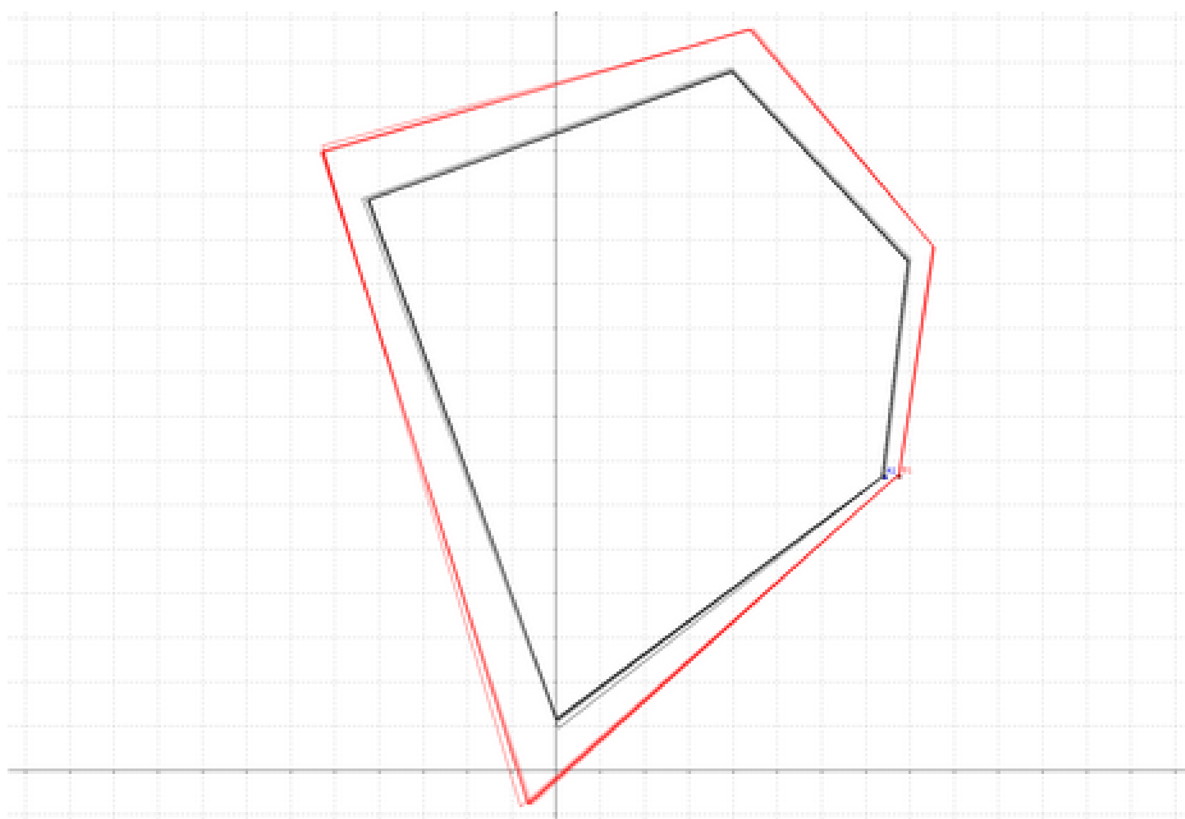
V primeru izvedbe npr. le treh meritev bi primera slik 22 in 25 kot del M opredelili pravilno, ne pa tudi primerov slike 26 ter 29.

## Ciklične točke

Za nekater točke velja, da po n iteracijah pristanejo nazaj na začetni točki ($z_0 = z_n$). Takšne točke imenujemo ciklične točke, saj ob iteraciji fraktalne formule rezultati formulirajo ponavljajoče se zaporedje. Število iteracij preden, se cikel ponovi, pa nam pove cikle orbite.

Za cikličnost točke, $z_0 = z_n$ ni absolutna obveza, dve točki $z_0$ in $c_0$ lahko sledita vizualno identični orbiti, dokler sta si dovolj blizu.

Nalednji primer prikazuje blžnji točki A1 in B1, obe sledita orbiti enake oblike in cikla.

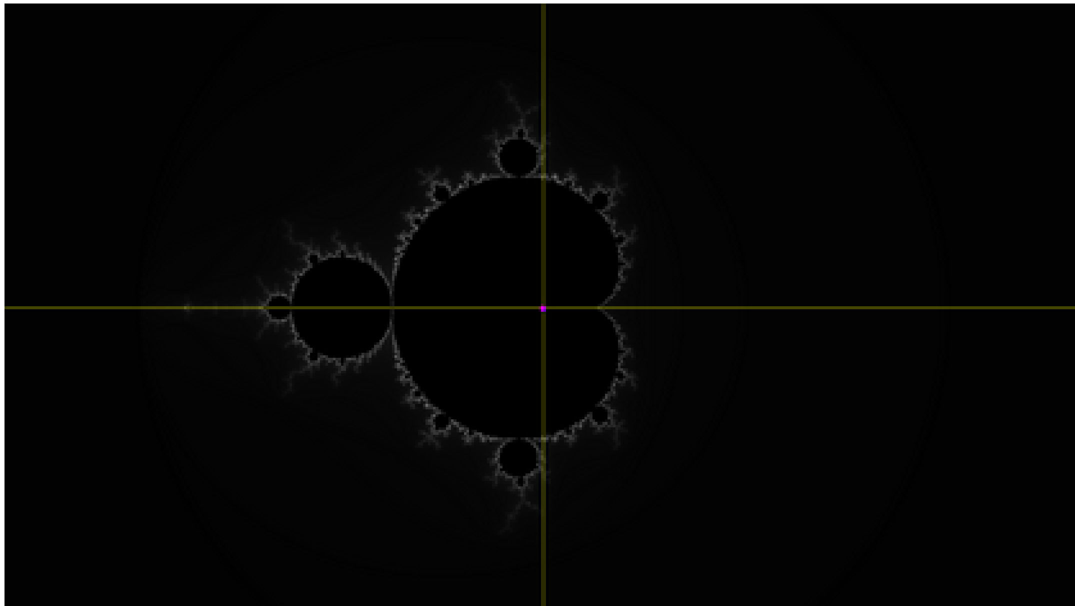


*Slika 30: Primer ciklične orbite.*

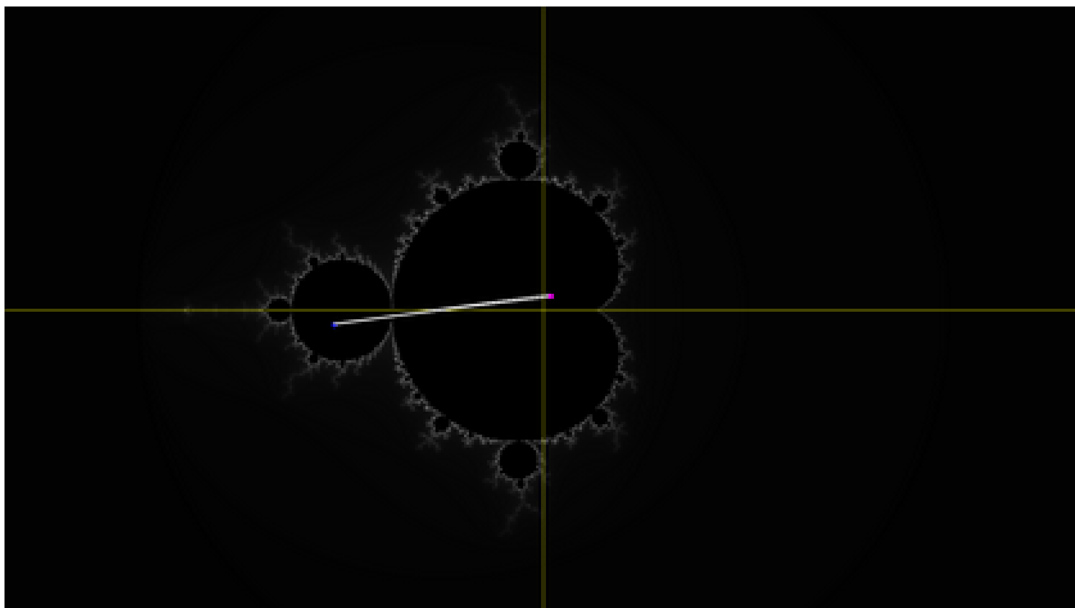
Neka orbita je lahko cikla ali višjega cikla. Za točke prvega cikla vedno velja $z_0 = z_n$ (se ne premikajo). Cikel orbite pove, med koliko točkami potuje $z_0$.

V primeru Mandelbrotove enačbe lahko najdemom orbite cikla vsakega naravnega števila.

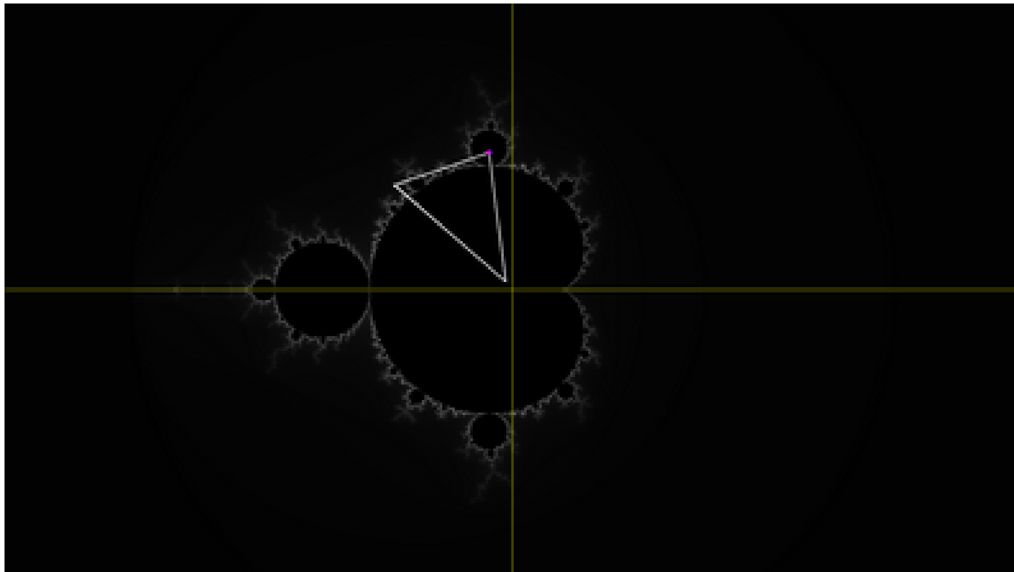
V primeru naslenjih slik rumeni črti predstavljata realno in kompleksno os. Modra točka je $z_0$ vijolična pa $z_{1000}$



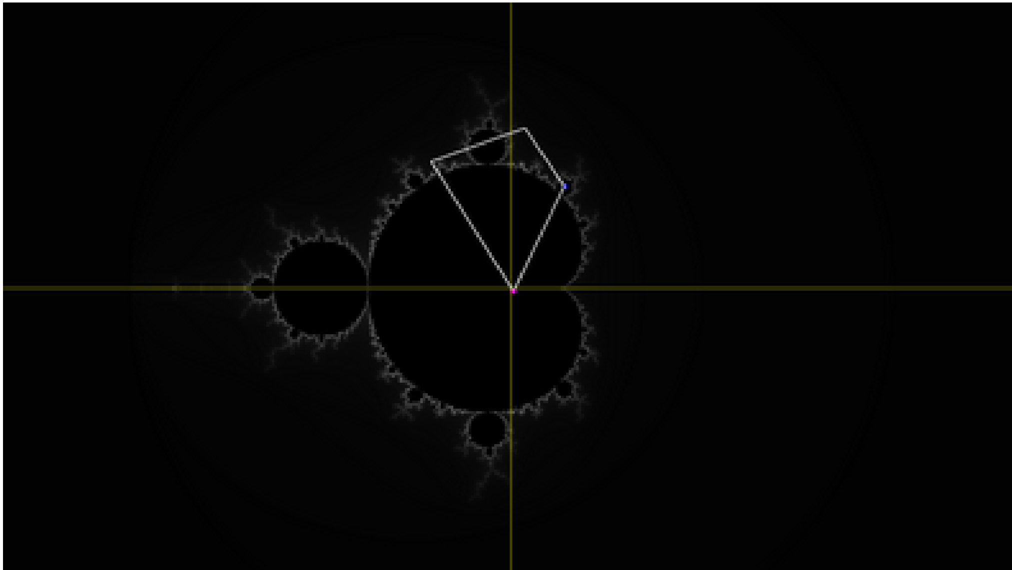
*Slika 31: Orbita prvega cikla.*



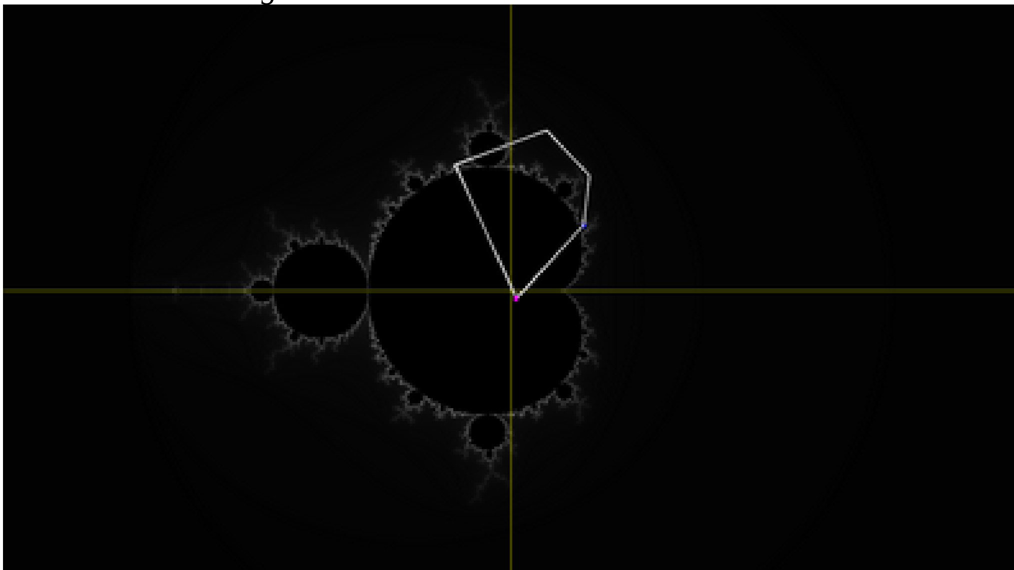
*Slika 32: Orbita drugega cikla.*



*Slika 33: Orbita tretjega cikla.*

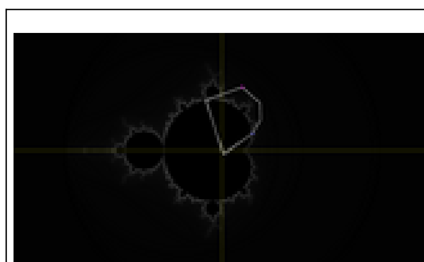


*Slika 34: Orbita četrtega cikla.*



*Slika 35: Orbita petega cikla.*

Trend se tako nadaljuje v neskončnost proti centru kardioida (0.25 + 0i). Stopnja orbite narašča z vsakim naslednjim krožnim izrastkom ob robi glavnega kardioida.

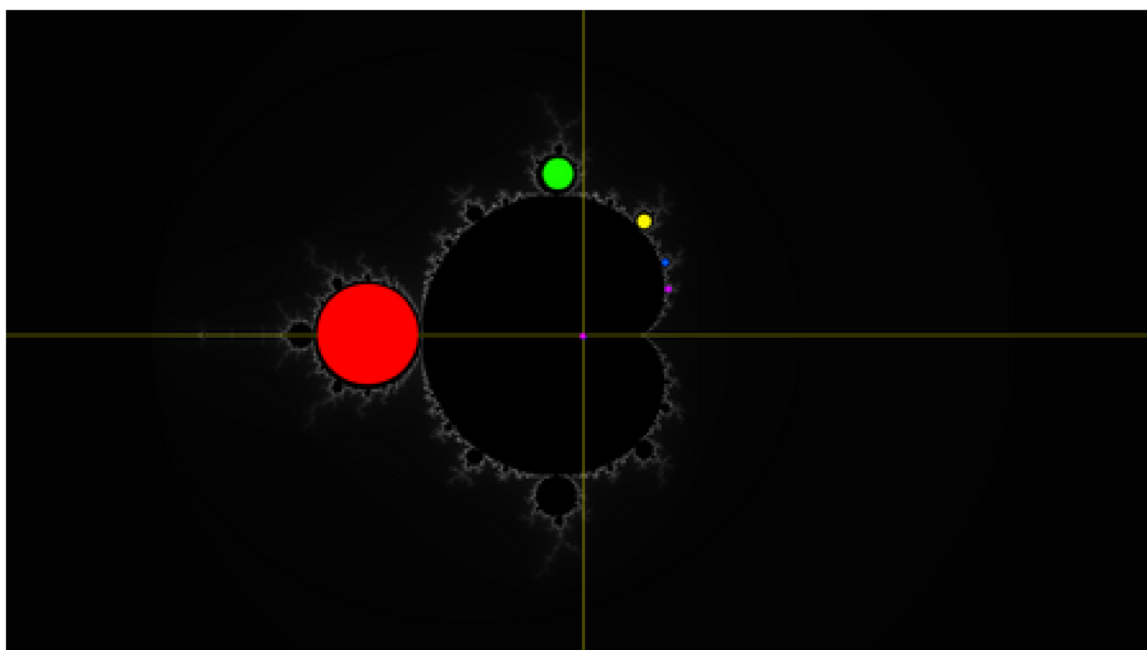
| | | |
|---|---|---|
|  *Slika 36: Šesti cikel.* | *Slika 37: Sedmi cikel.* | *Slika 38: Osmi cikel.* |

Orbite na slikah so približki popolnih orbit. Popolna orbita je sestavljena iz n točk, ki se ponavljajo. Takšne so pridobljene, če je $z_0$ na točno pravilni točki; v primeru slik 32-38 so središča okroglih izrastkov okoli glavnega kardioida.

