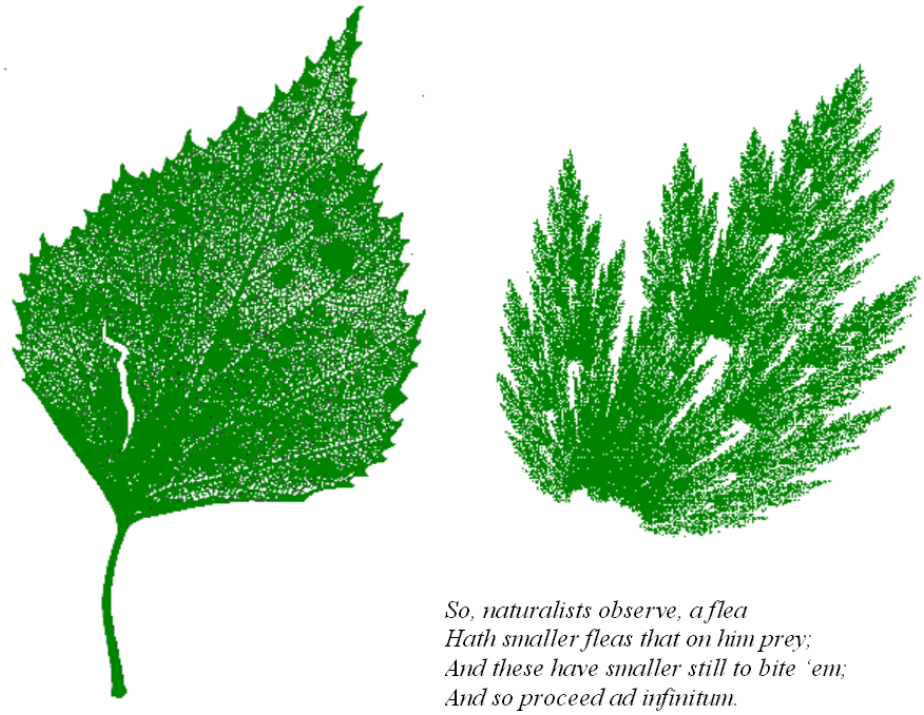
Wargentinsskolan, Östersund

Fraktaler
– Naturliga ting ritade med datorprogram



So, naturalists observe, a flea
Hath smaller fleas that on him prey;
And these have smaller still to bite 'em;
And so proceed ad infinitum.
Jonathan Swift (1667-1745)

David Larsson
Specialarbete i Matematik
2002-05-01
Handledare: Staffan Söderlind

Version i PDF-format, 27 december 2003

Innehåll

Denna version av specialarbetet i PDF-format är skriven i LaTeX, något som jag inte behärskade vid detta arbetes tillkomst. Texten är i princip identisk med den ursprungliga och har endast ändrats på ett fåtal ställen, mest i figurhänvisningar. Precis som i orginaltexten är namn på datorprogram, menyer och dialogrutor skrivna med text utan seriffer och metod- och variabelnamn i programkod skrivna med `skrivmaskinstext`

David Larsson
27 december 2003

1 Bakgrund

En gång i mina unga tonår lånade jag en bok på biblioteket. Jag hade då börjat intressera mig för olika programmeringsspråk för webben, framför allt HTML men när det blev för statiskt ville jag lära mig mer om Javascript och Perl. Jag lånade bok efter bok på biblioteket om dessa språk. Men det var när jag lånade min första (och hittills enda) bok om Perl som jag stötte på något jag aldrig sett förut.

Alldeles i början av arbetet fann jag en märklig bild, en bild med oändligt många detaljer. Intill fanns en sida med Perl-kod som jag inte förstod mig på. Koden skulle tydligen kunna generera bilden. Hur detta kunde gå till begrep jag inte då. Men bilden och dess lustiga namn, Mandelbrotfraktal", lagrade jag på någon bra plats i bakhuvudet. Detta var mitt första möte med den fraktala geometrin.

Under gymnasietiden har mina funderingar om dessa konstiga figurer växt. Flera gånger varje vecka har jag stött på dom genom att alla böcker i den läromedelsserie i matematik som vi använder har en del av en fraktal på framsidan. När så kursen i Matematik E stod på programmet och även min hjärna började räkna med komplexa tal föll många bitar på plats. Biblioteket stod till tjänst med böcker om fraktaler. Men bland innehållet i dessa var det inte Mandelbrots och Julias färgsprakande bilder som jag fastnade för, utan matematiska formler för att rita små svartvita krokiga linjer. Nog hade man någon gång under matematikundervisningen i skolan stött på något som kallades Kochkurvan. Man skulle följa en talspråklig algoritm något liknande

"Börja med en linje. Ersätt den mittersta tredjedelen med två lika långa linjer så att en liksidig triangel bildas. Upprepa detta för alla räta linjer som nu uppkommit.

Min erfarenhet av detta var att det blev väldigt mycket ritande, trianglar blev aldrig raka och tillslut var bilden så plottrig att man inte kunde se något resultat av ritandet. Detta gjorde att intresset dalade. Jag funderade någon gång på att skriva ett datorprogram och använda algoritmen ovan men jag fann att det skulle bli för svårt. Men vad jag fann i den bok om fraktaler som jag lånat var en enkel algoritm för att rita fraktala kurvor som Kochkurvan. Det var då jag slutligen bestämde mig för utformningen av detta arbete.

Detta var lite om min egen fraktalhistoria. Den allmänna dito är lite längre, egentligen mycket längre. Som så ofta har man för mycket länge sedan varit inne och nosat på området men aldrig riktigt fått tag i det. Citatet på förstasidan är från 1600-talet och kan, även om det inte är skrivet av en matematiker, tolkas som något liknande fraktaler. Nu vet vi ju att naturen endast är fraktal till en viss gräns, cellernas storlek sätter sedan stopp.

Många har de sedan varit, de matematiker som varit och försökt få grepp om detta. Kanske var Gaston Julia den som var närmast. Hade det ba-

ra datorgrafiken funnits på den tiden skulle det kanske ha varit han som införde fraktalteorin. Nu blev det inte så. I början av 80-talet hade datorerna hunnit ikapp matematikerna och Benoit Mandelbrot, polskfödd matematiker uppväxt i frankrike kunde rita upp sitt livsverk, den s.k. Mandelbrotfraktalen. Det var alltså Mandelbrot som införde namnet Fraktaler. Numera kan Mandelbrot resa runt världen över och föreläsa om sin skapelse. Och, som han själv svarade på frågan om hur det är att vara en känd professor som någon ställde när jag träffade honom: It's nice! Det skulle väl också kunna vara en beskrivning av hans bilder. They are really nice!

I sammanhanget får vi inte heller glömma den svenske matematikern Helge von Koch som 1904 skapade sin snöflingestjärna, den s.k. Kochkurvan. Detta arbete handlar egentligen mycket mer om Kochkurvor än om Mandelbrotfraktalen. Välkommen till en värld av affina transformationer och fraktalgeometri.

2 Problemområde och syfte

I sin bok *Fractals Everywhere* skriver Michael Barnsley om hur vissa fraktaler kan beskrivas med en speciell kod, en IFS-kod. Man kan lätt bli förvånad över hur många olika typer av bilder som en sådan kod kan beskriva. Dock är det svårare att ta fram denna kod. Som vi skall se i avsnitt 7 leder detta till stora ekvationssystem som måste lösas. Detta utgör ett hinder för att dessa underbara bilder skall kunna utforskas på ett enkelt sätt.

Efter att ha räknat ut några sådana ekvationssystem förstår man lätt att detta är en uppgift lämpad för en dator. En dator skulle kunna göra samma sak på bråkdelen av den tid det tar för en själv att göra det.

Syftet med det arbete som presenteras i denna rapport var att skapa ett datorprogram som kunde automatisera arbetet med att ta fram en IFS-kod. Dels att skriva ett program som bräknar en IFS-kod efter att punkters koordinater och hur dessa punkter ska transformeras har angetts. Dessutom att skriva ett program som utifrån en IFS-kod ritar upp motsvarande fraktal.

3 Avgränsningar

Hela detta arbete är inriktat på arbete med fraktaler som bygger på affina transformationer. Ett exempel på en sådan är Kochkurvan. Fraktaler som bygger på icke-linjära transformationer, t.ex. Mandelbrotfraktalen och Juliamängden berörs inte närmare i detta arbete.

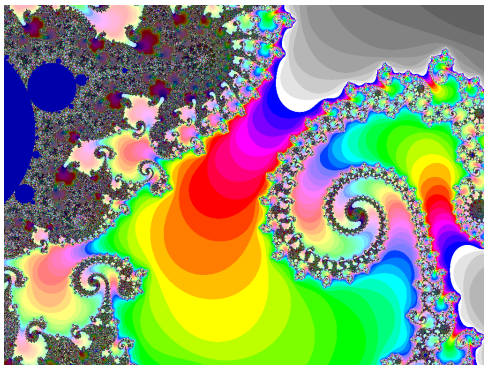
Hela arbetet är dessutom inriktat mot det datorprogram som skapas. Datorprogrammet beskrivs dels till sitt yttre, d.v.s. användargränssitt och de grundläggande funktioner som användaren stöter på vid körning av programmet. Dessutom beskrivs programmet till sitt inre, d.v.s. hur det är uppbyggt av algoritmer och beräkningar. I och med att programmet prioriteras leder detta till att avsnittet med fakta om allt det som kan sägas om fraktaler blir mera kortfattat. Det som här beskrivs är sådant som används i programmet och begrepp som dimension hos fraktaler har helt utelämnats. Detta arbete förutsätter alltså viss kunskap om grundläggande begrepp inom fraktalgeometrin.

4 Arbetsmetoder

Efter som syftet med detta arbete var att skapa ett datorprogram har den mesta tiden ägnats vid datorn. För att få grepp om ämnet som programmet skulle handla om, fraktaler med affina transformationer, var det dock naturligt att börja med att läsa litteratur om detta. Här har Bo E. Carlssons bok *Fraktaler, bilder av kaos och kosmos* varit till stor hjälp som en introduktion inför att läsa mera komplicerad engelsk litteratur.

Programmet Fractals har skrivits i Java och arbetet med att skriva källkoden har gjorts med de enkla inbyggda textredigeringsprogram som finns på datorn.

Figur 1: Exempel på icke-linjär transformation



5 Fraktala transformationer

Transformationer är helt grundläggande för fraktalgeometrin. Alla fraktaler är uppbyggda genom transformationer på något sätt. Man kan dock dela in fraktala transformationer i två huvudtyper: Icke-linjära och affina (linjära) transformationer.

5.1 Icke-linjära transformationer

Dessa transformationer innehåller termer av högre grad än 1. De bilder som kan bildas vid dessa transformationer är kanske de som de flesta som stött på ämnet förknippar med fraktaler. Bilden i figur 1 är en del av Mandelbrotfraktalen. Den är genererad genom iteration med formeln $z \rightarrow z^2 + c$ i det komplexa talplanet. Bilden ses som ett komplext talplan där varje pixel motsvarar ett komplext tal. Man itererar på varje punkt genom att sätta konstanten till det tal som pixeln i fråga motsvarar. Man undersöker hur snabbt z går mot oändligheten och ger pixeln färg med hänsyn till detta.

Mandelbrots bilder är bara ett exempel av många inom detta område, men eftersom det var Mandelbrot själv som införde hela fraktalbegreppet har hans bilder blivit särskilt vitt spridda.

5.2 Affina transformationer

Transformationer med grad 1 (eller lägre) kallas affina transformationer. I denna kategori kan man placera kurvor som t.ex. Helge von Kochs snöflingestjärna. De är ofta uppbyggda på så sätt att en sida i en figur skall ersättas med hela figuren. Då detta upprepas gång på gång bildas den fraktala kurvan.

Det är alltså de affina transformationerna som detta arbete och det program som skapats till detta handlar om. Michael Barnsley skriver i boken

Fractals Everywhere om något han kallar för fraktal tennis. Han införde något som han kallade för IFS-kod (IFS står för Iteration Function System). Denna kod kan användas för att beskriva en fraktal som bygger på affina transformationer. Det hela bygger på följande två formler:

$$X' = aX + bY + e \tag{1}$$

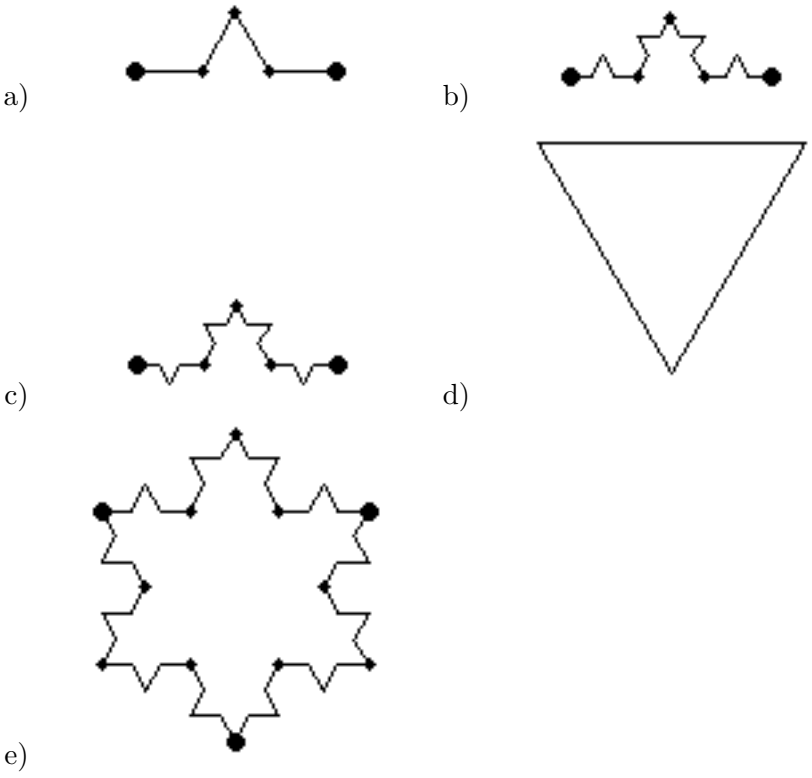
$$Y' = cX + dY + f \tag{2}$$

Den nya x-koordinaten beror alltså av både de gamla x- och y-koordinaten samt konstanterna a, b och e. Den nya y-koordinaten beror av både de gamla x- och y-koordinaten samt konstanterna c, d och f.

