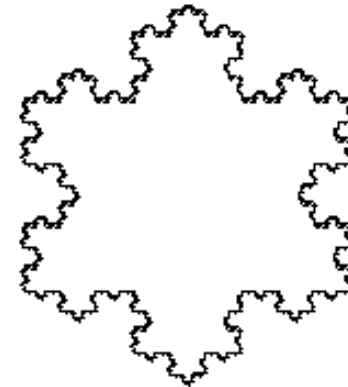
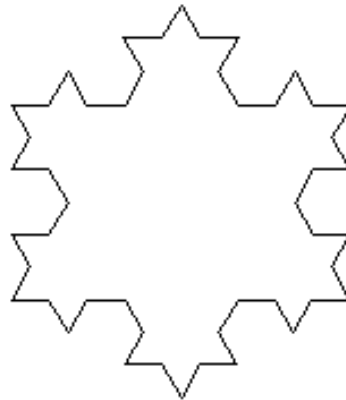
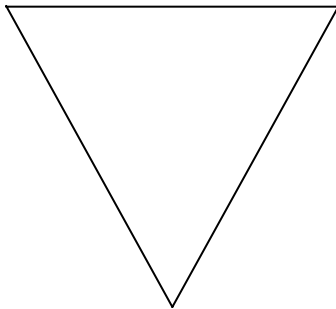
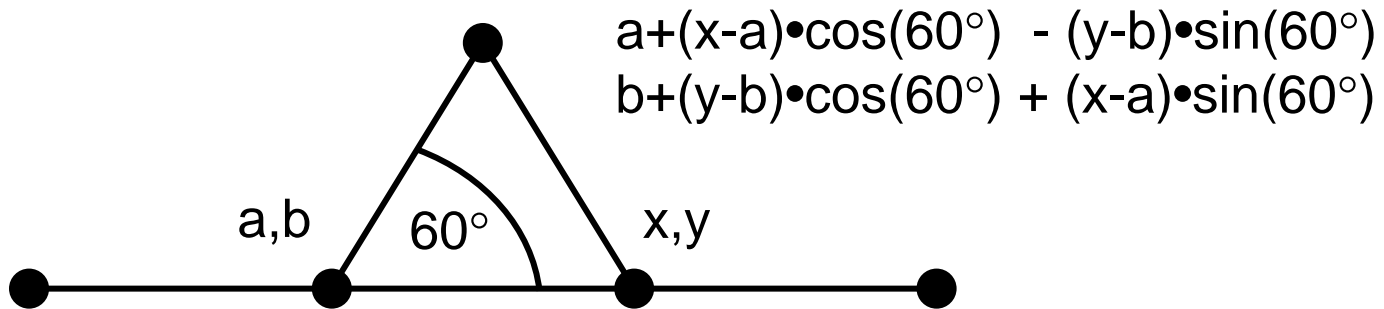


Kapitel 11: Fraktale

Selbstähnlichkeit



Koch'sche Schneeflocke



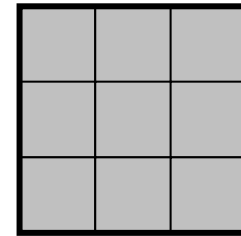
Fraktale Dimension

Ein selbstähnliches Objekt hat Dimension D , falls es in N identische Kopien zerfällt, skaliert mit dem Faktor $r = \frac{1}{N^{\frac{1}{D}}}$

Linie hat Dimension 1
N Teilstücke der Länge $1/N$



Fläche hat Dimension 2
N Teilflächen mit Kantenlänge $\frac{1}{N^{\frac{1}{2}}}$



Berechnung der Dimension

$$r = \frac{1}{N^{\frac{1}{D}}} \quad \Rightarrow \quad r \cdot N^{\frac{1}{D}} = 1$$

$$\Rightarrow N^{\frac{1}{D}} = \frac{1}{r} \quad \Rightarrow \quad \frac{1}{D} \cdot \log(N) = \log\left(\frac{1}{r}\right)$$

$$\Rightarrow D = \frac{\log(N)}{\log\left(\frac{1}{r}\right)}$$

Jede Kante der Koch'schen Schneeflocke zerfällt in $N=4$ Kopien, skaliert mit $r=1/3$.

$$D = \frac{\log(4)}{\log(3)} = 1.2618$$

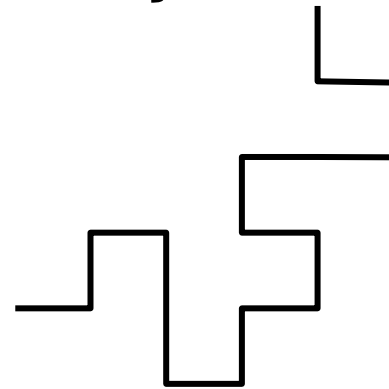
Lindenmayer-Systeme

Alphabet $\Sigma = \{r, u, l, d\}$

Regeln $f = \left\{ \begin{array}{l} r \rightarrow \text{rurddrur}, \\ u \rightarrow \text{ulurrulu}, \\ l \rightarrow \text{ldluuld}, \\ d \rightarrow \text{drdlldr} \end{array} \right\}$