V računskih približkih je le to nemogoče zaradi računske natančnosti in možnosti iracionalnosti začetnih vrednosti.

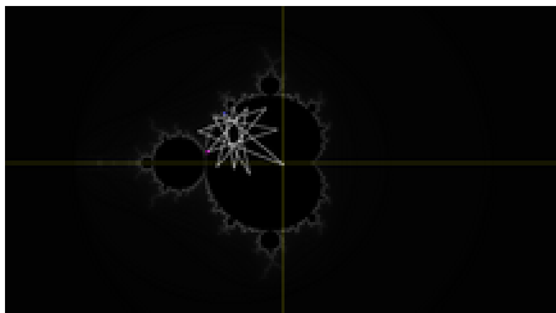
Centri obarvanih krogov predstavljajo idealne začetne vrednosti za popolne orbite:

- rdeč – drugega, slika 32,
- zelen – tretjega, slika33,
- rumen – četrtega, slika 34,
- moder – petega, slika 35,
- vijoličast – šestega, slika 36,
- izhodišče (0+ 0i) pa za prvi cikel.



*Slika 39: Idealne začetne vrednosti.*

Primeri še nekaj zanimivih orbit zanimivih oblik.



*Slika 40: Orbita primer 1.*

*Slika 41: Orbita primer 2.*

Vse ciklične in konvergentne točke imajo nekajskupnih, a precej uporabnih lastnosti:

a) točka $z_0$ bo vedno del množice,

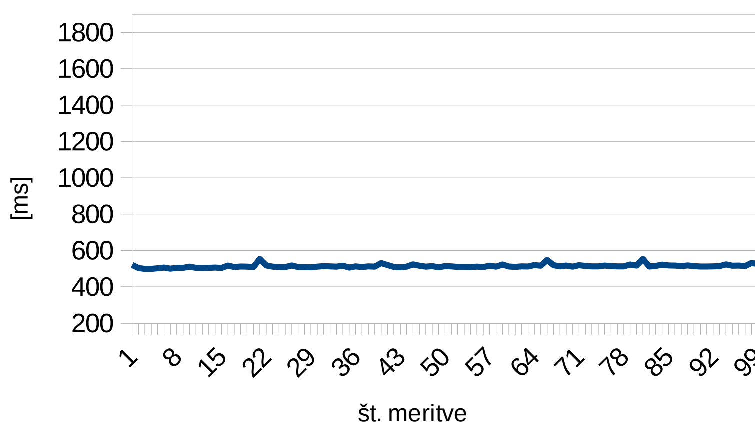
b) vse sosednje točke znotraj nekega radija r pravtako sledijo vedenju točke $z_0$ in so del množice,

c) preprosta detekcija,

lastnosti so lahko uporabne za predčasno prekinitev iteracijskega postopka.

Ciklične orbite zaznamo tako, da hranimo lokacijo $z_0$ in če opazimo približno ponovitev, prekinemo iteracijski postopek. Približne ponovitve so ključnega pomena, saj se zaradi omejene računske natančnosti računalnikov popolna ponovitev zgodi izjemno redko.

Za detekcijo konvergence v privlačno točko izmerimo razdalje med n zaporednimi iterati, če te upadajo, prekinemo izračun in točko označimo kot del KSF.

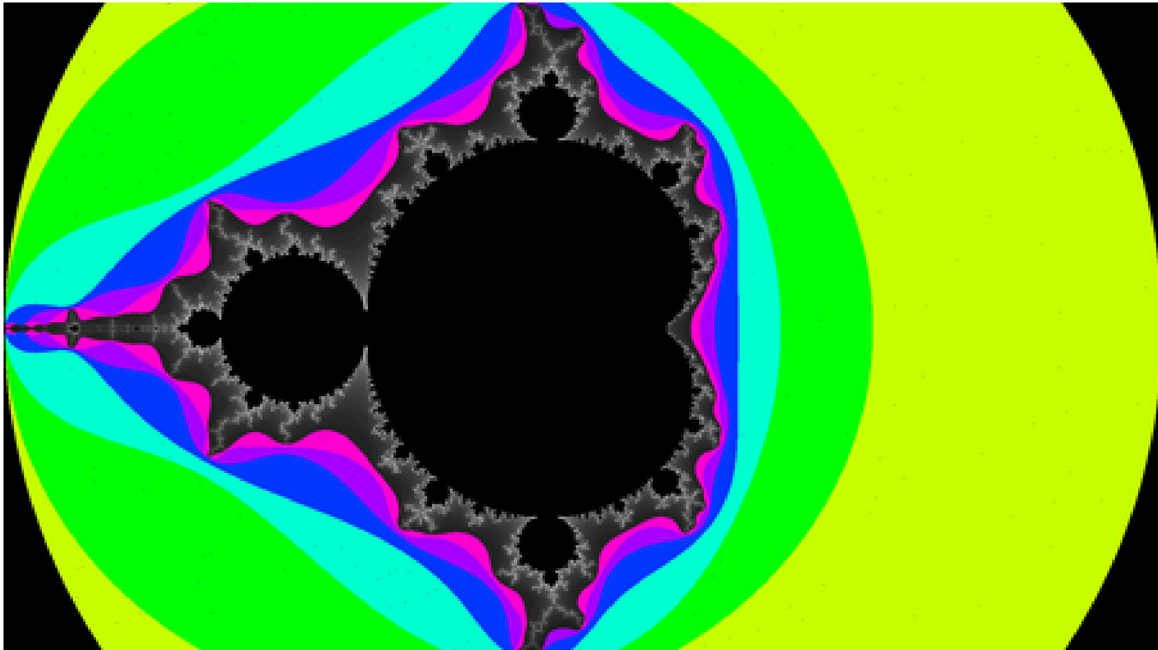



*Slika 42: Računski časi algoritma s predčasnimi prekinitvami, povprečje 514.12 ms.*

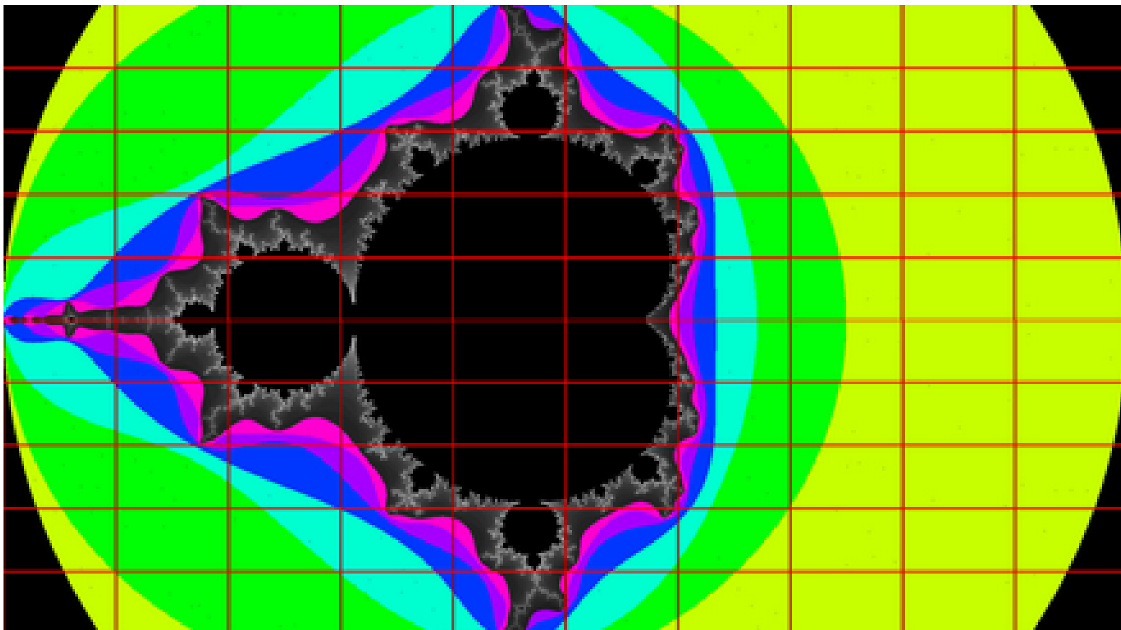
Z implementacijo detekcije konverzije in cikličnih orbit predčasno ustavimo vse nepotrene izračune neskončnih ciklov ter konverzij, ki predstavljajo večino računskega dela.

## DELJENJE NA SEKCIJE

Osnovni barvni algoritem uporablja linearno transformacijo za določanje barve vsaki točki glede na hitrost pobega (hitreje, kot divergira, temnejša bo točka). To pomeni, da si dve točki, obarvani z enako barvo, delita enako število iteracij pred pobegom.



*Slika 43: Točke z enakim $It_{esc}$.*



*Slika 44: Sekcije.*

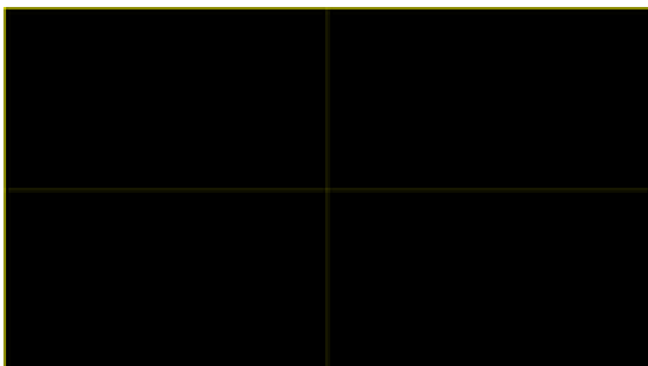
S postavitvijo mreže nad vidno polje razdelimo fraktal na sekcije. Nekatere od slednjih so v celoti prekrite z enako barvo. Vsaka sekcija je kvadraten del vidnega polja, omejen s štirimi stranicami. V primeru, da imajo vse točke, ki ležijo na njenih stranicah, enako število $It_{esc}$, bodo po toliko iteracijah končale tudi vse znotraj ležeče točke. Slednje velja le v primeru, da sta povečava in število $It_{max}$ v primernih razmerjih. V primeru visoke povečave in nizke $It_{max}$ slika v večini izgleda celotno enake barve.

*Slika 45: Slika s prenizkim številom $It_{max}$.*

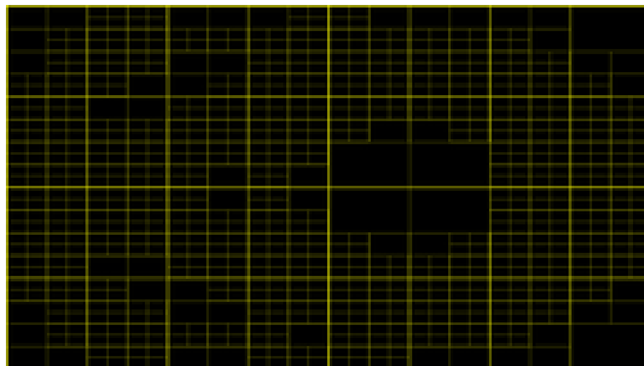
V idealnem primeru z izračunom roba ter zapolnitvijo notranjosti sekcije prihranimo izračun $a \times b - ((a-2) \times (b-2))$ točk, kjer a in b predstavljata širino ter višino trenutne sekcije.

Ko se zgodi, da izračun vseh robov vrne različne vrednosti, notranjosti ne moremo zapolniti. Tedaj lahko trenutno sekcijo razdelimo na štiri, 50 % manjše podsekcije, na katerih opravimo enak postopek preverjanja robov ter zapolnitve notranjosti; te delimo, dokler višina ali širina trenutne sekcije ni manjša od ene slikovne točke na zaslonu ali pa dosežemo maksimalno globino delitve. Takrat nam ne preostane drugega, kot ročen izračun vseh preostalih točk.

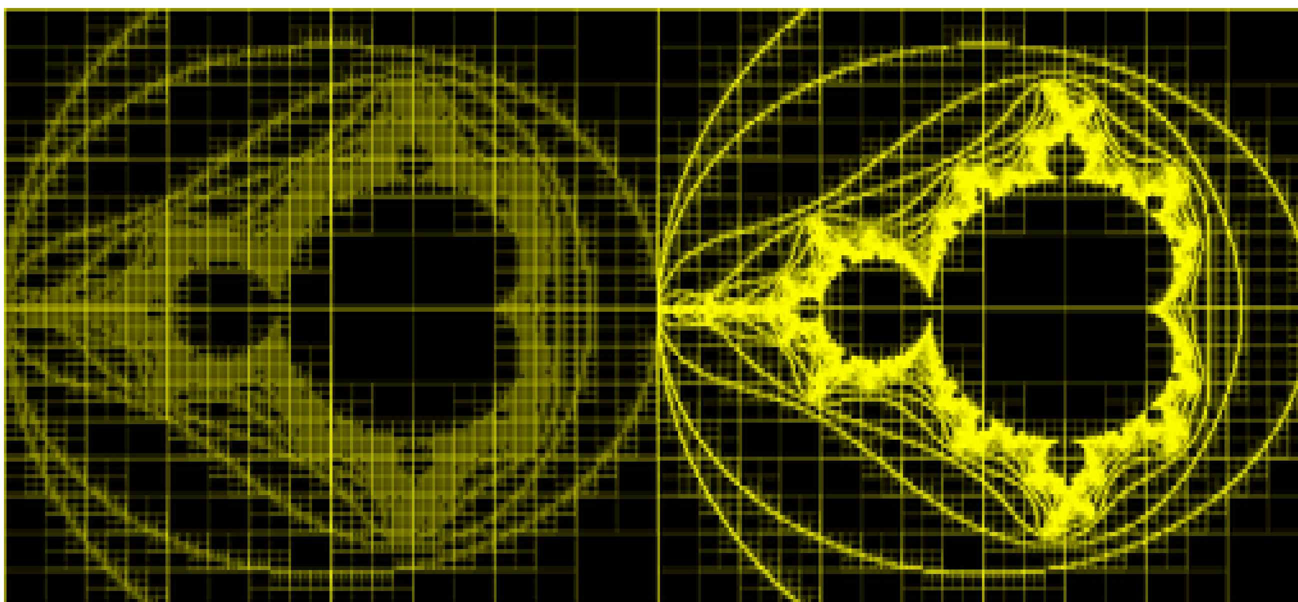
Naslednje slike prikazujejo delitev vidnega polja z višanjem globine.



*Slika 47: Globina 1.*

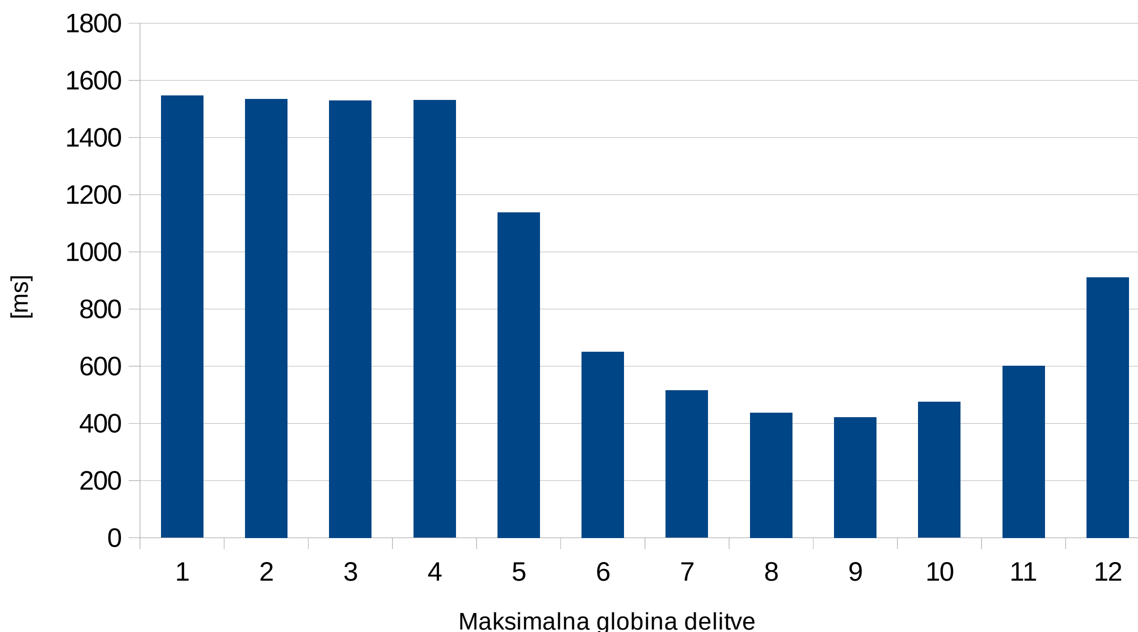


*Slika 46: Globina 5.*



*Slika 48: Globina 8.* *Slika 49: Globina 11.*

Višja maksimalna globina delitve ustvari velika števila podsekcij in s tem večje število robov, potrebnih preverjanja. Nižje globine pa večkrat potrebujejo ročni izračun vseh preostalih točk. Naslednja slika prikazuje računski čas v razmerju z maksimalno globino delitve.




