

ChaosII:Die von Kochs Schneeflocke

Bettina Beier, Rebecca Binni und Miriam Krause

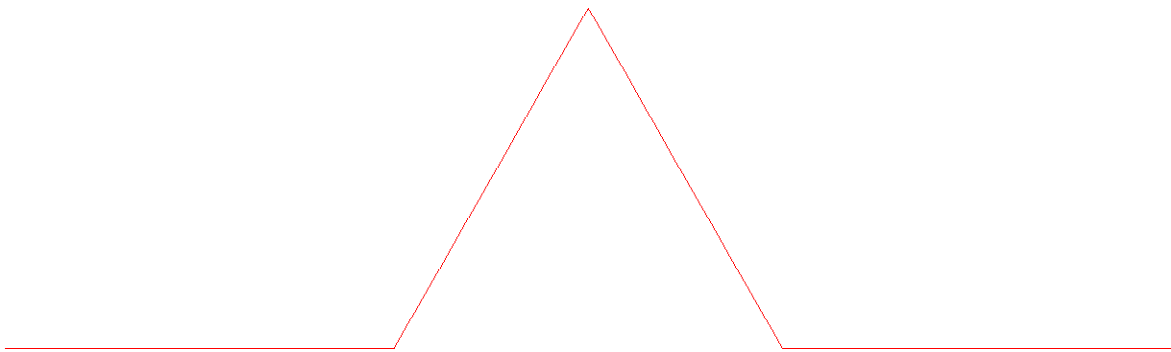
29.01.2003

```
> restart:
```

```
with(plots):
```

```
Warning, the name changecoords has been redefined
```

```
> plot([[0,0],[1/3,0],[1/2,sqrt(3)/6],[2/3,0],[1,0]],scaling=constant,axes=None);
```



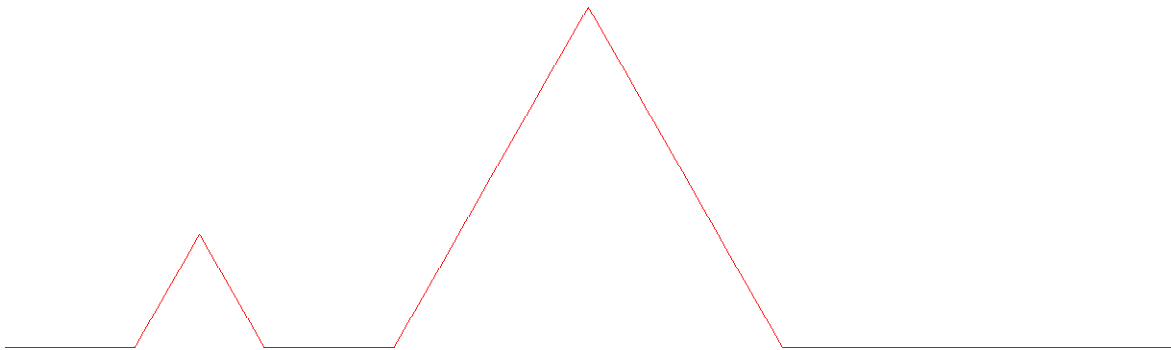
In diesem Beispiel zeichnet Maple einen Linienzug im Koordinatensystem, wobei wir fünf Punkte, deren Koordinaten wir aufgrund der Eigenschaft der Kochkurve berechnen haben, verbinden $(0/0)$, $(1/3/0)$, $(1/2/\sqrt{3}/6)$, $(2/3/0)$, $(1/0)$.

Das ist der erste Iterationsschritt. In jedem weiteren Schritt soll derselbe Vorgang auf die einzelnen Streckenteile angewandt werden.

Wie man im Folgenden sieht, benötigt man für den zweiten Schritt schon etwas kompliziertere

Berechnungen. Hier ein Beispiel für den zweiten Schritt auf einem Streckenteil.

```
> plot([[0,0],[1/9,0],[3/18,sqrt(3)/18],[2/9,0],[1/3,0],[1/2,sqrt(3)/6],[2/3,0],[1,0]],scaling=constrained,axes=None);
```



Man sieht, dass es relativ kompliziert wird, immer alle Zwischenpunkte von Hand auszurechnen. Aus diesem Grund haben wir versucht eine allgemeine Formel zu finden, die die Zwischenpunkte zu zwei gegebenen Randpunkten berechnet. Man schreibt eine Funktion, die ausgehend von zwei Punkten (Anfangs-, Endpunkt) eine Liste von fünf Punkten errechnet. Zu diesem Zweck muß man einmal von Hand fünf allgemeine Punkte berechnen, die der Kochkurve entsprechen. (x-, y-Komponente)

Die Strecken müssen in drei gleich große Teile gegliedert werden. Man will erreichen, dass man Maple zwei Anfangswerte eingibt und es sich somit automatisch um neue Punkte ausrechnet und diese verbindet (aus zwei werden fünf Punkte).

