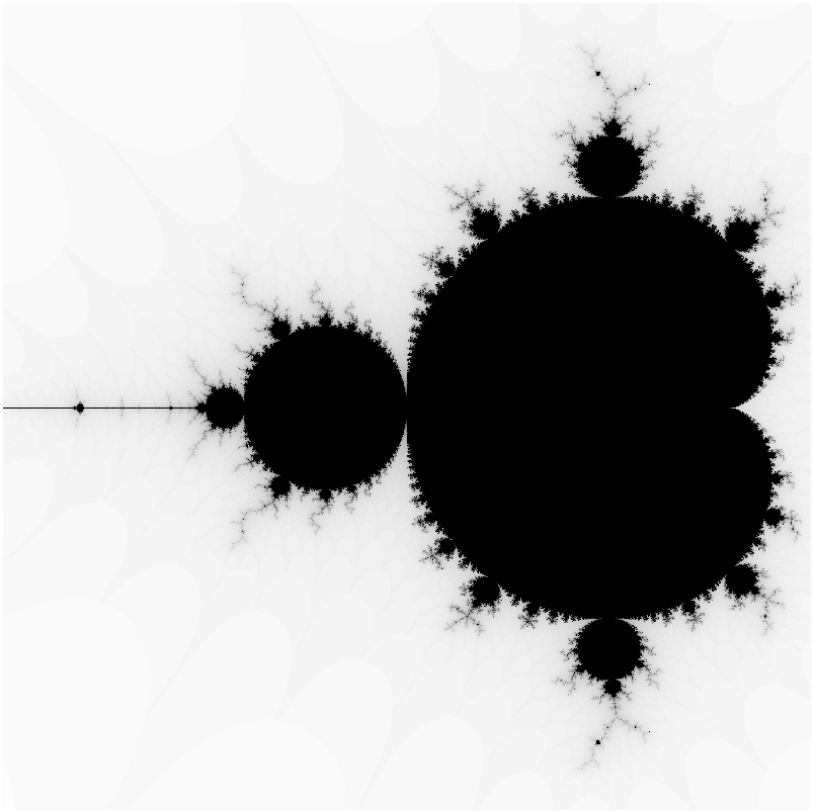
Studieretnings Case - Fraktaler

Victor Østergaard Nielsen, 2di

Maj 2024



Figur 1: Mandelbrotmængden

Tro- og loveerklæring

Jeg bekræfter herved med min underskrift, at denne studieretnings case er udarbejdet af mig. Jeg har ikke anvendt tidligere bedømt arbejde uden henvisning hertil, og opgaven er udfærdiget uden anvendelse af uretmæssig hjælp og uden brug af hjælpemidler, der ikke har været tilladt i forbindelse med projektet.

Victor Østergaard Nielsen d. _____

Underskrift: _____

Opgavens Problemstilling: Matematik værktøj til hjælp af forståelse af fraktaler

Hvordan kan fraktaler anvendes i programmering og matematik, og hvordan kan forståelsen af disse komplekse systemer øges ved hjælp af visuelle værktøjer?

Redegørelse

- Hvordan defineres fraktaler matematisk, og hvordan kan de visualiseres i en programmeringssammenhæng?
- Definer komplekse tal og deres sammenhæng med fraktaler.
- Teori omhandlende det objekt orienteret paradigme

Analyse

- Hvilke metoder anvendes typisk til visualisering af fraktaler i programmeringssprog og hvordan kan processen optimeres?
- Hvordan bygges en oop brugerflade

Vurdering og diskussion

- Afgør om hvorvidt programmet tilnærmer fraktalernes formler tilstrækkeligt
- Vurder og diskuter desuden på programmet og resultaterne

Perspektivering

- kystlinje

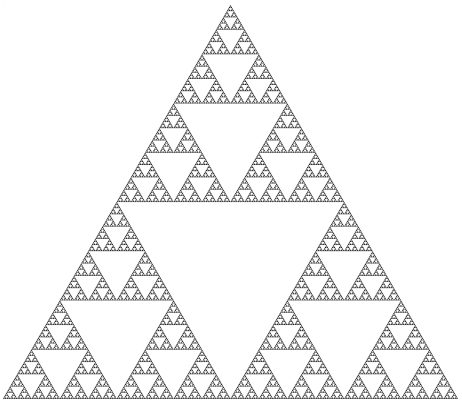
Indholdsfortegnelse

1 Hvad er fraktaler

For at kunne besvare problemformuleringen præcist, er en klar definition af en fraktal relevant. Selv om kilderne er lidt uenige, kan man groft sagt definere en fraktal således: En fraktal er en særlig type geometrisk form eller mønster, som besidder mindst én af følgende karakteristika: For det første kan den indeholde detaljer, uanset hvor meget man zoomer ind. Med andre ord forbliver strukturen kompleks på selv de mindste skalaer. For det andet er fraktaler så uregelmæssige og kaotiske i deres natur, at de ikke lader sig beskrive præcist med de klassiske geometriske begreber og metoder. Deres form er for kompliceret til de traditionelle geometriske rammer. I denne opgave vil der primært være fokus på den første slags.(Vestergaard, 2000)

1.1 Sierpinski-trekanten

Iblandt disse særlige geometriske former og mønstre af uendelig detalje er Sierpinski-trekanten en af de mest velkendte. For at skabe Sierpinski-trekanten skal denne algoritme følges punktligt: Antag en udfyldt ligesidet trekant. Nu forbindes hver af sidernes midtpunkter sammen, hvorefter denne trekant trækkes fra den forrige. Den resulterende figur, som består af 3 ligesidede trekanter, der er ligedannet med starttrekanten, er den første iteration af Sierpinski-trekanten. Hvis n er lig med iterationen, så er $n = 1$. For at skabe næste iteration $n = 2$ skal samme proces gentages for de 3 mindre trekanter. Denne proces kan ses på Figur 3 For at opnå Sierpinski-trekanten skal $n = \infty$, alt under $n = \infty$ er blot en tilnærmelse. Antag nu en Sierpinski-trekant, del figuren i 3 dele, forkast 2 af delene, skalér den sidste med en faktor på 2, den resulterende figur vil være lig med den oprindelige Sierpinski-trekant. Den opfylder derfor første kriterie i definitionen heraf og kan derfor betegnes som en fraktal. Sierpinski-trekanten er et eksempel på en fraktal, der viser en unik egenskab. Selvom dens kompleksitet stiger, nærmer dens areal sig 0, mens dens omkreds stiger mod ∞. Dette fænomen udstiller en interessant kontrast mellem dens faldene areal, og dens stigende omkreds. Således er Sierpinski-trekanten eksempel på en slags geometrisk paradoks.