*Slika 50: Računski časi programa pospešenega z deljenjem na sekcije.*

Povprečen računski čas glede na globino delitve.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1546 ms | 1534 ms | 1529 ms | 1530 ms | 1138 ms | 651 ms | 515 ms | 437 ms | 421 ms | 474 ms | 602 ms | 911 ms |

## DINAMIČNA LOČLJIVOST

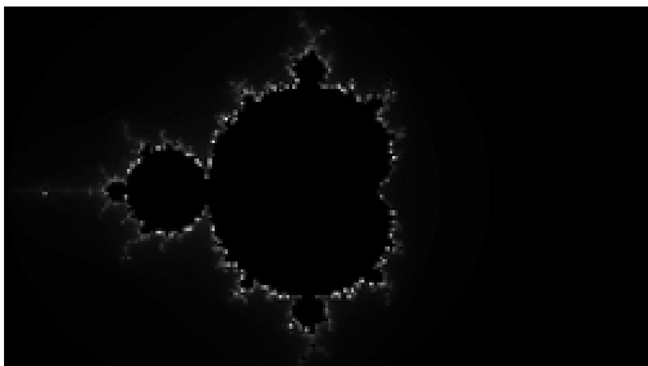
Osnovni algoritem izriše sliko po izračunu vseh točk v vidnem polju, torej je čas med uporabnikovim vnosom (premik/povečava vidnega polja) in izrisom nove posodoblje slike neposredno odvisen od števila točk na zaslonu.

Za boljšo uporabniško izkušnjo je odzivna hitrost na prvem mestu. Najpreprostejši način drastičnega znižanja računskega časa je nižanje ločljivosti slike in s tem manjšanje števila točk za izračun.

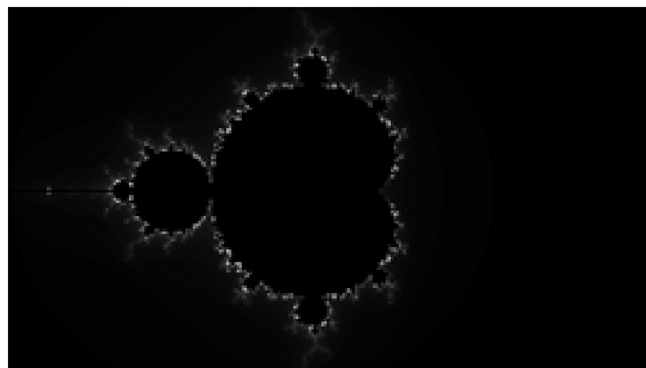
Ob uporabniškem vnosu lahko naročimo programu, da izriše sliko pri npr. 1/10 osnovne ločljivosti, sliko nato čez čas izostri. S tem uporabniku v trenutku omogočimo predogled naslednje slike, prav tako program postane veliko bolj odziven, saj so izrisi nizkoločljivostnih slik praktično takojšnji v primerjavi s slikami polne kvalitete.

Dinamična ločljivost (Sucessive refinement) je algoritem, ki deluje podobno kot deljenje na sekcije, le da je v tem primeru št. sekcij v šrini in višini vnaprej znano. Izračun je izveden na le eni točki sekcije in ta je nato v celoti pobarvana z pridobljeno barvo. [4]

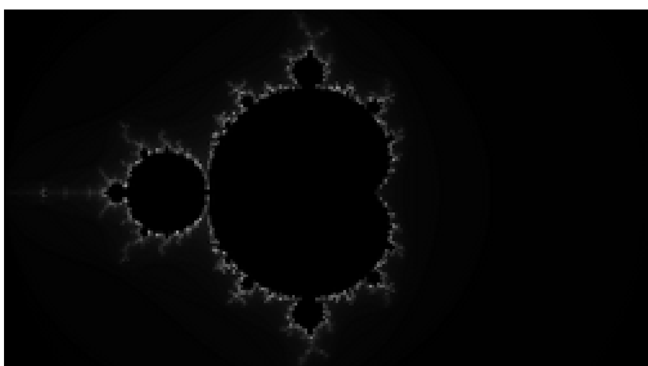
Algoritem najprej razdeli ravnino na npr. 64 x 64 sekcij, v vsaki izračuna po eno točko ter vso sekcijo pobarva in izriše sliko, zatem takoj nadaljuje z deljenjem ravnine na npr. 96 x 96 sekcij in ponovi računski postopek ter posodobi vidno polje. Ta postopek se ponavlja, dokler slika ni zadostne kvalitete ali ne pride do ponovnega uporabniškega vnosa, v tem primeru pa se postopek ponovi.



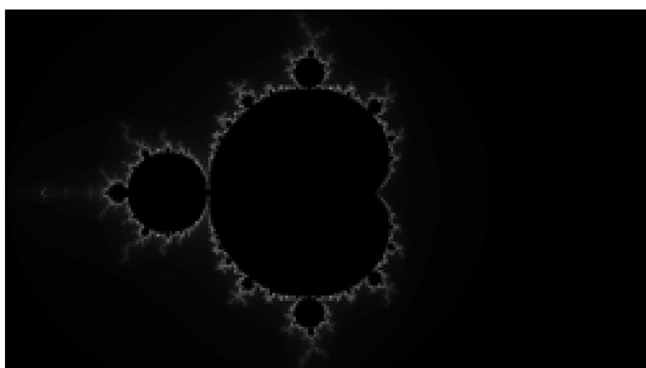
*Slika 51: Ločljivost 240 x 135.*



*Slika 52: Ločljivost 320 x 180.*



*Slika 54: Ločljivost 480 x 270.*



*Slika 53: Ločljivost 960 x 540.*

Prejšnje štiri slike so izrisane med trenutkoma, ko uporabnik požene program, in končnim izrisom slike polne ločljivost brez kakršne koli dodatne "računske cene". S tem se končni računski čas sicer ne spremeni, vendar omogoči takojšnji predogled končne slike.

Povprečen čas izrisa predogleda ločljivosti 10 % osnovne ločljivosti: 15.2 ms.
Povprečen čas izrisa slike polne ločljivosti: 1538.14 ms.

## DELJENJE ITERACIJ

Ob iteraciji točke z ta sledi svoji unikatni orbiti. V primeru, da je točka $z_1$ dovolj blizu točke z, lahko pričakujemo, da se bo le ta vedla podobno. S podobnimi orbitami si obe točki delita tudi podobno število iteracij pred pobegom (v primeru divergence).

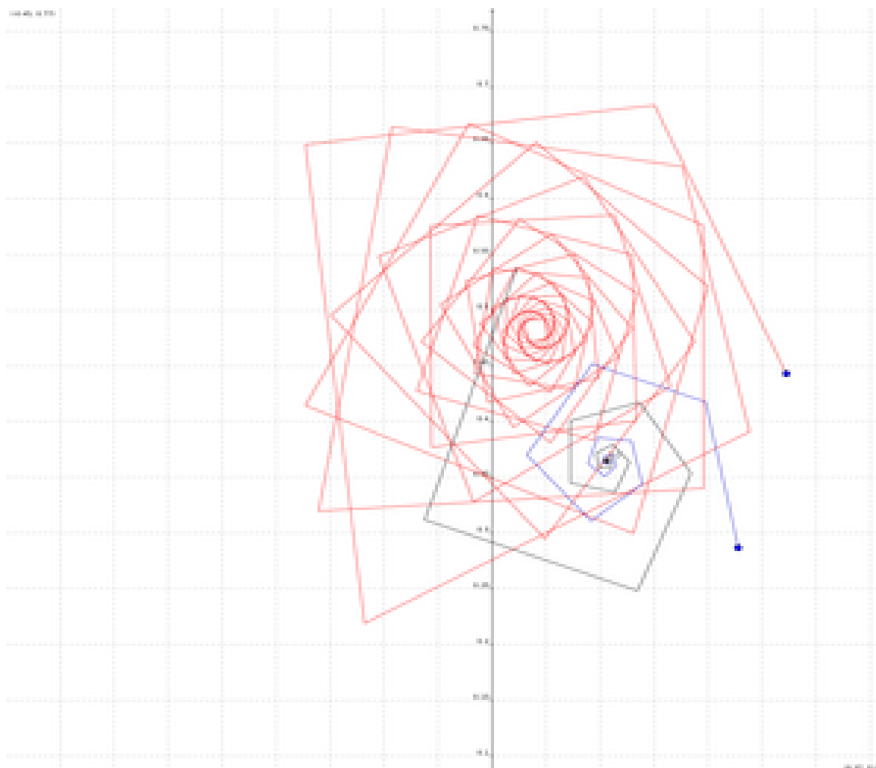
Z delitvijo vidnega polja na sekcije po 9 točk (3 x 3) ter izračunom centralne lahko izračunamo precej dober približek vseh ostalih 8 točk okoli centra.

Najprej izračunamo centralno točko s polno natančnostjo. Pri n-ti iteraciji shranimo vrednost $z_n$. Izračun vseh sosenjih točk pričnemo z uporabo $z_0 = z_n$ namesto 0 + 0i in zmanjšamo $It_{max}$ za n.

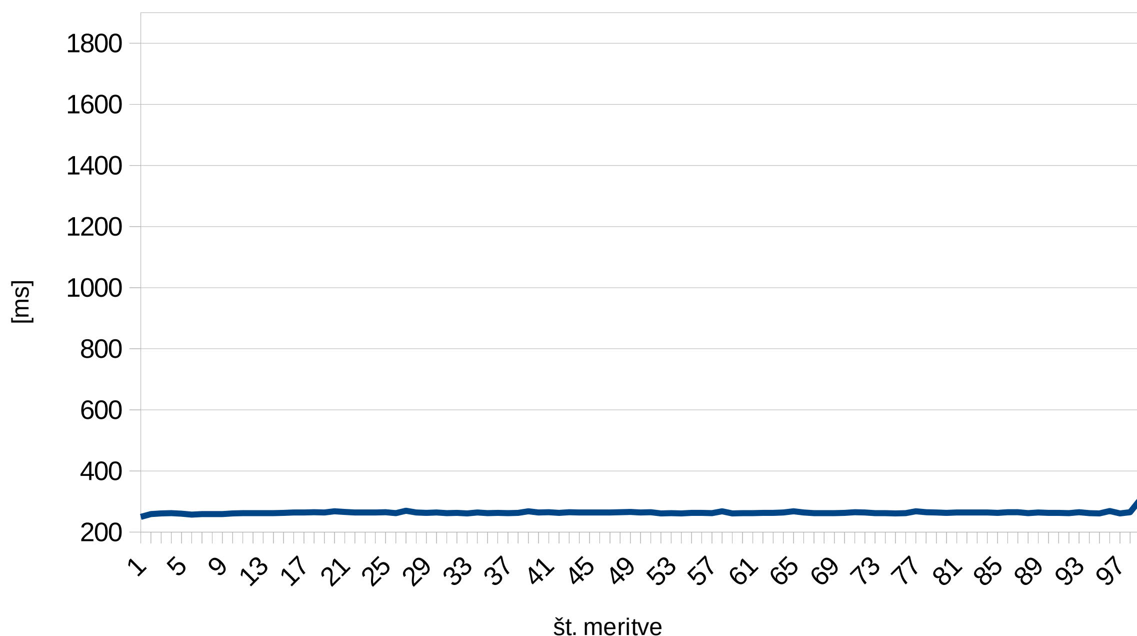
Višja, kot je vrednost n, manjša bo natančnost približka, saj preskočimo večji del izračunov.

| Primer za n = 50: | Točen izračun za $c_0$ | Približek za $c_0$ glede na $c_1$ |
|---|---|---|
| $c_1 = 0.27 + 0.44i$, $c_0 = 0.25 + 0.21i$ | $z_{100} = 0.18 + 0.32i$ | $z_{100} = 0.18 + 0.33i$ |

Rdeča orbita predstavlja referenčno orbito $c_1$. Modra predstavlja natančno orbito $c_0$. Črna predstavlja orbito približka $c_0$ glede na $c_1$. Čeprav približek izvede pol manj iteracij, ta izjemno hitro konvergira proti enaki točki kot $c_0$.



*Slika 55: Primer približka orbite*




*Slika 56: Računski časi programa z deljenjem iteracij, povprečje:* 263.62 *ms.*

Algoritem z visoko natančnostjo izračuna vsako deveto točko. Nato izračuna približke vseh 8 sosednjih točk pri n = 950. S tem prihranimo izračun 1.7mrd[2] iteracij.

[2] V osnovi je za izračun potrebnih 1080×1920×1000 = 2 073 600 000 iteracij.
Natančen izračun vsake devete točke: 1080×1920×1000 / 9 = 230 400 000
Izračun 50 iteracij ostalih točk: ((1080×1920) × (8/9) ) × 50 = 92 160 000
Skupno: 230 400 000 + 92 160 000 = 322 560 000
Razlika: 2 073 600 000 – 322 560 000 = 1 751 040 000

# IZRAČUN KORAKA

Osnovni algoritem [Alg1] za pridobitev kompleksne vrednosti vsake točke na zaslonu uporablja preslikavno funkcijo. Ta preslika vrednost n iz razpona [a-b] v nek drug razpon [c-d] tako, da ostanejo razmerja med a-n, n-b ter c-n, n-d enaka.

$$f(x, x_{min}, x_{max}, y_{min}, y_{max}) = \frac{x - x_{min}}{x_{max} - x_{min}} \times (y_{max} - y_{min}) + y_{min}$$

*Slika 57: Preslikavna funkcija.*

Legenda:
x - vrednost, ki jo preslikujemo,
$x_{min}$ - minimalna vrednost, ki jo lahko prevzame x,
$x_{max}$ - maksimalna vrednost, ki jo lahko prevzame x,
$y_{min}$ - minimalna vrednost razpona, v katerega slikamo x,
$y_{max}$ - maksimalna vrednost razpona, v katerega slikamo x,

Primer: Vrednost 5 želimo slikati iz razpona 0 do 10 v razpon 0 do 20

$$f(5, 0, 10, 0, 20) = 10$$

Iz formule je razvidno, da je preslikavna funkcija računsko neprijazna, saj zahteva skupno 6 osnovnih operacij, kar nanense na 12 operacij za točko. Preslikava x, y koordinat točke poteka ločeno.
Za en izris ločljivosti 1080p to nanese na  24.5 mil (1080 ×1920×12) operacij.

Ker gre za linearno transformacijo, bo razdalja med dvema zaporednima točkama  vedno enaka.

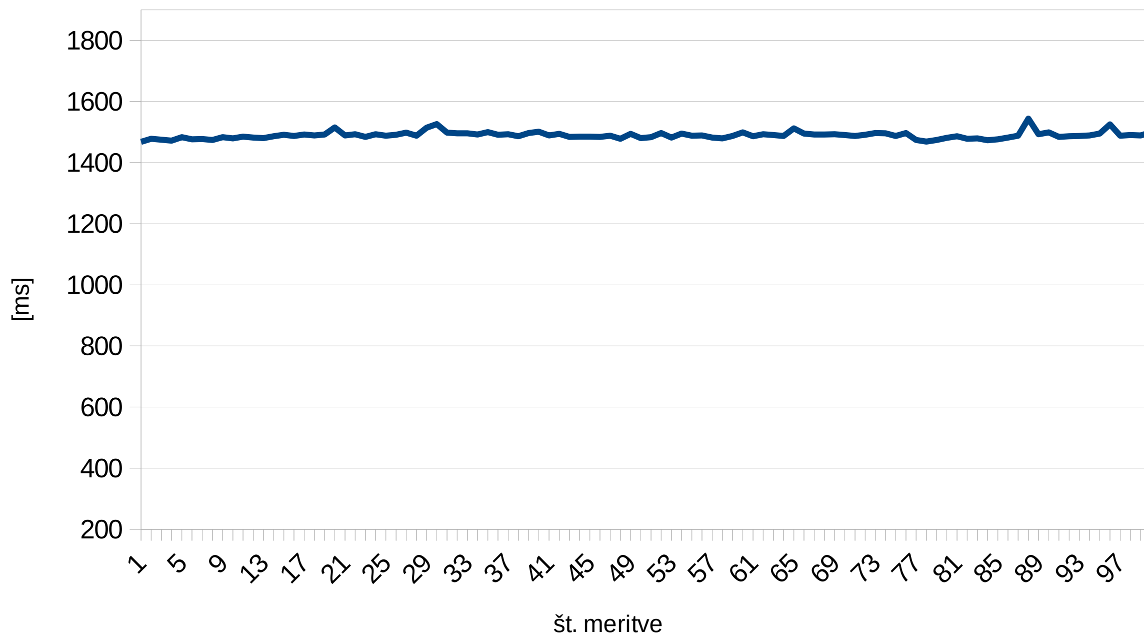
$$\Delta_n = z_{n+1} - z_n,\ \Delta_m = z_{m+1} - z_m$$

Velja: $\Delta_n = \Delta_m$ v primeru, da so točke n, n+1, m ter m+1 zaporedne in kolinearne.