För att använda dessa iterationsformler måste man först finna de sex konstanternas värden. Det är dessa som styr hela fraktalens utseende. I varje fraktal kan det dessutom finnas flera olika transformationer. Varje plats där hela figuren kan placeras in ger en transformation. Varje transformation har sina egna värden på de sex konstanterna. Sammanfattar man alla konstanter i en tabell med transformationerna som rader och konstanterna $a, \ldots, f$ i kolumnerna får man det som Barnsley kallade IFS-kod. Dessutom kan man ange sannolikheten för att en transformation skall köras. Antingen kan man ha en fast sannolikhet för varje transformation eller att man för varje transformation anger sannolikheter som skall användas då det skall bestämmas vilken transformation som skall köras härnäst.

Det är då man itererar med dessa formler den fraktala tennisen börjar. Med slumpen bestäms vilken transformation som skall köras och därmed också var nästa punkt skall plottas. Ändå resulterar allt till slut i mycket detaljerade bilder.

Figur 2: Kochkurvans generator och initierare



6 Grundläggande begrepp

För att studera olika fraktaler med affina transformationer använder man sig av ett antal begrepp som är viktiga att förstå. Både generator och initierare är fria översättningar av författaren för de engelska orden *generator* och *initiator* som används i Benoit Mandelbrots bok *The Fractal Geometry of Nature*.

6.1 Generator

En generator används för att på ett enkelt sätt visa vad som händer, eller åtminstone vad man skulle kunna tänka sig händer vid uppritningen av fraktalen, d.v.s. vid iterationen. Som vi ska se i avsnitt 5 kan man nämligen tänka sig en metod och iterera enligt en annan. Som exempel på generator skall vi ta en som kan användas till den s.k. Kochkurvan. Denna kurva beskrevs för första gången 1904 av den svenske matematikern Helge von Koch. Talspråkligt kan kurvans algoritm uttryckas som följer: Man utgår från en liksidig triangel. Varje sida delas sedan i tre bitar och mittdelen

ersätts med två lika långa linjer som tillsammans bildar en liksidig triangel på den ursprungliga triangelns sida.

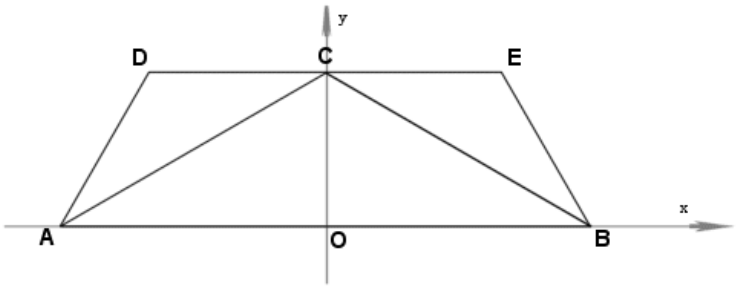
Kochkurvans generator kan skrivas enligt fig. 2a. De stora punkterna i ändarna markerar just att det är fråga om generatorns ändpunkter. De mindre punkterna markerar sidornas begränsningar. Vad visar då egentligen en generator? Jo, den talar om att en Kochkurva kan konstrueras genom att varje sida (rät linje, avgränsad med punkt) i generatorn ersätts med generatorn själv, här i förminskad form. Om man ännu tydligare vill visa vad som ska ske kan man använda sig av en mer specificerad generator som den i fig. 2b. Där är första transformationssteget utsatt men fortfarande är bara de ursprungliga sidornas start och ändpunkter utsatta. Denna andra generator kan vara bra om man ytterligare vill modifiera fraktalens form. Man skulle i detta fall kunna gör som i fig. 2c, att istället för att applicera generatorn på sidan som vanligt vända den upp och ner på två av transformationerna.

6.2 Initierare

Generatorn ovan leder till att endast en del av hela Kochkurvan ritas. För att få hela kurvan att ritas använder man sig av en initierare. Det är en figur som fungerar på ett liknande sätt som en generator. Skillnaden ligger i att varje sida i den ersätts med generatorn och inte med sig själv.

Den initierare som krävs för att rita Kochkurvan ser ut som fig. 2d. Initieraren anger alltså på vilka platser generatorer skall placeras ut. Detta framgår tydligt i fig. 2e där även generatorerna visas utplacerade i initieraren. Varje linje i initieraren som en generator placeras på kallas i fortsättningen för initierarlinje.

Figur 3: Skiss över första steget i transformationen



7 Exempel på en enkel affin transformation

För att få en inblick i arbetet med att ta fram en IFS-kod från en figur skall vi här titta på ett sådant exempel. I detta fallet beräknas alltså koden förhand, men det är detta förfarande som sedan skall överföras till ett datorprogram.

Ett koordinatsystem är inlagt i figur 3. Trianglarna ACO och BCO är båda halva liksidiga trianglar. Vi sätter sträckan AO till $\sqrt{3}$ l.e. Då antar AC och CO längderna 2 resp. 1 l.e. Med hjälp av likformighet mellan trianglarna ACB och ADC kan man sedan konstatera att sträckan DC är $\frac{2}{\sqrt{3}}$ l.e. lång (likformighet).

De utsatta punkterna får alltså följande koordinater:

$$\begin{array}{lll} A = (\sqrt{3}, 0) & B = (-\sqrt{3}, 0) & C = (0, 1) \\ D = (-\frac{2}{\sqrt{3}}, 1) & E = (\frac{2}{\sqrt{3}}, 1) & \end{array} \tag{3}$$

Transformationen utförs med hjälp av följande två ekvationer.

$$X'_{n+1} = aX_n + bY_n + e \tag{4}$$

$$Y'_{n+1} = cX_n + dY_n + f \tag{5}$$

Här finns det sex obekanta variabler. Dessa beräknas genom att man själv utför den första transformationen i figuren ovan. Det finns här två möjliga transformationer:

Punkt	**Transf. till**
A	A
C	D
B	C

och

Punkt	**Transf. till**
A	C
C	E
B	B

Sedan kan man använda de två formlerna (4) och (5) för att beskriva transformationen. Punkterna i första kolumnen i tabellen ovan har då koordinaterna (X_n, Y_n) och punkterna i andra kolumnen har koordinaterna

(X_{n+1}, Y_{n+1}) . Därmed får vi för varje transformation ett ekvationssystem med 6 ekvationer, 3 "x-ekvationer" och 3 "y-ekvationer".

Transformation 1:

$$\left.\begin{array}{rcl} -\sqrt{3} & = & a \cdot \left(-\sqrt{3}\right) + b \cdot 0 + e \\ -\frac{2}{\sqrt{3}} & = & a \cdot 0 + b \cdot 1 + e \\ 0 & = & a \cdot \sqrt{3} + b \cdot 0 + e \\ 0 & = & c \cdot \left(-\sqrt{3}\right) + d \cdot 0 + f \\ 1 & = & c \cdot 0 + d \cdot 1 + f \\ 1 & = & c \cdot \sqrt{3} + d \cdot 0 + f \end{array}\right\} \tag{6}$$

Detta system ger följande värden på de sex obekanta variablerna:

$$\left\{\begin{array}{rcl} a & = & 0.5 \\ b & = & -\frac{1}{2\sqrt{3}} \approx -0.28868 \\ c & = & \frac{1}{2\sqrt{3}} \approx 0.28868 \\ d & = & 0.5 \\ e & = & -\frac{\sqrt{3}}{2} \approx -0.86603 \\ f & = & 0.5 \end{array}\right. \tag{7}$$

Transformation 2:

$$\left.\begin{array}{rcl} 0 & = & a \cdot \left(-\sqrt{3}\right) + b \cdot 0 + e \\ \frac{2}{\sqrt{3}} & = & a \cdot 0 + b \cdot 1 + e \\ \sqrt{3} & = & a \cdot \sqrt{3} + b \cdot 0 + e \\ 1 & = & c \cdot \left(-\sqrt{3}\right) + d \cdot 0 + f \\ 1 & = & c \cdot 0 + d \cdot 1 + f \\ 0 & = & c \cdot \sqrt{3} + d \cdot 0 + f \end{array}\right\} \tag{8}$$

Detta system ger följande värden på de sex obekanta variablerna:

$$\left\{\begin{array}{rcl} a & = & 0.5 \\ b & = & \frac{1}{2\sqrt{3}} \approx 0.28868 \\ c & = & -\frac{1}{2\sqrt{3}} \approx -0.28868 \\ d & = & 0.5 \\ e & = & \frac{\sqrt{3}}{2} \approx 0.86603 \\ f & = & 0.5 \end{array}\right. \tag{9}$$

Detta kan sammanfattas i en tabell. Det vi fått fram nu är den så kallade IFS-koden för fraktalen.

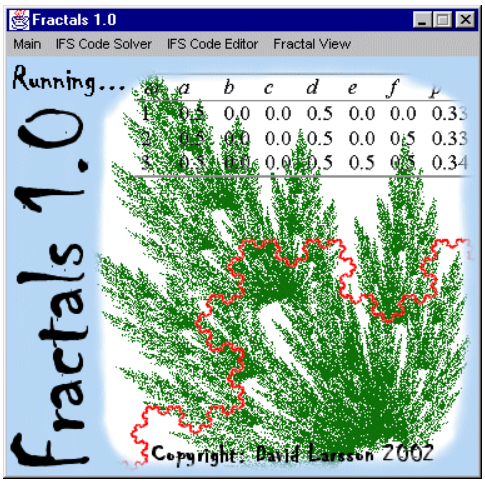
ω	a	b	c	d	e	f
1	$0,50000$	$-0,28868$	$0,28868$	$0,50000$	$-0,86603$	$0,50000$
2	$0,50000$	$0,28868$	$-0,28868$	$0,50000$	$0,86603$	$0,50000$

(10)

Figur 4: Den färdiga fraktala kurvan

I och med att IFS-koden är färdig kan nu den fraktala kurvan ritas. Med koden som räknades fram ovan får man något som liknar ett blomkålshuvud. Vid första anblicken kan det vara svårt att se den ursprungliga uppbyggnaden av trianglar, men det är faktiskt möjligt att hitta den. Origo i koordinatsystemet för bilden ligger som tidigare mitt på nedre sidan av bilden. På den del av varje "öglapå kurvan som ligger längst mot origo finns en spets från origo. Dessa spetsar motsvarar den trubbiga vinkeln C i skisserna.

Figur 5: Startfönstrets utseende



8 Fractals - En introduktion till programmets funktioner

Programmet Fractals är alltså det program som skall hjälpa till att utforska olika fraktala kurvor med affina transformationer. Detta avsnitt syftar till att ge en grundläggande kännedom om programmets olika funktioner så att läsaren sedan kan använda programmet. Programmeringstekniska aspekter och uppgifter om vilka algoritmer och metoder som används för att utföra beräkningar i programmet finns utförligt redovisade i avsnitt 9 - 15.

8.1 Startfönstret

Figur 5 visar det fönster som först visas på skärmen när programmet startas. Det innehåller förutom illustrationen en menyrad, varifrån alla programmets delar nås. Menyn Main innehåller programmets avslutningskommando.

Från menyn IFS Code Solver når man de två delar av programmet som tar fram en fraktal kurvas IFS-kod, nämligen Point Wizard (se avsnitt 8.2) och Generator Wizard (se avsnitt 8.3). Menyn Probability Editor ger tillgång till den programdel, med samma namn som menyn, som redigerar en IFS-kods sannolikheter (se avsnitt 8.4). Den sista menyn heter Fractal View och från den når man den del, med samma namn som menyn, som ritar upp en fraktal kurva med given IFS-kod (se avsnitt 8.5).

Tabell 1: Programmets kortkommandon

Programfunktion	**Kortkommando**
Point Wizard	`Ctrl+P`
Generator Wizard	`Ctrl+G`
Probability Editor	`Ctrl+E`
Fractal View	`Ctrl+F`
Avsluta programmet	`Ctrl+Q`

Figur 6: Knappar i Point Wizard

8.2 IFS Code Solver - Point Wizard

Point Wizard är en guide som leder till att en IFS-kod beräknas. Indata är olika punkters koordinater och hur dessa punkter sedan transformeras. Guiden är uppdelad i fyra steg: General Settings, Point Labels and Coordinates, Point Transformations och IFS Code Calculated.

8.2.1 Menyer och knappar

För att navigera i guiden finns det i Point Wizard ett antal knappar och menyer. För att få en överblick över funktionerna finns de sammanställda i tabell 2:

8.2.2 Step 1: General Settings

Det första steget innehåller fyra viktiga inställningar för de fortsatta beräkningarna i guiden.

Number of lines in initiator ställer in antal linjer i initieraren, eller med andra ord på hur många ställen generatorn skall placeras ut. Värden mellan 1 och 15 är tillåtna här.

Number of transformations per initiator line anger antal transformationer i generatorn. Värden mellan 1 och 15 är tillåtna här.