ru

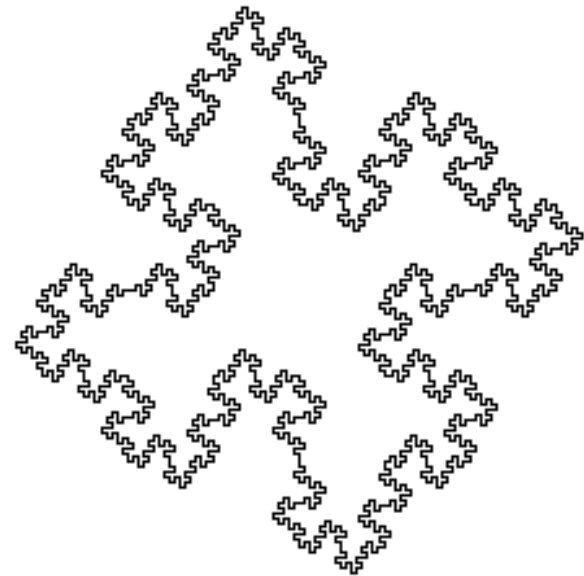
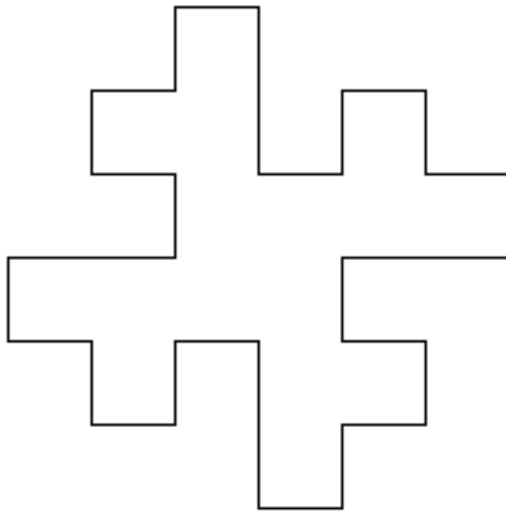


rurddrurulurrulu

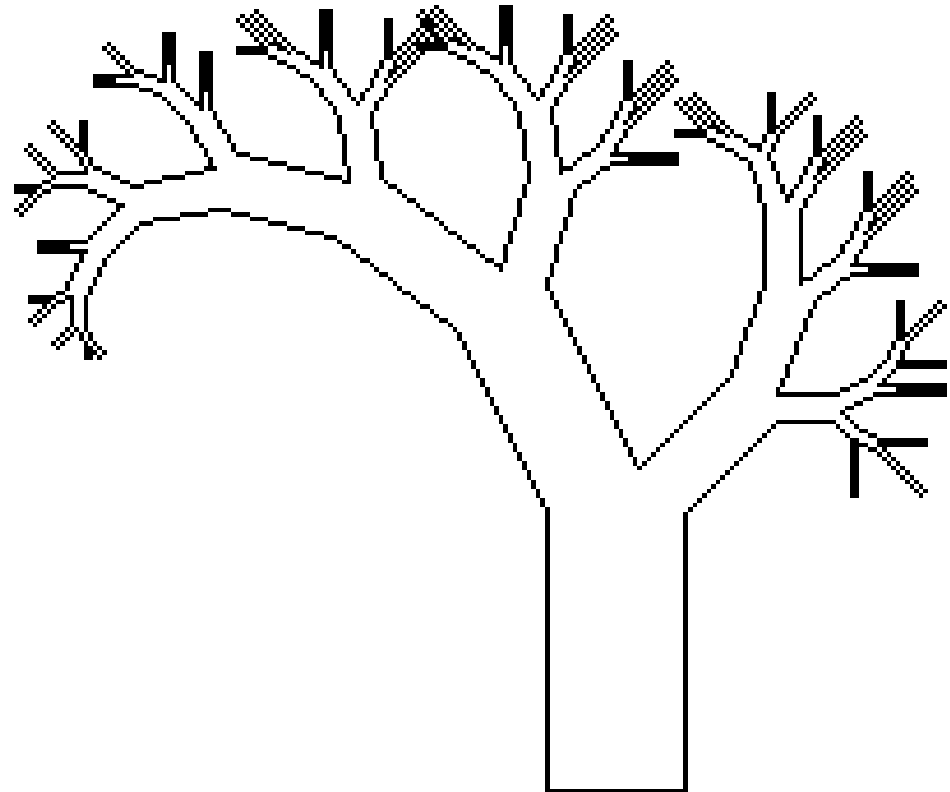
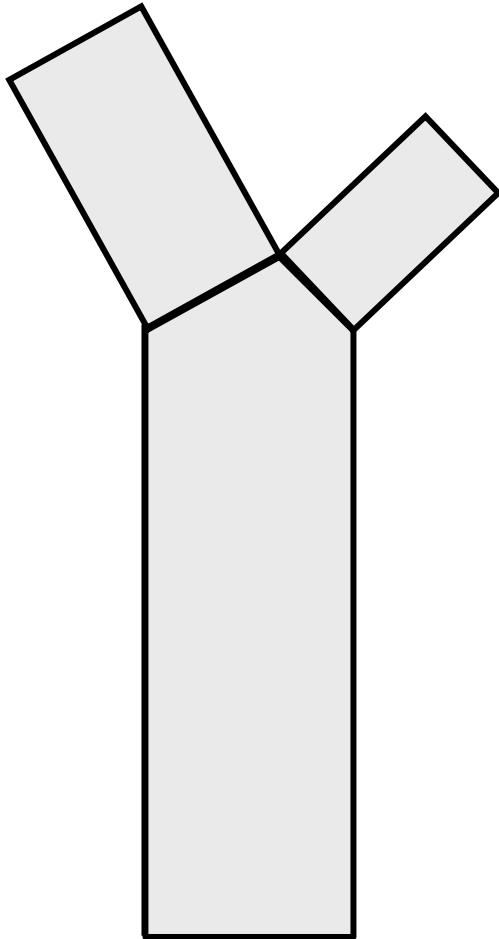
Quadratische Koch-Kurve