Die im Folgenden definierte Funktion `kochf` beinhaltet zwei Variablen und jeder der beiden Variablen ist als Punkt definiert. Die Funktion muss die beiden Punkte in eine Liste von neuen Zwischenpunkten übertragen, berechnet also zu zwei gegebenen Punkten die benötigten

Zwischenpunkte unterstellt eine Liste, die diese neuen Zwischenpunkte und die eingegebenen Randpunkte enthält.

```
> kochf := (A, B) -> [A, [(2/3)*A[1] + (1/3)*B[1], (2/3)*A[2] + (1/3)*B[2]],  
  [(1/2)*(B[1] + A[1]) + (sqrt(3)/6)*(A[2] - B[2]), (1/2)*(B[2] + A[2]) + (sqrt(3)/6)*(B[1] - A[1])],  
  [(1/3)*A[1] + (2/3)*B[1], (1/3)*A[2] + (2/3)*B[2]], B];
```

$$\text{kochf} := (A, B) \rightarrow \left[A, \left[\frac{2}{3}A_1 + \frac{1}{3}B_1, \frac{2}{3}A_2 + \frac{1}{3}B_2 \right], \right. \\ \left. \left[\frac{1}{2}B_1 + \frac{1}{2}A_1 + \frac{1}{6}\sqrt{3}(A_2 - B_2), \frac{1}{2}B_2 + \frac{1}{2}A_2 + \frac{1}{6}\sqrt{3}(B_1 - A_1) \right], \left[\frac{1}{3}A_1 + \frac{2}{3}B_1, \frac{1}{3}A_2 + \frac{2}{3}B_2 \right], B \right]$$

Will man mehrere Stufen der Kochkurve zeichnen, muss man zu je zwei berechneten benachbarten Zwischenpunkten wieder neue Zwischenpunkte berechnen.

Dazu benötigt man einen neuen Befehl, den "zip"-Befehl.

Dieser Befehl macht aus einer Liste zwei und greift dann immer parallel auf die Elemente zu.

Allerdings muß bei einer der Listen das erste (1) beider anderen das letzte (-1) Element entfernt werden, damit man sonst in den Ergebnislisten immer zwei gleiche Punkte stehen hat.

Beispiel:

In diesem Beispiel wird eine Liste *L* eingegeben. Aus dieser Liste wird eine neue Liste "Summe" berechnet, indem jeweils zwei aufeinanderfolgende Elemente aus *L* addiert werden.

```
> L := [1, 2, 3, 4, 5, 6];
```

```
L := [1, 2, 3, 4, 5, 6]
```

```
> Summe := zip((x, y) -> x + y, L[1..-2], L[2..-1]);
```

```
Summe := [3, 5, 7, 9, 11]
```

In unserem Fall ist die Funktion, die den "zip"-Befehl anwendet, die Kochfunktion. Also erzeugt sie Punktepaare aus denen Punktquintupel berechnet werden. Dader letzte (der fünfte) errechnete Punkt immer als Anfangspunkt der nächsten Teilkurve auftritt, kann man die Zahl der zu errechnenden Punkte auf vier einschränken. Also zu `kochf(x, y)` die Einschränkung `[1..4]`, die den fünften Punkt wegläßt, hinzufügen.

Um dies erhaltenen Punktquartupel wieder in eine "zip"-kompatible Form zu bringen, verwendet man den Befehl "op", der die erhaltenen Teillisten in einer Liste zusammenfaßt.

Damit die Punkte auch tatsächlich ausgerechnet werden, setzt man noch ein "evalf" davor.

Danach fügt man den bisher vernachlässigten letzten Punkt wieder hinzu. In diesem Fall sind Anfangs- und Endpunkt als $[0, 0]$ und $[0, 1]$ gewählt, sie können aber ohne Weiteres geändert werden.

Nachdem man nun eine Funktion zur Berechnung der Zwischenpunkte hat, muss man eine Prozedur schreiben, die mit der Angabe von zwei Punkten und einer Iterationsstufe (*i*) die gewünschten Schritte ausführt und somit die neuen Punkte automatisch berechnet. Anschließend soll man die einzelnen Punkte verbinden und somit die gewünschte Kurve zeichnen.

```
> Iteration:=proc(n,A,B)
  local i, punkte;
  punkte:=[A,B];
  for i from 1 to n do
    punkte:=evalf(zip((x,y)->op(kochf(x,y)[1..4]),punkte[1..-2],
    punkte[2..-1]));
    punkte:=[op(punkte),B];
  end do;
  plot(punkte,axes=none,scaling=constrained,color=blue);
end;
```

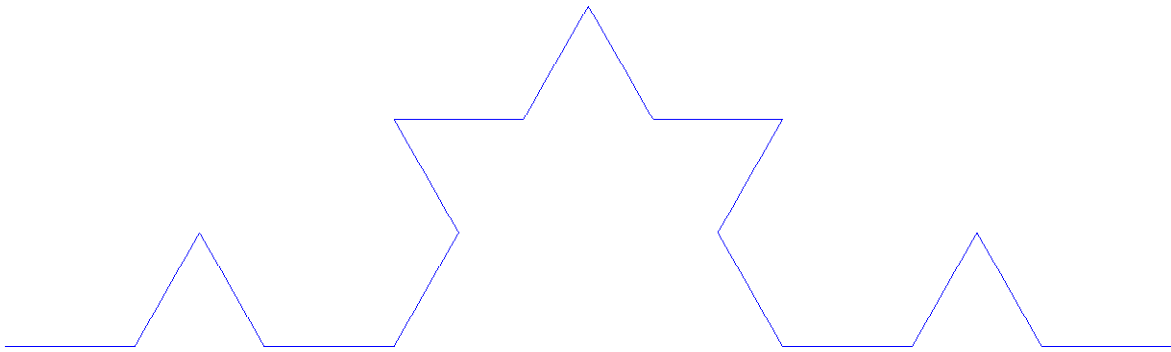
Die letzten Zeilen sind für die Ausgabe zuständig.