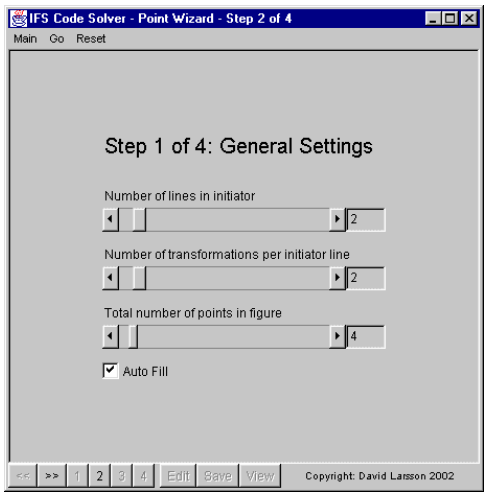
Total number of points in figure anger det totala antalet punkter som skall vara med i transformationsformlerna.

Auto Fill anger om man till alla initierarlinjer utom den första endast skall behöva ange två punkter och att datorn då beräknar de andra. Standardinställningen är att det skall vara så.

Tabell 2: Menyer och knappar i Point Wizard

Funktion	**Meny**	**Knapp**	**Kortk.**	**Beskrivning**
Redigera	Edit	Edit	Ctrl+E	Redigerar beräknad kod i IFS Code Editor. Endast aktiv i steg 4
Spara	Save	Save	Ctrl+S	Sparar beräknad kod på fil. Endast aktiv i steg 4
Rita	View	View	Ctrl+W	Ritar upp fraktalen i Fractal View. Endast aktiv i steg 4
Avsluta	Quit		Ctrl+Q	Avslutar guiden
Gå bakåt	Go back	<<	Ctrl+B	Går ett steg tillbaka. Inställningar i senare steg sparas ej
Gå framåt	Go forward	>>	Ctrl+F	Går ett steg framåt
Gå till steg 1	Go to step 1	1	Ctrl+1	Går till steg 1
Gå till steg 2	Go to step 2	2	Ctrl+2	Går till steg 2
Gå till steg 3	Go to step 3	3	Ctrl+3	Går till steg 3
Gå till steg 4	Go to step 4	4	Ctrl+4	Går till steg 4
Återställ sida	Reset Page		Ctrl+R	Återställer inställningarna på sidan till standardvärden
Återställ allt	Reset All		Ctrl+A	Återställer hela guiden. Börjar om på steg 1 med standardinställningar

Figur 7: Fönstrets utseende i steg 1



8.2.3 Step 2: Point Labels and Coordinates

Här anges etiketter och koordinater för de punkter som skall användas. Punktetiketter anges i första kolumnen under rubriken Label. Etiketterna behöver endast anges till första initierarlinjen. De läggs automatiskt till i de andra fälten. Etiketten får bestå av valfria tecken men får vara högst 10 tecken lång.

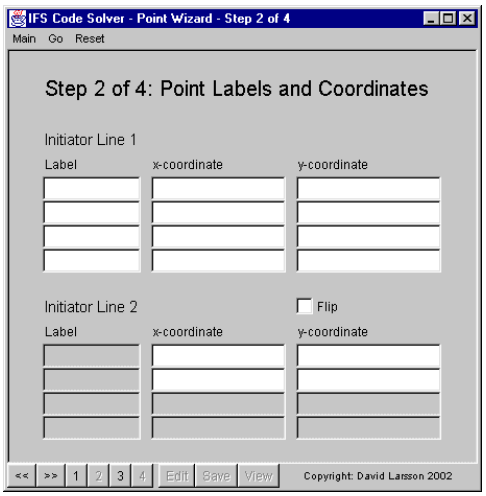
Till första linjen i initieraren måste alla koordinater anges. I resterande linjer behöver, om inte Auto Fill är avbockad, endast koordinater för de två första punkterna anges. För att gå vidare måste alla vita fält vara ifyllda korrekt. Det är alltid de tre först angivna punkterna som sedan skall transformeras. Detta gör att ordningsföljden för angivna punkter är viktig.

Från och med Initiator Line 2 finns det en ruta med texten Flip för varje linje vilket vänder generatorn tvärs längdriktningen. Funktionen mirror, d.v.s. att vända generatorn i längdriktningen åstadkoms genom att start och ändpunkt byter plats. För mer information om flip och mirror hänvisas till avsnitt 8.3.3.

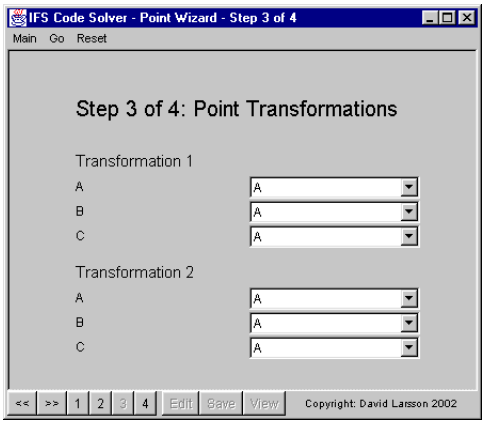
8.2.4 Step 3: Point Transformations

Här anges för varje transformation på vilket sätt de tre transformationspunkterna skall transformeras. Förfarandet är enkelt, punkten som skall transformeras står skriven till vänster på raden och i valrutan till höger väljer man till vilken punkt som den skall transformeras. Ett gott råd till hela denna programdel är att ha en skiss med punkterna och ev. också transformationernas väg utritade.

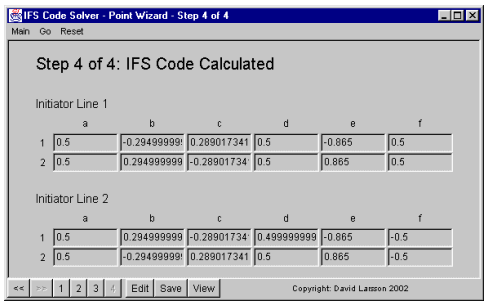
Figur 8: Fönstrets utseende i steg 2



Figur 9: Fönstrets utseende i steg 3



Figur 10: Fönstrets utseende i steg 4



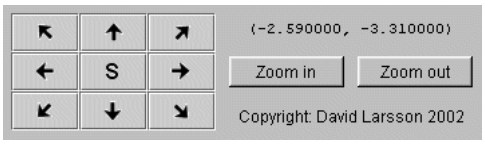
Om de angivna transformationerna inte leder till att en IFS-kod kan beräknas visas felmeddelandet Error: Invalid transformation!

8.2.5 Step 4: IFS Code Calculated

Detta steg saknar inställningar. IFS-koden har beräknats och den visas i fönstret. Nu kan koden antingen redigeras, sparas eller omsättas till en fraktal bild med hjälp av tidigare nämnda kommandon.

Guiden är slut men det går att med hjälp av navigationsknappar och menyer gå tillbaka valfritt antal steg och börja om därifrån.

Figur 11: Navigationskomponenter i Generator Wizard



8.3 IFS Code Solver - Generator Wizard

Generator Wizard är den mest avancerade programdelen i Fractals, eftersom den gör mycket arbete automatiskt. Den är precis som Point Wizard upplagd som en guide som användaren går igenom steg för steg. Navigationen genom guiden sker med hjälp av menyer och en knapprad nedtill i fönstret. Knappar och menyer är i stort sett identiska med dem som finns i Point Wizard, med undantag för att antal steg här är flera och att antalet sådana menyalternativ och knappar därför är flera.

Grundidén med Generator Wizard är att användaren anger generatorer och initierare grafiskt och att programmet sedan själv väljer ut lämpliga punkter i figurerna och beräknar IFS-koden med hjälp av dessa. Detta gör att man kan ta fram IFS-kod för relativt krångliga fraktaler med många transformationer och initierarlinjer. Den lämpar sig också bäst när punkter i generatorn ofta får koordinater som inte är heltal. Detta eftersom koordinater här kan anges på polär form.

Generellt kan man säga att Generator Wizard lämpar sig bäst till alla fraktala kurvor som tas fram genom den talspråkliga algoritmen ersätt sidan med hela figuren, medan Point Wizard lämpar sig bra till transformationer på t.ex. löv där det är svårare att tänka sig räta sidor.

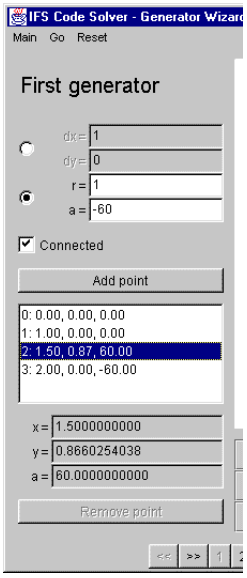
8.3.1 Ritytan och dess inställningar

Den högra delen av fönstret i Generator Wizard är det samma genom hela guiden. Överst finns en vit rityta". Här visas angivna punkter grafiskt som en generator och som initierare. Under denna yta finns det två inställningsmöjligheter för ytan, sidoförflyttning och zoom in och ut.

Längst till vänster av dessa inställningar finns en navigationspanel som är avsedd för förflyttningar av den bilden som ritas. Pilknapparna anger riktningen och mittknappens utseende anger förflyttningens längd. S (Small) innebär korta steg, B (Big) innebär stora steg. Växla mellan dessa lägen genom att trycka på knappen. Observera att det är mittpunken som flyttas i angiven riktning, bilden flyttas alltså åt motsatt håll. Med andra ord, det är fönstret som flyttas och det avbildade koordinatsystemet ligger fast.

Till höger om navigationspanelen finns det två knappar, Zoom in och Zoom out. Dessa gör precis det som det låter som, zoomar in och ut i den

Figur 12: Fönstrets utseende i steg 1



ritade skissen. Ovanför dessa knappar finns en display som visar vilka koordinater som muspekaren befinner sig i då den rörs över skissen.

8.3.2 Step 1: First Generator

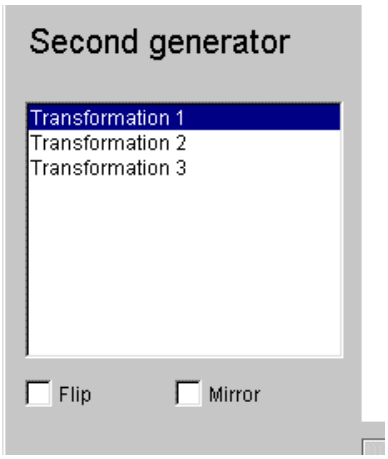
Detta steg skapar en första generator. Detta görs genom att man anger punkt för punkt. Det som ges som indata är nästa punkts relativa placering i förhållande till den senaste. Detta kan göras med antingen rektangulära eller polära koordinater. Man kan välja mellan dessa för varje ny punkt och växla mellan dem i samma generator. Valet görs genom två de runda s.k. radioknapparna t.v. om de fyra översta textfälten. Om den översta av dessa radioknappar är markerad är fälten dx och dy aktiva. Här anges avstånd i x- resp. y-led till nästa punkt.

Om den nedersta radioknappen markeras blir istället fälten r och a aktiva. I dessa anges avståndet till nästa punkt samt vinkeln mellan linjen som skall ritas dit och x-axeln. Det är alltså polära koordinater med nuvarande punkt som pol som anges.

Under textfälten finns en kryssruta med etiketten Connected . Denna anger om det skall dras någon linje mellan aktuell punkt och den nya punkten, d.v.s. om några punkter skall transformeras till denna sträcka. Standardläget är att en linje skall dras, och om rutan bockas av återgår den till standardläget då nästa punkt skall anges.

När alla inställningar för punkten är gjorda används knappen Add point

Figur 13: Fönstrets utseende i steg 2



för att lägga till punkten i punktlistan och i illustrationen till höger. Punktlistan under knappen visar alla punkter med x- och y-koordinater samt vinkel på linjen till punkten. Observera att första punkten inte kan påverkas av användaren och att den alltid är placerad i origo. Då ett alternativ i listan markeras visas data om punkten med fler decimaler i de tre textfälten x, y och a under listan. Dessa fält är inte tillgängliga för användaren och kan inte användas för redigering av tidigare angivna punkter. När en punkt lagts till med knappen Add point kan den inte redigeras.

Längst ner finns knappen Remove point, som används för att ta bort punkter i punktlistan. Det är bara den sista punkten i listan som kan tas bort och då antalet punkter går under två blir knappen inaktiv. Detta för att den första punkten ej kan påverkas.

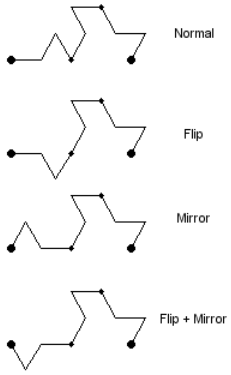
För att gå vidare från detta stag måste ett par kriterier uppfyllas. Dels måste det finnas minst en linje i den angivna generatorn. Dessutom får inte, av förklarliga skäl, startpunkten ha samma koordinater som slutpunkten. Avståndet mellan dessa skulle då bli 0 och om denna applicerades på sidorna i generatorn skulle figuren bli oändligt stor.

8.3.3 Step 2: Second Generator

Här ritar programmet själv ut första transformationen i fönstret. Det som användaren här kan göra är att ange om någon av transformationerna ska vända på generatorn åt något håll. I figur 14 är det den första transformationen (längst till vänster) som förändras med kommandona flip och mirror.

Flip spegelvänder bilden som om man skulle placera en spegel på linjen mellan start och ändpunkt. Man kan säga att man vänder generatorn tvärs längdriktningen, med en tänkt linje mellan start och ändpunkt som axel.

Figur 14: Funktion hos flip och mirror



Mirror spegelvänder bilden som om man skulle placera en spegel vid någon av ändpunkterna vinkelrätt mot en tänkt linje mellan start och ändpunkt. Generatorn vänds här i längdriktningen, start och ändpunkt byter plats.