Jeder Kantenzug besteht aus 8 Kopien,
skaliert mit Faktor 1/4

$$D = \frac{\log(8)}{\log(4)} = 1.5$$



Bäume



Java-Applet zu Lindenmayer

[~cg/2006/skript/Applets/Fraktale2/App.html](#)

Komplexe Zahlen

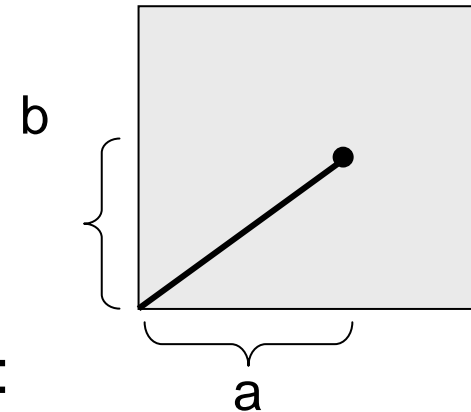
Komplexe Zahl $z = a + bi$

$$i^2 = -1$$

Realteil a

Imaginärteil b

Betrag von $z = \sqrt{a^2 + b^2}$



Quadrieren einer komplexen Zahl:

$$z^2 = (a + bi) \cdot (a + bi) = a^2 + 2abi + b^2i^2$$

\Rightarrow Realteil $a^2 - b^2$

\Rightarrow Imaginärteil $2ab$

Iteration

Sei c komplexe Zahl

Sei $f(z) := z^2 + c$

Betrachte $0, f(0), f^2(0), f^3(0), \dots$

| -1.5 | 0.2 | -0.5 | 0.2 |
|-------------|------------|-----------|----------|
| 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| -1.500000 | 0.200000 | -0.500000 | 0.200000 |
| 0.710000 | -0.400000 | -0.290000 | 0.000000 |
| -1.155900 | -0.368000 | -0.415900 | 0.200000 |
| -0.299319 | 1.050742 | -0.367027 | 0.033640 |
| -2.514467 | -0.429014 | -0.366422 | 0.175306 |
| 4.638493 | 2.357487 | -0.396466 | 0.071527 |
| 14.457877 | 22.070379 | -0.347930 | 0.143283 |
| -279.571438 | 638.381719 | -0.399474 | 0.100294 |

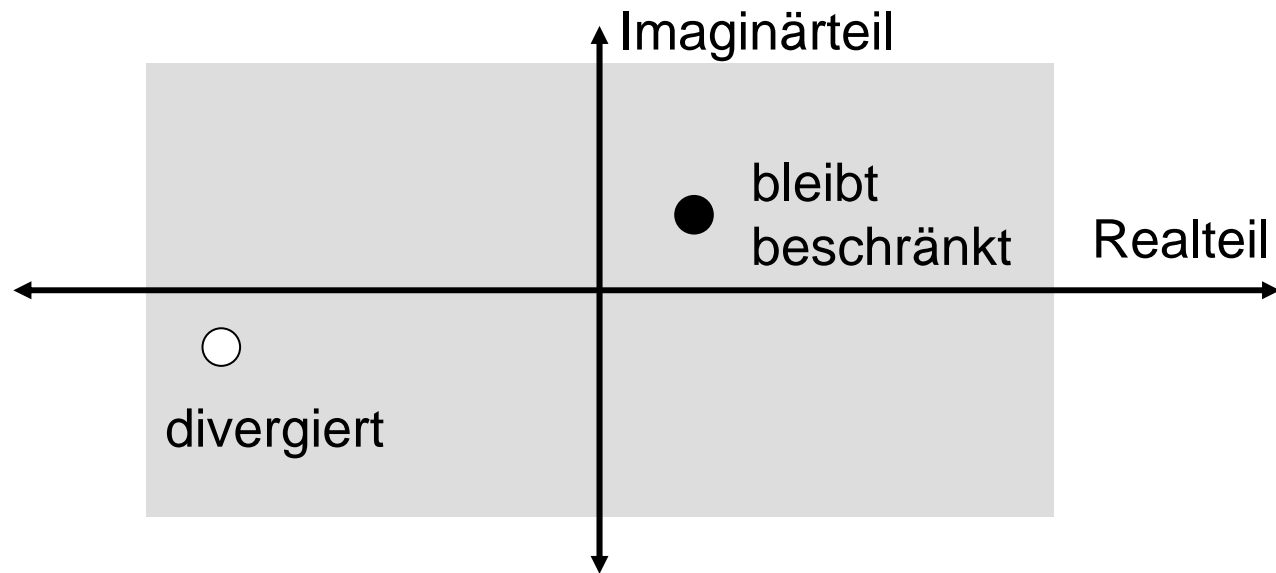
Mandelbrotmenge

Die zu c gehörende Folge kann ...

- konvergieren
- Zyklus durchlaufen
- sich chaotisch verhalten
- gegen unendlich streben

Die Menge der komplexen Zahlen c ,
die bei Startwert $z=0$
zu einer beschränkten Folge führen,
heißt Mandelbrotmenge.