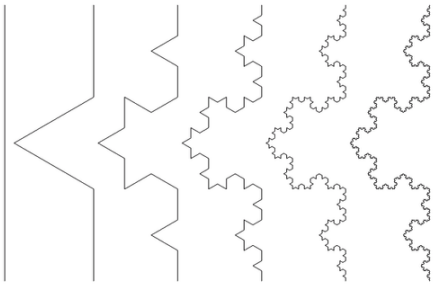


Figur 2: Sierpinski-trekanten, iteration $n > 8$, fra: (Stanislaus, 2013)

Figur 3: Sierpinski-trekanten, iteration $n = 0..4$, fra: (Saperaud, 2006)

1.2 Koch's kurve

Koch's kurve er et andet eksemplar på en selvsimilær figur. Kurven blev først betragtet af den svenske matematiker Niels Fabian Helge von Koch (H. von Koch) i begyndelsen af 1900-tallet. Kurven dannes ved følgende fremgangsmåde: Start med et linjestykke, fjern den midterste tredjedel, og indsæt to linjestykker i stedet, så der dannes en spids med en vinkel på 60 grader, som vist i tegningen til højre. Gentag derefter processen for hver af de fire resulterende linjestykker i den nye figur, ved at erstatte den midterste tredjedel med en spids. Dette resulterer i den tredje figur set til højre. Processen fortsættes gentagne gange. igen siges det at alt under $n = \infty$ er blot en tilnærmelse, hvis n er antallet af iterationer.



Figur 4: Koch's kurve, $n = 0..5$

1.3 Definition af dimension

De fleste mennesker støder på begrebet dimensioner i deres hverdag. Udtrykket bliver ofte brugt når man beskriver et objekts udstrækning. For eksempel når man spørger: "Hvad er dimensionerne på kassen?" Her refererer man til bordets længde, højde og bredde. Et andet eksempel er når man taler om "3D-film". Her opfatter man dybden i billedet udover det flade billede. Man siger, at en linje har én dimension (kun længde). En flad firkant har to dimensioner (længde og bredde). Og endelig har en massiv kasse tre dimensioner (længde, bredde og højde). For at kunne beskrive fraktaler skal forståelsen af dimensioner udvides. De selvsimilære fraktaler har den særlige egenskab, at de er sammensat af mindre dele, som hver især ligner den samlede fraktal, bare i mindre skala. Et godt eksempel er Koch's kurve, som illustreret i figur 4 i forrige afsnit. kurvens mindre grene ligner den overordnede snefnug, bare i mindre format. Denne egenskab ved selvsimilære fraktaler gør, at deres dimension ikke nødvendigvis er et helt tal som 1, 2 eller 3. I stedet kan deres dimension være et ikke-helt decimaltal, som giver et mere nøjagtigt mål for deres kompleksitet.

1.3.1 Hausdorff dimension

Lad os antage, at n er antallet af de formindskede kopier, som fraktalen består af, og lad s være faktoren, man skal gange de formindskede kopier med for at genskabe den fulde fraktal. Angående s, så refererer dette til den lineære forstørrelse. Med andre ord er s forholdet mellem længden af den samlede fraktal og længden af de enkelte formindskede kopier. s kaldes skaleringsfaktoren. Dimensionen for en selvsimilær fraktal er det tal d, som opfylder:

$$n = s^d$$

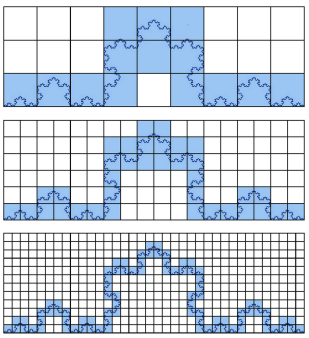
Denne definition af dimension kaldes også for *Hausdorff dimension*

1.3.2 Minkowski–Bouligand dimension

Hausdorff dimension er kun gældenende for selvsimilære figure idet de her den særlige egenskab, at de er sammensat af mindre dele, som hver især ligner den samlede figur. Minkowski-Bouligand dimensionen, også kendt som box-counting dimensionen, er en måde at kvantificere fraktale strukturer på. Den beregner dimensionen D af et sæt ved at tælle antallet $\mathrm{N}(\epsilon)$ af gitterceller (bokse) af given størrelse ϵ, der indeholder dele af sættet, hvor D defineres som:

$$D = \lim_{\epsilon \to 0} \frac{\log(\mathrm{N}(\epsilon)}{\log(\frac{1}{\epsilon})}$$

Denne definition dimension tager højde for både selvsimilaritet og ikke selvsimilære figure. Som set på figuren til højre er denne metode kun relevant for kanten af fraktalen, det er en en slags måde at beskrive at der fortsat vil være detalje uanset hvormeget man zoomer ind. (Vestergaard, 2000)



Figur 5: Illustration af box-counting på Koch's kurve, fra: (Pilgrim & Taylor, 2018)

2 Matematiske fraktaler

Før der kan arbejdes med matematiske fraktaler, er en basal viden indenfor komplekse tal såvel som det komplekse plan nødvendig.

2.1 Komplekse tal

Komplekse tal ($\mathbb{C}$) betragtes som en udvidelse af de reelle tal ($\mathbb{R}$) og er bestående af to dele:

$$z = a + b \cdot i, \quad a \in \mathbb{R},\ b \in \mathbb{R}$$

Her er a den reelle del og $b \cdot i$ er den imaginære del. Den imaginære del repræsenteres typisk ved bogstavet i, som er defineret som kvadratroden af et negativt tal. det vil sige at:

$$i^2 = -1$$

Da det ikke er vigtigt for problemformuleringen hvorfor dette giver mening vil denne del undlades. Ligesom de reelle tal kan man ligeledes udføre addition, subtraktion og multiplikation samt andre operationer på komplekse tal:

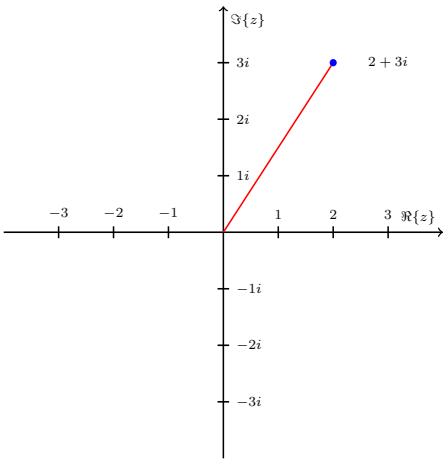
Givet: $z_1 = a + bi$ og $z_2 = c + di$

$$\begin{aligned} z_1 + z_2 &= (a + bi) + (c + di) = a + bi + c + di = (a + c) + (b + d)i \\ z_1 - z_2 &= (a + bi) - (c + di) = (a - c) + (b - d)i \\ z_1 \cdot z_2 &= (ac - bd) + (ad + bc)i \end{aligned}$$

Addition, subtraktion er lige ud af landevejen. Multiplikation kan virke lidt kryptisk, men det virker hvis man multiplicere hver del af det første komplekse tal med hver del af det andet komplekse talsiden og husker at $i^2 = -1$ (Cuemath, n.d.)

2.2 Det komplekse plan

Den traditionelle måde at beskrive tal er en linje, lad os kalde denne linje for den *reelle linje*. På denne linje findes alle de naturlige tal ($\mathbb{N}$), alle hele tal ($\mathbb{Z}$), alle Rationelle tal ($\mathbb{Q}$) og alle de Reelle tal ($\mathbb{R}$). Men denne tallinje kan ikke beskrive de komplekse tal ($\mathbb{C}$), der er derfor brug for et nyt system til at indeholde de komplekse tal. Antag et koordinatsystem hvor x-aksen er den *reelle linje*, og hvor y-aksen er den imaginære akse, beskrevet ved den imaginære del af de komplekse tal. Da der ikke er tale om hverken koordinater, eller x og y-akser kan denne talemåde erstattes med hhv. *det komplekse plan* den *reelle akse* og den *imaginære akse*. Det komplekse plan er også tiltalt som et Argand-diagram, efter den schweiziske matematiker Jean-Robert Argand, der i 1806 offentliggjorde hans idé om netop denne geometriske fortolkning af komplekse tal. til højre ses det komplekse tal $z = 2 + 3i$ på et Argand-diagram. Når de komplekse tal vises grafisk, giver det bedre mening hvordan addition, subtraktion og multiplikation virker.



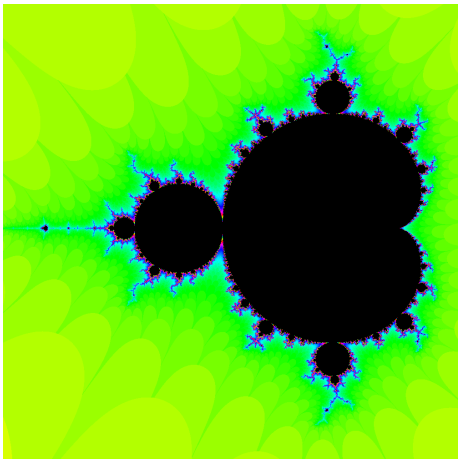
Figur 6: $z = 2 + 3i$ i Argand-diagram

2.3 Mandelbrotmængden

Med en grundforståelse indenfor komplekse tal og det komplekse plan kan mandelbrotmængden nu introduceres. Mandelbrotmængden er givet ved følgende rekursive ligning:

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

I Mandelbrotmængden er c en kompleks konstant. Den repræsenterer punktet på det komplekse plan, som du tester for medlemskab i Mandelbrotmængden. Mandelbrotmængden er en mængde af komplekse tal c, hvor iterationen $z_{n+1} = z_n^2 + c$ divergerer, når den itereres fra $z_0 = 0$. Med andre ord, hvis absolutværdien af z_n forbliver begrænset, når n nærmer sig uendelig, så betragtes c som værende i Mandelbrotmængden. Hvis absolutværdien af z_n derimod vokser uden grænse, er c ikke i Mandelbrotmængden. Denne relativt simple ligning viser sig at have overraskende stor kompleksitet og en verden af unikke fraktale mønstre. Billeder af Mandelbrot-mængden, som set til højre, er meget berømte inden for matematikken. (Vedelsby, 2023)



Figur 7: Mandelbrotmængden, $n = 100$

3 Det objekt orienteret paradigme

3.1 Indkapsling

Indkapsling er et princip inden for objektorienteret programmering (OOP), der betyder, at man skjuler objektets interne implementering og tilstand fra omverdenen. Dette opnås ved at begrænse direkte adgang til objektets felter (data) og metoder (funktioner). I stedet eksponeres et veldefineret grænseflade, der kontrollerer, hvordan data kan tilgås og ændres. På denne måde beskyttes objektets integritet og stabilitet. (Khanna, 2021) I Java er dette opnået ved brug at *access modifiers*. I Java findes der 3 typer: `public`, `private` og `protected`. Herunder ses forskellene:

`public`	Adgang til indholdet for alle klasser
`private`	Adgang til indholdet er kun mulig i den erklærede klasse
`protected`	Adgang til indholdet i den samme klasse og i alle underklasser

Da de interne variabler skal ifølge princippet holdes `private` eller `protected`, men hvis disse variabler skal eksponeres, skal det ske gennem såkaldte *getters* og *setters*. Herunder ses en simpel klasses hvor dette princip er inkorporeret:

```
1  public class Person {
2      private String CPR;
3
4      public Person(String CPR) {
5          this.CPR = CPR;
6      }
7
8      public String getCPR() {
9          return CPR;
10     }
11
12     public void setCPR(String newCPR) {
13         CPR = newCPR;
14     }
15 }
```

Mens der i dette eksempel er fri adgang til variablen `CPR` gennem de defineret *getters* og *setters*, vil der let kunne pålægges regler for, hvem der kan ændre variablen eller hvornår det kan ske. `protected` er kun relevant hvis en klasse har underklasser.

