Computergrafik mit DERIVE

Dr. Maria Koth, Universität Wien

Inhalt

0. Einleitung

Das Computeralgebraprogramm DERIVE verfügt nicht nur über hervorragende numerische, symbolische und graphische Fähigkeiten, sondern kann darüber hinaus auch als einfach erlernbare Programmiersprache für den Mathematikunterricht angesehen werden. Insbesondere kann man mit Hilfe der Programmierbefehle von DERIVE relativ einfach reizvolle Grafiken erzeugen. Im folgenden werden Beispiele für solche Computergrafiken vorgestellt.

Grundsätzlich sollten für alle betrachteten Beispiele die folgenden Menüvoreinstellungen gewählt werden:

OPTIONS PRECISION APPROXIMATE
In jedem der folgenden Programme wird eine große Zahl von Eckpunktskoordinaten von darzustellenden Punkte berechnet. Um Rechenzeit und Speicherplatz zu sparen, ist es daher unbedingt nötig, den APPROXIMATE-Modus einzustellen.

OPTIONS STATE CONNECTED
Damit die Punkte auf dem Bildschirm durch Strecken verbunden werden, muß im Graphikmenü beim Menüpunkt OPTIONS STATE der Modus CONNECTED eingestellt werden.

AXES NO
DERIVE ist so voreingestellt, daß die Koordinatenachsen auf dem Bildschirm gezeichnet werden. Da diese beim Betrachten der Bilder eher störend sind, sollte man sie mit Hilfe des Graphikmenüpunkts AXES NO ausblenden.

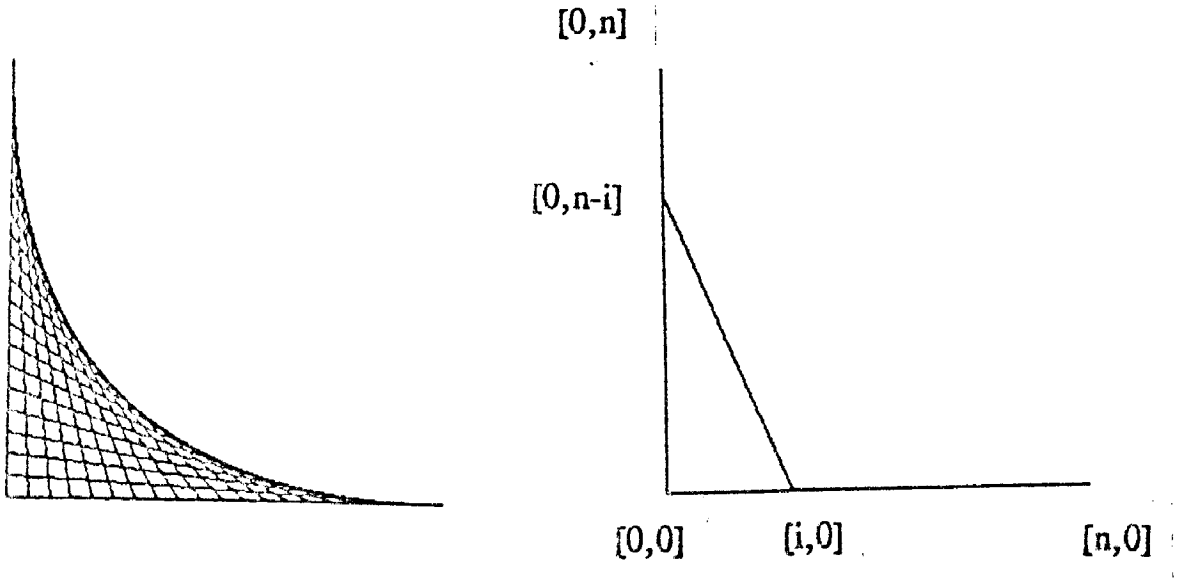
OPTIONS COLOR AUTO NO
DERIVE ist so voreingestellt, daß Graphiken, die nacheinander dargestellt werden, in verschiedenen Farben erscheinen. Die meisten der hier vorgestellten Bilder sind aus mehreren Streckenzügen zusammengesetzt. Will man ein solches Bild in einer einzigen Farbe zeichnen, so muß man den automatischen Farbwechsel durch Anwählen des Graphikmenübefehls OPTIONS COLOR AUTO NO abschalten.