Visualisierung der Mandelbrotmenge



Implementation

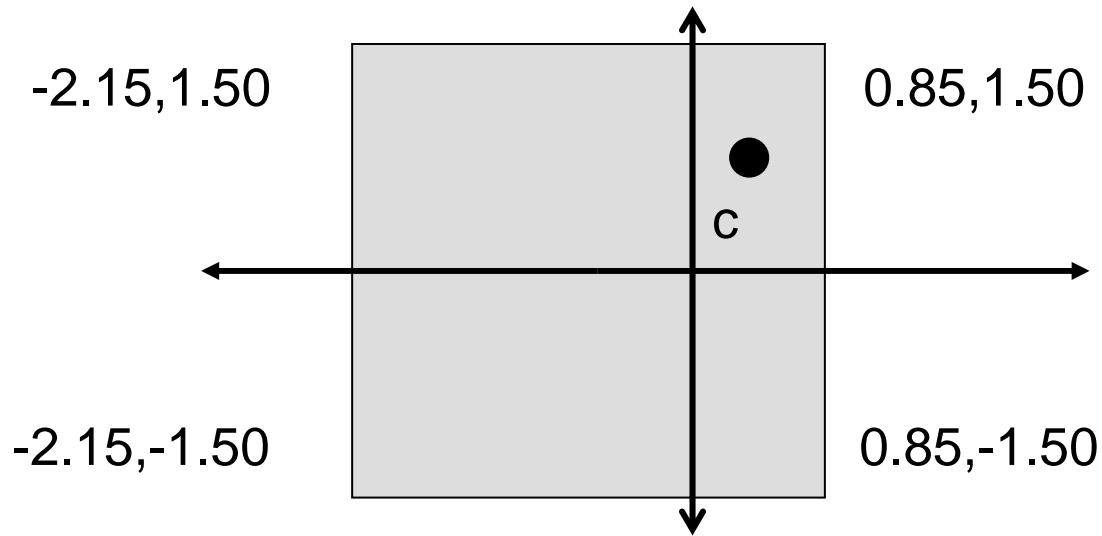
```
public Complex f(Complex z, Complex c){
    double re, im;
    re = z.re*z.re - z.im*z.im + c.re;
    im = 2*z.re*z.im+c.im;
    return new Complex(re,im);
}

int iter = 0;
Complex z = new Complex(0,0);
while (betrag(z) < 2 && iter < 100) {
    z = f(z,c);
    iter++;
}
```

iter<100: divergiert mit Sicherheit

iter>100: bleibt vermutlich beschränkt

Visualisierung



bei 300 Pixeln \Rightarrow Schrittweite $(0.85+2.15)/300 = 0.01$

```
public Complex (Point p, Complex start, double schritt){  
    this.re = start.re + schritt*p.x;  
    this.im = start.im - schritt*p.y;  
}
```

Mandelbrot-Programmcode

```
Point p;
double schritt = (ende.re-start.re)/WIDTH;

for (p.x=0, c.re=start.re; p.x < WIDTH;
     p.x++, c.re+=schritt)
for (p.y=0; c.im=start.im; p.y < HEIGHT;
     p.y++, c.im+=schritt){
    Complex z = new Complex(0,0);
    int iter=0;
    while((betrag(z)<2.0) && (iter<100)){
        z = f(z,c);
        iter++;
    }
    if (betrag(z) < 2.0) setPixel(p);
```


S/W-Mandelbrotmenge

$$\text{farbe}(c) := \begin{cases} \text{weiß, sobald Betrag} > 2 \\ \text{schwarz, falls Betrag} \\ \text{nach 100 Iterationen} < 2 \end{cases}$$

$$\text{farbe}(c) := \begin{cases} \text{iter}\%2, \text{ sobald Betrag} > 2 \\ \text{schwarz, falls Betrag} \\ \text{nach 100 Iterationen} < 2 \end{cases}$$

bei mehr Iterationen wird schwarze Fläche weiß

Farbige Mandelbrotmenge

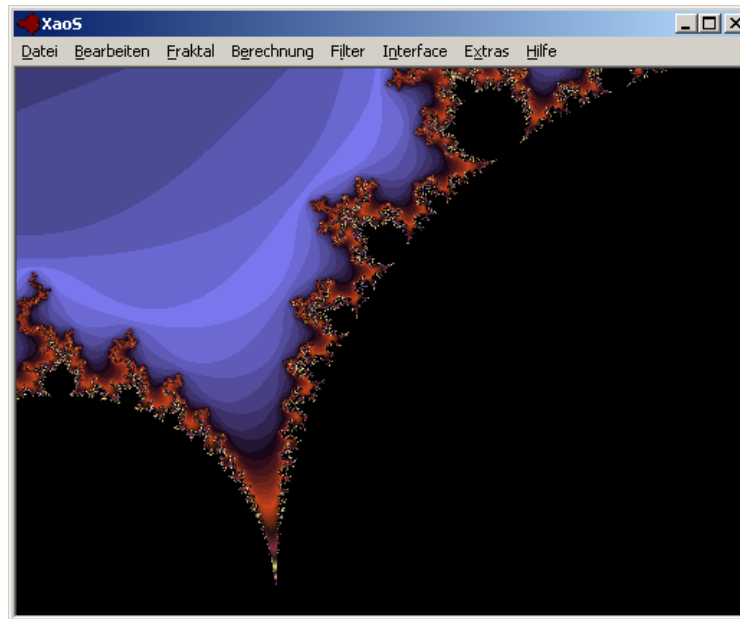
Es seien k Farben verfügbar:
 $col[0], col[1], \dots, col[k-1]$

$$farbe(c) := \begin{cases} col[iter \% k], & \text{sobald Betrag} > 2 \\ \text{schwarz, falls Betrag} & \text{nach 100 Iterationen} < 2 \end{cases}$$

bei mehr Iterationen wird schwarze Fläche bunt

Java-Applet zur Mandelbrotmenge

[~~cg/2006/skript/Applets/Fraktale2/App.html](http://cg/2006/skript/Applets/Fraktale2/App.html)