3.2 Nedarvning

Nedarvning er et andet meget vigtigt princip indenfor objektorienteret programmering (OOP). Nedarvning i objektorienteret programmering er en slags mekanisme, hvor en klasse kan arve egenskaber og metoder fra en anden klasse. Den nye klasse, der arver, kaldes en underklasse eller en *child*, mens klassen, der bliver arvet fra, kaldes en superklasse eller en *parrent*. Underklassen kan tilgå alle de `public` og `protected` variabler og metoder fra superklassen, men kan også tilføje sin egen funktionalitet. Ved at overskrive funktionalitet eller bygge nye funktioner. herunder ses et kort eksempel hvor en nedarvet klasse overskrive en metode:

```
1  class Dyr {
2      String navn;
3
4      Dyr(String navn) {
5          this.navn = navn;
6      }
7
8      void lavLyd() {
9          System.out.println("Dyr laver en lyd");
10     }
11 }
12
13 class Hund extends Dyr {
14     Hund(String navn) {
15         super(navn); // Kalder constructor af superklassen
16     }
17
18     @Override // Kun i Java, unlad i Processing
19     void lavLyd() {
20         System.out.println("hunde lyde :)");
21     }
22 }
```

Her overskrives `lavLyd()` metoden for at gide hunden en unik lyd, bemærk at underklassen arver `navn` værdien fra superklassen.

3.3 SOLID

SOLID er et velkendt akronym for fem principper for god objektorienteret design, det vil primært arbejdes med de første 2 principper, men de er alle præsenteret herunder for overblikkets skyld:

S - Single Responsibility Principle: En klasse bør have ét, velafgrænset ansvar

O - Open/Closed Principle: Softwareenheder bør være åbne for udvidelse, men lukkede for modifikation.

L - Liskov Substitution Principle: Underklasser skal kunne erstatte deres overklasser.

I - Interface Segregation Principle: Bedre at have mange specifikke interfaces end én generel.

D - Dependency Inversion Principle: Moduler på højere niveauer bør ikke afhænge af moduler på lavere niveauer. Begge bør afhænge af abstraktioner.

3.3.1 Single Responsibility Principle

Single Responsibility Principle siger, at en klasse bør have ét, velafgrænset ansvar. En klasse med mere end et ansvar bliver sværere at forstå, ændre og teste. F.eks. skal `Hund` kassen ikke have indflydelse/ansvar over `Menneske` klassen.

3.3.2 Open/Closed Principle

Open/Closed Principle siger, at softwareenheder (klasser) bør være åbne for udvidelse, men lukkede for modifikation. Det vil sige, at man bør kunne tilføje ny funktionalitet uden at ændre eksisterende kode. Dette er typisk gjort ved at designe med abstrakte basisklasser og bruge arv og komposition. Og dermed have *søsterklasser*, altså mange klasser som arver fra den samme superklasse.

4 Visualisering af mandelbrotmængden

I programmet vil der primært tages udgangspunkt i mandelbrotmængden og hvordan den oversættes til kode, programmerings sproget som vil bruges til fremstilling er Java 21.

4.1 Oversættelse af: $z_{n+1} = x_n^2 + c$ til Java

Da ligningen er rekursiv, ville et `while`-loop nok være et godt sted at starte:

```
1 int n = 0;
2 while(n < i) {
3     n++;
4 }
```

Hvor `n` er den nuværende iteration og `i` er det maksimale antal tilladte antal iterationer. Lad os nu inddrage det komplekse tal c som `ca` og `ca`, refererende til hhv. den reelle del og den imaginære del. Da c var lig det komplekse tal vi ønsker at undersøge, vil hhv. `a` og `b` holde hver del af dette komplekse tal mens der itereres over `while`-loopet:

```
1 double a = dmap((double) x, 0D, (double) img.width, (double) (-2.5 / z + ox), (double)
      (2.5 / z + ox));
2 double b = dmap((double) y, 0D, (double) img.height, (double) (-2.5 / z + oy), (double)
      (2.5 / z + oy));
3 double ca = a + cao;
4 double cb = b + cbo;
5 int n = 0;
6 while(n < i) {
7     n++;
8 }
```

`dmap()` *er bare en* `map()` *funktion til typen* `double`
Her defineres `a` og `b` som hjørnet at det vindue vi kigger i, vinduet er defineret med en zoom variabel `z` og et x og y offset (`ox` og `oy`). `2.5` er igen bare et fastlagt offset for at starte det rigtige sted. Nu kan den næste iteration udregnes. z_{n+1} er givet i koden som `aa` og `bb` stående for hver det af det komplekse tal, `cao` og `cbo` er igen bare offsets, disse er normalt sat til 0:

```
1 ...
2 while(n < i) {
3     double aa = a * a - b * b;
4     double bb = 2 * a * b;
5     a = aa + ca;
6     b = bb + cb;
7     n++;
8 }
```

`aa = a * a - b * b` og `bb = 2 * a * b` er givet ud fra formlen $(a + bi)^2 = a^2 - b^2 + 2abi$. Derfor må z_{n+1} være lig: $(a, b) = (aa + ca, bb + cb)$ som implementeret i kode i linje 5-6 herover. hvis $a + b$ eksploderer i størrelse er tallet ikke en den at mandelbrotmængden, dette kan også tilføjes til koden:

```
1 ...
2 while(n < i) {
3     double aa = a * a - b * b;
4     double bb = 2 * a * b;
5     a = aa + ca;
6     b = bb + cb;
7     if (abs((double) (a + b)) > 16) {
8         break;
9     }
10    n++;
11 }
```

Nu kan den nuværende kode indsættes i et dobbelt `for`-loop til at lave denne udregning for alle pixels i billedet:

```
1 for (int x = 0; x < img.width; x++) {
2     for (int y = 0; y < img.height; y++) {
3         // forrige kode
4     }
5 }
```

Alle billedets pixels farves efter hvor mange iterationer det tog før den eksploderer i størrelse, hvis tallet forblev lille, bliver dan farvet sort, antag `colorMode()` er sat til `HSB`. Det giver den endelige mandelbrotmængde tegne funktion:

```
PImage img = p.createImage(w / sf, h / sf, RGB);
img.loadPixels();
for (int x = 0; x < img.width; x++) {
    for (int y = 0; y < img.height; y++) {
        double a = dmap((double) x, 0D, (double) img.width, (double) (-2.5 / z + ox),
            (double) (2.5 / z + ox));
        double b = dmap((double) y, 0D, (double) img.height, (double) (-2.5 / z + oy),
            (double) (2.5 / z + oy));
        double ca = a + cao;
        double cb = b + cbo;
        int n = 0;
        while (n < i) {
            double aa = a * a - b * b;
            double bb = 2 * a * b;
            a = aa + ca;
            b = bb + cb;
            if (abs((double) (a + b)) > 16) {
                break;
            }
            n++;
        }
        int bright = (int) (map(n, 0f, 100f, 0f, 255f));
        if (n == i) {
            bright = 0;
        }
        int pix = x + y * img.width;
        img.pixels[pix] = p.color(((bright + 150) * 2) % 255, bright == 0 ? 0 : 255,
            bright == 0 ? 0 : 255);
    }
}
img.updatePixels();
return img;
```