"plot" zeichnet die Kurve, wobei die Achsen des Koordinatensystems weggelassen, und die Maßstäbe von x- und y-Achse gleich gewählt werden.

In der Zeile "Iteration()" wird die gewünschte Iterationsstufe eingegeben.

Damit erhält man dann zum Beispiel für die zweite Iterationsstufe:

```
> Iteration(2,[0,0],[1,0]);
```



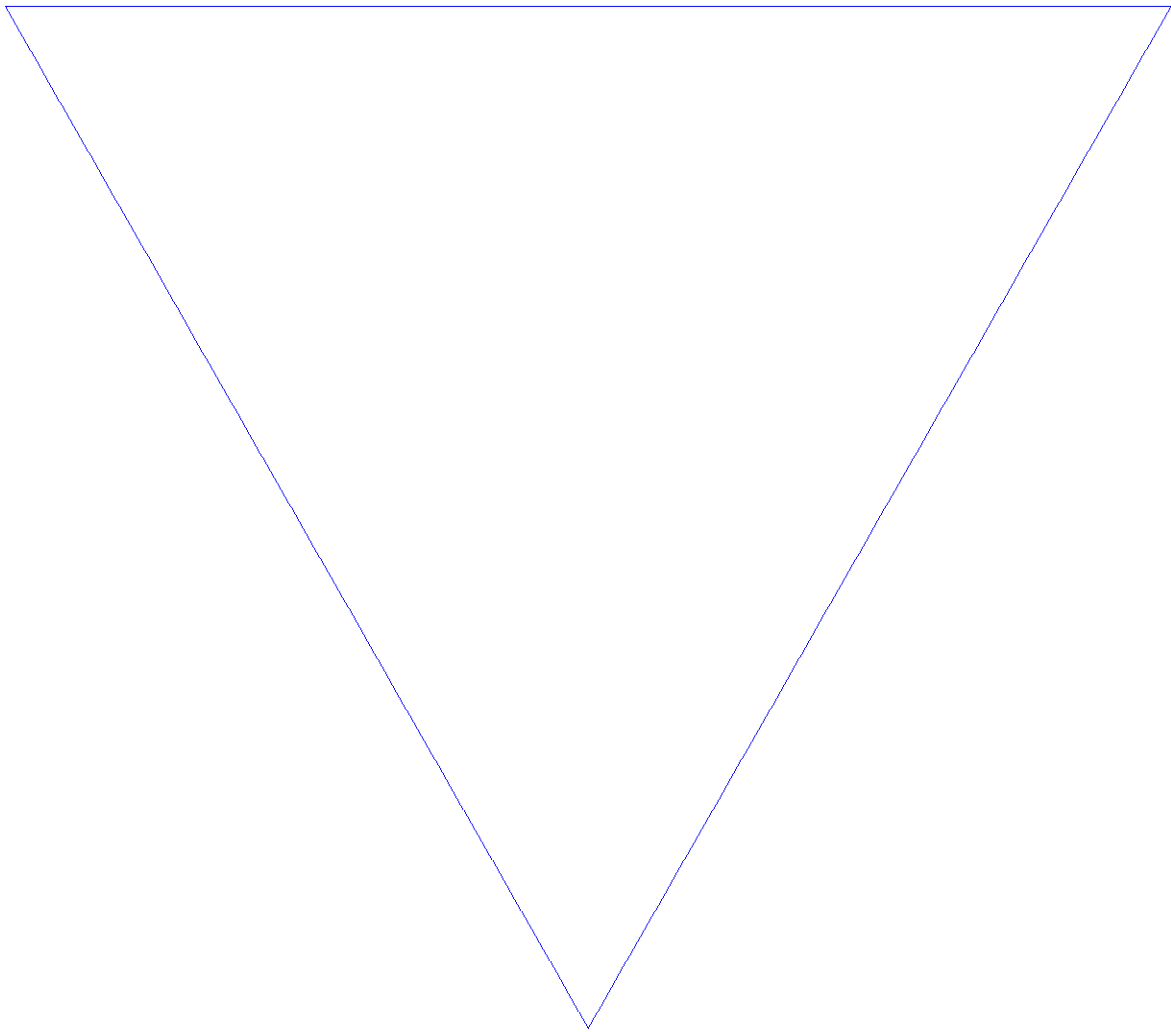
[DieerstenvierIterationstufenkannmanauchalsFilmdarstellen:

```
> P:=table():
  for i from 0 to 4 do
    P[i]:=display(Iteration(i,[0,0],[1,0]));

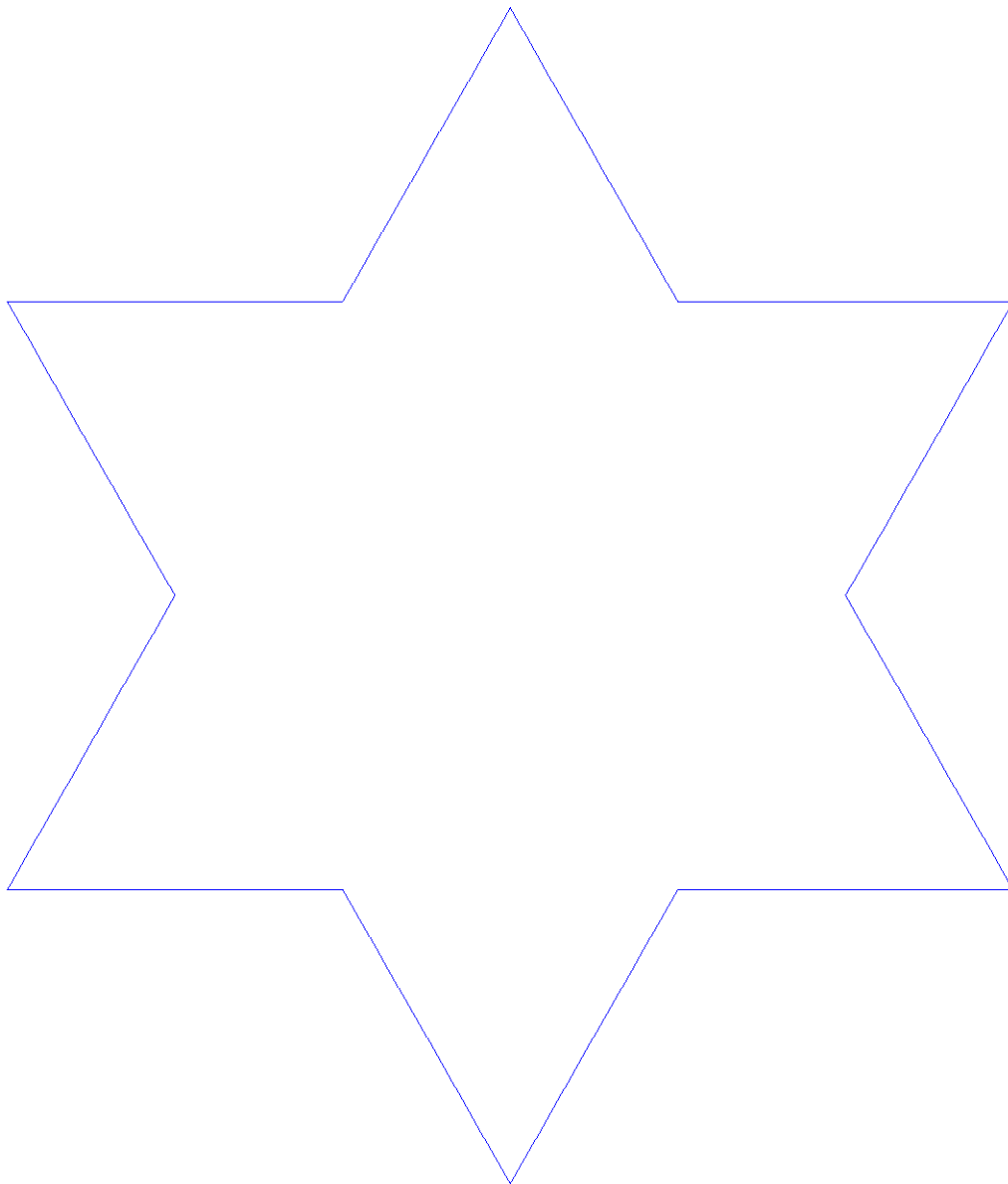
  end do:
plots[display]( convert(P,list), insequence=true);
```