XaoS von Jan Hubicka

<http://xaos.sourceforge.net/index.php>

Invarianz bzgl. Funktion

Sei $f(z) = z^2$

Wähle Startwert z_0

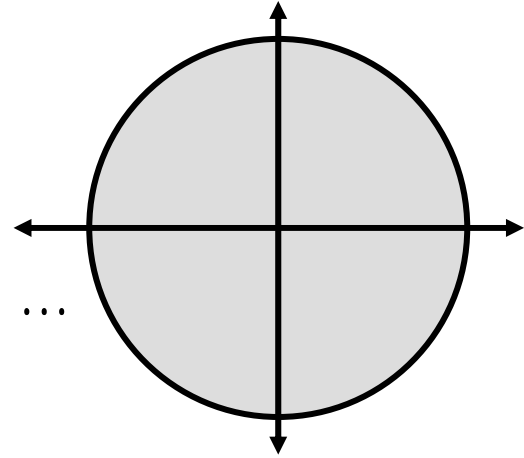
Betrachte $f(z_0), f(z_0)^2, f(z_0)^3, \dots$

$|z_0| > 1$ Sequenz divergiert

$|z_0| < 1$ Sequenz konvergiert gegen 0

$|z_0| = 1$ Sequenz bleibt auf Einheitskreis

Menge der Kreispunkte ist invariant bzgl. f



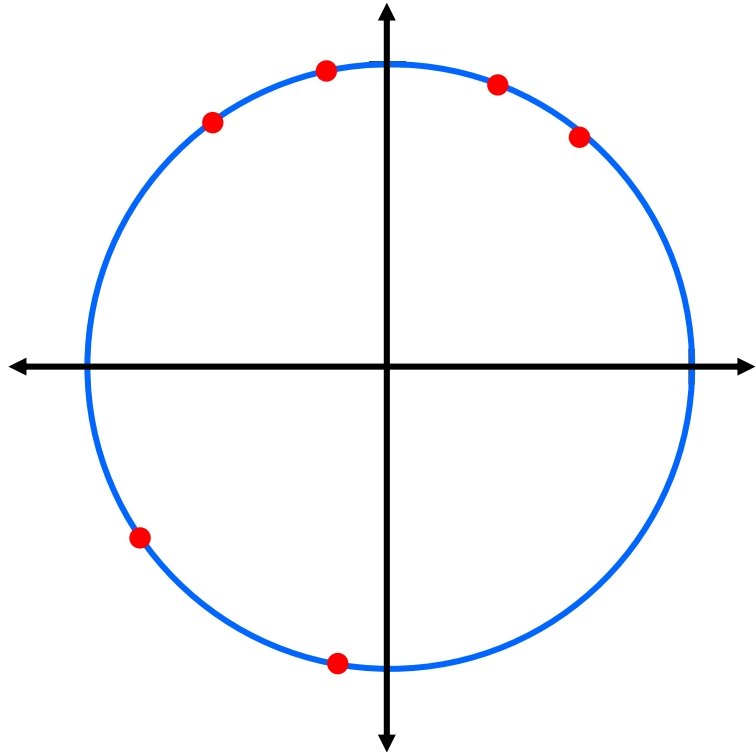
Julia-Menge

Sei c eine komplexe Zahl

Sei $f(z) = z^2 + c$

Die bzgl. f invarianten Punkte bilden die
Julia-Menge für c

Julia-Menge für $c=0$



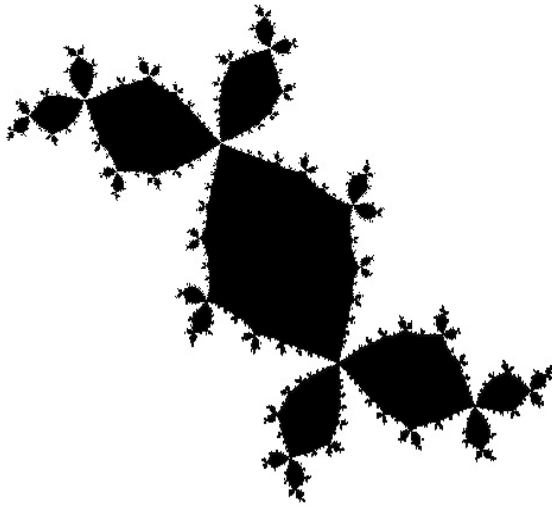
| | |
|----------|----------|
| 0.60000 | 0.80000 |
| -0.28000 | 0.96000 |
| -0.84320 | -0.53760 |
| 0.42197 | 0.90660 |
| -0.64387 | 0.76512 |
| -0.17084 | -0.98529 |
| ... | |
| ... | |

Julia-Menge, die 1.

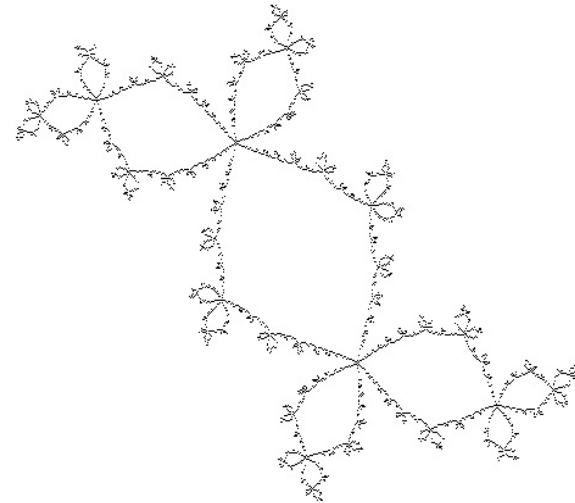
```
// berechne Gebiet fuer festes c
// fuer jeden Startpunkt teste auf Konvergenz
Point p;
double schritt = (ende.re-start.re)/WIDTH;
for (p.x=0; p.x < WIDTH; p.x++)
for (p.y=0; p.y < HEIGHT; p.y++){
    Complex z = new Complex(p,schritt);
    int iter=0;
    while((betrag(z)<2.0) && (iter<100)){
        z = f(z,c);
        iter++;
    }
    if (betrag(z) < 2.0) setPixel(p);
}
```