I detta steg kan man ange om en transformation skall vara vänd med flip och/eller mirror. En transformation väljs i listan och sedan anges status för flip och mirror med hjälp av kryssrutorna nedanför listan. Om generatorn ser ut som man vill redan från början är det bara att gå vidare direkt.

8.3.4 Step 3: Initiator Shape

Det tredje steget skall ange en initierare till fraktalen. Detta stegs inställningar har precis samma utseende och funktion som i första steget. Villkoret för att gå vidare är även här att initieraren innehåller minst en linje.

8.3.5 Step 4: Initiator Settings

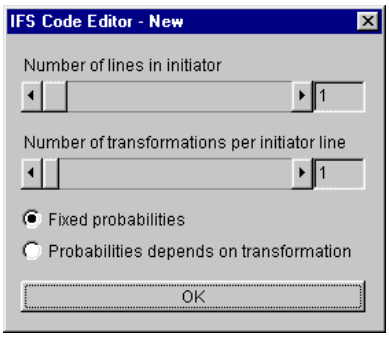
Detta steg motsvarar steg 2, med den skillnaden att det är en initierare som redigeras. Utseende och funktion hos inställningar är annars desamma.

8.3.6 Step 5: Rotation etc.

Detta steg ställer in fraktalens grundläggande position i fönstret. Dessa inställningar kan i viss mån göras vid uppritandet av fraktalen. Skillnaden då är att de inställningarna inte lagras i IFS-koden.

Tillgängliga inställningar här är att fraktalen kan vridas i en angiven vinkel. Dessutom kan den skalas om. En sträcka på 1 längdenhet i detta program motsvarar en pixel i Fractal View. Ibland kan det vara bra att skala upp figuren lite redan här. Dock är standardinställningen i Fractal View 100

Figur 15: Dialogruta till menyalternativet New



gångers förstoring. Under Displacement kan en sträcka i x- respektive y-led anges om hela figuren skall flyttas i planet.

Då ändringar i detta steg skett kan bilden t.h. uppdateras genom att trycka på knappen Preview. Tänk på att bilden kan skalas om även den. Använd muspekarens koordinater för att se till att bilden blir så stor som önskas.

8.3.7 Step 6: Code Calculated

Detta steg innehåller inga inställningar. Koden är beräknad och nu är den färdig att sparas, ritas upp eller redigeras.

Ett tryck på knappen Edit with IFS Code Editor startar ett editorfönster där koden kan redigeras. Härifrån kan man också skriva ut den som HTML-fil. Knappen Save IFS Code sparar koden på en fil som sedan kan öppnas i någon annan del av programmet. Knappen View with Fractal View startar ett fönster där fraktalen kan ritas upp.

I och med detta är guiden slut. Skulle man nu vilja börja om från valfritt steg är det bara att använda stagens knappar och menyalternativ.

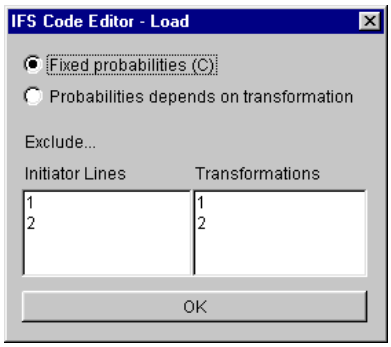
8.4 IFS Code Editor

Med hjälp av IFS Code Editor kan man redigera och skriva egen IFS-kod. När programmet startas visas ett grått fönster med en menyrad upptill. För att börja arbeta med IFS-kod använder man sig av menyalternativen i menyn File.

8.4.1 Menyn File

I menyn File finns det, för utom avslutningsalternativet, fem menyval som ger tillgång till programmets grundläggande funktioner.

Figur 16: Dialogruta till menyalternativet Open



New används för att skriva in en ny IFS-kod. När detta alternativ väljs visas en dialogruta som i fig. 18. De två översta inställningarna känns igen från Point Wizard. För information om dessa, se avsnitt 8.2.2. Enda skillnaden är att här är antalet linjer i initieraren begränsat till 10. Den nedersta inställningen bestämmer vilken modell för sannolikheter som skall användas i koden. Fixed probabilities betyder att varje transformation har en fast sannolikhet oavsett vilken transformation som tidigare utförts. Probabilities depends on transformation innebär att man för varje transformation specificerar sannolikheter för var och en av transformationerna att dessa skall köras efter den aktuella. Sannolikheten för att en transformation skall köras är alltså beroende av vilken transformation som tidigare körts.

Open öppnar IFS-kodsfiler till redigering. Det första man möter efter att ha valt alternativet är den dialogruta där en fil kan väljas. När detta är gjort öppnas en ny dialogruta där ytterligare inställningar kan göras. Den första inställningen är likvärdig med den sista i dialogrutan New. Markören (C) vid ett av alternativet står för current och visar att alternativet används i den öppnade filen. Om alternativet utan (C) väljs kommer inte sparade sannolikhetsvärden att visas i editorn. Den nedre delen av denna dialogruta, Exclude, gör det möjligt att ta bort vissa initierarlinjer och transformationer. Det som skall tas bort markeras. Minst ett av alternativen i vardera listan måste vara omarkerat för att man ska kunna fortsätta.

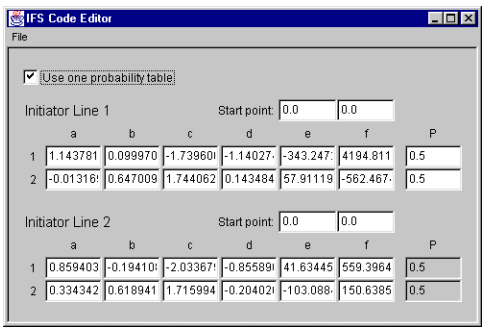
Save sparar en IFS-kodsfil. Observera att detta alternativ alltid öppnar en ruta där ett filnamn kan anges. Det är inte så att en fil som öppnas och redigeras sedan bara kan sparas med detta alternativ. Att öppna en fil innebär egentligen bara att filens data läses in och görs tillgängligt för redigering. Vill man redigera filen i vanlig mening kan man spara filen med samma namn som den öppnade.

Save HTML sparar en HTML-fil med aktuell IFS-kod i tabellform. Detta är ett praktiskt sätt att få ut en IFS-kod i klartext från programmet.

Tabell 3: Kortkommandon i IFS Code Editor

Funktion	**Kortkommando**
New	`Ctrl+N`
Open	`Ctrl+O`
Save	`Ctrl+S`
Save HTML	`Ctrl+H`
View	`Ctrl+W`

Figur 17: Editorfönstret i IFS Code Editor



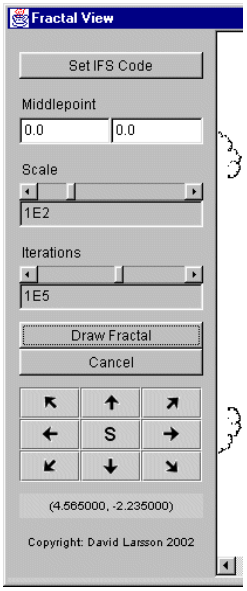
View öppnar Fractal View med vilket fraktalen som koden beskriver kan ritas. För mer information om detta, se avsnitt 8.5.

8.4.2 Editorfönstret

När kommandot New eller Open har använts och alla inställningar gjorts på ett korrekt sätt visas i grundfönstret ett rutnät som i fig. 20.

Allra överst finns det en kryssruta med texten Use one probability table. Denna anger om alla generatorer skall ha samma sannolikhetstabell. Om denna ruta är förbockad kommer sannolikhetsfälten till första linjen i initieraren vara de enda sannolikhetsfält som är tillgängliga att redigera. Från de första förs data automatiskt över till de andra sannolikhetsfälten. När kryssrutan bockas för visas en varning för att data kan komma att raderas. Då ersätts nämligen alla sannolikhetsfält med de sannolikheter som finns i första initierarlinjen. Att avmarkera rutan gör dock endast att alla fält åter blir tillgängliga. Fältens värde ändras inte. Under kryssrutan följer sedan en IFS-kodstabell för varje initierarlinje. Om en fil har lästs in med kommandot Open visas inlästa data i sina respektive textfält enligt vad som angetts vid inläsningen. Till höger om initierarlinjens rubrik finns det två fält för startpunktens x- och y-koordinater.

Figur 18: Del av fönster till Fractal View



8.5 Fractal View

Fractal View har till uppgift att utifrån en given IFS-kod rita en fraktal kurva. Till höger syns en del av programmets fönster; det som fattas är ritytan som fraktalerna ritas på.

Den vänstra delen av fönstret består alltså av en inställningspanel. Överst finns knappen Set IFS Code. Den öppnar en dialogruta för att öppna en fil och används när man vill ange eller ändra den IFS-kod som skall användas av programmet. Från vissa av delprogrammen i Fractals kan man gå direkt till Fractal View utan att spara koden på fil. Då går denna knapp inte att använda.

Under knappen finns två fält för att ange ritytans mittpunktskoordinater. Ritytans längd och bredd är 1200 pixlar och för att kunna se hela omges den av rullningslister. Man skulle kunna se ritytan som ett papper på vilket koordinatsystem med origo i olika positioner och olika skalor på axlarna skulle kunna ritas. Inställningen Middlepoint anger vilka koordinater ritytans mittpunkt får i det aktuella koordinatsystemet, den punkt som man vid zoomning inåt närmar sig. Standardvärdet är $(0,0;0,0)$, vilket innebär att mittpunkten är origo i koordinatsystemet.

Nästa inställning, Scale, har också med ovan nämnda koordinatsystem att göra. Den anger förstoringsgraden och anges i potensform. Standardvärdet är 1E2, d.v.s. 100 gångers förstoring, vilket kan vara lagom om man har en IFS-kod som ger en figur med längd och bredd i storleksordningen 1 l.e.

Iterations anger antal iterationer som skall utföras vid uppritningen. Om figuren innehåller en initierare med fler än en sida, vilket alltså medför flera uppsättningar IFS-koder, så utförs angivet antal iterationer för varje sida i initieraren. Detta innebär att antalet iterationer kan bli mångdubbelt fler än inställningen visar. Standardvärdet här är 1E5, d.v.s. 50000. Om detta är ett lämpligt värde för en kurva beror på skalan. Stor skala kräver fler iterationer för att bilden inte skall bli kornig. Beakta dock att ett stort antal iterationer kan ta lång tid att utföra och programmet kan i värsta fall helt sluta att svara. Bäst är att börja med ett litet värde här och sedan gå uppåt. På detta sätt märker man hur mycket den aktuella datorn mäktar med.

För att sedan rita fraktalen, tryck på knappen Draw Fractal. Längst ner i fönstret finns ett liggande stapeldiagram som visar hur beräkningsarbetet fortskrider. Sedan visas resultatet på ritytan. Knappen Cancel används om iterationerna av någon anledning skall avbrytas. Knappen kan t.ex. användas om antalet iterationer blivit för stort.

Under dessa två knappar finns en navigationspanel som är avsedd för små förflyttningar av den ritade bilden. Pilknapparna anger riktningen och mittknappens utseende anger förflyttningens längd. S (Small) innebär korta steg, B (Big) innebär stora steg. Växla mellan dessa lägen genom att trycka på knappen. Det är mittpunken som flyttas i angiven riktning, bilden flyttas alltså åt motsatt håll. Med andra ord, det är fönstret som flyttas och det avbildade koordinatsystemet ligger fast. Observera att tryck på dessa knappar innebär att hela bilden ritas (beräknas) om. Knapparna är mest användbara i kombination med litet antal iterationer. Eftersom förflyttningen är i fönstret är lika stor oavsett skala är knapparna ett bra komplement till att göra justeringarna med mittpunktsinställningen. Utför justeringarna med ett lågt antal iterationer och öka sedan antalet när placeringen är den önskade.

9 Något om programmeringen i stort

9.1 Programspråk

Programmet Fractals är skrivet i programmeringsspråket Java. Det går inte att köra som en applet. Detta p.g.a. begränsningar t.ex. för att skriva till fil, vilket alltså av säkerhetsskäl ej är tillåtet för en applet. För att kunna köra programmet krävs det alltså, som tidigare nämnts att en JavaVM finns installerad på datorn, något som annars finns inbyggt i webbläsaren när program körs som applets.

Java är ett objektorienterat programmeringsspråk som har många stora likheter med C++. Operatorer och syntax är i mångt och mycket identiskt med motsvarande i C++. Vad som skiljer språken åt är javas stora inbyggda klassbibliotek med färdiga klasser. Sådana klassbibliotek finns naturligtvis också för C++, men de är inte inbyggda i språket. Här är Windowsprogrammering ett bra exempel.

Objektorienteringen innebär att varje fysisktöbjekt ses som ett objekt i koden också. Fönster, menyer knappar dialogrutor o.s.v. är objekt. Ett objekt är alltså en modell av ett verkligt eller tänkt föremål. För att beskriva dessa objekt används klasser. En klass är en beskrivning av ett eller flera objekt med samma egenskaper.

Arvsmekanismen är en annan viktig del i objektorienteringen. Den medför att egenskaper kan ärvas genom klasser. En minibuss är en bil som är ett fordon. Minibussen har egenskaper som både ett fordon och en bil har. Utöver detta har den också egna egenskaper som varken ett fordon eller en bil har. Överfört till programmeringen, en dialogruta är ett fönster och har många egenskaper lika med ett fönster. Dock har den vissa egenskaper som skiljer den från ett fönster. Man säger att dialogrutan är sen subklass till fönstret.

På detta sätt, med subklasser och arv, byggs javas klassbibliotek upp. Man kan också skapa egna klasser som ärver egenskaper av varandra. Programmet Fractals är uppbyggt av ett antal klasser. Varje klass sparas som en fil med ändelsen .class.