Zu beachten ist, daß bei manchen Programmbeispielen neue Befehle der DERIVE-Version 3 verwendet werden, und diese daher unter DERIVE2.58 nicht lauffähig sind.

1. Einfache Liniengrafiken

1.1 Liniengrafiken über einem rechten Winkel

Gegeben sind zwei gleich lange Strecken mit gemeinsamem Anfangspunkt, die aufeinander normal stehen. Jede der beiden Strecken wird in n gleich lange Teilintervalle zerlegt. Der jeweils i-te Teilungspunkt der ersten Strecke wird mit dem (n-i)-ten Teilungspunkt der zweiten verbunden. Dabei entsteht die unten dargestellte Liniengrafik.



Will man diese Grafik durch ein DERIVE-Programm beschreiben, so kann man als Endpunkte der beiden Strecken der Einfachkeit halber die Punkte [0,0], [n,0] sowie [0,n] wählen. Das fertige Bild kann nämlich bequem mit Hilfe der Menübefehle CENTER und SCALE so verschoben bzw. gestaucht werden, daß es am Bildschirm Platz hat. Die i-te zu zeichnende Strecke wird in diesem Fall durch den Vektor [[i,0],[0,n- i]] beschrieben. Läßt man i die Werte 0 bis n durchlaufen, so wird die obige Grafik erzeugt. Das zugehörige DERIVE-Programm besteht nur aus einer einzigen Zeile:

GRAFIK1(n) := VECTOR([[i,0],[0,n-i]], i, 0, n)

Die Anzahl der zu zeichnenden Strecken wird beim Funktionsaufruf gewählt. Tippt man, zum Beispiel, GRAFIK1(20) ein, und vereinfacht man diesen Ausdruck mit dem Menübefehl SIMPLIFY, so wird eine Liste mit Koordinaten für die entsprechende Grafik mit 21 Linien auf dem Bildschirm ausgegeben. Markieren dieser Liste und zweimaliges Anwählen des Menübefehls PLOT bewirkt, daß die Grafik am Bildschirm gezeichnet wird. Mit Hilfe der Menübefehle SCALE und CENTER kann man die Zeichnung so verkleinern und verschieben, daß sie in passender Größe in der Mitte des Bildschirms plaziert ist.