Der Rand des berechneten Gebiets ist die Julia-Menge

Beispiel für Julia-Menge



Konvergenzgebiet



Rand

Rückwärts statt vorwärts

$$r := r^2$$

$$r := \sqrt{r}$$

| | | |
|--------|--------|---------|
| 0.5000 | 1.0000 | 1.500 |
| 0.2500 | 1.0000 | 2.250 |
| 0.0625 | 1.0000 | 5.062 |
| 0.0039 | 1.0000 | 25.629 |
| 0.0001 | 1.0000 | 656.849 |
| ... | ... | ... |

| | |
|--------|--------|
| 0.8000 | 1.2000 |
| 0.8944 | 1.0954 |
| 0.9457 | 1.0466 |
| 0.9724 | 1.0230 |
| 0.9861 | 1.0114 |
| ... | ... |

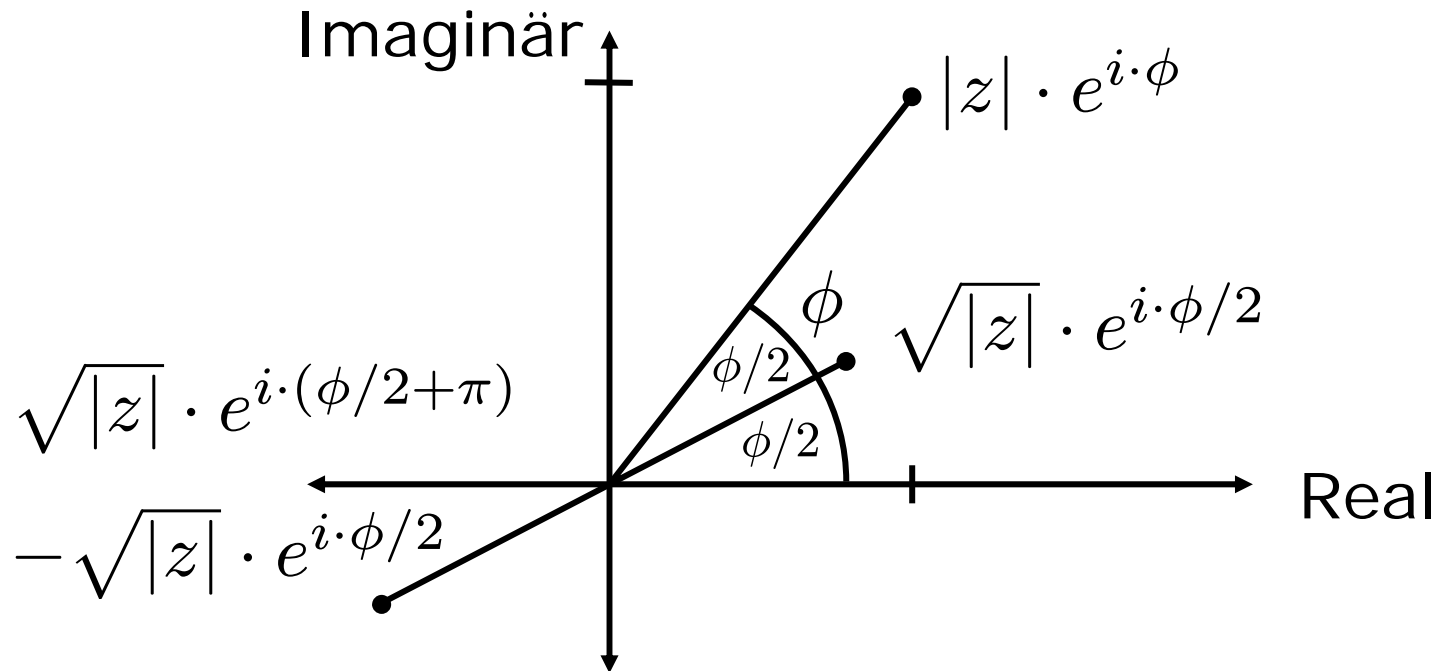
Julia-Menge, die 2.

$$f(z) = z^2 + c \Rightarrow f^{-1}(z) = \pm \sqrt{z - c}$$

$$\text{Sei } z = a + b \cdot i$$

$$\sqrt{z} = \begin{cases} \sqrt{\frac{|z|+a}{2}} + i \cdot \sqrt{\frac{|z|-a}{2}} & \text{falls } b \geq 0 \\ \sqrt{\frac{|z|+a}{2}} - i \cdot \sqrt{\frac{|z|-a}{2}} & \text{falls } b \leq 0 \end{cases}$$

Wurzel aus komplexer Zahl z



Julia-Menge, die 2.

```
Complex backward_random{Komplex z, Komplex c}{
    // berechne r = Wurzel aus z-c
    // wuerfel Vorzeichen fuer r
    return r
}

Complex z = new Complex(1.0,0.0);
for (k=0; k<50; k++) z = backward_random(z,c);

for (k=0; k<10000; k++){
    z = backward_random(z,c);
    p.x = (int)(z.re-z.start.re)/schritt;
    p.y = (int)(z.im-z.start.im)/schritt;
    setPixel(p);
}
```