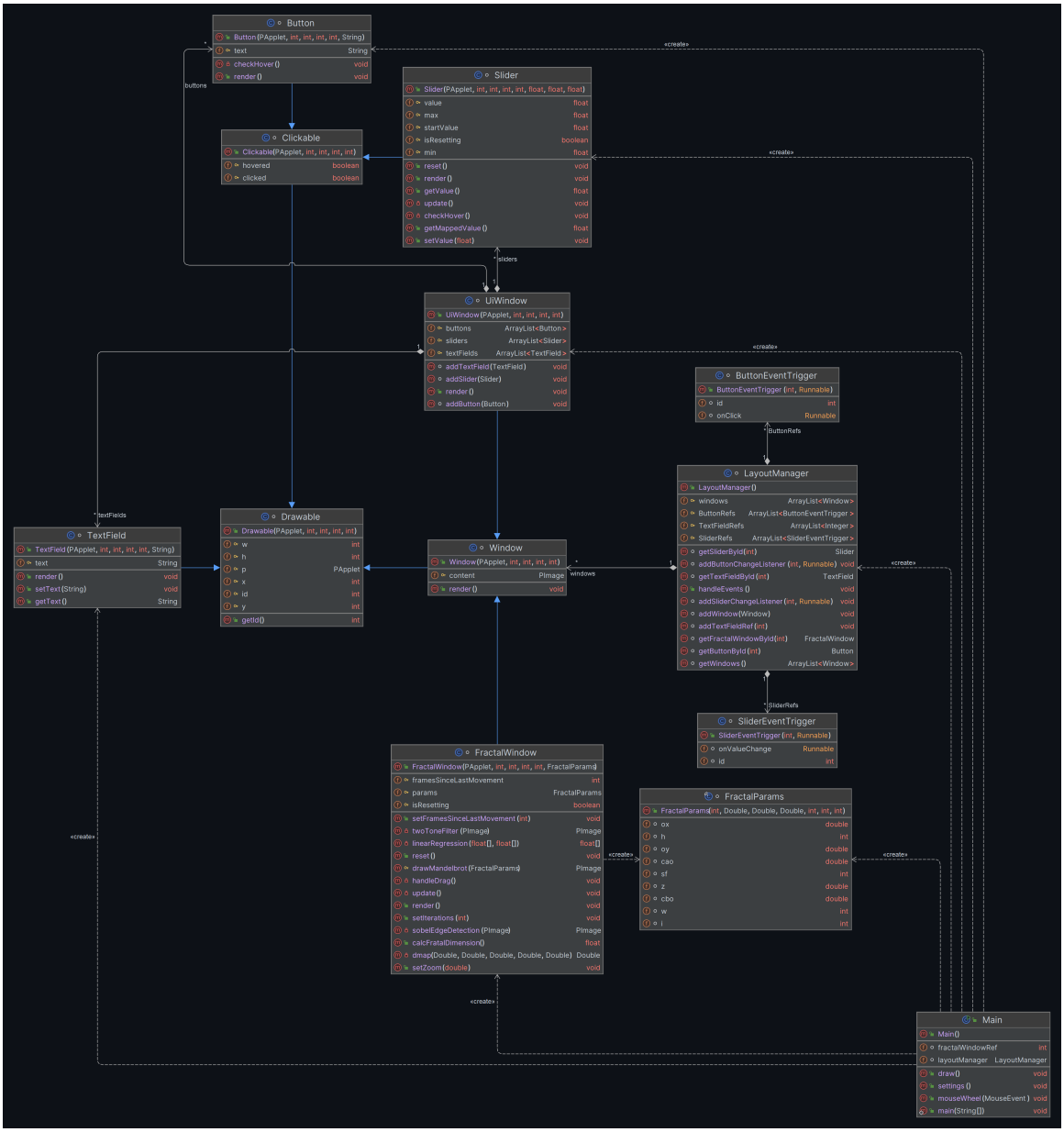
Dette er bare 1 funktion i 1 klasse, da målet er at lette forståelsen af fraktaler er en intuitiv og let betjent brugerflade nødvendig. I næste kapitel ses opbygningen af denne brugerflade.

4.2 Datatyper: `float` vs `double`

I Java skal man angive typer til sine variabler, der findes mange typer, som f.eks. `boolean`, `int`, `String`, `float` og `double`. Mens de fleste typer har hvert deres afgrænset ansvar, f.eks. `String` er kun til tekst, mens `boolean` er en binær variabel, der kun kan antage værdien enten true eller false. `int` er en heltalstype, der bruges til at repræsentere hele tal inden for et bestemt interval. `float` og `double` holder begge tal, der har et decimalkomma, og de anvendes til at repræsentere decimaltal. Deres forskel er størrelsen af de tal såvel som den præcision af tal, de kan holde. Typen `float` er 4 bytes og kan beskrive tal i intervallet $[3.4 \cdot 10^{-38}; 3.4 \cdot 10^{38}]$ med visse undtagelser. Den har en præcision på omkring 6-7 decimaler. Typen `double` er 8 bytes og kan beskrive tal i intervallet $[1.7 \cdot 10^{-308}; 1.7 \cdot 10^{308}]$ med visse undtagelser. Den har en højere præcision på omkring 15-16 decimaler, hvilket gør den mere nøjagtig end `float`-typen, men den kræver også mere hukommelse. Da programmet skal kunne zoome ind på en meget lille del at mandelbrotmængden er det oplagt at bruge typen `double` når fraktalen skal udregnes idet det vil øge funktionaliteten af programmet. som set i kodestykket herover er typen `double` brugt i alle de steder hvor nøjagtighed er yderst vigtigt.