9.2 Kodens utseende

Ovan nämndes det att java-kod har stora likheter med C++-kod. Dock kan samma kod se ut på väldigt många sätt beroende på vem som skrivit den. Variabel- och klassnamn kan i stort sätt väljas fritt och mellanslag kan läggas till var man vill.

Under arbetet med programmet har dock några enkla reglerför kodens utseende använts, och de sammanfattas nedan.

Kommentarer: En informativ kommentar används först i koden. Markering: `/* */`. Vidare används `//` för att kommentera, rubrikkommen-

tarer skrivs med VERSALER.

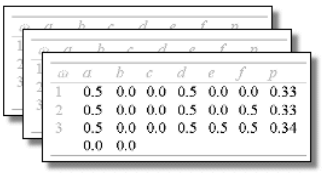
Variabelnamn: Variabelnamn börjar med liten bokstav och varje delord (förutom första) inleds med stor bokstav. Undantag: Förkortningar i början, t.ex. `IFSCode`

Klassnamn: Som variabelnamn fast med stor bokstav i början

Metodnamn: Som Variabelnamn

Block: Vid t.ex. `for`- och `if`-satser skrivs huvudet (`for`, `if` etc.) på en rad. På raden under skrivs blockmarkören ensam indragen som huvudet. Sedan följer blocket indraget två kolumner längre än huvudet. Sist kommer avslutningsmarkören på egen rad indragen som huvudet.

Figur 19: Schematisk bild över ifs-filens struktur. Endast tal med svart färg lagras



10 Filformatet IFS

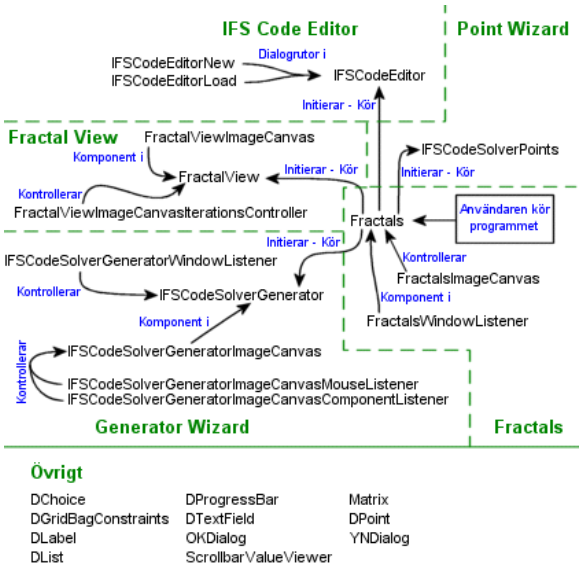
När man i Point Wizard, Generator Wizard och IFS Code Editor sparar en fil innehållande ren IFS-kod sparas den i en fil med filändelsen `.ifs`. Filen är en binärfil, d.v.s. ingen textfil som går att redigera enkelt. Det som sparas binärt är ett flyttalsfält i tre dimensioner (`double[][][]`), ungefär som en tredimensionell tabell. Det innehåller IFS-kod för varje linje i initieraren. Varje sådan IFS-kod har ett visst antal transformationer, vilka i sin tur består av 6 tal för variablerna $a, \ldots, f$, samt ett antal sannolikheter. I filen lagras också lämpliga startpunkter för iterationer med IFS-koden.

Figur 19 visar en schematisk bild över filstrukturen, där man tänker sig filen, den tredimensionella tabellen, som en samling pappersark med en tvådimensionell tabell på var och ett av arken. Den första indexeringen i det lagrade fältet skulle med denna modell ange vilket av arken som avses. Den andra indexeringen skulle vilken rad i den tvådimensionella tabellen på det aktuella arket som avsågs. Med andra ord anges aktuell transformation.

Den tredje indexeringen anger då vilken specifik variabel inom aktuell transformation som avses. Raderna kan här variera i längd beroende på vilken modell för sannolikheter som har valts i den aktuella filen. Antingen kan varje rad, som i figur 19, bestå av 7 element, de sex transformationsvariablerna samt en sannolikhet för transformationen. Annars har varje rad, utöver de sex transformationsvariablerna, lika många element ytterligare som antalet transformationer i generatorn.

Den sista raden i varje tvådimensionell tabell består av två element, x- resp. y-koordinat för lämplig startpunkt vid iteration. När filen läses är det alltså viktigt att programmet inte ser den sista raden som ytterligare en transformation. Detta skulle leda till att programmet försöker läsa fler element än vad som finns på den raden. Detta är förklaringen till att loopar på andra indexet i IFS-kodsfält endast går till näst sista raden.

Figur 20: Programmets klasser och dess funktion



11 Programmets uppbyggnad klass för klass

I figur 20 åskådliggörs varje klass funktion i programstrukturen, alltså vilka relationer som råder klasserna emellan. För att det skall bli lättare att överblicka skissen är den indelad efter de olika programdelarna.

Nederst i skissen finns en lista med övriga klasser, mestadels komponenter. Detta är komponenter som inte är specifika för någon programdel och används eller skulle kunna användas i alla delprogram. De är mestadels förbättringar och förenklingar av javas standardklasser där dessa har ärvts med. Detta gäller `DChoice`, `DGridBagConstraints`, `DLabel`, `DList`. `DPoint` är en enklare variant av javas klass Point med flyttalskoordinater. Här har inget arv använts.

`DProgressBar` är en förloppsindikator som skrivits för programmet. `OKDialog` och `YNDialog` är dialogrutor som kan visas med olika textmeddelanden. `ScrollbarValueViewer` är en komponent som består av en panel och två komponenter: En rullningslist och en etikett där rullningslistens värde visas.

Klassen `Matrix` är en beskrivning av en matris. Som vi skall se i avsnitt 12.2 används matriser för att beräkna IFS-kod. Denna klass innehåller en metod för att beräkna determinanten till den matris den beskriver.

12 IFS Code Solver - Point Wizard

Point Wizard är alltså det program som beräknar en IFS-kod utifrån indata i form av punkter i en figur och hur dessa skall transformeras mellan varandra. Här följer först en genomgång av programmet och sedan några mera ingående beskrivningar av komplicerade delar som finns i programmet.

12.1 Programmets gång

Point Wizard består egentligen av två speciellt viktiga metoder som båda dirigerar ut anrop av övriga metoder i programmet. Dessa är konstruktorn och händelsemetoden `actionPerformed` tillhändelselyssnaren `ActionListener`

12.1.1 Konstruktorn

När programmet skall startas från huvudfönstret i Fractals anropas konstruktorn i Point Wizard. Konstruktorn är den metod som skall initiera klassens variabler och den har alltid formen `public klassnamn(`*`ev. argument`*`)`. I detta fall skickas inga argument. Det första anropet i konstruktorn, `super()` anger att konstruktorn i superklassen `Frame` skall köras, d.v.s. att fönstrets standardvariabler skall initieras. En fönsterlyssnare för att kunna fråga om användaren verkligen vill stänga fönstret om denna trycker på X:et uppe till höger i fönstret läggs därefter till.

Efter detta börjar initieringen av fönstrets olika komponenter, en del av programmeringen som inte har så mycket annat än just programmeringen att göra. Komponenterna initieras, lyssnare kopplas till dessa för att upptäcka och handla vid t.ex. knapptryck. Till sist läggs all i olika paneler med hjälp av olika layouter, d.v.s. man bestämmer var de olika komponenterna skall ligga i förhållande till varandra

Den allmänna komponentinitieringsfasen består av två metodanrop, metoderna `initMenuBar()` och `initBottomFrame()` anropas. Dessa initierar menyraden överst resp. knappraden nederst. Själva metoderna följer genast efter konstruktorn i koden och lämnas härefter ingen uppmärksamhet. Efter detta sätts variabeln `step` till 1 vilket anger att aktuellt steg är steg nummer 1. Efter detta anropas också metoden `handleEnabledDisabled-Components()` som ser till att rätt knappar och menyer för steget är tillgängliga för användaren. Metoddefinitionen finns längre ned i koden och anger status för komponenterna enligt en konstant fältvariabel `enableDisable` som finns deklarerad bland de konstanta variablerna i början av koden.

Sedan följer ett metodanrop av `initStep1()` som initierar första stegets inställningspanel. Metoddefinitionen följer omedelbart efter definitionen till `initBottomFrame()`. Den som vill veta mera om exakt vad denna metod gör hänvisas till kommentarerna i koden.

12.1.2 Händelselyssnaren `ActionListener`

Lyssnarmetoderna i grafiska, objektorienterade program som Fractals är mycket viktiga. De är de som känner av vad användaren gör i fönstret och lyssnarmetoderna skall då utföra det som skall göras. Lyssnaren `ActionListener` är, som alla andra lyssnare, ett gränssnitt som implementeras i klassdeklarationen. Detta gränssnitt gör inget annat än att det kräver att en metod med utseendet `public void actionPerformed(ActionEvent e)` deklareras. Vad som däremot ej är bestämt är vad metoden själv skall göra, det specificerar man själv för varje program.

För att det skall hända något då användaren t.ex. trycker på en knapp måste man ange att komponenten skall skicka information om händelser till lyssnaren. Detta görs genom att metoden `addActionListener` anropas. Den tar en klass som har implementerat gränssnittet `ActionListener` som argument och metoden finns för alla grafiska komponenter. Eftersom det i detta fall är den egna klassen som har implementerat gränssnittet skriver man `addActionListener(this)`.

Hela metoden består av selektionssatser (`if`- och `else`-satser). Man letar med hjälp av dessa rätt på den komponent som skickat ett meddelande om en händelse och utför då det som då ska hända. Det börjar med knappen för att gå ett steg framåt. Om denna knapp använts måste man också veta vilket steg som är det nuvarande. När detta är klart anropas en speciell metod för att utföra själva stegbytet. Namnet på metoden har formen `goToStep1()` där talet 1 naturligtvis byts ut mot det steg som metoden går till. Dessa metoder finns definierade strax ovanför actionPeformed i koden.

Det finns för det mesta två olika delar i dessa metoder, en för de fall då man kommer från det närmast föregående steget. Då skall inställningar från steget läsas in och bearbetas. Om nästa stags panel redan har initieras måste den tas bort innan den nya läggs till. Den andra delen är för den händelse att man vill gå från något steg längre fram tillbaka till steget. Då måste man på nytt ladda stegets inställningspanel. I metoden anropas också alltid `handleEnabledDisabledComponents()` som beskrevs i avsnitt 12.1.1.

I dessa `goTo`-metoder anropas också metoder som har namnformen `initStep1()`. De gör precis vad det låter som, initierar stegens inställningspaneler och placerar ut komponenterna. I avsnitt 12.1.1 nämndes det på tal om just metoden att info om dessa kan fås genom kommentarer i källkoden. De innehåller mest programmeringstekniska detaljer med layouter och paneler, som inte har så mycket med fraktaler att göra. Metoddefinitionerna till dessa metoder följer i nummerordning omedelbart efter definitionen till `initBottomFrame()`.

Efter framåtknappen följer bakåtknappen och varje stegs knapp. Här anropas direkt ovan beskrivna metoder och därför beskrivs inte dessa mera ingående. Därefter följer satser som skall exekveras om användaren tryckt på någon av Edit-, Save- eller View-knapparna. Här är det save-alternativet

som kräver en närmare beskrivning. De andra startar endast IFS Code Editor resp. Fractal View genom att anropa respektive konstruktor med IFS-koden som argument. Save däremot skall spara IFS.koden på fil. Den börjar med att en fildialog visas där ett filnamn kan anges. Då denna dialogruta stängs på något sätt, vanligen genom att användaren trycker OK, fortsätter exekveringen. Programmet kontrollerar om någon fil verkligen angetts. Har detta inte skett har användaren tryckt avbryt och hela sparprocessen avbryts. Här läggs även filändelsen .ifs till om användaren inte gjort det. Programmet kontrollerar också så att filen inte redan finns och om den gör det visas en dialogruta. Sedan skrivs hela IFS-kodsobjektet ut på den angivna filen.

Efter detta återstår det tre komponenter vars händelser behandlas av `ActionListener`. Det gäller först menyalternativet Quit som stänger fönstret. Först frågas användaren om fönstret skall stängas och om svaret då blir ja så stängs fönstret. Alternativet Reset Page återställer aktuellt stegs inställningspanel. Där kontrolleras vilket det aktuella steget är och sedan återställs alla komponenter i det steget till standardvärden. Det sista alternativet är Reset All som återställer hela guiden till startläget. Här går man tillbaka till steg 1 och initierar dess inställningspanel. Inget görs här direkt åt redan angivna data men de skrivs successivt över när användaren går vidare i guiden.

12.1.3 Övriga lyssnare

Efter metoden `actionPerformed` följer ytterligare två lyssnarmetoder, som dock är mycket mindre till omfattningen. Den första hör till lyssnaren `TextListener` och har hand om att automatiskt föra över punktetiketter till alla fält i steg 2 om man har mer än en initierarlinje. Den andra har hand om den automatiska ifyllnaden av fält i steg 2. Den anropar metoden `calculateFieldValues()` som finns beskriven i avsnitt 12.3.

12.2 Beräkning av IFS-kod

Det finns två programdelar i Fractals som beräknar IFS-kod. Det är Point Wizard och Generator Wizard, som båda ligger under underrubriken IFS Code Solver. Dessa båda program beräknar IFS-koden på samma sätt även om de hämtar indata på lite olika sätt. Själva beräkningsfasen är i båda programmen placerad i en speciell metod `calculateIFSCode()`. Detta är en ganska komplicerad metod att förstå utan den bakgrundsteori som här ges.