Java-Applet zur Julia-Menge

[~cg/2006/skript/Applets/Fraktale2/App.html](#)

Iterierte Funktionensysteme

Beschreibe den Bildinhalt durch
System von affinen Transformationen

Pro Transformation:

2×2 Matrix A

2×1 Vektor b

Anwendungswahrscheinlichkeit w

Bilde $Ax+b$ mit Wahrscheinlichkeit w

Matritzen für Farn

| | A | b | w |
|-------|---|---|------|
| r_1 | $\begin{pmatrix} 0.85 & 0.04 \\ -0.04 & 0.85 \end{pmatrix}$ | $\begin{pmatrix} 0.0 \\ 1.6 \end{pmatrix}$ | 85 % |
| r_2 | $\begin{pmatrix} 0.20 & -0.26 \\ 0.23 & 0.22 \end{pmatrix}$ | $\begin{pmatrix} 0.0 \\ 1.6 \end{pmatrix}$ | 7 % |
| r_3 | $\begin{pmatrix} -0.15 & 0.28 \\ 0.26 & 0.24 \end{pmatrix}$ | $\begin{pmatrix} 0.0 \\ 0.44 \end{pmatrix}$ | 7 % |
| r_4 | $\begin{pmatrix} 0.00 & 0.00 \\ 0.00 & 0.16 \end{pmatrix}$ | $\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$ | 1 % |

Farn-Algorithmus

Wähle Startpixel p in Zeichenfläche

```
while(noch_nicht_zufrieden) {
```

```
    wähle Transformation  $f$  gemäß  $w$ 
```

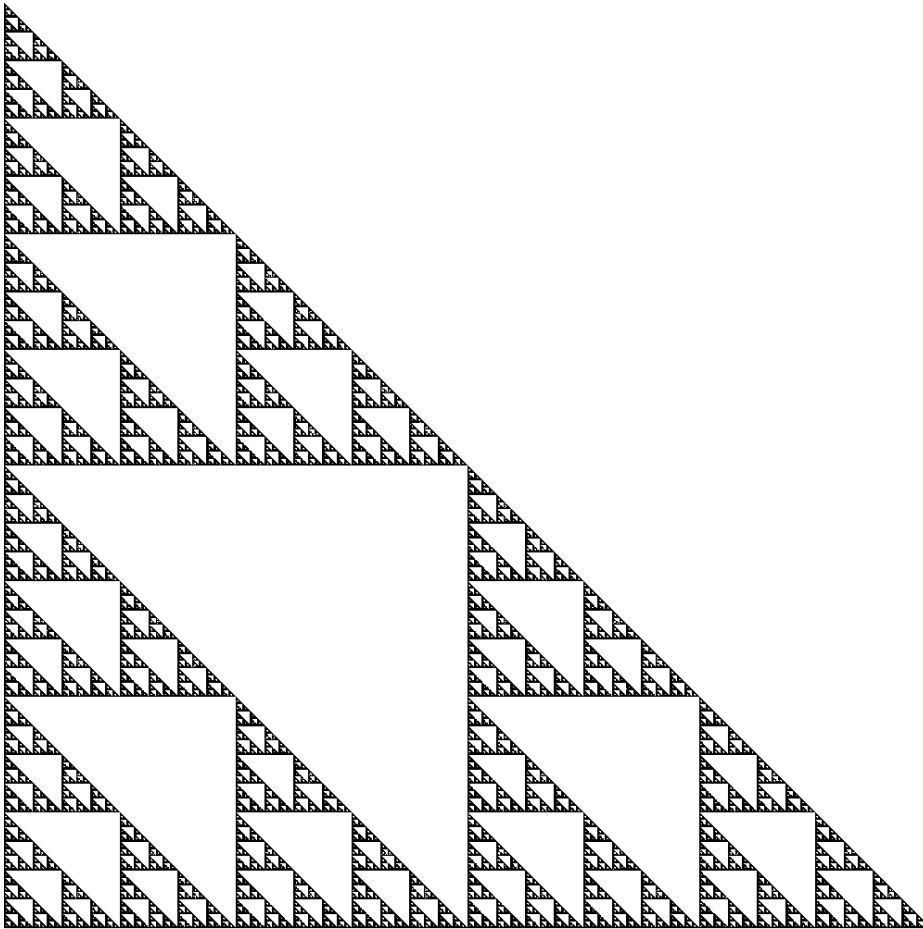
```
     $p = f(p);$ 
```

```
    setPixel( $p$ );
```

```
}
```



Sierpinsky-Dreieck

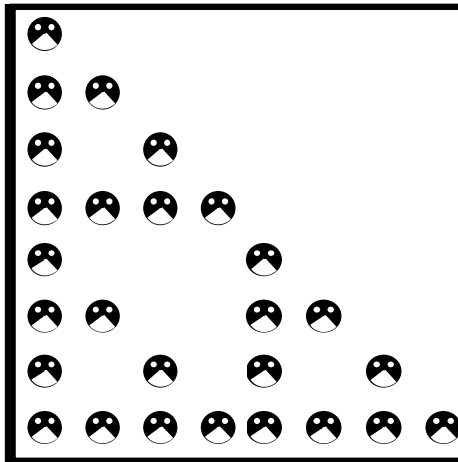


| | |
|-------|-------|
| D_1 | |
| D_2 | D_3 |

D enthält sich
3mal verkleinert.