Naj bo $\Delta_0 = z_{0,0} - z_{1,0}$ , kjer je $z_{0,0}$ zgornje levo ogljišče vidnega polja, točka $z_{1,0}$ pa njen desni sosed.
Naj bo $\Delta_1 = z_{0,0} - z_{0,1}$ , kjer je $z_{0,0}$ zgornje levo ogljišče vidnega polja, točka $z_{0,1}$ pa njen spodnji sosed.

Z uporabo $\Delta_0$ in $\Delta_1$ lahko izračunamo katerokoli točko $z_{n,m}$ brez uporabe preslikavne funkcije.

Ko program  izračuna točko $z_{0,0}$ lahko sedaj pridobi točko $z_{0,1}$ tako, da točki $z_{0,0}$ preprosto prišteje $\Delta_1$. Ob izračunu zadnje točke $z_{0,1079}$, lahko pridobi naslednjo točko $z_{1,0}$ tako, da ponovno točki  $z_{0,0}$ prišteje $\Delta_0$. Tako lahko pridobi vsako točko ravnine z le 6 klici preslikavne funkcije (po 2 za vsako izmed slednjih točk $z_{0,0}$ , $z_{0,1}$ , $z_{1,0}$).




*Slika 58: Računski časi algoritma z izračunanim korakom, povprečje:* 1489.33 ms.

Pospeški niso velik,i saj se znebimo le 24 mil. osnovnih aritmetičnih operacij, ki pa jih računalnik izvaja izjemno učinkovito.

# RAČUNALNIŠKA OPTIMIZACIJA

V matematičnem in splošnem delu so bile predstavljene optimizacije, s katerimi si prihranimo čas in zmanjšamo število računskih operacij za izris slike. Naslednji del pa bo prikazal nekaj načinov, kako najbolj izkoristiti računsko moč računalnika.

## VEČNITNOST

Trenutni program se izvaja v le eni programski niti, kar nudi preprosto in učinkovito možnost za izboljšavo – večniti program.

V zelo posplošenem pomenu nam večnitnost omogča vzporedno izvršitev večih kopij našega programa. Nit si lahko predstavljamo kot kos celotnega programa, ki se izvaja na enem od procesorjevih jeder, ločen od preostale celote.

Večina modernih procsorjev ima več kot 4 jedra, kar jim omogoča hkratno izvršitev več niti in/ali programov. Vsako procesorjevo jedro ponavadi izvaja več procesov. To počnejo tako, da izvajajo vsak program za le kratek časovni interval, nato pa pričnejo z delom na drugem. Dolžina in pogostost teh intervalov je odvisna predvsem od pomembnosti procesa; proces, ključen za delovanje operacijskega sistema, bo vedno pridobil več in daljše intervale v primerjavi z uporabnišim programom, npr. Microsoft Word. Ta hirearhija procesov je ključnega pomena za odzivnost sistema ter hitro in učinkovito delovanje ključnih delov rač. sistema.

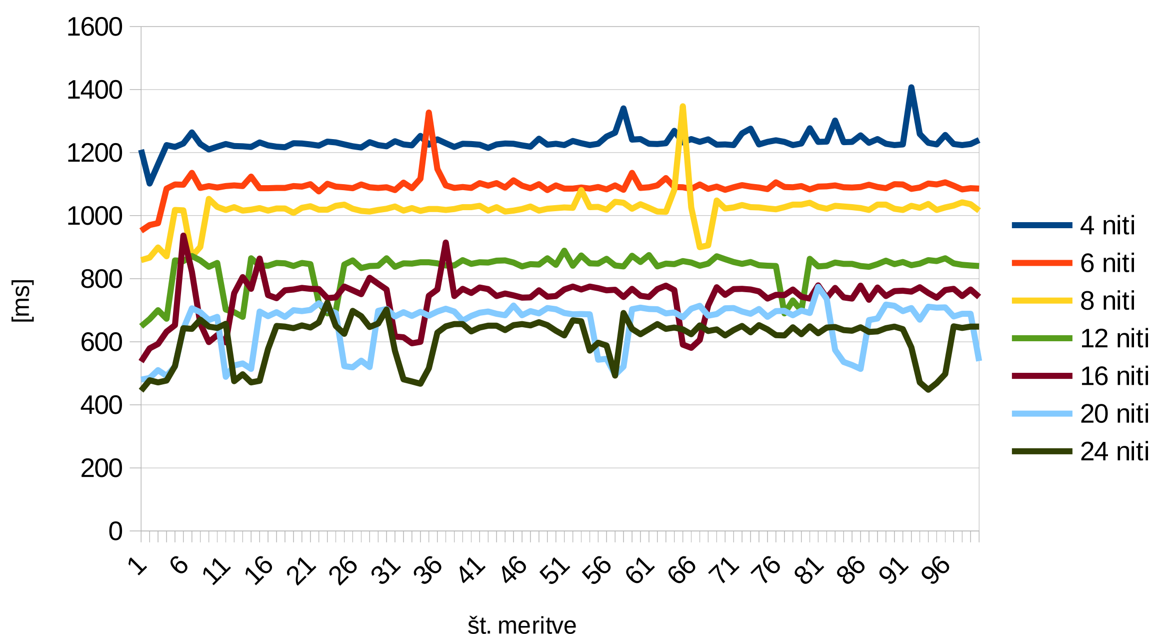
V idealnem svetu, kjer bi procesor izvajal le eno nalogo brez kakršnih koli motenj, bi v primeru, ko naš program razdelimo na štiri “podprograme” ali niti in vsako od teh pognali na svojem procesorjevem jedru, lahko pričakovali 4x znižan izvršilni čas programa. To je zato, ker je vsako procesorjevo jedro skoraj popolnoma nedovisno od ostalih. Če jedro A izvaja program A, ne bo imelo nikakršnih vplivov na izvršitev programa B na jedru B.
Delitev računskega dela med več niti je relativno preprosta.

Za izris KSF je potreben izračun velikega števila iteracij. Te lahko razdelimo na štiri enake dele, nato pa enakomerno zaposlimo vsako jedro procesorja s svojo četrino. V tem primeru vsako od jeder opravlja 25 % osnovnega računskega problema in bo temu primerno končalo 75 % hitreje; ker vsako od jeder opravlja enako nalogo in konča v istem času bo celoten program končal 75 % hitreje.

Pri pisanju večnitnih programov so pazljivo delo z vzporednimi nitmi, nitna varnost, testiranje in optimizacija ključnega pomena.

Ključnega pomena je koordinacija več niti tako, da so med seboj neodvisne, hkrati pa ne poskušajo dostopati do istih podatkov in imajo še vedno dostop do vseh podatkovnih virov.

V primeru, da ena ali več niti poskušajo dostopati do enakega podatka takšen, dogodek v večini primerov privede to sesutja programa. Pogost problem, na katerega sem večkrat naletel pri izdelavi programa, so tudi ti. "smrtne zanke". Gre za problem sinhronizacije, ko nit A prične čakati na konec izvedbe niti B brez zavedanja, da nit B prav tako čaka na konec izvedbe niti A, kar privede do neskončnega čakanja.




*Slika 59: Računski časi večnitnega programa.*

Z višanjem št. niti je opazno upadanje računskega časa. Čeprav pridobitve začnejo upadati nad 16 nitmi, je glede na rezultate 24 niti program najboljša izbira.

| Št. niti | 4 | 6 | 8 | 12 | 16 | 20 | 24 |
|---|---|---|---|---|---|---|---|
| Povrečje | 1233.28 ms | 1092.61 ms | 1018.19 ms | 829.84 ms | 738.72 ms | 656.51 ms | 613.11 ms |

V prejšnji implementacij je vsaka izmed niti prejela enak delež računskega dela. Vendar to še ne pomeni, da je vsaka nit opravila enako količino dela. Ker je točke, ki so del KSF navadno najteže izračunati (iteracijski postopek pripeljejo do konca), bo nit, ki prejme največ točk, ki so del KSF, končala najkasneje.

Primer delitve točk KSF:

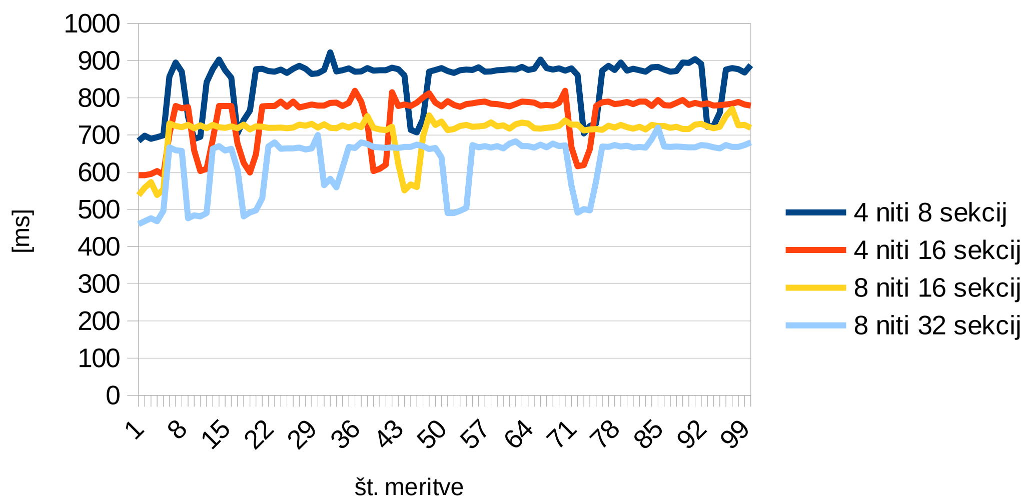
- nit a prejme 10 %,
- nit b prejme 15 %,
- nit c prejme 75 %.

V tem primeru bosta nit a in b že davno opravili svoje delo, medtem ko c še vedno računa. V trenutni implementaciji v takšnem primeru a in b čakata, da c opravi svoje delo, ter zapravljata dragocen procesorski čas.

Takšen scenarij je razrešen z delitvijo dela. Pred začetkom izračunov se delo razdeli na večjo količino manjših nalog (v primeru izračuna fraktalov - sekcij ravnine). Ob začetku izračunov vsaka nit prejme eno izmed nalog; ko jo opravi, ji je dodeljena nova. Seveda tako problem ni popolnoma razrešen, vendar so razlike minimalizirane. Vse niti si v tem primeru delijo skupno zbirko nalog, v primeru, da nit a zaostane, lahko preostale prevzamejo njeno delo.

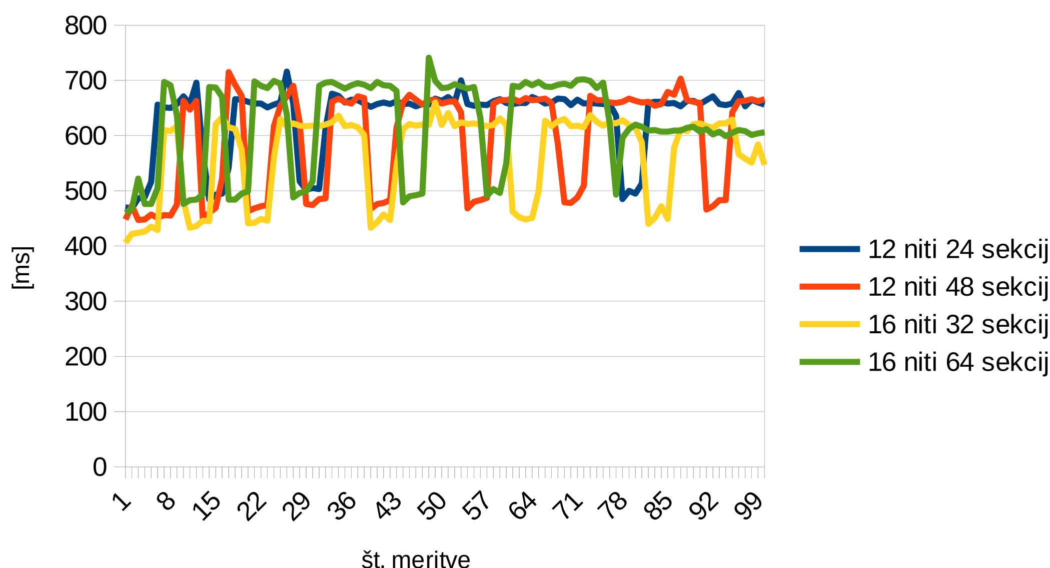
Velikost naloge je v tem primeru ključnega pomena. Če je med n niti porazdeljeno ogromno izjemno majhnih nalog, bodo upočasnile sistem. Upočasnitev je posledica sinhronizacije niti, saj ko je niti dodeljena naloga, mora biti sinhronizirana z nitjo, ki ji dodeljuje naloge, kar pomeni čakanje. Več majhnih nalog pomeni hitro izvedbo in večjo količino sinhronizacij.

Pri izračunu M to ni večja težava saj so naloge precej časovno zahtevne.




*Slika 60: Računski časi algoritma z deljenim delom 1*

| Konfiguracija | 4 niti 8 sekcij | 4 niti 16 sekcij | 8 niti 16 sekcij | 8 niti 32 sekcij |
|---|---|---|---|---|
| Povprečen čas | 844.85 ms | 751.22 ms | 709.15 ms | 627.09 ms |




*Slika 61: Računski časi večnitnega algoritma z deljenim delom 2*

| Konfiguracija | 12 niti 24 sekcij | 12 niti 48 sekcij | 16 niti 32 sekcij | 16 niti 64 sekcij |
|---|---|---|---|---|
| Povprečen čas | 632.21 ms | 591.67 ms | 566.11 ms | 616.59 ms |

Trend nižjih računskih časov z višanjem števila niti se ohranja, saj procesor procesom iste pomembnosti v večini dodeljuje enake količine ter dolžine delovnih intervalov. Z višanjem števila niti povišamo število intrvalov in posledično količino procesorjeve moči, ki jo program s tem pridobi.

Pokaže pa se tudi upočasnitev z višjimi števili sekcij. Postavitev 16 niti ter 64 nalog je dokaj počasnejša od 16 niti za 32 nalog.

## POSPEŠEVANJE Z GPE

Grafične kartice ali GPE so ključni sestavni del vsakega računalnika, saj omogočajo izris slike na zaslon. V zgodnjih dneh računalništva so za izris skrbeli sami CPE, saj so bile grafične zahteve še nizke, zasloni so imeli zelo omejeno število barv ter so bili izjemno nizkih resolucij v primerajvi z današnjimi.

Z razvojem zaslonov večjih resolucij, podporo več barv ter grafično zahtevnejših programov so sčasoma CPE postali prešibki za hkratno procesiranje splošnih in grafičnih nalog. Grafične kartice so bile ustvarjene z namenom odložitve grafičnih nalog s strani CPE. Hiter razvoj strojne in programske opreme je v nekaj desetletjih privedel do grafično procesnih enot, kot so Nvidia RTX 4090 s 16384 jedri, ki delujejo na frekvencah do 2.5 GHz. Vsako grafično jedro je z vidika zmožnosti opravljanja računov skoraj enakovredno procesorskim[3].

Grafične kartice so v osnovi namenjene za obdelavo ogromne količine podatkov, temu primerno imajo več tisoč procesorskih jeder, ki lahko v vzporedni postavi delajo in obdelujejo vsakovrstne probleme, povezane primarno z grafičnimi nalogami.

V primerjavi z modernimi procesorji so po številu FLOP večstokrat hitrejše. V primeru "povprečnega” procesorja"[4], ki v najboljšem primeru izvede do 50 GFLOPS je RTX 4090 1650x hitrejša pri okoli 82.5 TFlops, vredne omembe pa so tudi Nvidine profesionalne grafične kartice serije A100, ki lahko dosežejo tudi do 312 TFLOPS. [5][6]

Problem izračuna KSF je idealen primer naloge za grafično procesno enoto. Imamo ogromno količno podatkov (več milijonov kompleksnih števil), nad katerimi je potrebno izvesti več tisoč opearcij (iterirati fraktalno formulo).