Die gewünschte Schneeflocke erhält man nun, indem man drei Kurven aneinandersetzt. Dies erreicht man mit dem `display`-Befehl. Für die Iterationsstufe 0 ergibt sich ein Dreieck, ab der 1. Stufe erhält man dann die Schneeflocke.

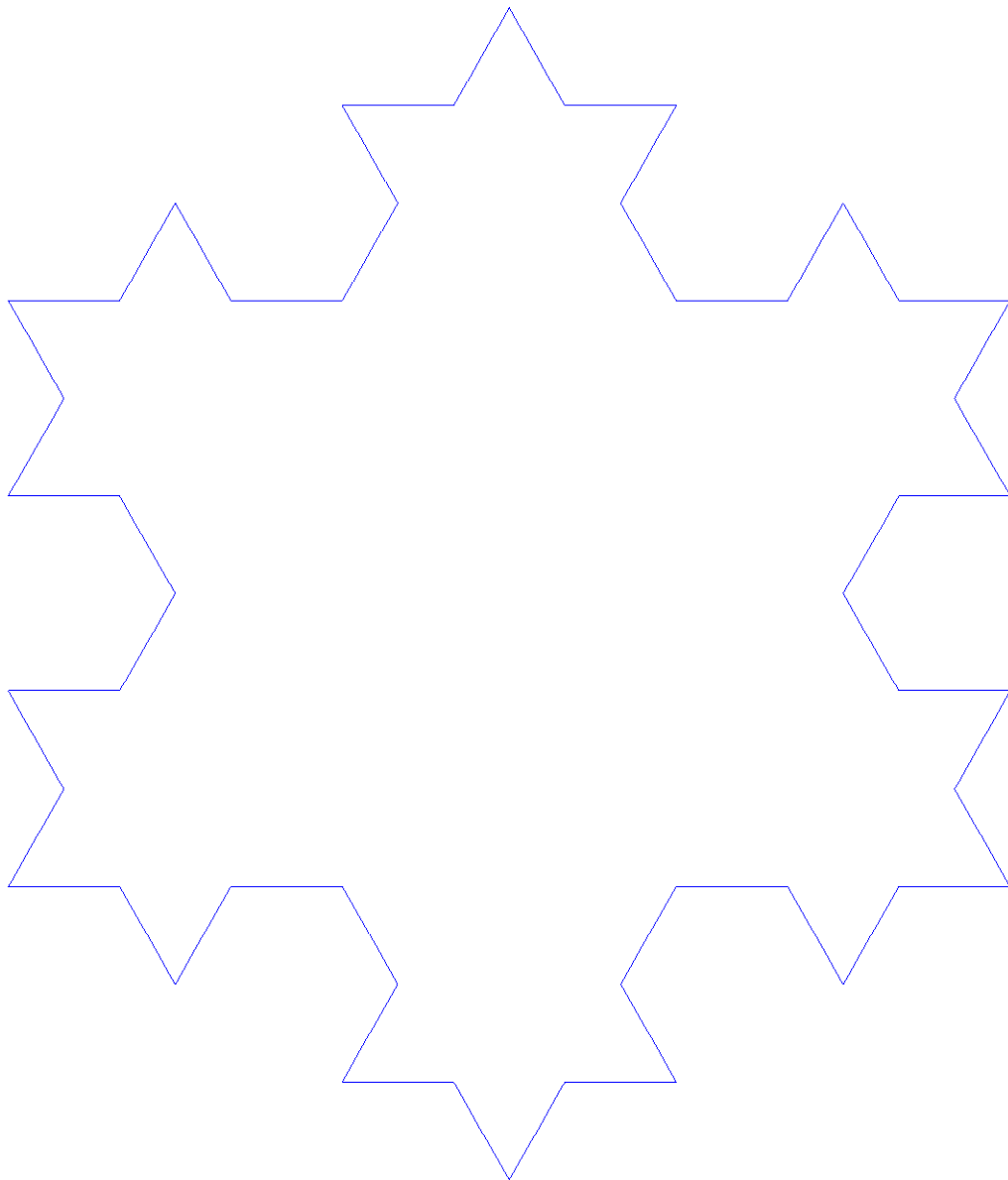
```
> display(Iteration(0,[0,0],[1,0]),Iteration(0,[1,0],[1/2,-sqrt(3)/2]),Iteration(0,[1/2,-sqrt(3)/2],[0,0]));
```