Algorithmus für Sierpinsky

```
D := Rechteck mit zufälligem Inhalt
while (noch_nicht_zufrieden) do {
  D' := mit Faktor ¼ verkleinerte Version von D
  kopiere D' in den 2., 3., 4. Quadranten von D
}
```



Java-Applet zu Iterierten Funktionensystemen

[~~cg/2006/skript/Applets/IFS/fraktal.html](http://www.cg/2006/skript/Applets/IFS/fraktal.html)

Fraktale Kompression

fixiere 8 Transformationen

pro Ziel-Block suche bestes Urbild

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Identität

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

90° Drehung

$$\begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$$

180° Drehung

$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

270° Drehung

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Spiegelung
an x

$$\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$$

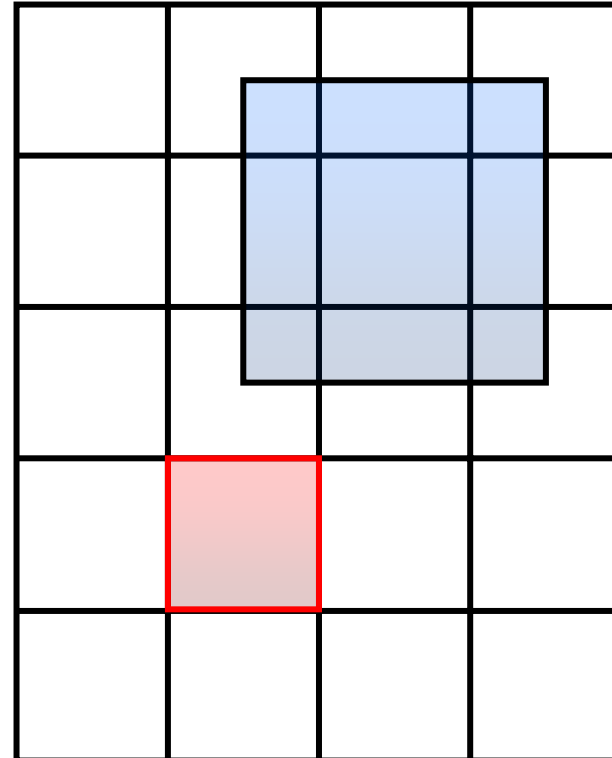
Spiegelung
an y

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Spiegelung
an Diagonalen

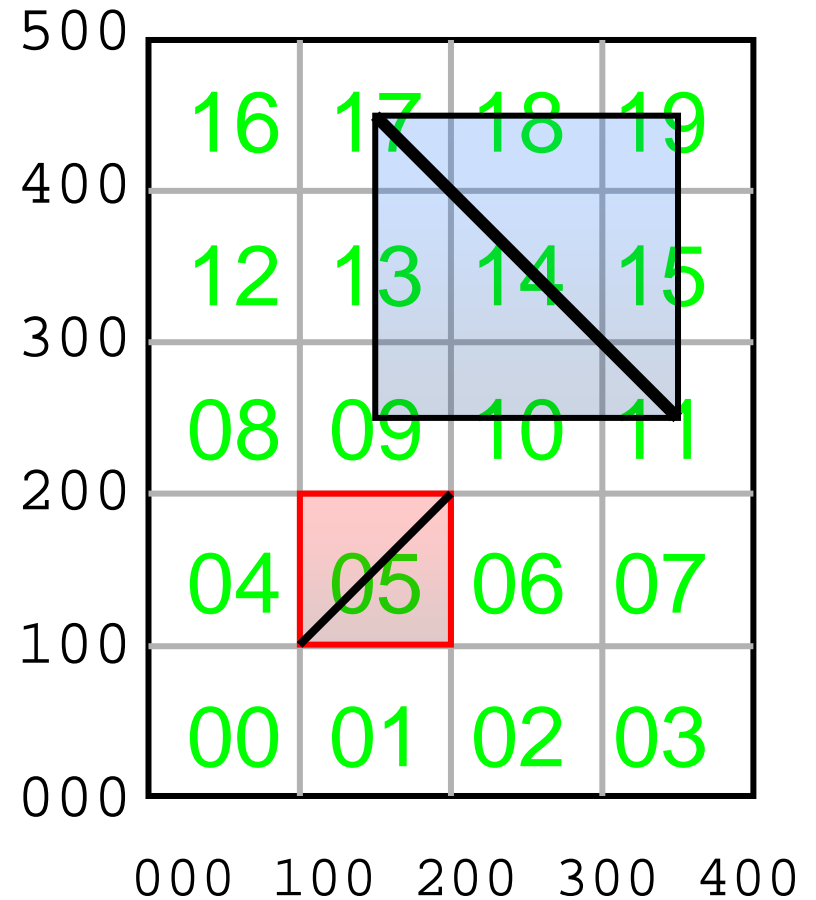
$$\begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}$$

Spiegelung an
anderer Diagonalen



Liste der Abbildungen

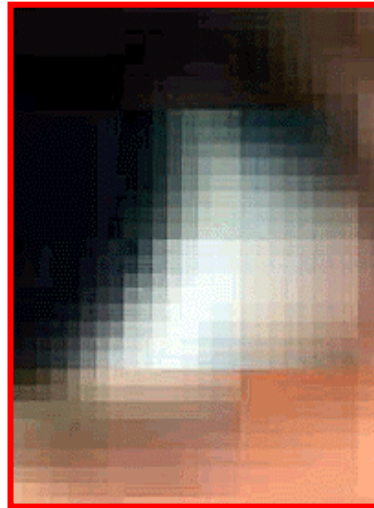
| Typ | Ursprung | Ziel |
|-----|----------|------|
| 1 | 150,250 | 5 |
| ... | | |



Iterated Systems



jan.tif



Ausschnitt aus
jan.jpg



Ausschnitt aus
jan.tif