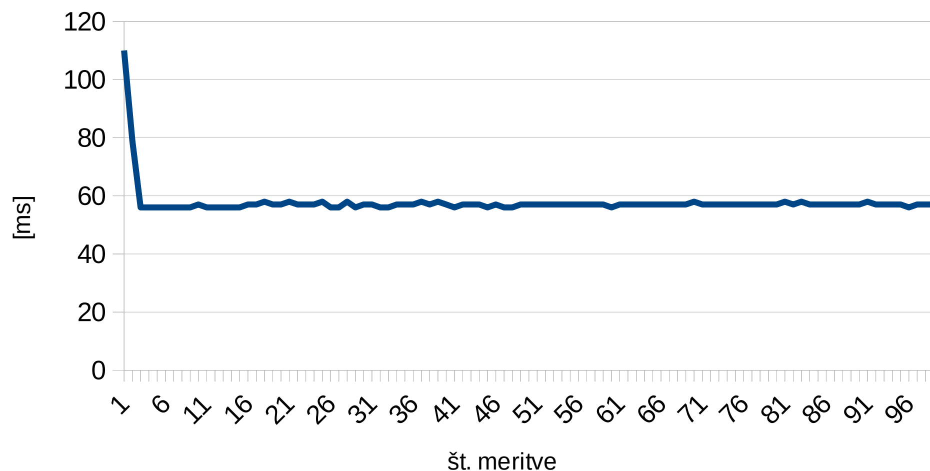
CUDA vmesnik nam omogoča  prav to; namesto uporabe procesorjevih jeder lahko ustvarimo program, ki učinkovito porazdeli računsko delo na jedra grafične kartice in s tem prinese ogromen računski pospešek.

Čeprav GPE nudijo ogromne pospeške pri izvajanju vsakovrstnih računskih nalog, niso primerne za to nalogo. Trenutno knjižnica za podatkovne tipe arbitrarne natančnosti, namenjena izvedbi na GPE, ne obstaja, kar omeji program na natančnost standardnih tipov, kot so double in float.

Za pisanje kode, namenjene izvedbi na GPE, je potrebno upoštevati veliko dejavnikov, ki niso prisotni pri pisanju kode za CPE. Za stvaritev idealnega algoritma, primernega izvedbi na GPE, je potrebno poznavanje arhitekture grafične enote in hitrosti PCIe vmesnika; Paziti moramo na pasovno širino grafičnega spomina, enakomerno porazdelitev računskega dela in primerno napisan program.

Zaradi velikega števila procesorskih jeder grafičnih enot je potrebno eksperimentirati s števlnimi faktorji, kot so število niti za hkratno izvedbo, saj razmerje števila niti ter procesorskih jeder 1:1 ni vedno najhitrejše (točna razmerja so odvisna od arhitekture grafičnega procesorja).
Vsaka nit mora biti popolnoma neodvisna od ostalih. Za dostop do skupnih virov morajo biti vse niti pravilno sinhronizirane, da se izognemo nepotrebnim zaustavitvam zaradi čakanja.

Prenos podatkov med GPE in CPE mora bit pazljivo načrtovan, saj prepogosti prenosi upočasnijo delovanje progama. Večji, a redkejši prenosi pa program upočasnijo, ker porabijo več časa za premik podatkov s sistemskega pomnilnika v pomnilnik GPE. V idealnem scenariju, kjer je edina omejitev hitrost PCIe vmesnika med CPE in GPE, je za hitrost vseh vrednosti, potrebnih za izračun množice, potrebnih skoraj 20 milisekund.




*Slika 62: Računski čas programa, izvedenega na GPE, povprečje: 57.3 ms.*

**Meritve hitrosti GPE pospešenega programa so bile izvedene na drugem sistemu [Spec2].**

2 milijona niti, izvedenih na 2304 jedrih GPE, omogoča skoraj realnočasoven izris brez kakršnih koli dodatnih optimizacij. Z dodatno optimizacijo lahko preprosto dosežemo še nižje računske čase, vendar bo takšen program omejen s strani računske natančnosti.

[3] Jedra CPE so specializirana za izjemno hitro in natančno delo na eno nitnih programih, medtem ko so grafična jedra specializirana za več tisoč nitne programe z prioriteto na količini izračunov in ne toliko na hitrosti ali zanesljivosti.

[4] Merjenje FLOPS na procesorskih enotah je bolj kompleksno v primerjavi z GPE zato točni podatki o le teh ne obstajao. V tem primeru je bil FLOPS pridobljen z slednjo formulo
$FLOPS = \text{Št. Jeder} \times \text{Frekvenca} \times \text{Št. Ukazov na urin cikel} \times \text{Št. Float procesnih enot}$
$(50GFLOP \simeq 4 * 3.0Ghz * 2 * 2)$

## SIMD UKAZI

V računalništvu poznamo dva vrsti izvršitve ukazov: skalarne in vektorske.

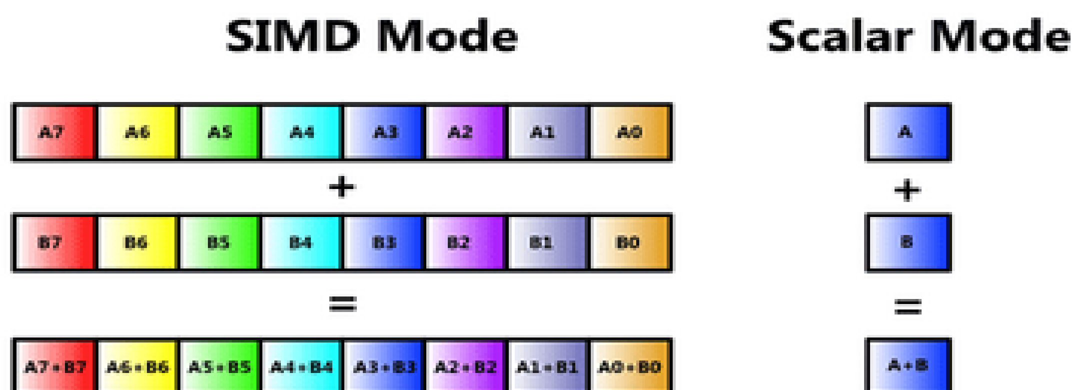
Pomembno je omeniti, da imata v kontekstu računalniških operacij skalar in vektor drugačne pomene kot v matematičnih okoljih. V smislu matematike ali fizike je skalar numerična vrednost ali enota, npr. 2, -5, 6 m, 1 kg, vektor pa geometrični objekt, ki ima smer in magnitudo, npr. sila, hitrost.

V računalništvu so koncepti podobni, skalar predstavlja eno samo numerično vrednost, vektor pa je zbirka skalarjev in jih predstavlja več.

| | skalar | vektor |
|---|---|---|
| mat. | 3.14 | sila teže na jabolko |
| rač. | S = 3.14 | V = {1, 2, 3, 4, 5} |

Procesorji so sestavljeni iz raznih sestavnih delov, najpomembnejši izmed njih je ALU, ki predstavlja možgane računalnika, saj se v njih izvajajo vse računske in logične operacije. ALU je specializiran in izvaja izključno skalarne operacije npr. 1 + 1 = 2. Prednost ALU je univerzalnost, ki omogoča ogromno število operacij na raznovrstnih podatkih (standardni ukazni set x86-64 je sestavljen iz več tisoč raznovrstnih ukazov, točne številke so odvisne od starosti in arhitekture specifičnega procesorja).

Poznamo pa tudi vektorsko procesne enote ali VPE. Ločene od glavnega ALU so enote specializiranje za procesiranje vektorskih vrednosti, npr. {1, 2, 3, 4} + {4, 3, 2, 1} = {5, 5, 5, 5}. Sicer so manj univerzalne (najnovejša zbrika AVX512 ima le nekaj sto ukazov, točno število odvisno od procesorja), vendar omogočajo procesiranje več podatkov vzporedno, kar omogoča ogromno procesno moč. Preprosta adicija dveh števil na ALU vzame enako število[5] urnih ciklov kot adicija dveh vektorjev na VPE.




*Slika 63: Vektorsko in skalarno procesiranje.*

[5] Operacije nad manjšim številom podatkov bodo v nekaterih primerih hitreje izvedene na ALU, SIMD ukazi so namenjeni procesiranju velikih količin podatkov, in ob pravilni implementacij prinesejo izjemne rezultate v obliki hitrejših izvršilnih časov.

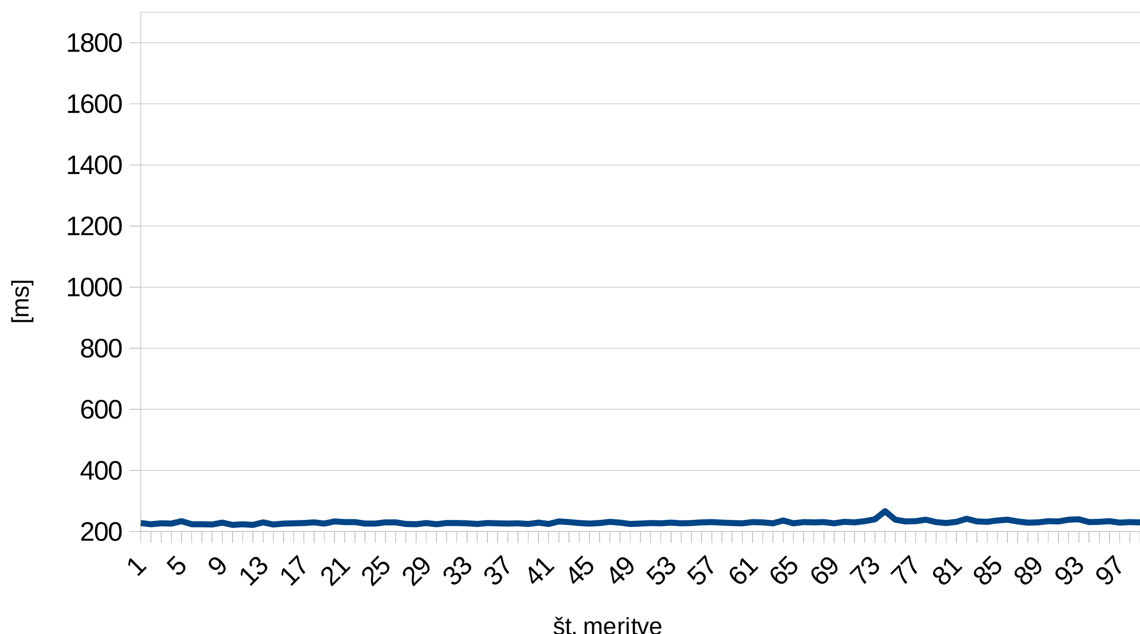
SIMD ukazi so namenjeni vektrostko-procesnim enotam in nam omogočajo nadzor nad njihovim delovanjem. Ukazna zbirka AVX512 ponuja 32, 512 bitnih registrov, v katere lahko shranimo kakršnokoli numerično ali logično vrednost. Z drugimi besedami standard AVX512 prinese VPE, ki lahko hkrati hrani do 32 vektorjev, vsak izmed njih pa lahko hrani do 512 bitov podatkov.

Pisanje SIMD programov je lahko kompleksna naloga, saj je za ustvarjanje učnikovitih programov potrebno globoko razumevanje uporabljene SIMD arhitekture (SSE, AVX, AVX2 ...).

Program mora podatke procesirati vektorsko. Procesorji imajo omejeno število registrov, nad katerimi lahko izvajajo operacije, zato je potrebno njihovo uporabo natančno načrtovati. Za prenos podatkov iz sistemskega spomina v registre morajo biti le ti, za preprečevanje zgrešitev predpomnilnika spominsko poravnani, kar lahko prinese večje preglavice ob uporabi kompleksnejših podatkovnih struktur.

Za najučinkovitejšo izrabo sistemskih virov je potrebno poznavanje tudi specifičnih SIMD ukazov platforme, ki v večini primerov niso podprti z večjimi prevajalniki, zato je pogosto potrebno pisati surov strojni jezik. V takšnih primerih se ni mogoče zanašati na optimizacije prevajalnika, kar pomeni, da sta hitrost in učinkovitost programa odvisni zgolj od programove kakovosti.

Trenutni program deluje na števili tipa double, izmed katerih vsako zavzame 64 bitov, torej lahko en register/vektor hrani do 8 takšnih vrednosti. To omogoči, da z uporabo VPE in SIMD ukazov iteriramo 8 točk hkrati, kar teoretično predstavlja 8x krajši računski čas.

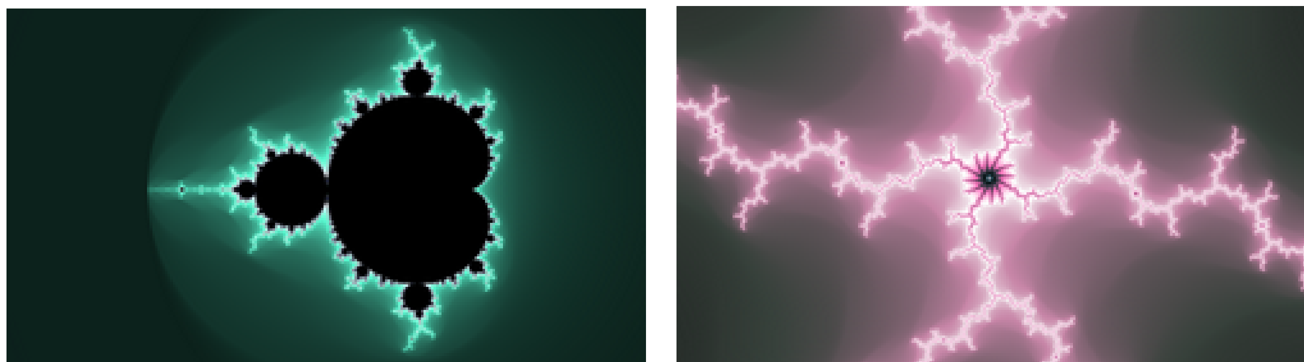


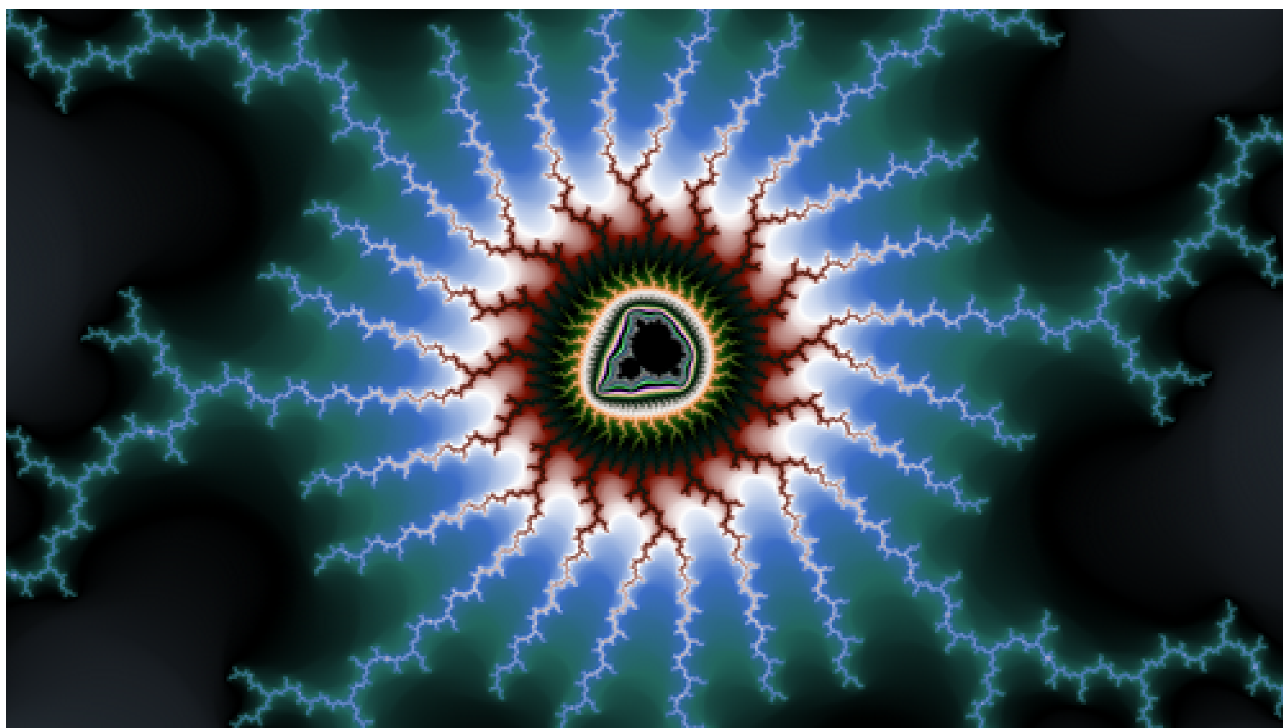

*Slika 64: Računski čas SIMD algoritma, povprečje: 229.2 ms.*

# BARVNI ALGORITMI

Za vse, ki ste že kadarkoli videli sliko Mandelbrotove množice si upam trditi, da ta ni bila dolgočasno črno-bela, kot je bilo predstavljeno do sedaj.