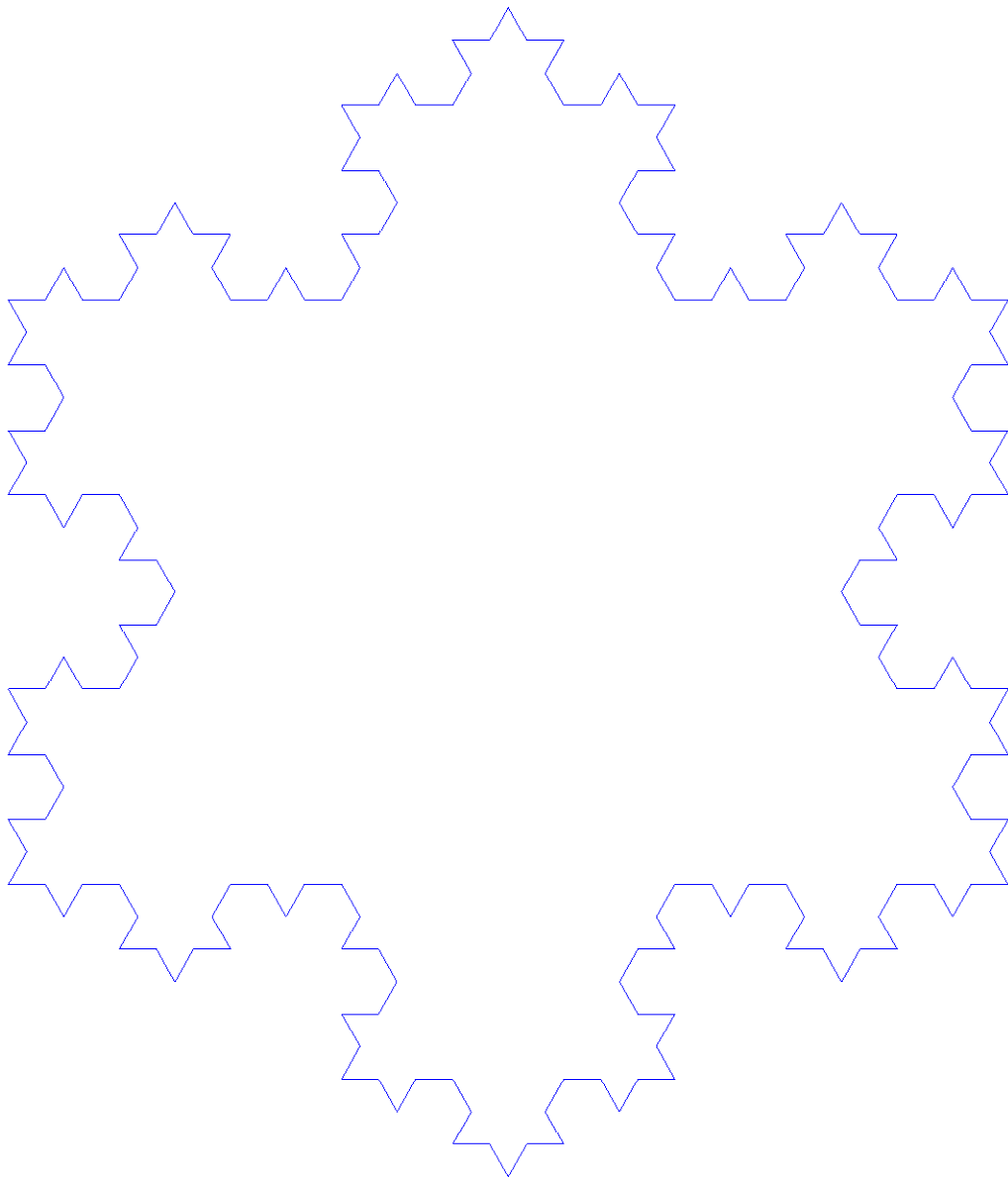
```
> display(Iteration(1,[0,0],[1,0]),Iteration(1,[1,0],[1/2,-sqrt(3)/2]),Iteration(1,[1/2,-sqrt(3)/2],[0,0]));
```



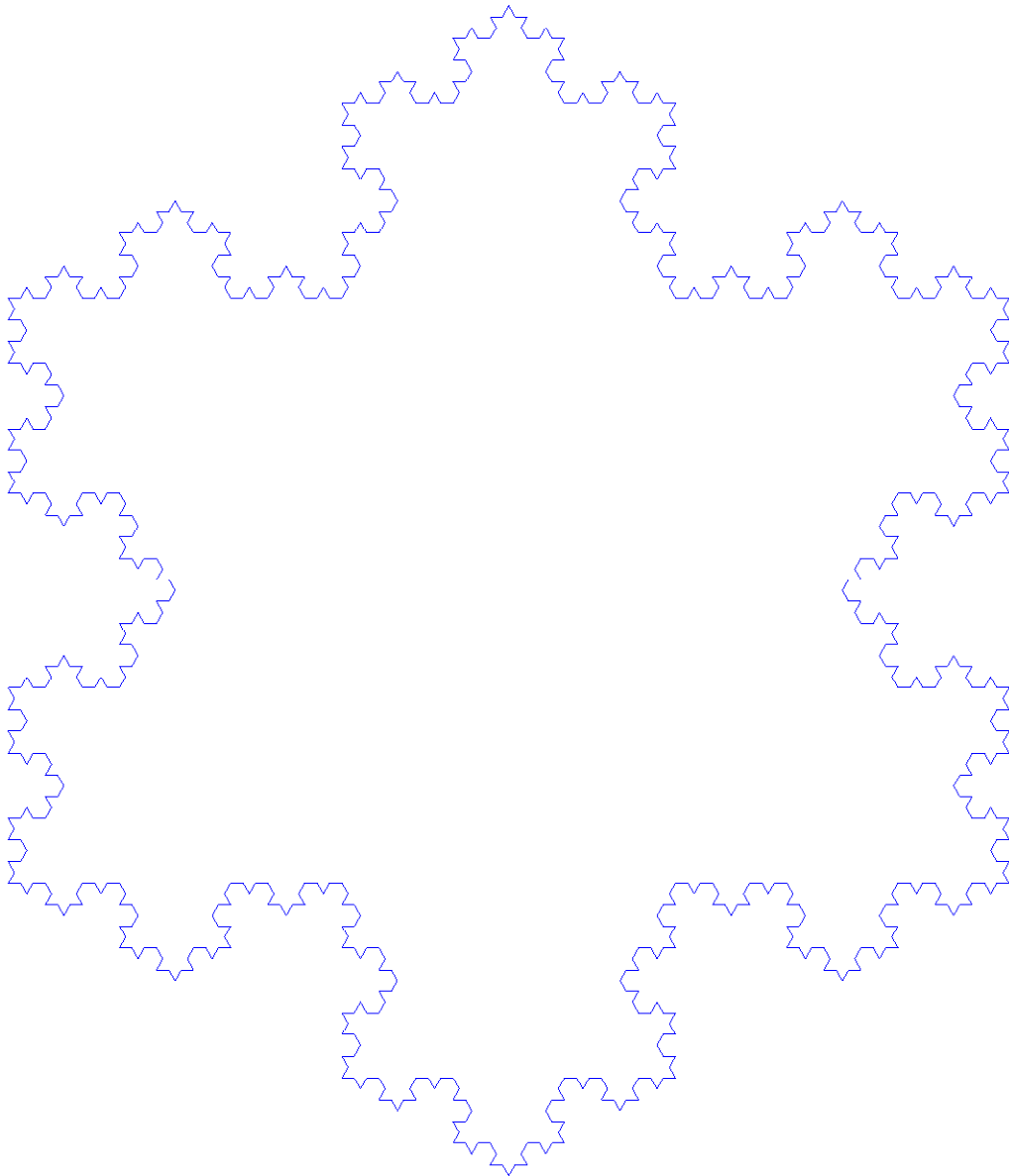
```
> display(Iteration(2,[0,0],[1,0]),Iteration(2,[1,0],[1/2,-sqrt(3)/2]),Iteration(2,[1/2,-sqrt(3)/2],[0,0]));
```



```
> display(Iteration(3,[0,0],[1,0]),Iteration(3,[1,0],[1/2,-sqrt(3)/2]),Iteration(3,[1/2,-sqrt(3)/2],[0,0]));
```



```
> display(Iteration(4,[0,0],[1,0]),Iteration(4,[1,0],[1/2,-sqrt(3)/2]),Iteration(4,[1/2,-sqrt(3)/2],[0,0]));
```