5 OOP anvendt

Herunder ses UML klasse diagrammet af det endelige værktøj:



Figur 8: UML klasse diagram af koden

For at overholde *Single Responsibility Principle* Skal klasser som `Main` ikke styre specifikke ting men hellere stå for at køre programmet som helhed, såvel som opstart af programmet. Programmet er opbygget af vinduer administreret af en `LayoutManager` som har til ansvar at holde og vinduerne i programmet og håndtere kommunikation mellem vinduerne. Ethvert `Drawable`, som er programmet mest abstrakte basisklasse, har et `id` som bruges som reference når et element skal kommuniere eller kalde en funktion i et andet element sikker og uden at bryde OOP principperne. For at gøre dette indeholder `LayoutManager` en slags kommunikations protokol til at håndtere knappernes og skydernes begivenheder og deres respektive handlinger. Først skal f.eks en knap konstrueres:

```
1 Button resetButton = new Button(this, 50, 100, 100, 50, "Reset");
```

Da `Button` arver fra `Clickable` som arver fra `Drawable` skal parametrene `PApplet`, `x`, `y`, `w`, `h`, `text` defineres. Når objeket er lavet skal id'et gemmes i en lokal variabel til senere brug:

```
1 int resetButtonRef = resetButton.getId();
```

Nu kan knappen tilføjes til vinduet så den kan blive tegnet på skærmen:

```
uiWindow.addButton(resetButton);
layoutManager.addWindow(uiWindow);
```

Nu skal `LayoutManager` vide hvad der skal ske når knappen trykkes, denne aktion kan udtrykkes som en lambda-funktion (en ikke navngivet funktion) som bruges som argument i en `changeListener`:

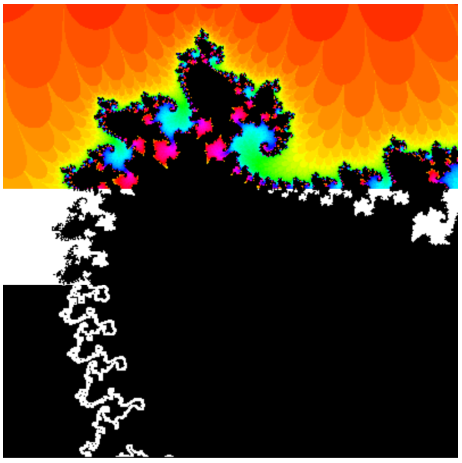
```
layoutManager.addButtonChangeListener(
        resetButtonRef,
        () -> {
            Button el = layoutManager.getButtonById(resetButtonRef);
            FractalWindow fractal = layoutManager.getFractalWindowById(fractalWindowRef);
            Slider caoSliderEl = layoutManager.getSliderById(caoSliderRef);
            Slider cboSliderEl = layoutManager.getSliderById(cboSliderRef);
            if (el instanceof Button && fractal instanceof FractalWindow) {
                ((Button) el).clicked = false;
                fractal.reset();
                caoSliderEl.reset();
                cboSliderEl.reset();
            }
        });
```

På den måde holdes funktionaliteten inde i `LayoutManager` klassen som har til formål at tillade kommunikation mellem elementer. Samme tilgang bruges til de andre interaktive elementer i programmet. Programmet overholder også *Liskov Substitution Principle* idet f.eks klassen `Clickable` kunne erstatte klassen `Drawable` idet funktionaliteten er bevaret (i dette tilfælde uden overhovedet at overskrive). Programmet overholder også indkapslings princippet idet der ikke er direkte adgang til variablerne f.eks. i `LayoutManager` klassen er de interne reference `ArrayList<...>` ikke tilgængelige, men klassen tilbyder metoder til at tilføje og finde et element, dette gør den endelige kode bedre, mere sikker og mere læsbar.

6 Minkowski–Bouligand dimension i praksis

Formlen for Minkowski–Bouligand dimension:

$$D = \lim_{\epsilon \to 0} \frac{\log(\mathrm{N}(\epsilon)}{\log(\frac{1}{\epsilon})}$$



Figur 9: Billede processering til box-counting

ikke giver mening i praksis, da vores boksstørrelse skal tilnærme sig 0 og derfor vil udregningen tage uendelig lang tid. vi bliver altså også har nået til at lave en tilnærmelse, ligesom alt andet i "fraktal-land". Dette gøres ved brug af lineær regression. i denne regression defineres et koordinat system hvor x-aksen er lig $\log(\frac{1}{\epsilon})$ og y-aksen er lig $\log(\mathrm{N}(\epsilon))$. i dette tilfælde er ϵ lig nedskaleringsfaktoren af billedet så: $\frac{\text{startBilledstørrelse}}{\text{nedskaleringsfaktoren}}$ = billedstørrelse og $\mathrm{N}(\epsilon)$ er mægnden af hvide pixels i billedet. Men vi kan ikke bruge det oprindelige billed da formlen kun virker med kanten af fraktalen, af den grund skal fraktal billedet gennem en billede processerings algoritme, på billedt til højdre ses de 2 skridt i denne process, først skal alt hvad der ikke er sort sættet til hvidt, derefter skal der køres en edge detection algoritme på billedet, nu kan de hvide pixels optælles og derefter kan en lineær regression udføres hvor hældningskoefficienten er ca. lig D. Herunder analyseres programmets implementation at de 2 lag i programmets billede processerings algoritme samt programmets implementation at lineær regression.

6.1 Sort-hvid filter

```
private PImage twoToneFilter(PImage img) {
    PImage img2 = p.createImage(img.width, img.height, RGB);
    img2.loadPixels();
    for (int x = 0; x < img.width; x++) {
        for (int y = 0; y < img.height; y++) {
            int c = img.pixels[x + y * img.width];
            if (p.red(c) == 0 && p.green(c) == 0 && p.blue(c) == 0) {
                img2.pixels[x + y * img.width] = p.color(0);
            } else {
                img2.pixels[x + y * img.width] = p.color(255);
            }
        }
    }
    img2.updatePixels();
    return img2;
}
```

Denne simple algoritme kigger på hver pixel og erstatter hver pixel som ikke er sort med hvid, da `pixels[]` ikke er et 2 dimensionalt liste i Processing skal y værdigen ganges med bredden af billedet. denne funktion er en del af `FractalWindow` klassen.