Barvni algoritmi sicer niso ključni del izračuna. Za osnovno vizualizacijo je osnovna preslikavna funckija dovolj, vendar prinese slike slabše kvalitete. Za slike višje kakovosti so potrebni zahtevnejši algorimi, ki pa prinesejo dodatne izračune, kar jih dodatno upočasni. V naslednjem delu bodo predstavljene tri metode za učinkovito barvanje osnovne slike.





*Slika 65: Primeri lepo obarvane Mandelbrotove množice; slike so posnete v programu [Xaos].*

Barvanje množice prinaša vizualno bolj prijeten rezultat, a ta pride s ceno dodatnih izračunov. Za določanje barv obstaja neskončno mnogo forumul vsaka od njih prinese drugačen rezultat, zato je način kako želimo obarvati sliko,  precej subjektiven.

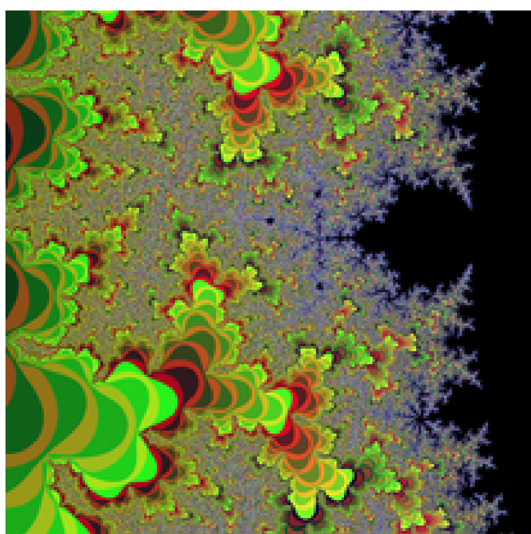
Do sedaj so bile vse slike barvane z najosnovnejšo in računsko najcenejšo funkcijo, ki prejme IttPobega neke točke in vrne monokromatsko barvo, ki ji pripada.

$$f(x)=\begin{cases} r=\mathrm{It}_{max},\ g=0,\ b=0;\ x=0 \\ r=map(x,0,\mathrm{It}_{max},0,255),\ g=r,\ b=r;\ x>0 \end{cases}$$

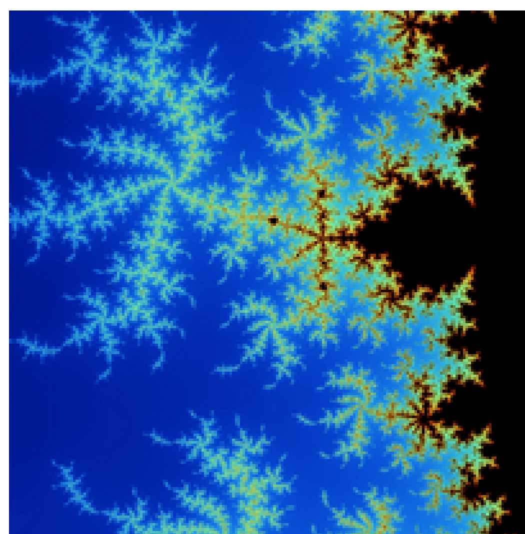
R, g b vrednosti predstavljajo RGB barvne kanale trenutne točke, map pa je preslikavna funkcija [Izračun koraka].

Za preprosto barvno sliko ustreza že katerakoli funkcija, za katero velja f(x) = $\mathbb{R}^3$, kjer je x število iteracij pred pobegom, funkcija pa vrne 3 različne številske parametre.

Barvni algoritmi se delijo na dve skupini, lomljene ter gladke. Vsi algtoritmi in formule, pri katerih so prehodi med različnimi oddtenki barv očitno sodijo med lomljene. Ti zaradi hitrih prehodov med barvami včasih ustvarijo slike slabše kvalitete, predvsem ob robovih množice, kjer se $\mathrm{It}_{esc}$ izjemno hitro spreminja, kar ustvari televizijski statiki podoben učinek.



*Slika 66: Barvna statik.*

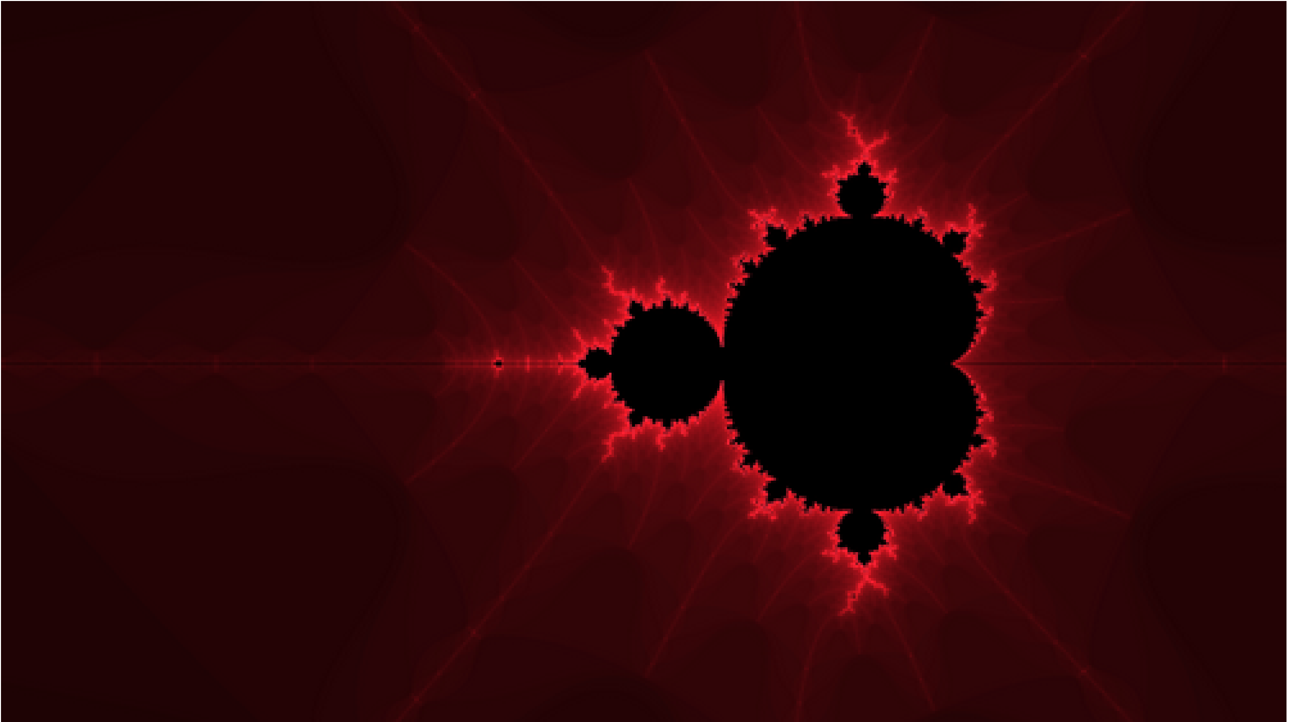


*Slika 67: Gladek barvni algoritem.*

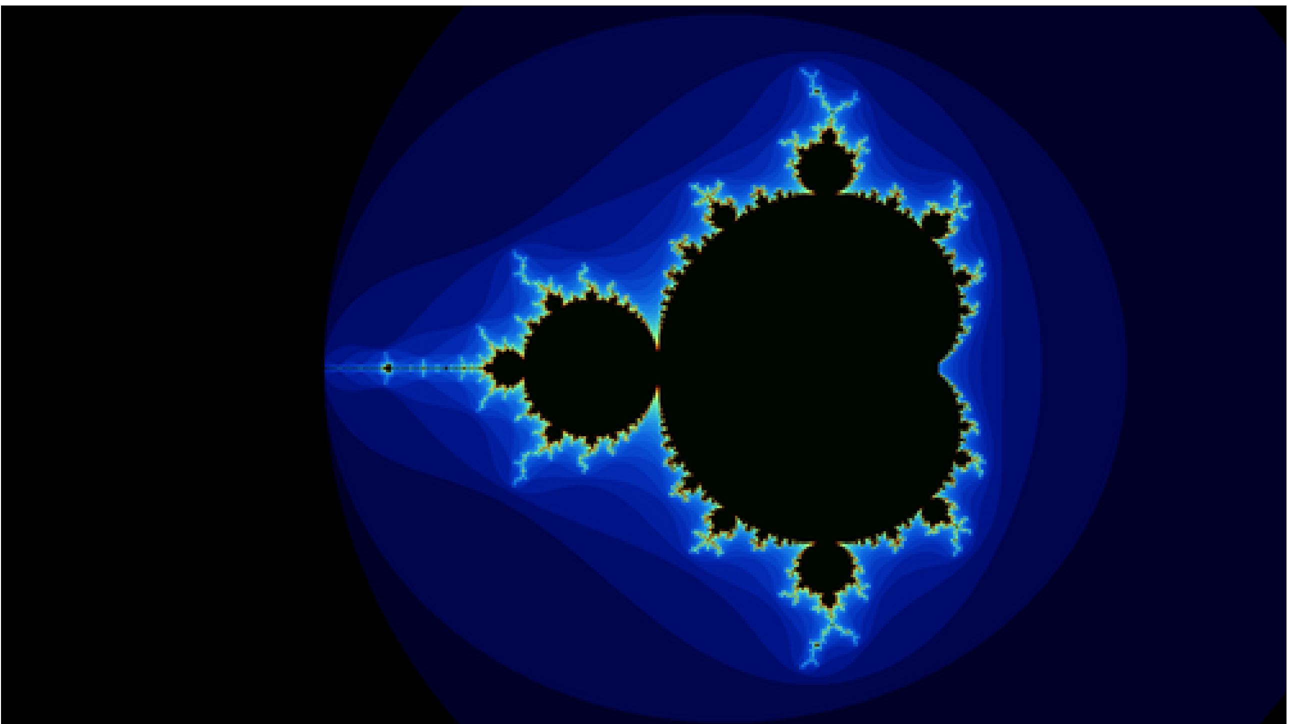
Gladki algoritmi so tisti pri katerih prehod med barvami ni viden ali pa je minimaliziran.
Barvnih algoritmov ni mogoče ravno optimizirati, lahko pa smo pazljivi pri njihovi sestavi. Za hiter algoritem se je pametno izogibati operacijam, kot so korenjenje, fakultete, kotne funkcije, logaritmi ipd.

Z uporabo le osnovnih aritmetičnih operacij se lahko ustvari vizualno zanimiv in hiter barvni algoritem. Demonstrirani bodo na spodnjih slikah.
Legenda:

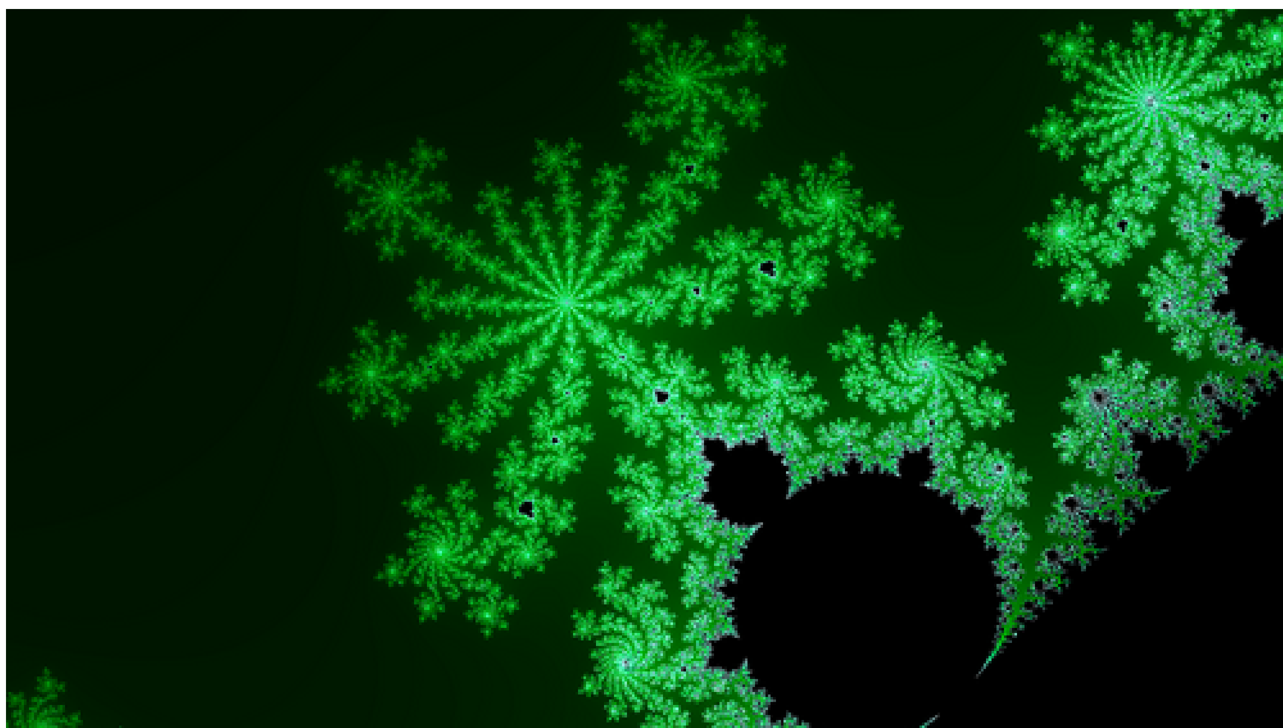
t = preslikavna funkcija($\mathrm{It}_{esc}$, 0, $\mathrm{It}_{max}$, 0, 255)
x = $\mathrm{It}_{esc}$ / $\mathrm{It}_{max}$



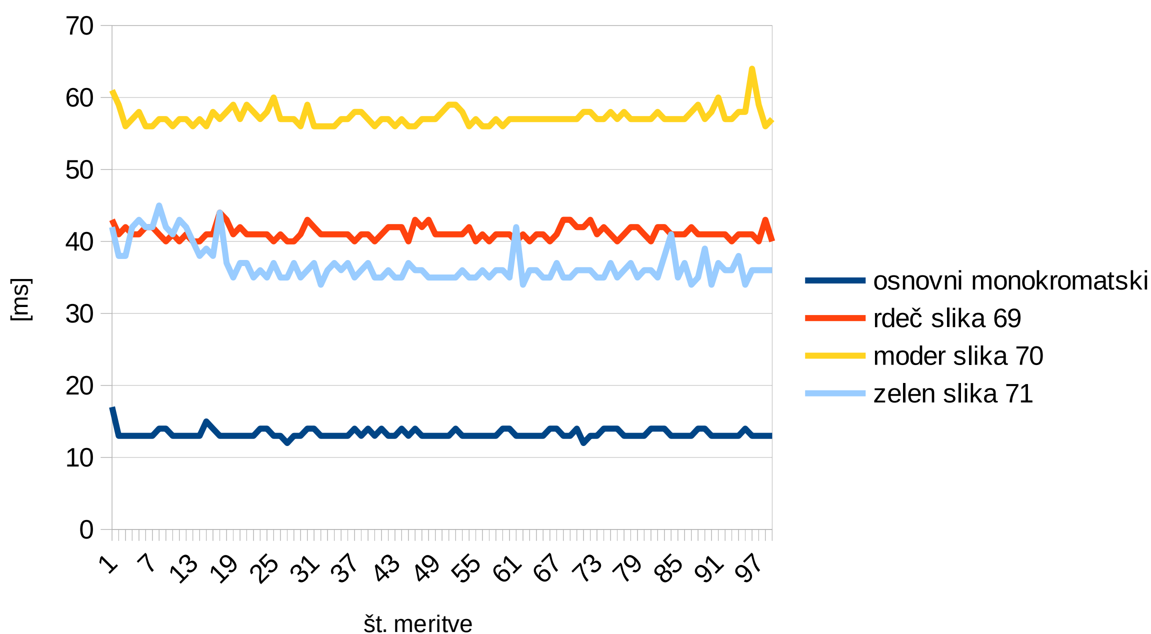
*Slika 68: $r = 2 \times t$, $g = (r/2) \times 0.22$, $b = (r/2) \times 0.33$*



*Slika 69: $r = 9 \times (1-x) \times x^3$ , $g = 15 \times (1-x)^2 \times x^2$ , $b = 8.5 \times (1-x)^3 \times x$*



*Slika 70: r = 1/(-x) + 2, g = 2 × x, b = (r+g) × 0.5*




*Slika 71: Računski časi barvnih algoritmov*

Osnovni barvni algoritem je najpreprostejši in tudi najhitrejši. Sledi mu algoritem slike 70 s 7 osnovnimi aritmetičnimi operacijami. Najpočasnejši in najzahtevnejši pa je algoritem slike 69 z 18 osnovnimi aritmetičnimi operacijami.

| Algoritem | Osnoven | Rdeč (slika 68) | Moder (slika 69) | Zelen (slika 70) |
|---|---:|---:|---:|---:|
| Povprečen čas | 13.3 ms | 41.2 ms | 57.3 ms | 36.8 ms |

## FRAGMENTNI SENČNIKI

Fragmenti senčnik je majhen program, namenjen izvedbi na GPE. Njegova naloga je določanje barv in drugih vizualnih lastnosti neke točke na zaslonu. Uporabljajo jih primarno programi, ki delajo s 3D grafikami, kjer izračunajo barvo neke točke na zaslonu glede svetlost, obliko, teksturo in druge lastnosti objektov v neki 3D sceni.