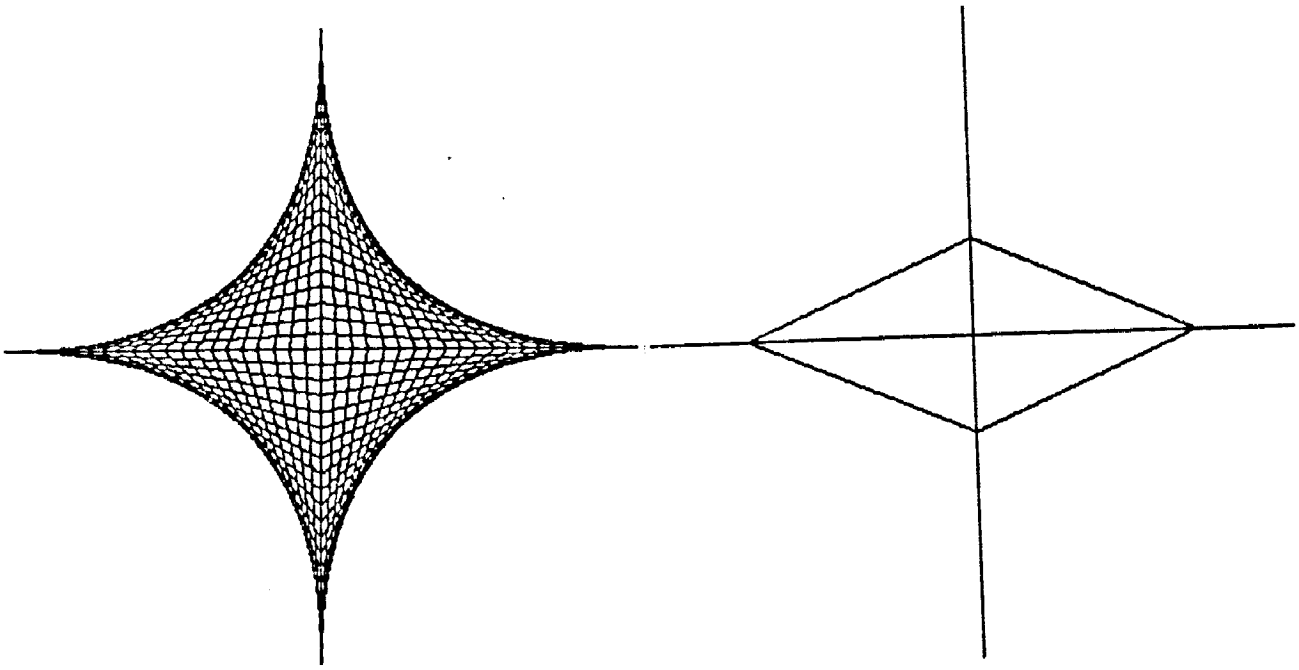
Dieses einfache Beispiel zeigt deutlich die guten Möglichkeiten der DERIVE-Programmierung. Um diese Liniengrafik in der Programmiersprache BASIC oder PASCAL zu beschreiben, wäre ein mehrere Zeilen umfassendes Programm erforderlich. Bereits im Programm müßten die Koordinaten der zu zeichnenden Punkte unter Berücksichtigung der verwendeten Bildschirmauflösung so festgelegt werden, daß die Grafik innerhalb des Bildschirms Platz hat. In DERIVE dagegen reicht ein einziger einfacher Befehl aus, um dieses Bild zu erzeugen. Lage, Größe und Farbe der Darstellung können bequem mit Hilfe der Menübefehle im nachhinein verändert werden.

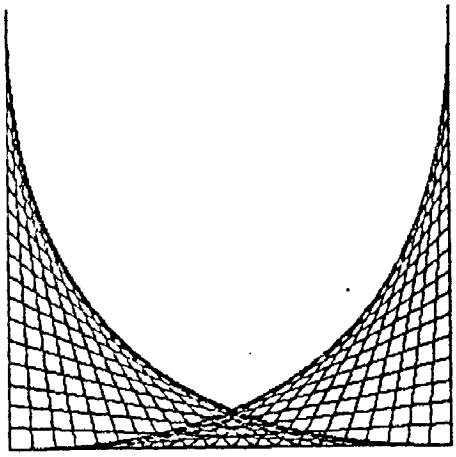
Analog dazu kann man auch die folgenden Grafiken durch einfache DERIVE-Programme beschreiben.

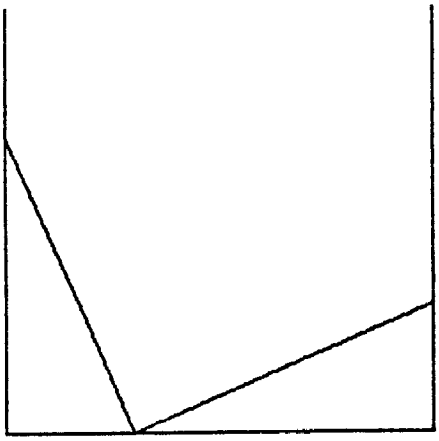
GRAFIK2(n) := VECTOR([[i,0],[0,n-i],[-i,0],[0,-n+i],[i,0]], i, 0, n)