6.2 Sobel-kantdetektion

```
private PImage sobelEdgeDetection(PImage img) {
    PImage edgeImg = p.createImage(img.width, img.height, RGB);
    for (int x = 1; x < img.width - 1; x++) {
        for (int y = 1; y < img.height - 1; y++) {
            int topLeft = img.pixels[y * img.width + x - 1 - img.width];
            int top = img.pixels[y * img.width + x - img.width];
            int topRight = img.pixels[y * img.width + x + 1 - img.width];
            int left = img.pixels[y * img.width + x - 1];
            int right = img.pixels[y * img.width + x + 1];
            int bottomLeft = img.pixels[y * img.width + x - 1 + img.width];
            int bottom = img.pixels[y * img.width + x + img.width];
            int bottomRight = img.pixels[y * img.width + x + 1 + img.width];

            float gx = (-topLeft + topRight - 2f * left + 2f * right - bottomLeft +
                bottomRight) / 4.0f;
            float gy = (-topLeft - 2f * top - topRight + bottomLeft + 2f * bottom +
                bottomRight) / 4.0f;

            float gradient = PApplet.sqrt(gx * gx + gy * gy);
            edgeImg.pixels[y * edgeImg.width + x] = p.color(PApplet.min(gradient * 255,
                255));
        }
    }
    edgeImg.updatePixels();
    return edgeImg;
}
```

Sobel-kantdetektion er en algoritme, der finder kanter i et billede ved at beregne gradientet for hver pixel. Den bruger to separate filtreringskerner, en for horisontal gradient (`gx`) og en for vertikal gradient (`gy`), til at udregne ændringen i intensitet i x- og y-retningen for hver pixel. For hver pixel i billedet, undtagen den yderste kant, beregnes værdierne for de omkringliggende pixels ved hjælp af en 3x3 maske, og disse bruges til at beregne det horisontale og vertikale gradient. Derefter findes den samlede gradient ved at tage kvadratroden af summen af kvadraterne af `gx` og `gy`, og denne værdi opdateres i det resulterende billede. Højere gradientværdier indikerer stærkere kanter, hvilket gør det muligt at identificere kanter i originalbilledet. Siden billedet har været igennem et sort-hvid filter på det tidspunkt vil det resulterende gradient næsten altid være perfekt hvid.

6.3 Lineær regression

```
1  private float[] linearRegression(float[] x, float[] y) {
2      float sumX = 0;
3      float sumY = 0;
4      float sumXY = 0;
5      float sumXX = 0;
6      for (int i = 0; i < x.length; i++) {
7          sumX += x[i];
8          sumY += y[i];
9          sumXY += x[i] * y[i];
10         sumXX += x[i] * x[i];
11     }
12     float a = (sumY * sumXX - sumX * sumXY) / (x.length * sumXX - sumX * sumX);
13     float b = (x.length * sumXY - sumX * sumY) / (x.length * sumXX - sumX * sumX);
14     return new float[] { a, b };
15 }
```

Denne kode beregner koefficienterne a og b for en lineær regressionsmodel ved hjælp af mindste kvadraters metode. Den itererer over to lister x og y, der indeholder henholdsvis de uafhængige og afhængige variable. Inden for løkken akkumuleres summer, der er nødvendige for beregning af a og b, såsom summen af x, y, xy og x^2. Efter løkken beregner den a og b ved hjælp af formler, der er afledt fra mindste kvadraters metode, hvor a repræsenterer hældningen af regressionslinjen og b repræsenterer skæringspunktet med y-aksen. Metoden returnerer et liste indeholdende a og b, som er koefficienterne for den lineære regressionsmodel. Denne kode er baseret på princippet om, at i en lineær regressionsmodel minimerer den bedst egnede linje summen af kvadraterne af de lodrette afstande mellem de observerede og forudsagte værdier.

6.3.1 BEVIS for a og b i mindste kvadraters metode

Residualet for et givet punkt (x_i, y_i) i en lineær model $y = ax + b$, hvor a er hældningen og b er skæringspunktet med y-aksen, er lig:

$$r_i = y_i - ax_i - b$$

Så vil kvadratsummen af alle punkter være lig:

$$K(a,b) = \sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} (y_i - ax_i - b)^2$$

Nu differentieres K med hensyn til både a og b og sættes til 0:
Lad os starte med K_b'

$$0 = K_b'(a,b)$$

$$0 = \sum_{i=1}^{n} 2(y_i - ax_i - b)(-1) \quad \Leftrightarrow \quad \text{Differentieret med hensyn til } b$$

$$0 = -2\sum_{i=1}^{n} (y_i - ax_i - b) \quad \Leftrightarrow \quad \text{sætter 2 og -1 ud foran}$$

$$0 = \sum_{i=1}^{n} (y_i - ax_i - b) \quad \Leftrightarrow \quad \text{Da } K_b' = 0 \text{ kan } -2 \text{ forkortes}$$

$$0 = \sum_{i=1}^{n} y_i - a\sum_{i=1}^{n} x_i - \sum_{i=1}^{n} b \quad \Leftrightarrow \quad \text{Da det er tilladt at splitte summen op:}$$

$$0 = \sum_{i=1}^{n} y_i - a\sum_{i=1}^{n} x_i - nb \quad \Leftrightarrow \quad \text{Da } b \text{ er konstant}$$

$$nb = \sum_{i=1}^{n} y_i - a\sum_{i=1}^{n} x_i \quad \Leftrightarrow \quad \text{plus } n \cdot b \text{ på bægge sider}$$

$$b = \bar{y} - a\bar{x} \quad \Leftrightarrow \quad \text{divider med } n \text{ på begge sider, summene bliver til gennemsnit}$$

Lad os nu udregne K'_a

$$
\begin{aligned}
0 &= K'_a(a,b) \\
0 &= \sum_{i=1}^{n} 2(y_i - ax_i - b)(-x_i) && \Leftrightarrow && \text{Differentieret med hensyn til } a \\
0 &= -2\sum_{i=1}^{n} x_i(y_i - ax_i - b) && \Leftrightarrow && 2 \text{ bliver til } -2 \text{ da vi rykker 2 og } -x_i \text{ ud} \\
0 &= \sum_{i=1}^{n} x_i(y_i - ax_i - b) && \Leftrightarrow && \text{Da } K'_a = 0 \text{ kan } -2 \text{ forkortes} \\
0 &= \sum_{i=1}^{n} x_i(y_i - ax_i - \bar{y} + a\bar{x}) && \Leftrightarrow && \text{Da } b = \bar{y} - a\bar{x} \\
0 &= \sum_{i=1}^{n} x_i(y_i - \bar{y}) - a\sum_{i=1}^{n} x_i(x_i - \bar{x}) && \Leftrightarrow && \text{Jeg splitter parentesen i y og x dele} \\
a\sum_{i=1}^{n} x_i(x_i - \bar{x}) &= \sum_{i=1}^{n} x_i(y_i - \bar{y}) && \Leftrightarrow && \text{Plus } a\sum_{i=1}^{n} x_i(x_i - \bar{x}) \text{ på begge sider} \\
a &= \frac{\sum_{i=1}^{n} x_i(y_i - \bar{y})}{\sum_{i=1}^{n} x_i(x_i - \bar{x})} && \Leftrightarrow && \text{Divider } \sum_{i=1}^{n} x_i(x_i - \bar{x}) \text{ på begge sider}
\end{aligned}
$$