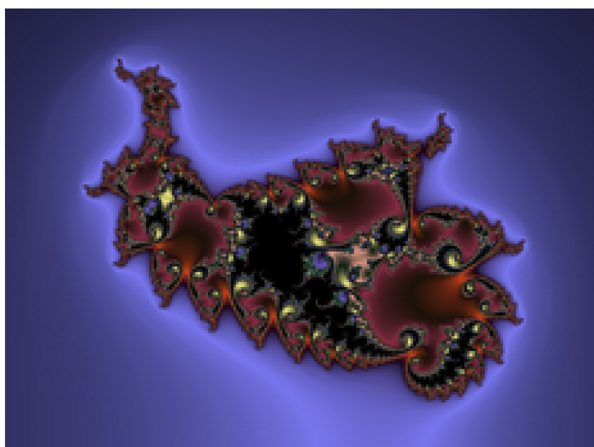
GPE ob izrisu slike za vsako točko zaslona požene po en fragmentni senčnik, zato je v kombinaciji z velikim številom procesnih jeder grafične enote izračun barve vseh točk na zaslnou časovno zanemarljiv v primerjavi s povprečno 14ms potrebnih za izračun barve vseh točk na CPE.

Fragmentni senčnik v primeru izrisa fraktalov omogoča odložitev izračuna barv na GPE, s čimer se znebimo vseh upočasnitev s strani tudi veliko kompleksnejših barvnih algoritmov. Prav tako nam omogočijo uporabo slikovnih učnikov za končne vizualne izboljšave.

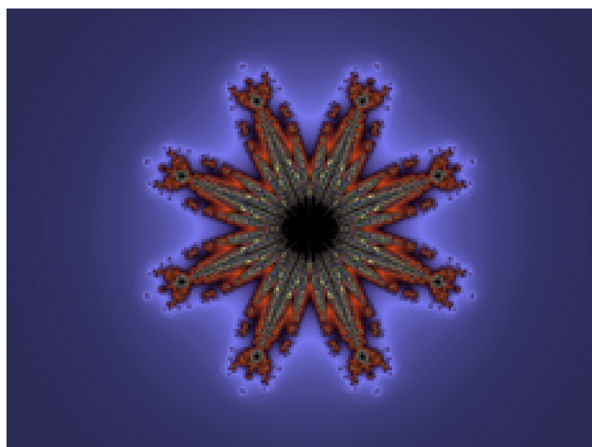
# ŠIRJENJE NA DRUGE FRAKTALE

Z izjemo poenostavitve računov in simetrije so vse do sedaj naštete metode univerzalne in v teoriji omogočajo učinkovio pospeševanje kateregakoli KSF.

Univerzalnost algoritmov omogoča preprosto učinkovito implementacijo novih KSF v končni program.



*Slika 73: KSF Feniks [Xaos].*



*Slika 72: KSF Osmica [Xaos].*

# UPORABLJENE TEHNOLOGIJE

Med izdelavo programa so bila uporabljena razna orodja in tehnologije. V prototipni fazi, je bila večina testnih programov razvitih v programskem okolju Processing, ki s preprostim Javi podobnim jezikom omogoča hitro in učinkovito razvijanje in materiliziranje idej. V fazi razvoja končnega programa pa je bil v uporabi primarno urejevalnik besedil Vim v kombinaciji z jezikom C++.

Slika 74: Processing.

**Processing**

Processing je začetnikom namenjena knjižnica in programsko okolje, namenjeno razvoju programskih umetnosti, vizualizacije idej ipd. Grajen je na jeziku Java, ki je prav tako primarni jezik razvojnega okolja. Podpira vse funkcionalnosti Jave z dodatkom svojih metod in uporabniških knjižic.

Okolje sta ustvarila Reas Casey in Ben Fry v letu 2001; od takrat ostaja Processing eno najboljših orodij za začetnike v programiranju, pa tudi kot za že izkušene programerje, ki zaradi njegove odprtokodne narave lahko razvijajo nove knjižnice, in velike projekte z uporabo Javinih funkcionalnosti. [8]

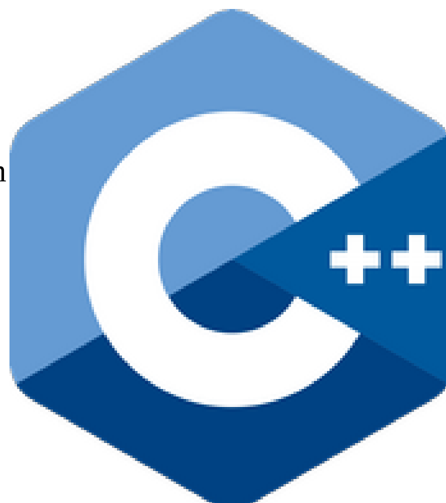
**Vim**

Vim ali Vi-improved je brezplačni odprtokodni tekstovni urejevalnik. Nastal je 2. novembra 1991 kot izboljšana verzija Bill Joyevega Vi tekstovnega urejevalnika. Zaradi njegove velike konfigurativnosti kot dejstva, da je danes vključen v skoraj vsak sistem tipa UNIX, je postal eden najpriljubljenejših tekstovnih urejevalnikov. Zasnovan je za uporabnike tipkovnic in s tem prinaša veliko učinkovitost tistim, ki so se pripravljeni naučiti uporabniškega vmesnika. V osnovi je VIM konzolni urejevalnik besedil in s tem pridobi zelo popularen pri administratorjih in nadzornikih rač. omrežij. Obstaja tudi manj znana namizna aplikacija Nvim. [9]

Slika 75: Vim.

**C++**

C++  je eden najhitrejših programskih jezikov. Ustvarjen je bil kot dodatek jeziku C. Danes je eden svetovno najpopularnejših jezikov. Po sintaksi je podoben C, C# in Javi. Poleg s Cjem primirljivimi hitrostmi, visokim nadzorom nad pomnilnikom in procesorjem pa ponuja še razrede in omogoča  objektno programiranje. Je medplatformni jezik in deluje na večini sistemov Windows, Linux, Mac. [10]



*Slika 76: C++.*

*Slika 77: GCC.*

**GCC**

GCC (gnu compiler collection) je zbirka prevajalnikov. GCC je odprtokodni program z GNU GPL licenco in je trenutno pod razvojem Free software foundation (FSF). Dandanes je industrijski standard in največji prevajalnik za jezike C, C++, Fortran ipd. Je ključni del GNU programske verige in je vključen v skorajda vsak Linux sistem.

V tej raziskovalni nalogi je bil uporabljen g++ prevajalnik, namenjen prevajanju jezika C++. [11]

**NVIDIA CUDA**

CUDA (Compute unified device architecture) je strojno-opremski API, ki uporabnikom omogoča izvedbo nekaterih programov na Nvidiinih grafičnih karticah. S tem omogoča drastično pospeševanje delovanja programov z utilizacijo tisočih  jeder ter masivne pretočnosti pomnilnika grafičnih kartic. CUDA Toolkit je paket programske opreme, ki vsebuje vse potrebno za izdelavo grafično pospešene aplikacije. Prinaša knjižnice za komunikacijo z grafično enoto, prevajalnik za jezike C ali C++ ter CUDA razvijalno okolje. [12]

*Slika 78: CUDA.*

**AVX2/512**
Intel AVX2 (Advanced Vector extensions 2.0) je eden od novejših dodatkov Intel-ovemu ukaznemu setu. Razširja osnovni AVX z 256 bitnimi float, intiger in FMA ukazi. Namenjen je predvsem izboljšanju hitrosti kodeksov in programov za procesiranje slik in videa, ki ponavadi potrebujejo večje računske hitrosti. AVX2 je bil nasleden z AVX512, ki v osnovi prinese 512 bitne registre, vendar je podprt le na peščici CPE v primerjavi z njegovim predhodnikom. [13]

**MPFR**
MPFR (Multiple-precision floating-point) je opdprtokodna knjižnica jezika C za izvajanje računskih operacij s poljubno natančnostjo. Stoji na GMP (GNU multiple precision) knjižnici, vendar jo nadgradi z večjo učinkovitostjo in bolje definirano semantiko. [14]

**SFML**
SFML ali Simple and fast multimedija library je programska knjižnica. Namenjena je ustvarjanju hitrega in preprostega API za razvoj multimedijskih aplikacij. Podpira delo z zvokom, omrežjem in paketi, sistemom, grafiko, I/O napravami ipd.
V tej nalogij je bil uporabljen le grafični del knjižnice, ki z abstrakcijo kontekstov knjižnice OpenGL poenostavi razvoj grafičnih aplikacij.
Podpira razvoj programov v več jezikih: C++, C, Ruby, Java, C#, python… Kot odprtokodni program si funkcionalnost lahko vsakdo priredi za razvoj programa po svojih željah. [15]

*Slika 79: SFML.*

# PREDSTAVITEV APLIKACIJE

S pomočjo novo pridobljenega znanja o optimizaciji algoritmov sem usvaril končno aplikacijo, ki koristi najboljše od skoraj vseh do sedaj omenjenih optimizacij. Glede na okoliščine lahko izrisuje tudi več kot 60 sličic na sekundo.

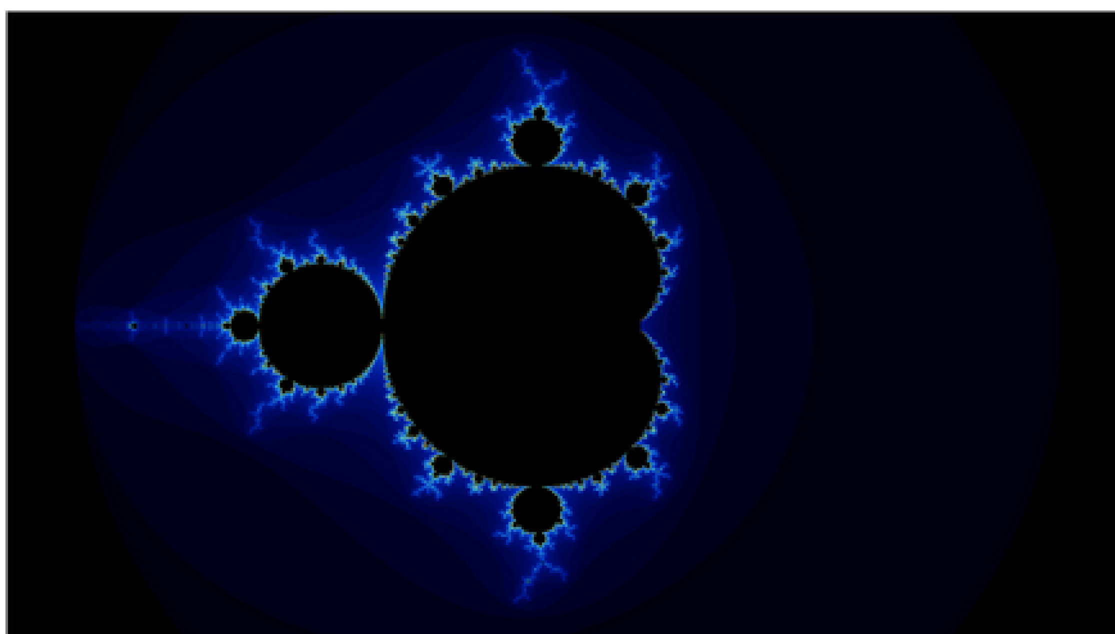
Ob zagonu se uporabniku izpišejo navodila za uporabo, saj program nima uporabniškega vmesnika, nato pa se prikaže še glavno okno.

```
==================================================
                    WFractal




Opis                                  Tipka
==================================================
Za premikanje:                        WASD
Za spremembo barvnega algoritma:      CTRL+1...9
Za spremebo fraktala:                 SHIFT+1/2/3
Zajem vidnega polja:                  F
Povečaj/manjšaj iteracije:            J/K
Ponastavi vidno polje:                0
Prosti pad:                           SPACE
```

*Slika 80: Navodila za uporabo.*



*Slika 81: Osnovni pogled.*

Uporabnik lahko v programu z uporabo tipk WASD pomakne vidno polje za 20 % v katero koli smer, enako lahko stori z levim klikom miške. Miškin koleščček je uporabljen za povečavo v točko, ki se nahaja pod miškinim kazalcem.

Z uporabo kombinacije tipk CTRL ter številk v številčni vrstici je mogoče zamenjati barvni algoritem. Podobno SHIFT in številke 1, 2, 3 omogočajo menjavo fraktalne formule.

Tipka F omogoča izvoz trenutne scene v .png formatu.

Med raziskovanjem se lahko zgodi, da je število iteracij premajhno, kar lahko uredimo s kombinacijo tipk K in J za višanje in nižanje števila iteracij.

Pritisk preslednice zažene prosti pad. Program izbere naključno točko in prične povečevati vanjo. Pritisk tipke 0 pa ponastavi vidno polje na osnovni pogled (slika 81).

Delovanje končnega programa je obrazloženo v prilogi [Kon]

# ANALIZA REZULTATOV

Ob začetku izdelave raziskovalne naloge je bilo postavljenih nekaj hipotez, ki so bile med raziskovanjem ovrednotene ter posledično potrjene ali ovržene.

## ANALIZA HIPOTEZ

Prva hipoteza se glasi, da bo nastal program, zmožen izračuna slike Mandelbrotove množice pri povečavi $10^{100}$ v manj kot eni sekundi. Sicer je program zmožen izrisa pri tej povečavi, le da traja veliko dlje kot eno sekundo. Zato prvo hipotezo **ovržem**.

Druga hipoteza je bila, da bo program, izveden na GPE najhitrejši. Kot samostojna optimizacija izvajanje programa na grafični enoti prinese največje rezultate, program se izvede okoli 32x hitreje,

in to zgolj s preprosto pretvorbo algoritma. To pa ne pomeni, da program, izveden izključno na CPE, ni sposoben takšnih hitrosti. Hipotezo zato torej **potrdim**.

Tretja hipoteza je predpostavila preprostost razširitve optimzicaij med KSF. Večina optimizacij z izjemo simetrije ter poenostavitve izračunov je dokaj univerzalnih in jih lahko uporabimo pri pospešku večine fraktalov. S tem hipotezo **delno potrdim**.

Četrta hipoteza pravi, da so omejene natačnosti podatkovnih tipov večji problem pri višjih povečavah. Omenjena natančnost res predstavlja problem pri višjih povečavah, saj je takrat izračun praktično nemogoč. Vendar preprostemu programu namenjenemu izrisu fraktalov, osnovni podatkovni tipi prinesejo alternativo, ki lahko v kombinaciji z GPE prinese izjemno hiter izračun. Hipotezo **potrdim**.

Cilj naloge je bil najti najučinkovitejši način izračuna KSF s primarnim fokusom na Mandelbrotovi množici. Najdenih je bilo veliko pristopov, od katerih je vsak na različne načine omogočal znižanje računskih časov.
Vsak izmed analiziranh algoritmov je bil časovno ovrednoten preko stotih meritev z uporabo standardne visokonatančnostne časovne knjižnice chrono.
Vsi rezultati meritev izrisa slike 1080p pri $It_{max}$ = 1000 so zbrani in predstavljeni v spodnji tabeli.

| Algoritem: | Povprečen računski čas | Pospešek |
|---|---|---|
| Osnovni neoptimiziran | 1538.14ms | 0% |
| Računsko poenostavljen | 1501.1ms | 2.4% |
| Z uporabo simetrije | 825.06ms | 46.3% |
| Z predčasnimi prekinitvami | 514.12ms | 66.6% |
| Z deljenjem na sekcije | 421.15ms | 72.6% |
| Z delitvijo iteracij | 263.62ms | 82.86% |
| Z izračunom koraka | 1489.33ms | 3.2% |
| Večniten - osnoven | 613.11ms | 60.1% |
| Večniten z deljenjem nalog | 566.11ms | 63.2% |
| Pospešen z GPE | 57.3ms | 96.3% |
| Z uporabo SIMD ukazov | 229.2ms | 85.1% |
| Končni program | 10.1ms | 99.35% |

Najučinkovitejši so bili računalniški pristopi, saj z deljenjem dela med več procesorskimi enotami vsaka opravi manjšo količino izračunov in temu primerno konča veliko hitreje. To najbolje izraža izračun na GPE, kjer je naloga razdeljena na preko večtisoč procesnih enot. Navkljub izjemnim rezultatom s strani grafičnih enot, končni program ne uporablja GPE zaradi njihove omejene natančnosti.

Pristopi, ki manjšajo skupno število izračunov, so prav tako pokazali ogromen potencial. Če uporabimo simetrijo, je mogoče preskočiti do 1/2 vseh izračunov.