Som vi såg i avsnitt 7 leder beräkningarna tillslut till ett antal ekvationssystem. Då ett ekvationssystem på formen

$$\begin{cases} ax + by + cz &= p \\ dx + ey + fz &= q \\ gx + hy + iz &= r \end{cases}, \tag{11}$$

där x, y och z är de obekanta variablerna skrivs på matrisform år det följande utseende:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} p \\ q \\ r \end{pmatrix}, \tag{12}$$

vilket vi i fortsättningen skriver

$$AX = C. \tag{13}$$

Det finns några olika sätt att lösa en sådan ekvation. Ett av dem är med hjälp av Cramers regel. Denna regel kan på allmän form skrivas

$$x_k = \frac{1}{\det A} \begin{vmatrix} a_{11} & \cdots & a_{1,k-1} & b_1 & a_{1,k+1} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2,k-1} & b_2 & a_{2,k+1} & \cdots & a_{2n} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & \cdots & a_{n,k-1} & b_n & a_{n,k+1} & \cdots & a_{nn} \end{vmatrix}, \tag{14}$$

där

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}. \tag{15}$$

Överfört på vårt ekvationssystem gör det att om variabeln x skall beräknas ersätts den första kolonnen i A med X. Determinanten för denna nya matris divideras med determinanten för den ursprungliga matrisen A. Om Variabeln y skall ber
äknas ersätts den andra kolonnen i A med X o.s.v.

Vidare beräknas determinanten för en kvadratisk matris med n rader och n kolonner genom

$$\det A = \begin{cases} a_{11} & \text{då } n = 1 \\ a_{11}a_{22} - a_{12}a_{21} & \text{då } n = 2 \\ \begin{array}{l} a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} \\ \quad -a_{11}a_{23}a_{33} - a_{12}a_{21}a_{33} - a_{13}a_{22}a_{31} \end{array} & \text{då } n = 2 \\ \sum_{j=1}^{r} a_{ij}(-1)^{i+j} \det D_{ij} & \text{då } n > 3 \end{cases} \tag{16}$$

där D_{ij} betecknar determinanten till den matris man får om man i A stryker rad i och kolonn j. Om determinanten för matrisen A i matrisekvationen ovan skulle vara lika med 0 saknar ekvationssystemet entydiga lösningar, d.v.s. antingen saknar det lösningar helt eller också har systemet oändligt många lösningar.

För varje transformation i fraktalen får vi nu ett ekvationssystem med tre x-ekvationer genererade av formeln $X' = aX + bY + e$ där det alltså finns tre obekanta: a, b och e (motsvarande x, y och z ovan). De konstanter som finns är X', X, Y och 1 (motsvarande p, a, b och c ovan). Vi får också för varje transformation ett ekvationssystem med tre y-ekvationer på motsvarande sätt. Vad man finner när alla ekvationssystem (inom samma initierarlinje) är uppställda är att matrisen A i matrisekvationen alltid är den samma. Detta förstås enkelt, eftersom samma tre punkter transformeras i alla transformationer, och det är dessa punkters koordinater som lagras i matris A. Däremot transformeras punkterna till olika koordinater, men detta påverkar matrisen C och inte A.

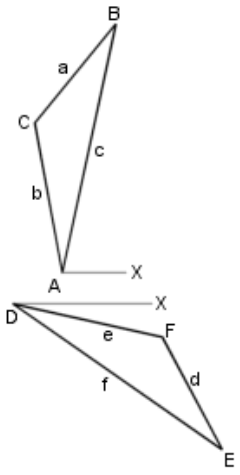
Detta medför alltså att endast en matris A behöver skapas för varje initierarlinje. I programmet kallas denna matris för `coefficientMatrix`. Denna är ett tredimensionellt flyttalsfält (`double[][][]`). Däremot måste det skapas en resultatmatris (motsvarande C ovan) för varje transformation och för varje komponent (X och Y) i koordinaterna. I programmet kallas dessa variabler för `resultMatrixX` och `resultMatrixY`. Båda är tvådimensionella flyttalsfält (`double[][]`), vari det första indexet anger vilken transformation (vilket ekvationssystem) som matrisen skall användas i och det andra indexet speglar själva matrisen.

Vad som händer i slutet av metoden är det som ovan beskrivits i allmänna matematiska termer. Man itererar för varje transformation. Itererar inom transformationsiterationerna också för varje kolonn i koefficientmatrisen (A). Där görs två kopior av koefficientmatrisen med metoden `arrayCopy`, en för x-ekvationssystemet och en för y-ekvationssystemet. I kopiorna ersätts kolonner som beskrevs ovan och konstanterna beräknas och placeras in i IFS-koden. Här finns det en hjälpvariabel m som placerar det beräknade talet i rätt kolumn i IFS-kodstabellen.

Det sista som händer i metoden är att lämpliga starpunkter läggs in på sin plats, något som alltså inte har något direkt samband med själva beräkningen av koden. Dessa punkter hämtas från de punkter som användaren angett som indata till programmet. Efter detta tas nästan hela metoden om för nästa initierarlinje. De matriser som använts vid beräkningen av aktuell kod skrivs nu över.

Metoden `arrayCopy` kan behöva en noggrannare förklaring. Den kopierar alltså ett tvådimensionellt flyttalsfält. Om detta skulle göras på vanligt sätt genom att initiera en ny fältvariabel och ge denna det aktuella fältets värde skulle bara referenser till fältet kopieras. Detta skulle innebära att när kolonner i kopiorna sedan ersätts skulle detta ske även i den ursprungliga matrisen. Det skulle kort sagt bli fel. `ArrayCopy` gör däremot en s.k. djup kopia, den kopierar matriserna element för element. Då uppnår man vad man ville, att kopiera matrisens värden och inte en referens till densamma.

Figur 21: Skiss till koordinatberäkningarna



12.3 Koordinatberäkningar

I fig. 21 är A, B och C tre punkter i en generator. När då denna generator ska appliceras på olika linjer i initieraren kommer generatorns skala att ändras och eventuellt vrids hela generatorn. Observera dock att inga vinklar inne i själva generatorn ändras; de båda trianglarna till vänster är ju likformiga.

Programmet skall alltså under förutsättning av att användaren angivit koordinater för tre punkter i generatorn på en specifik linje i initieraren, samt angivit motsvarande de två första punkterna för samma generator applicerad på en annan av initierarens sidor, kunna beräkna den tredje punktens motsvarighet i den senare nämnda generatorn. Alltså, sträckan AB är initierarens första linje och punkten C är en tredje punkt i generatorn. Observera att de trianglar som här är utritade inte är generatorer och att det därmed kan finnas flera punkter i generatorn mellan varje utsatt punkt i skissen.

När nu koordinaterna för tre punkter i den första generatorn är definierade önskar man applicera samma generator på en annan sida i initieraren. Denna sida motsvarar i skissen sträckan DE. Koordinaterna för generatorns alla punkter måste så beräknas. Koordinaterna för punkterna D och E är redan givna, däremot måste koordinaterna för punkten F beräknas. Detta görs genom att längden av sträckan DF, i fortsättningen kallad e, beräknas.

$$\begin{gathered}\lceil\text{likformighet}\rceil \Rightarrow \frac{b}{e} = \frac{c}{f} \Rightarrow e = b \cdot \frac{f}{c} \Rightarrow \\ e = \sqrt{(x_C - x_A)^2 + (y_C - y_A)^2} \cdot \frac{\sqrt{(x_E - x_D)^2 + (y_E - y_D)^2}}{\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}}\end{gathered} \tag{17}$$

Vidare beräknas vinkeln mellan e och x-axeln:

$$v_b - v_e = v_c - v_f \Rightarrow v_e = v_b - v_c + v_f \tag{18}$$

När nu avstånd och vinkel är kända kan koordinaterna för den sökta punkten F bestämmas:

$$F = (x_d + e\cos v_e,\ y_d + e\cos v_e) \tag{19}$$

De ovan nämnda beräkningarna utförs sedan för alla ytterligare punkter i generatorn och för varje sida i initieraren förutom den första.

Dessa beräkningar utförs i metoden `calculateFieldValues()`. Den börjar med att kontrollera vilka data som finns angivna och vilka som därmed kan beräknas. Sedan använder den ovanstående metoder för att beräkna övriga punkter, vilkas koordinater sedan skrivs ut.

12.4 Vinklar

I Point Wizard och i några av de andra programdelarna behöver man ta fram en vinkel mellan en rät linje och x-axeln. Då används följande modell, där v är den sökta vinkeln, Δx är avståndet i x-led och Δy är avståndet i y-led mellan två punkter på linjen:

$$v = \begin{cases} \arctan\frac{\Delta y}{\Delta x} & \text{då } \Delta x > 0 \\ 0 & \text{då } \Delta x = 0 \text{ och } \Delta y = 0 \\ \text{sgn}\Delta y \cdot 90 & \text{då } \Delta x = 0 \text{ och } \Delta y \neq 0 \\ 180 & \text{då } \Delta x < 0 \text{ och } \Delta y = 0 \\ \text{sgn}\Delta y \cdot \left(180 - \left|\arctan\frac{\Delta y}{\Delta x}\right|\right) & \text{då } \Delta x < 0 \text{ och } \Delta y \neq 0 \end{cases} \tag{20}$$

Den andra raden tål att förklaras. En vinkel skall då ges för en linje mellan två punkter som ligger på samma ställe. Normalt skulle detta leda till ett odefinierat svar, vinkeln kan ju faktiskt vara vilken som helst. För enkelhetens skull sätts den här till 0 grader. Vinkeln används bland annat när en figur skall vridas. Då hämtas en linjes aktuella lutning med denna formel och sedan adderas vridningsvinkeln. Om då denna formel ger vinkeln 0 grader i detta fall så adderas vridningsvinkeln på det och linjens nya vinkel blir densamma som vridningsvinkeln.

Denna formel finns som en egen metod i Point Wizard och Generator Wizard med namnet `getAngle`. Argumenten är namngivna `dx` (motsvarande Δx) och `dy` (motsvarande Δy).

13 IFS Code Solver - Generator Wizard

Detta delprogram är utan tvekan det mest avancerade av de fyra i Fractals, både till automatisering och källkodslängd. Dock finns det på många ställen stora likheter mellan detta och Point Wizard. Ibland är hela metoder lika eller nästan lika. Också guideformen är en likhet som speglas väl i stegens initieringsmetoder och i händelsemetoden `actionPerformed`.

13.1 Programmets gång

13.1.1 Konstruktorn och stegens initieringsmetoder

Konstruktorn i denna klass gör det vanliga som skall göras. Den gör grundläggande inställningar, anropar metoder för att initiera och lägga till menyraden och knappraden nederst. Sedan anropar den också initieringsmetoden för första steget. Detta behandlas nedan. Också navigationspanel, knappar för zoomning och själva ritytan initieras och placeras ut här. När allt detta är färdigt visas fönstret.

I slutet av konstruktorn initieras talredigeringsobjekt sam några listor för att lagra användarens inställningar. Först gäller det den tvådimensionella listan pointData1 som lagrar information om punkterna som anges i steg 1. Varje rad i detta fält motsvarar en punkt. Raderna innehåller i nämnd ordning x-koordinat, y-koordinat, avstånd från förra punkten, vinkel på en eventuell linje från förra punkten samt om det skall vara någon sådan linje (talet 1 lagras då, annars 0). I konstruktorn förbereds också listorna för funktionerna flip och mirror. Sist görs rätt navigationskomponenter tillgängliga genom anropet av `handleEnabledDisabledComponents()`.

Stegens initieringsmetoder är namngivna precis som i Point Wizard. Om dessa metoder skrevs det i samma sammanhang att dessa metoder innehöll mestadels initiering och utplacering av komponenter i paneler med layouter. Samma sak gäller här, och därför är det obefogat att gå igenom dessa noggrant här.

13.1.2 Händelselyssnaren `ActionListener`

Som vanligt är metoden `actionPerformed` en mycket viktig metod i programmet. Härifrån styrs många av de funktioner som det grafiska användargränssnittet har att erbjuda. Denna metod börjar på samma sätt som i Point Wizard, med knapparna för att gå bakåt, framåt och till ett specifikt steg. Alla dessa förflyttningar sker med anrop av vad som här tidigare kallats goTo-metoder. Eftersom ett liknande system finns i Point Wizard ska vi inte gå in på detta mera i detalj. Det kan dock nämnas att i metoderna `goToStep2()` och `goToStep4()` börjar programmet, om steg 1 resp. 3 är det aktuella, med en kontroll av givna indata. Kraven för att gå vidare finns specificerade i avsnitt 8.3.2 resp. 8.3.4.

Efter detta börjar de stegspecifika komponenternas satser, uppdelade efter just steg. Först ut är knappen Add point i steg 1. Denna skall lägga till en angiven punkt i första punktlistan, lägga till data om punkten i den lista som finns i första steget, samt rita om ritytan. När detta är gjort i ovan nämnd ordning kontrolleras också så att inte antal punkter är 100, för då får det inte plats fler i första punktlistan. Om antalet punkter är 100 spärras knappen Add point.

Från första steget kommer också knappen Remove point. Denna gör precis det motsatta som beskrivits ovan. Även här kontrolleras punktantalet i efterhand. Om det finns en punkt i listan spärras knappen eftersom den första punkten ej kan röras. Om punktantalet är 99 görs knappen Add Point åter tillgänglig.

När sedan komponenterna från steg tre skall behandlas händer i stort sett samma sak som ovan beskrivits. Skillnaden är att det är initierare som behandlas och att punkterna därför lagras i tredje punktlistan. Dessutom är antalet punkter här begränsade till 50. I övrigt sköts kontrollerna för tillgänglighet hos Add Point resp. Remove Point på samma sätt som ovan.