[Dieslässt sich auch als Filmdarstellen:

```
> P:=table():
  for i from 0 to 5 do

    P[i]:=display(Iteration(i,[0,0],[1,0]),Iteration(i,[1,0],[1/2,-s
    qrt(3)/2]),Iteration(i,[1/2,-sqrt(3)/2],[0,0]));

  end do:
  plots[display]( convert(P,list), insequence=true);
```

[Einigeweitere Beispiele:

```
> P:=table():
  for i from 0 to 4 do

    P[i]:=display(Iteration(i,[0,0],[1,0]),Iteration(i,[1,0],[1/2,sq
    rt(3)/2]),Iteration(i,[1/2,sqrt(3)/2],[0,0]));
```

```

end do:
plots[display]( convert(P,list), insequence=true);
> P:=table():
for i from 0 to 4 do

P[i]:=display(Iteration(i,[0,0],[1,0]),Iteration(i,[1,0],[1,-1])
,Iteration(i,[1,-1],[0,-1]),Iteration(i,[0,-1],[0,0]));

end do:
plots[display]( convert(P,list), insequence=true);
> P:=table():
for i from 0 to 4 do

P[i]:=display(Iteration(i,[0,0],[1,0]),Iteration(i,[1,0],[1,1]),
Iteration(i,[1,1],[0,1]),Iteration(i,[0,1],[0,0]));

end do:
plots[display]( convert(P,list), insequence=true);
> P:=table():
for i from 0 to 4 do

P[i]:=display(Iteration(i,[0,0],[1,0]),Iteration(i,[1,0],[2,-1])
,Iteration(i,[2,-1],[1,-2]),Iteration(i,[1,-2],[0,-2]),Iteration
(i,[0,-2],[-1,-1]),Iteration(i,[-1,-1],[0,0]));

end do:
plots[display]( convert(P,list), insequence=true);
> P:=table():
for i from 0 to 4 do

P[i]:=display(Iteration(i,[0,0],[1,0]),Iteration(i,[1,0],[2,-1])
,Iteration(i,[2,-1],[2,-2]),Iteration(i,[2,-2],[1,-3]),Iteration
(i,[1,-3],[0,-3]),Iteration(i,[0,-3],[-1,-2]),Iteration(i,[-1,-2]
,[-1,-1]),Iteration(i,[-1,-1],[0,0]));

end do:
plots[display]( convert(P,list), insequence=true);
> restart:
with(plots):
Warning, the name changecoords has been redefined
> kochf:=(A,B)->[A,[(2/3)*A[1]+(1/3)*B[1],(2/3)*A[2]+(1/3)*B[2]],
[(2/3)*A[1]+(1/3)*B[1],(1/2)*(B[2]+A[2])+(sqrt(3)/6)*(B[1]-A[1])
],[ (1/3)*A[1]+(2/3)*B[1],(1/2)*(B[2]+A[2])+(sqrt(3)/6)*(B[1]-A[1]
)], [(1/3)*A[1]+(2/3)*B[1],(1/3)*A[2]+(2/3)*B[2]],B];