Ker večina točk pobegne ali konvergira dokaj hitro, je večji del računskega časa porabljen za izračune cikličnih in konvergentnih točk. Predčasne prekinitve naslovijo prav to težavo in preprečijo večji del nepotrebnih izračunov ter znižajo računske čase do 66 %. Najučinkovitejši način nižanja števila računov pa predstavlja deljenje iteracij, ki v idealnem primeru zniža računske čase tudi do 82% saj se znebi kar 84% vseh potrebnih izračunov.

Opzimizacije poenostavitev formule in izračun koraka so prinesle najmanjši pospešek, saj se znebijo le manjšega dela vseh računskih operacij v primerjavi z ostalimi pristopi. Še vendo pa so ključnega pomena zaradi preprostosti implementacije ter njihovega, četudi majhnega, prispevka h končnemu rezultatu.

Z učinkovito rabo vseh naštetih pristopov je končni program sposoben izrisa v le 10 milisekundah, kar je veliko manj od ciljnih 33 ms. Do povečave okoli $10^{16}$ so le redki izrisi počasnejši od 10 ms. Ob prehodu na visokonatančnostne podatkovne tipe pa ta čas postane dokaj nestabilen in v idealnem primeru lahko pod eno sekundo izriše sliko do povečav okoli $10^{40}$ x. Na točne številke izjemno vplivata $It_{max}$ in regija, ki jo izrisujemo.

Odkrita je bila idealna kombinacija, ustvarjen je bil realnočasovni program. S tem lahko potrdim, da je bilj cilj naloge uspešno dosežen.

# ZAKLJUČEK

Med izdelavo raziskovalne naloge sem poglobil svoja znanja s področja matematike in računalništva. Na poti sem naletel na mnogo težav – od premnogo neuspešnih idej do neštetih nedelujočih računalniških programov.

Med raziskavo sem odkril izjemno uporabne nove tehnologije in koncepte, kot so SIMD ukazi, uporaba GPE za splošne računske naloge, pisanje fragmentnih senčnikov, večnitnost in še bi lahko našteval. Z ustvarjenim programom sem dokaj zadovoljen, čeprav vidim še sto in en način, kako bi ga lahko izboljšal.

Ugotovil sem, da za hitro delovanje programa za izris fraktalov ni dovolj zgolj izjemno močan računalnik, kot sem si mislil na začetku. Za dobrim programom je veliko več kot samo hiter procesor.

Marsikdaj me je presenetilo, kako lahko raznovrstni matematični triki skrajšajo računske čase skoraj bolj učinkovito kot računalniški pristop.

Med raziskovanjem mi je uspelo odkriti nov pristop za krajšanje računskih časov. Gre za metodo, predstavljeno v poglavju Deljenje iteracij. Kar me je pri njej najbolj presenetilo, je, da uspešno pridobiva približke točk s preskokom tudi več kot 90 % vseh iteracij.

Izdelana naloga mi je prinesla mnoga nova znanja, ki jih bom lahko še mnogokrat uporabil, morda celo pri naslednji raziskovalni nalogi.

# VIRI IN LITERATURA

[1 Paul Kirvan, What is a computer instruction?, URL: https://www.techtarget.com/whatis/definition/instruction, Datum pridobitve: 24.2.2024
]

[2 Benôıt B. Mandelbrot, Fractals: Form, chance, and dimension, 1977, Založba W.H. Freeman & Company
]

[3 Main cardioid and period bulbs, 8.3.2024, URL: https://en.wikipedia.org/wiki/Mandelbrot_set#Main_cardioid_and_period_bulbs, Datum pridobitve: 11.12.2023
]

[4 Robert P. Munafo, Sucessive refinement, 20.8.2023, URL:http://www.mrob.com/pub/muency/successiverefinement.html, Datum pridobitve: 20.12.2023
]

[5 TechPowerUp, NVIDIA GeForce RTX 4090, URL: https://www.techpowerup.com/gpu-specs/geforce-rtx-4090.c3889, Datum pridobitve: 8.1.2024
]

[6 TechPowerUp, NVIDIA A100 PCIe 80GB, URL: https://www.techpowerup.com/gpu-specs/a100-pcie-80-gb.c3821, Datum pridobitve: 8.1.2024
]

[Xaos, Jan Hubička and Thomas Marsh, Xaos, URL: *https://xaos-project.github.io/*
]

[8, Processing foundation, Processing overview, URL: *https://processing.org/overview,* Datum pridobitve: 15.2.2024
]

[9, Vim online, What is Vim?, URL: *https://www.vim.org/about.php,* Datum pridobitve: 15.2.2024
]

[10, W3Schools, C++ Introduction, URL: *https://www.w3schools.com/cpp/cpp_intro.asp,* Datum pridobitve: 15.2.2024
]

[11, GNU Compiler Collection, URL: https://en.wikipedia.org/wiki/GNU_Compiler_Collection, Datum pridobitve: 15.2.2024]

[12, Nvidia Corporation, CUDA, URL: https://developer.nvidia.com/cuda-zone, Datum pridobitve: 15.2.2024
]

[13, Intel Corporation, Intel® Advanced Vector Extensions 2, URL: https://edc.intel.com/content/www/us/en/design/ipla/software-development-platforms/client/platforms/alder-lake-desktop/12th-generation-intel-core-processors-datasheet-volume-1-of-2/009/intel-advanced-vector-extensions-2-intel-avx2/, Datum pridobitve: 15.2.2024
]

[14, The GNU MPFR Library, URL: https://www.mpfr.org/, Datum pridobitve: 15.2.2024
]

[15, Laurent Gomila, Simple and Fast Multimedia Library, URL: https://www.sfml-dev.org/, Datum pridobitve: 15.2.2024
]

## Viri slik

Slika 1: Sierpińskijev trikotnik, Datum dostopa: 8.11.2023, URL: https://upload.wikimedia.org/wikipedia/commons/thumb/4/45/Sierpinski_triangle.svg/1920px-Sierpinski_triangle.svg.png

Slika 2: Sierpińskijeva preproga, Datum dostopa: 8.11.2023, URL: https://blogs.ams.org/visualinsight/files/2014/07/sierpinski_carpet_small.jpg

Slika 4: Mandelbrotova množica, Datum dostopa: 8.11.2023, URL: https://i.imgur.com/gB4or6i.jpg

Slika 5: Praprot, Datum dostopa: 8.11.2023, URL: https://thereisnocavalry.wordpress.com/2012/08/09/fractals-in-nature/

Slika 6: Romanesco brokoli, Datum dostopa: 8.11.2023, URL: https://thereisnocavalry.wordpress.com/2012/08/09/fractals-in-nature/

Slika 11: Kochova krivulja pri 1, 2, 3, 4, 5 iteracijah, Datum dostopa: 12.1.2024, URL: https://i.stack.imgur.com/JpP9k.png

Slika 64: Vektorsko in skalarno procesiranje, Datum dostopa: 1.12.2023, URL: https://ru.algorithmica.org/cs/arithmetic/img/simd-vs-scalar.gif

Slika 67: Barvna statika, Datum dostopa: 2.3.2024, URL: https://solarianprogrammer.com/images/2013/02/28/mandelbrot_piece_Z2.png

Slika 68: Gladek barvni algoritem, Datum dostopa: 2.3.2024, URL:
https://solarianprogrammer.com/images/2013/02/28/mandelbrot_smooth_Z2.png

Slika 75: Processing, Datum dostopa: 15.2.2024, URL:
https://upload.wikimedia.org/wikipedia/commons/c/cb/Processing_2021_logo.svg

Slika 76: Vim, Datum dostopa: 15.2.2024, URL:
https://upload.wikimedia.org/wikipedia/commons/thumb/9/9f/Vimlogo.svg/800px-Vimlogo.svg.png

Slika 77: C++, Datum dostopa: 15.2.2024, URL:
https://upload.wikimedia.org/wikipedia/commons/thumb/1/18/ISO_C%2B%2B_Logo.svg/1200px-ISO_C%2B%2B_Logo.svg.png

Slika 78: GCC, Datum dostopa: 15.2.2024, URL:
https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/GNU_Compiler_Collection_logo.svg/868px-GNU_Compiler_Collection_logo.svg.png

Slika 79: CUDA, Datum dostopa: 15.2.2024, URL:
https://www.incredibuild.com/wp-content/uploads/2021/03/Asset-1901.png

Slika 80: SFML, Datum dostopa: 15.2.2024, URL:
https://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/SFML_Logo.svg/1200px-SFML_Logo.svg.png

# PRILOGE

## Delovanje osnovnega programa

Naslednji del programa je bilj uporabljen za izvedbo časovnih meritev vseh optimizacij. Program izmeri in izpiše čas v milisekundah, ki je bil potreben za izračun števila $It_{esc}$ za vsako točko slike ločljivosti 1080p.

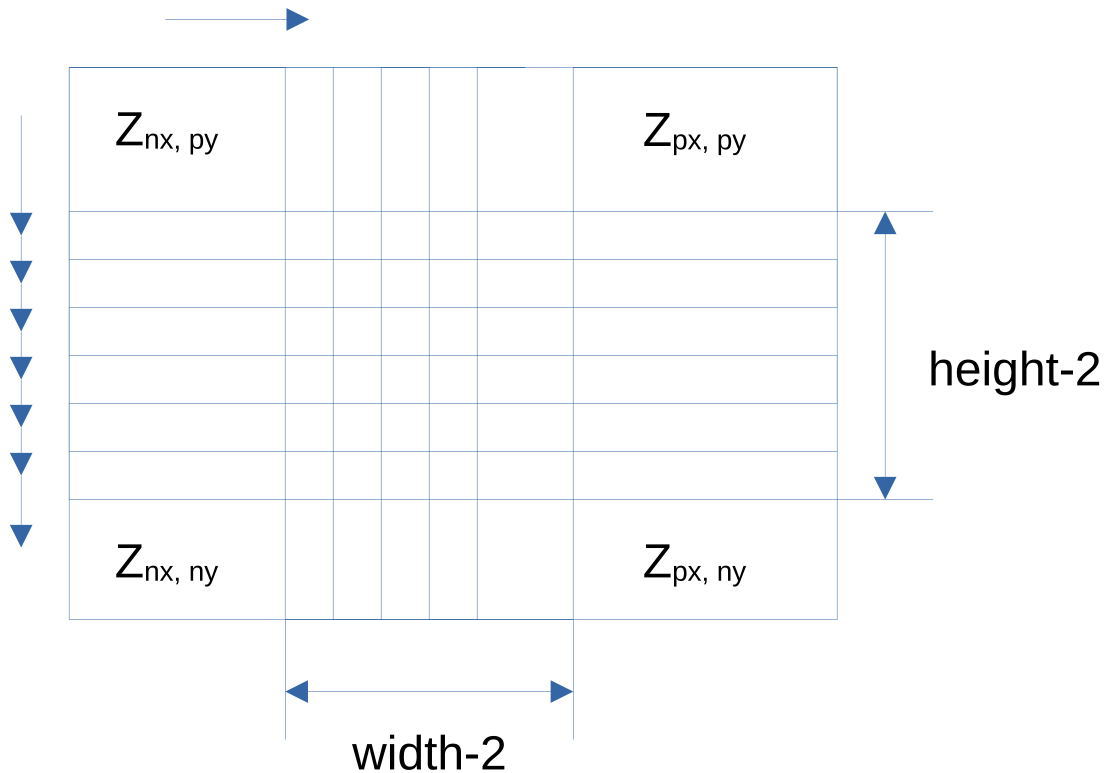
```
1  int main() {
2     yoff = 0;
3     xoff = 0;
4     width = 1920;
5     height = 1080;
6     nx = (-width / 2);
7     px = (width / 2);
8     ny = (-height / 2);
9     py = (height / 2);
10     z =300;
11     maxIteration =1000;
12
13     static int* data = new int[1920 * 1080];
14
15     for(int meritve = 0; meritve < 100; meritve++) {
16        auto start = std::chrono::high_resolution_clock::now();
17
18        for (int i = 0; i < 1920; i++) {
19           for (int j = 0; j < 1080; j++) {
20              data[j * 1920 + i] = renderers::getItterationsDefault(i, j);
21           }
22        }
23
24        auto end = std::chrono::high_resolution_clock::now();
25        int timeInMilli = std::chrono::duration_cast<std::chrono::milliseconds>(end – start).count();
26
27        std::cout << timeInMilli << std::endl;
28     }
29
30     functions::drawSet(data);
31     return 0;
32  }
```

Program začne z initializacijo javnih spremenljivk:

yoff - translacija vidnega polja po Y osi (+ navzdol, - navzgor),
xoff - translacija vidnega polja po X osi(+ desno, - levo),
width - širina zaslona
height - višina zaslona
nx, px, ny, py - prestavljajo minimalne in maksimalne koordinate točk v x in y smeri,
definiraj del kompleksne ravnine nad katerim naj se izvedejo izračuni.
nx, py – zgornji levi kot. px, ny- spodnji desni kot.
z - povečava,

maxIteration = $It_{max}$, maksimalno število iteracij, ki jih lahko program izvede na eni točki
data – dvodimenzionalno polje ki hrani podatke o številu iteracij za vsako točko.

Program nato izvede 100 meritev, vsakič iterira skozi vsako izmed točk na zaslonu, pokliče algoritem za izračun št. iteracij. Iteriranje poteka najprej po stolpcu, nato po vrstici. Za vsak izračunan stolpec se pomakne za eno točko v desno.




Vrnjen podatek shrani v polje data. Ko izračun opravi za vse točke, izpiše čas v milisekundah.
Po končanih stotih meritvah opcionalno pokliče funkcijo drawSet, ki izračunane podatke vizualizira na zaslonu (izriše fraktal).

## Algoritem 1

Osnovni algoritem kot vhod prejme točko na zaslonu, predstavljeno s koordinatama x in y, ki sta parameter metode. Ti sta nato pretvorjeni v točko kompleksne ravnine, ki se razteza od nx do px po realni ter od ny do py po imaginarni osi. Funkcija vrne 0 v primeru, da je točka element M, sicer pa število $It_{esc}$.

```
1  int renderers::getItterationsDefault(float x, float y) {
2      double xo = functions::mapValue(x, 0, 1920, nx / z, px / z);
3      double yo = functions::mapValue(y, 0, 1080, ny / z, py / z);
4
5      double r = xo;
6      double i = yo;
7      int c = 0;
8      double a, b, d;
9
10     while (sqrt(r * r + i * i )<2 && ++c < maxItteration) {
11         a = r * r;
12         b = 2 * r * i;
13         d = i * i * -1;
14
15         r = a + d + xo;
16         i = b + yo;
17     }
18
19     if (c == maxItteration) {
20         return 0;
21     }
22
23     return c;
24 }
```

## Delovanje končnega programa

Dokler je mogoče s strani računske natančnosti, program izvaja izračune na GPE. V primeru presega računske natančnosti float podatkovnega tipa program prestopi na računanje s CPE.

Tedaj program zažene 8 niti, ki ostanejo aktivne, dokler uporabnik programa ne zapre. Ustvarjene niti so zadolžene za izračun množice, medtem pa jih nadzoruje in sinhronizira glavna nit.

Ob uporabniškem vnosu, ki zahteva ponoven izračun množice (povečava, premik, menjava formule), glavna metoda razdeli vidno polje na 9x9 sekcij, ki jih nato postavi v “čakalno vrsto”. Nato so vse čakajoče niti obveščene o novem računskem delu.

Vsaka čakajoča nit prevzame svojo sekcijo za izračun. Najprej izvede deljenje na skecije z največ 4 delitvami. Ob vsaki delitvi pridobljene vrednosti hrani za izris z dinamično ločljivostjo. Vse izračune izvaja na osmih vrednostih hkrati z uporabo SIMD ukazov in deljenja iteracij.

Ko se “čakalna vrsta” izprazni, se vse niti zaustavijo in pričnejo čakati na novo računsko delo.

## Sistemske specifikacije

## Sistem:

**Operating System Details:**

- Linux distribution: Ubuntu 22.04.1
- Kernel version: 6.5.0-21-generic
- System architecture: x86_64

**CPU Information:**

- Model: Intel Core i7-1065G7
- Number of cores: 4
- CPU max MHz: 3900.0000
- CPU min MHz: 400.0000

**Memory (RAM) Information:**

- Model:  TEAMGROUP-SD4-3200
- Size: 16 Gib
- Clock: 3200Mhz (0.3ns)

Prevajalnik:
g++ (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0,

Prevajalni parametri:
-oo -mavx512f -march=native

---

## Sistem 2:

**Operating System Details:**

- Windows 10 Pro
- System architecture: x86_64

**CPU Information:**

- Model: Intel Core i7-10750H
- Number of cores: 6
- CPU max MHz: 5100.0000
- CPU min MHz: 2600.0000

**Memory (RAM) Information:**

- Model:  CRUCIAL-16GB-DDR4-2600
- Size: 32Gib
- Clock: 2600Mhz

**Graphics (GPU) Information:**

- Model:  Nvidia Geforce RTX 2070 Max-q
- Memmory: 8Gib GDDR6
- Theoretical TFLOPS: 10.92 (half), 5.46 (full).

Prevajalnik:
Visual Studio integrated compiler