I steg 5 är det knappen Preview som skickar händelser till metoden. Då anropas metoden `setPointData5` som beräknar femte punktlistan utifrån de data som angivits i steget. Denna metods definition finns i koden en bit nedanför sista steginitierarmetoden `initStep6()`. Metoden anropas också när steg 5 initieras för att behandla de redan ifyllda värdena. När denna metod fått exekveras ritas också ritytan om. I steg 6 finns det tre knappar vars händelser här behandlas. Dock är alla dessa tre likvärdiga med dem som beskrivits i Point Wizard, och en beskrivning av dessa finns i avsnitt 12.1.2.

Sist i `actionPerformed` finns satser för zoomningsknapparna och navigationspanelen. Zoomningsknapparna dubblerar respektive halverar skalan. De har en gräns för vilka tal skalan ska ligga inom för att de skall fungera. Denna ligger på 10^6 resp. 10^{-6} . Om dessa gränser uppnås förlorar knapparna sin funktion.

För att hantera navigationspanelen används dubbla loopar för rader och kolumner. Först kontrolleras om någon tryckt på mittknappen. I så fall skall ikonen på mitten växla mellan ett S och B och steglängden skall ändras till det motstående värdet. Både ikoner och steglängder finns deklarerade som konstanta variabler i början av koden.

Sedan kontrolleras om det är någon av de andra knapparna som användaren tryckt på. I så fall beräknas de nya koordinaterna för mittpunkten med hjälp av det konstanta fältet `directions`. Det består av talen 1, 0 och -1 och anger vad x- och y-koordinaterna skall multipliceras med för utom steglängden för att bilden skall flyttas åt rätt håll. En förflyttning av mittpunkten uppåt åt vänster motsvarar i directions talparet -1, 1 (negativ förflyttning i x-led, positiv i y-led).

13.1.3 Övriga lyssnare

Lyssnarmetoden itemStateChanged håller i första och tredje steget reda på två saker. Först de två radioknapparna för att välja mellan rektangulär och polär inmatning av data. Metoden får då i uppgift att göra rätt fält tillgängliga. Det andra metoden gör i dessa steg är att visa utförligare information om de punkter som visas i listan i respektive steg. Då en punkt väljs skriver metoden ut data i de tre textfälten omedelbart nedanför listan med punkter.

I steg 2 och 4 är denna lyssnare viktigast, eftersom alla komponenter där styrs av den. Då en transformation/initierare väljs ur listan gör lyssnaren att rutorna flip och mirror bockas för enligt tidigare inställningar. Då någon av kryssrutorna används gör lyssnaren också att inställningen sparas i en speciell variabel för att kunna plockas fram nästa gång transformationen/initieraren markeras i listan.

Metoden textValueChanged, kopplad till textListener, utför i steg 1 och 3 en kontroll av de koordinater, såväl rektangulära som polära beroende på vilket alternativ som är valt, som användaren anger. Om koordinaterna är angivna på ett felaktigt sätt eller ligger utanför de tillåtna intervallen gör detta att knappen Add Point blockeras.

13.2 Koordinatsystem

Koordinatsystem används på några olika ställen i programmet. I samtliga fall används fönstrets mittpunkts koordinater i det bakomliggande koordinatsystemets koordinater. Med det bakomliggande koordinatsystemet menas det koordinatsystem som genom fönstret delvis skall avbildas.

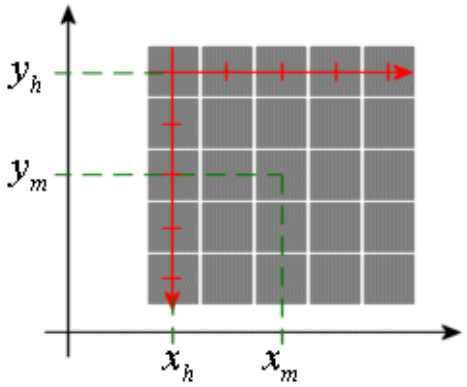
När en punkt skall ritas i fönstret måste dess koordinater anges. Det är då koordinaterna i det bakomliggande koordinatsystemet som anges. För att kunna rita punkten i fönstret krävs det att man kan översätta dess koordinater till pixelkoordinater. Det koordinatsystem som pixlarna i fönstret ger upphov till är som alltid när det gäller datorgrafik uppochnedvänt. Detta gör att den övre vänstra punkten, från och med här kallad hörnpunkten, har koordinaterna $(0, 0)$. För att få reda på pixelkoordinaterna kan man ta reda på avståndet i x- och y-led till hörnpunkten.

Skalan s anger förstoringsgraden, eller med andra ord hur många pixelbredder som ska motsvara en längdenhet i det tänkta koordinatsystemet som fönstret ska visa en del av. För att få pixelkoordinater måste man alltså multiplicera avståndet med s. Dessutom måste man p.g.a. det uppochnedvända koordinatsystemet multiplicera uttrycket för y-koordinaten med -1. Slutligen får vi alltså följande formler:

$$x_p = s \cdot (x_b - x_k) \tag{21}$$

$$y_p = s \cdot (y_k - y_b), \tag{22}$$

Figur 22: Översikt över de två koordinatsystemen



där x_p och y_p är pixelkoordinater, x_b och y_b är koordinater i det bakomliggande koordinatsystemet och x_h och y_h är koordinater för hörnpunkten.

Dessa formler översätter alltså de koordinater som anges till pixelkoordinater. Observera att eftersom det handlar om pixlar kan pixelkoordinaterna endast vara heltal. Därmed måste de erhållna värdena avrundas.

I formeln används fönstrets hörnkoordinater, men i programmet anger man dess mittkoordinater. Vi måste således finna ett sätt att ta reda på hörnpunktens koordinater. Vi vill alltså kunna uttrycka den översta vänstra pixelns koordinater med hjälp av pixelsystemets specificerade mittpunktskoordinater. För att kunna göra detta måste vi även känna den rådande skalan och hur många rader och kolumner av pixlar som finns i pixelsystemet. Känner vi dessa kan vi först, utan att ta hänsyn till skalan, se att avståndet i x-led mellan mitt- och hörnpunkterna är $x_m - x_k = \frac{w-1}{2}$ där w anger antal pixelkolumner. I y-led får uttrycket utseendet $y_k - y_m = \frac{h-1}{2}$.

Skalan s anger alltså hur många pixelbredder som ska motsvara en längdenhet i det tänkta koordinatsystemet som fönstret ska visa en del av. Härav följer att det inverterade värdet betecknar hur bred en pixel är uttryckt i det bakomliggande koordinatsystemets längdenheter. Om vi nu multiplicerar högerleden i formlerna för avstånd mellan mitt- och hörnpunkter ovan med s^{-1} och sedan löser ut hörnpunktens x- resp. y-koordinat får vi följande samband:

$$x_k = x_m - \frac{w-1}{2s} \tag{23}$$

$$y_k = y_m + \frac{h-1}{2s} \tag{24}$$

Koordinaterna för hörnpunkten behöver enligt formlerna bara räknas om när fönstrets storlek, mittpunktens koordinater och/eller skalan ändras.

Man kan trots allt fråga sig varför man inte skulle kunna ange fönstrets position med hjälp av dess hörnpunkt. Detta för att slippa räkna ut denna i efterhand. Dock är det en stor fördel att använda sig av mittpunkten vid zoomning, då denna förblir oförändrad.

14 IFS Code Editor

Detta är alltså programmet för att redigera IFS-kod. Vi går igenom programmet del för del.

14.1 Konstruktorn

Konstruktorn i programmet är relativt enkel. Den tar som argument emot ett tredimensionellt flyttalsfält (`double[][][]`) som innehåller en IFS-kod. Det är inte nödvändigt att skicka med någon kod. I vanliga fall gör man inte det och anger då `null` som argument. De gånger som denna variabel används är då redigeringsfunktionen i Point Wizard och Generator Wizard används. Vi kommer till va som händer då denna variabel används.

Som vanligt anropas först superklassens konstruktor med `super()`. Sedan sker grundläggande fönsterinställningar och initiering av fönsterlyssnare. Sedan initieras den ruta med rullningslister som upptar hela fönstret och som editorformulären placeras i.Efter detta initieras också menyraden och placeras i fönstret.

Sedan kommer vi till den eventuellt angivna IFS-koden. Här skapas, om IFS-kod finns angiven, direkt ett editorformulär. Dessutom spärras programmet så att just detta programfönster inte kan användas för att redigera någon annan IFS-kod Det är helt enkelt kommandona för att öppna och skapa nya filer som spärras. Däremot kan man naturligtvis fortfarande använda kommandona Save, Save HTML och View.

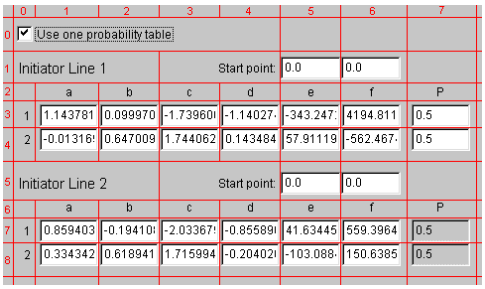
14.2 Metoden `buildEditorPanel`

Denna metod bygger upp ett nytt editorformulär och har fem argument. Det första, `fixed`, anger vilken sannolikhetsmodell som skall användas i koden. Värdet `true` här anger att fasta sannolikheter skall användas, annars skall de vara beroende. Se mera om detta i avsnitt 8.4.1.

Nästa argument, `probabilitiesAsBefore` anger om sannolikhetsmodellen är den samma som angetts i filen. Som framgick i avsnitt 8.4.1 kan detta ändras i den dialogruta som visas under öppnandet av en fil. Om värdet är `true` skall alltså sannolikheter läsas in från den bifogade IFS-koden, annars ska sannolikhetsfälten lämnas tomma.

Argumentet `IFSCode` talar för sig själv vad det innebär. Detta gäller även de två sista. Anledningen till att de finns med är för att metoden skall kunna användas både när ett editorformulär skapas från en fil genom kommandot Open och när ett helt nytt formulär skapas genom kommandot New. I det senare fallet sätts argumentet `IFSCode` till `null` och de två sista argumenten bifogar de värden som användaren angett i den dialogruta som visas när kommandot New används.

Figur 23: Layoutens koordinater



Metoden består till största delen av initiering av komponenter och utplacering av desamma i editorformulärets panel. Det som kan kräva en del tankeverksamhet här är att beräkna i vilken rad och kolumn i layouten som komponenterna hamnar. Med hjälp av figur 23 kan man dock ganska enkelt ta fram vilka koordinater som det j:te fältet på den i:te raden i tabellen (rubrikraden inräknat) för den h:te initierarlinjen. Första initierarlinjens tabell ligger på rad 2. Varje tabell upptar lika många rader som antalet transformationer (t) plus 2 rubrikrader. Därmed får vi koordinaterna $((t+2)\cdot h+2,\ j)$

14.3 Lyssnaren `ActionListener`

Klassen `IFSCodeEditor` implementerar tre olika lyssnargränssnitt: `ActionListener`, `ItemListener` och `TextListener`. det resulterar i att tre lyssnarmetoder måste användas.

I metoden `actionPerformed` behandlas de sex menyalternativen i menyn File. Först ut är New (`newFile` i koden eftersom `new` är ett reserverat ord i Java och variabler får därför inte heta så). Programmet börjar med att kontrollera om något editorformulär redan finns i fönstret. i så fall har kontrollvariabeln värdet `true` och användaren varnas för att detta kommer att raderas. Sedan anropas konstruktorn i klassen `newDialog` som öppnar en separat dialogruta, se avsnitt 14.5. När denna har stängts anropas metoden `buildEditorPanel` (se avsnitt 14.2) och om detta är det första editorformuläret som öppnas i detta fönster görs vissa menyalternativ nu tillgängliga efter detta.

Satserna som skall exekveras om användaren använder kommandot Open är något fler och mer komplicerade än ovan beskrivna. De börjar dock precis som med New med att kontrollera om något editorformulär finns i fönstret. Sedan visas en görs en fildialog i ordning. Sedan följer en `do`-sats, d.v.s. en upprepningssats där ett villkor kontrolleras efter att satserna i blocket körts och inte före som är fallet med en `while`-sats. Villkoret finns där upprepningsblocket tar slut.

Hursomhelst, fildialogen visas och sedan kontrolleras genom kontroll av filnamnsvariabeln att användaren inte tryckt på Avbryt-knappen. Sedan försöker programmet att läsa in ett tredimensionellt flyttalsfält från filen. Om inte detta går visas ett felmeddelande. Sedan tar en omfattande kontrollprocess vid. Kontrollvariabeln genom hela inläsningen och kontrollen heter `error`. Denna är satt till `false` från början, men så fort ett fel påträffas sätts den till värdet `true`. Detta leder till att kontrollprocessen avbryts, eller rättare sagt spärras resterande kontroller. Villkoret för att utföra varje specifik kontroll är nämligen att inget fel får ha skett.

När kontrollerna är färdiga har upprepningssatsen körts en gång. Villkoret för att en upprepning skall ske är att ett fel ska ha inträffat. Omvänt är alltså villkoret för att få gå vidare att inget fel får ha inträffat. Om upprepning sker öppnas fildialogen på nytt och en ny fil kan väljas.

En dialogruta initieras (se avsnitt 14.5) men visas inte om både antal initerarlinjer och antal transformationer är 1, då dialogrutan är helt onödig. Annars visas den och användarens inställningar lagras. Nästa steg är att göra verklighet av användarens inställningar om borttagning av initierarlinjer och/eller transformationer. Detta går till så att alla initierarlinjer och transformationer gås igenom i turordning. Om aktuell linje eller transformation finns i respektive borttagningslista tas denna bort. För att slippa kontrollera hela borttagningslistan varje gång används en speciell fältvariabel som håller rätt på vilken position i de båda listorna som skall kontrolleras. Listorna är ordnade i nummerordning och om t.ex. en transformation tas bort ändras denna variabel så att nästa position i den kontrolleras.