```

$$kochf := (A, B) \rightarrow \left[A, \left[\frac{2}{3}A_1 + \frac{1}{3}B_1, \frac{2}{3}A_2 + \frac{1}{3}B_2 \right], \left[\frac{2}{3}A_1 + \frac{1}{3}B_1, \frac{1}{2}B_2 + \frac{1}{2}A_2 + \frac{1}{6}\sqrt{3}(B_1 - A_1) \right], \right. \\ \left. \left[\frac{1}{3}A_1 + \frac{2}{3}B_1, \frac{1}{2}B_2 + \frac{1}{2}A_2 + \frac{1}{6}\sqrt{3}(B_1 - A_1) \right], \left[\frac{1}{3}A_1 + \frac{2}{3}B_1, \frac{1}{3}A_2 + \frac{2}{3}B_2 \right], B \right]$$

```

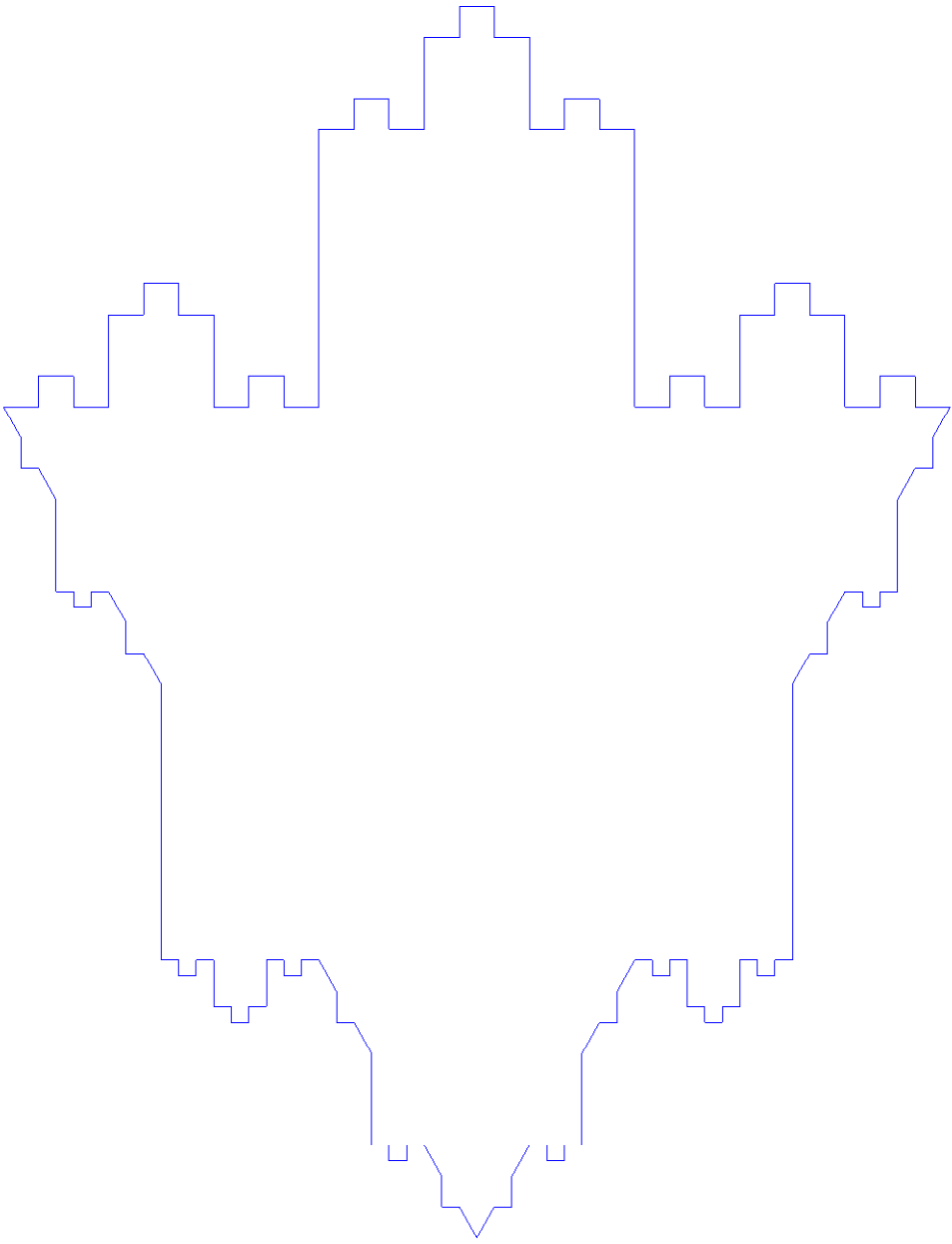
> Iteration:=proc(n,A,B)
  local i, punkte;
  punkte:=[A,B];
  for i from 1 to n do
    punkte:=evalf(zip((x,y)->op(kochf(x,y)[1..5]),punkte[1..-2],
    punkte[2..-1]));
    punkte:=[op(punkte),B];
  od;
  plot(punkte,axes=none,scaling=constrained,color=blue);
end:

> P:=table():
  for i from 0 to 4 do
    P[i]:=display(Iteration(i,[0,0],[1,0]));
  end do:
plots[display]( convert(P,list), insequence=true);

> P:=table():
  for i from 0 to 4 do
    P[i]:=display(Iteration(i,[1,0],[2,1]));
  end do:
plots[display]( convert(P,list), insequence=true);

> display(Iteration(3,[0,0],[1,0]),Iteration(3,[1,0],[1/2,-sqrt(3)
/2]),Iteration(3,[1/2,-sqrt(3)/2],[0,0]));

```



[>