Nu er løsningen af a og b i mindste kvadraters metode bevist ■

7 Forbedringsforslag

Øget nøjagtighed i programmet vil øge programmets funktionalitet fordi selv med datatypen `double` løber man hurtig ind i begrænsninger da zoom funktionaliteten ikke er lineær. Øget nøjagtighed vil også øge præcisionen af dimension udregningerne. Dog vil udregninger ved en eventuel øget nøjagtighed også øge tiden det vil tage at udregne en opdatering, det er altså en balance mellem tid og nøjagtighed. Det vurderes at programmet har slået en god balance idet man aldrig skal vente, det hele køre i reeltid ihvertfald på en moderat kraftig laptop. Et andet område hvor programmets kunne forbedres er mængden af fraktaler man kan udforske og beregne fraktal dimension på, f.eks. kunne nogle at de mere simple fraktaler fra første afsnit inddrages og visualiseres. Men dette vurderes alligevel til at være udenfor opgaves krav. Programmets brugervenlighed vurderes til at være meget høj idet alt kan betjenes med musen direkte fra brugerfladen.

8 Kystlinjer

Lidt ligesom fraktaler har kystlinjer også den egenskab at de har uendelig detalje. Hvis du kigger med en detalje på 1 meter vil du få en anden længde end hvis du kigger ned til 1 millimeter detalje, på denne måde kan vi også her bruge box-counting til at tilnærme en fraktal dimension. f.eks. er kystlinjen af Norges lig ca. 1,31. (Ribeiro & Miguelote, 1998) Men uanset hvor langt du zoomer ind vil der altid være mere detalje, dog vil du i den virkelige verden være begrenset af atomer, bøjer og mange andre ting.

9 Opsamling

Opgavens problemformulering lød således: *Hvordan kan fraktaler anvendes i programmering og matematik, og hvordan kan forståelsen af disse komplekse systemer øges ved hjælp af visuelle værktøjer* For at løse denne problemformulering blev der lavet et visuelt udforsknings værktøj af mandelbrotmængden i reeltid, der har til formål at øge forståelsen af disse komplekse matematiske fænomener. Der blev også udforsket og implementeret en måde at udregne fraktal dimension af mandelbrotmængden ved at kigge på forskellige størrelser af billeder. Mens der naturligvis er plads til forbedring så er programmet både funktionelt og intuitivt som det er nu.

References

Cuemath. (n.d.). Multiplying complex numbers. https://www.cuemath.com/numbers/multiplying-complex-numbers/

Khanna, M. (2021, December). Encapsulation in oop: Definition and examples. https://scoutapm.com/blog/what-is-encapsulation

Pilgrim, I., & Taylor, R. (2018, November). Fractal analysis of time-series data sets: Methods and challenges. https://doi.org/10.5772/intechopen.81958

Ribeiro, M., & Miguelote, A. (1998). Fractals and the distribution of galaxies. *Brazilian Journal of Physics*, *28*. https://doi.org/10.1590/S0103-97331998000200007

Saperaud, M. (2006, November). Evolution of the sierpinski triangle in five iterations. https://da.wikipedia.org/wiki/Sierpinski-trekant#/media/Fil:Sierpinski_triangle_evolution.svg

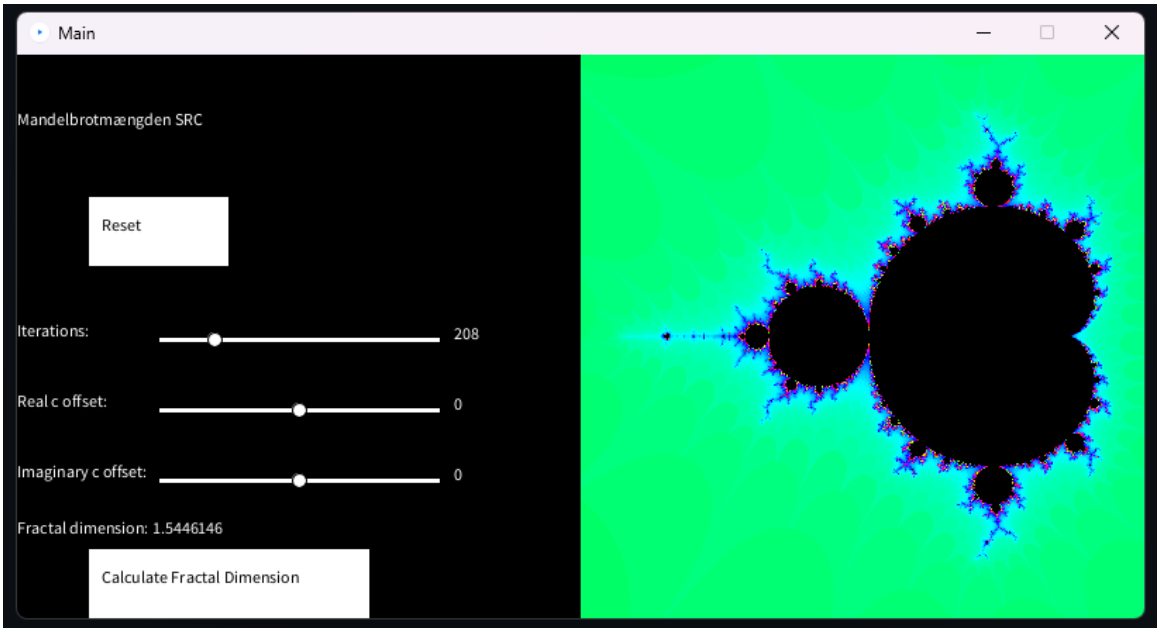
Stanislaus, B. (2013, May). Sierpinski-trekant. https://da.wikipedia.org/wiki/Sierpinski-trekant#/media/Fil:Sierpinski_triangle.svg

Vedelsby, J. (2023, October). Mandelbrotmængden var et gennembrud for computeren. https://videnskab.dk/naturvidenskab/mandelbrotmaengden-var-et-gennembrud-for-computeren/

Vestergaard, E. (2000, January). Fraktaler. https://www.matematiksider.dk/fractal.html

10 Bilag

10.1 Programmet



Figur 10: Det færdige program

10.2 Kilde koden

https://github.com/victorDigital/SRC