Genom hela öppningsprocessen har IFS-koden sparats i en temporär variabel `tempIFSCode` eftersom det inte är säkert att den går igenom hela kontrollen. Nu är det emellertid dags att lägga koden i sin riktiga variabel `IFSCode`. När detta är gjort anropas metoden `buildEditorPanel` (se avsnitt 14.2) och sedan händer samma sak som ovan beskrivits för kommandot New.

Det tredje kommandot som här behandlas är Save. Detta fungerar på precis samma sätt som i Point Wizard och mer info om detta finns i avsnitt 12.1.2. Det finns dock ett undantag, och det är det inledande anropet av metoden `readForm()` inlagt i en `if`-sats. Denna metod, vars definition finns i koden omedelbart innan `actionPerformed`, kontrollerar så att alla fält innehåller korrekta värden. Om så inte är fallet returnerar den värdet false, och `if`-satserna i `actionPerformed` gör att densamma avslutas utan åtgärd. I metoden `readForm` används metoden `valueOK()` för att kontrollera textfälten. Denna metod finns definierad omedelbart innan `readForm` och kontrollerar om varje värde går att läsa in korrekt utan fel.

Kommandot Save HTML börjar med samma satser som kommandot Save. Däremot följer ett block med satser där HTML-kommandon skrivs ut på den angivna HTML-filen. Detta är dock inget som är nödvändigt att beskriva närmare. Avslutningsvis finns koden till kommandona View och Quit. Hur dessa fungerar har tidigare beskrivits.

Tabell 4: Informationsöverförande metoder i dialogrutorna

Funktion	**Finns i**	**Returnerar**
`fixedProbabilities()`	New och Load	Om sannolikheterna skall vara fasta
`getInitiatorLines()`	New	Antal initierarlinjer
`getTransformations()`	New	Antal transformationer
`getSelectedIndexes()`	Load	En tvådimensionell lista över vilka initierarlinjer/transformationer som skall tas bort. Den första raden anger initierarlinjer och den andra transformationer

14.4 Övriga lyssnarmetoder

Det finns alltså ytterligare två lyssnarmetoder i denna klass. De har båda att göra med funktionen One probability table.

Metoden `itemStateChanged` som hör till `ItemListener` bevakar funktionens kryssruta överst i editorformuläret. Om denna bockas för kommer en varningsdialogruta upp. Sedan görs alla sannolikhetsfält förutom i första initierarlinjen otillgängliga för användaren och värdena ersätts med värden från första initierarlinjen.

Metoden `textValueChanged` som hör till `TextListener` är den som ser till att sannolikhetsvärdena ändras i alla initierarlinjer då de ändras i den första.

14.5 Dialogrutorna New och Load

Dessa två dialogrutor används då ett nytt editorformulär skall skapas, antingen helt nytt eller från fil. Dessa dialogrutors funktion har ganska ingående beskrivits tidigare (se avsnitt 8.4.1) och behöver därför inte beskrivas så noggrant här.

Vad man kan titta på är sättet på vilket användarens inställningar hämtas till programmet då denne tryckt på OK-knappen. Detta sker med en uppsättning metoder i varje klass som enbart har till uppgift att returnera en inställning. Tabell 4 ger en översikt över dessa.

15 Fractal View

Denna programdel består av en inställningspanel och en rityta. Vi går här först igenom huvudfönstret med inställningspanelen och sedan förklaras ritytan i ett eget avsnitt.

15.1 Huvudfönstret

I huvudfönstret kan alla nödvändiga inställningar göras för fraktaluppritningen. Själva uppritningen sker i den speciella ritytan `FractalViewImageCanvas`.

Huvudfönstret initieras som vanligt i konstruktorn. Denna består till allra största del av initiering av komponenter och utplacering av dessa i fönstret. I slutet av konstruktorn kommer ett avsnitt som kontrollera om någon IFS-kod getts som argument. Det normala är att så inte skett, argumentet har satts till `null`. Som tidigare används detta när ett annat delprogram i Fractals skall använda sig direkt av Fractal View. De satser som finns här i slutet av konstruktorn spärrar i så fall knappen Set IFS Code, så att fönstret endast kan användas till den aktuella IFS-koden.

Vidare finns det två lyssnarmetoder. Den första är `actionPerformed` som här tar hand om knapptryck från programmets knappar. Första knappen ut är Set IFS Code. Dess satser i lyssnarmetoden innehåller en upprepningssats som i princip överensstämmer med den som beskrivits för öppningskommandot i IFS Code Editor, se avsnitt 14.3. När dessa körts följer ett anrop av metoden `editProbabilities()`. I denna skrivs sannolikheterna i IFS-koden om för att passa för iterationerna. Vad som skrivs in i sannolikhetsfälten nu är den sammanlagda sannolikheten för alla element fram till det aktuella. En sannolikhetslista som har utseendet $0.2, 0.3, 0.1, 0.4$ skrivs alltså om till $0.2, 0.5, 0.6, 1.0$ i denna metod. Vid iterationerna slumpas ett tal mellan 0.0 och det sista talet i listan. Sedan kontrolleras om detta tal är mindre än varje tal i de omskrivna sannolikhetstabellerna och därmed får man fram en fördelning av transformationer enligt de specificerade sannolikheterna.

Sedan följer satser för knappen Draw. Det som händer här är att givna indata för mittpunkten kontrolleras. IFS-koden kontrolleras ej här, men eftersom knappen Draw är otillgänglig tills en IFS-kod angetts är detta inte nödvändigt. Sedan skickas parametrarna till ritytan genom anrop av dess metod `setParameters` och sedan ges kommando att rita genom att anropa metoden `draw()`, också den i ritytans klass `FractalViewImageCanvas`.

Då knappen Cancel görs som enda sats ett anrop av ritytans metod `cancel()`. Se mera om denna i avsnitt 15.2. Efter detta följer hanteringen av navigationspanelen. Denna finns också beskriven i avsnitt 13.1.2. Efter detta följer lyssnarmetoden `adjustmentValueChanged` som känner av om de båda rullningslisterna ändrats. I så fall ändras också det värde som står

utskrivet under dessa.

Sist i koden finns en metod `run()` definierad. Klassen `FractalView` implementerar gränssnittet `Runnable` som gör att man kan ha två parallella programtrådar som exekverar samtidigt. Förutom den vanliga är det som specificeras i metoden `run`. run måste finnas i klasser som implementerar gränssnitten, men den får innehålla vad som helst, doch består den oftast av en upprepningssats ytterst. Så är fallet också här. Denna `run`-metod styr förloppsindikatorn för iterationer genom att regelbundet (25 gånger per sekund) kontrollera värdet på antal utförda iterationer i ritytan. Detta sker inte som vanligt genom metodanrop utan variablerna nås direkt. Detta för att spara datorkraft då iterationer pågår. Anropet `sleep(40)` anger att tråden skall vänta 40 ms och sedan åter kontrollera värdet.

15.2 Ritytan

Denna klass saknar händelselyssnare, eftersom det inte finns några komponenter i den. Däremot finns det ett antal andra metoder. Som vanligt börjar man med en konstruktor. Denna tar ett argument, en etikett för att visa muspekarens koordinater. Komponenten är initierad och visas i huvudfönstret Fractal View. Det som skickas med här är en referens till denna så att ritytan kan skriva ut koordinaterna i fönstret. I konstruktorn lagras referensen och en inställning för muspekaren görs. Ytterligare några variabler initieras. Sedan startas den parallella tråden som skall ligga och vänta på att få iterera.

Nästa metod i koden är `setParameters`. Denna tar emot fyra argument: en IFS-kod, en mittpunkt, skalan och antal iterationer. Dessa överförs till klassens egna variabler. I samband med detta sätts också variabeln `fixed` efter vilken sannolikhetsmodell som används.

Nästa metod är `draw()`. Den anropas då hela fraktalen skall ritas om. Det enda denna metod gör är att sätta variabeln `redraw` till `true`. Detta gör att den parallella tråden aktiveras och ritar om fraktalen.

Nästa metod är just metoden `run()`. Det första som finns i denna är repetitionssatsen som hela tiden körs. I denna kontrolleras först om variabeln `redraw` har värdet `true`, d.v.s. om fraktalen skall ritas om. Om den inte ska det sker inget annat än att tråden väntar 40 ms och sedan försöker igen. Om fraktalen däremot skall ritas om börjar satserna i metoden att exekveras.

Först tillverkas en ny bild i som ännu inte visas på skärmen. Sedan utförs iterationerna för varje initierarlinje. Nuvarande koordinater sätts enligt den i IFS-koden angivna startpunkten. Därefter kan iterationerna börja i en repetitionssats som tas om en gång för varje iteration. För att måste ett par villkor vara uppfyllda. Först och främst får inte alla iterationer vara utförda och dessutom får inte avbrytflaggan vara satt, se nedan.

Det första som görs på varje varv är att slumpvis välja en transformation. Detta sker på olika sätt beroende på vilken sannolikhetsmodell som används.

Det är variabeln `fixed` som väljer vilken av alternativen som skall väljas. Sedan utförs det som är den egentliga transformationen. Här måste variabler för nuvarande värden och nästkommande värden användas eftersom båda formlerna använder de nuvarande x- och y-värdena. Om då x-satsen ändrar x-värdet används det nya x-värdet och det gamla y-värdet i y-satsen. Detta gör att transformationen blir felaktig. Slutligen beräknas vilken pixel som skall plottas med hjälp av metoden `translateCoordinates`. Denna tillämpar formler som tagits fram i avsnitt 13.2.

Efter dessa förhållandevis långa metoder följer ett stort antal korta metoder. Den första, `showCoordinates`, anropas som enda metod då en musrörelse registreras i händelselyssnaren `FractalViewImageCanvasMouseMotionListener`. Den visar muspekarens koordinater på skärmen genom etiketten som tagits som argument i konstruktorn till ritytan. Sedan följer metoder för att beräkna hörnpunkt, ange och hämta mittpunkt samt ange och hämta skala. Sist kommer metoden `cancel` som sätter flaggan `cancel` till `true`. Detta innebär att om den parallella tråden itererar kommer dess repetitionssatser nästa gång de skall börja om att hindras de från detta och omritningen avbryts.

16 Sammanfattning

Så finns det nu ett program som heter Fractals och som utgör det dataverktyg som kan utgöra en stor hjälp vid undersökningar av fraktaler. Det leder till stora tidsbesparingar då en IFS-kod skall beräknas. Det gör det enkelt att redigera kod, kanske då framför allt sannolikheterna. Det blir också enkelt att rita upp fraktaler, från fil eller direkt från t.ex. redigeringsprogrammet.

Att säga att programmet nu är färdigt vore nog att gå ett steg för långt. För det första, ett dataprogram blir (i alla fall nästan) aldrig färdigt. Vad man kan göra är att bestämma sig för att det är tillräckligt färdigt för att det skall gå att använda. Sedan kan man ju alltid fortsätta att förbättra ett program för att sedan släppa det under ett nytt versionsnummer. Det program som nu skapats har verisonsnummer 1.0 och det finns många lediga nummer uppåt för eventuella uppdateringar.

För det andra, programutvecklingen har inte drivits i form av ett projektarbete. Normalt sker programutveckling i ett helt arbetslag. På så sätt kan man skriva några programdelar var och på så sätt lägga ner mera arbete på varje detalj. Den största fördelen med att arbeta i grupp vid programmeringsarbete är dock kanske möjligheten till utbyte av ider under arbetet. Likaså kan man diskutera programmets utformning och få tips och ideer om hur programmet på bästa sätt skall utformas för att användargränssnittet skall bli så lättförståeligt och praktiskt att använda.

När man, som vid arbetet med Fractals arbetar ensam mister man dessa fördelar av grupparbete. Detta behöver nödvändigtvis inte betyda att programmets kvalitet och funktion blir sämre, men arbetet tar förmodligen längre tid. Specialarbetet är enligt betyget motsvarande 20 arbetstimmar. Detta är dock något som man nästan som en regel överskrider. Detta gäller även detta arbete där tiden har blivit mångdubbelt längre.

Att arbetet med programmet tagit lång tid är dock inget jag ångrar. Jag har genom hela arbetet tyckt att det var intressant. Visserligen har jag inte alltid varit vän med kompilatorn då den spottat ut obegripliga felmeddelanden sedan den synat min kod. Men jag ser det som en problemlösning, att sova på saken brukar hjälpa! Sammanfattningsvis kan jag ju ändå säga att arbetet varit roligt.

Att man sedan genom arbetet har lärt sig en hel del inom matematiken, dels naturligtvis inom fraktalgeometrin, men även t.ex. inom matrisalgebran kan man bara se som något positivt som man får med på köpet. Sedan har jag säkert genom detta arbete fått mera rutin i mitt programmeringsarbete allmänt. Att känna att man klarar att skriva ett större program gör att det inte känns lika stort att skriva något lika stort någon gång i framtiden.

Jag hoppas slutligen att fler än jag skall få tillfälle att använda detta program. Flera som kan använda det och komma med synpunkter på dess utformning. Detta är inget löfte, men vem vet, programmet kanske kommer i någon ny version någon gång i framtiden. Vem vet?

17 Referenser

Andersson, Kjell, Blomberg, Åke, Matriser och determinanter Studentlitteratur

Bergendal, Gunnar, Brinck, Inge (1968), Linjär algebra Studentlitteratur

Carlsson, Bo E. (1992), Fraktaler, bilder av kaos och kosmos

Mandelbrot, Benoit B (1983), The Fractal Geometry of Nature New York: W. H. Freeman and Company

Thompson, Jan (1991), Wahlström & Widstrands Matematiklexikon Wahlström & Widstrands förlag

Forskning och Framsteg 6/1989 Sidorna 40-43