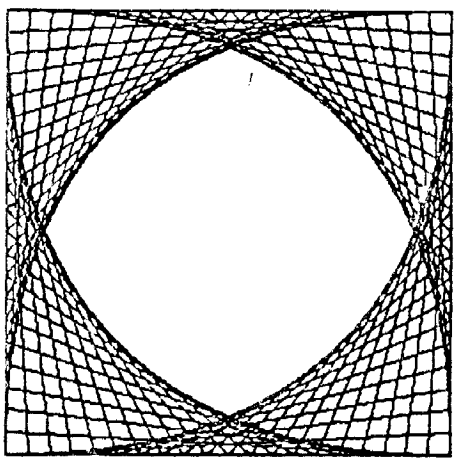
GRAFIK3(n) := VECTOR([[0,n-i],[i,0],[n,i]], i, 0, n)

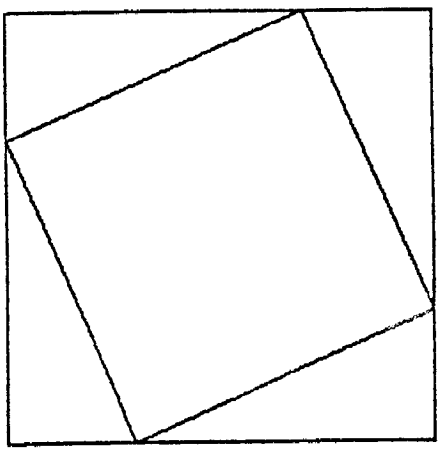
GRAFIK4(n) := VECTOR([[0,n-i],[i,0],[n,i],[n-i,n],[0,n-i]], i, 0, n)





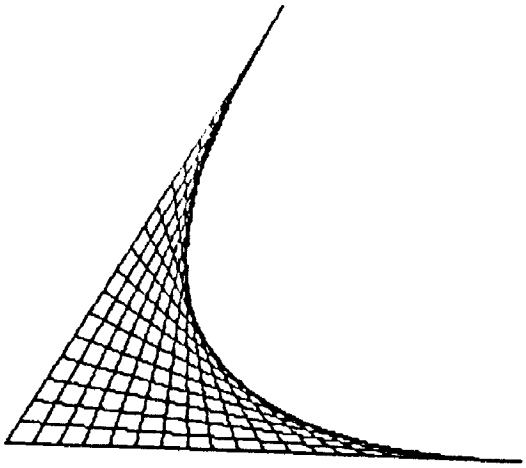


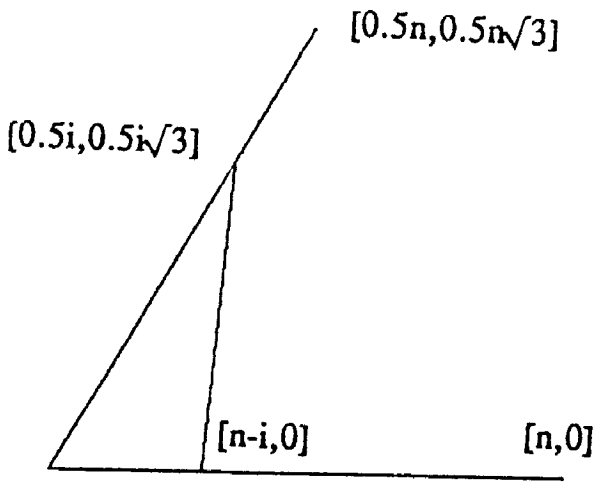




1.2 Liniengrafiken über einem 60°-Winkel

Geht man von zwei Strecken aus, die einen Winkel von 60° miteinander einschließen, so bilden diese Strecken die Seiten eines gleichseitigen Dreiecks. Aus dem Unterstufenunterricht ist den Schülern bekannt, daß die Höhe eines gleichseitigen Dreiecks mit Seitenlänge n die Länge $0.5*n*\sqrt{3}$ hat.





Als Endpunkte der beiden Strecken kann man daher die Punkte [0,0], [n,0] sowie $[0.5n,0.5n*\sqrt{3}]$ wählen, der i-te Teilungspunkt der zweiten Strecke ist in diesem Fall gegeben durch $[0.5i,0.5i*\sqrt{3}]$. Das zugehörige DERIVE-Programm lautet:

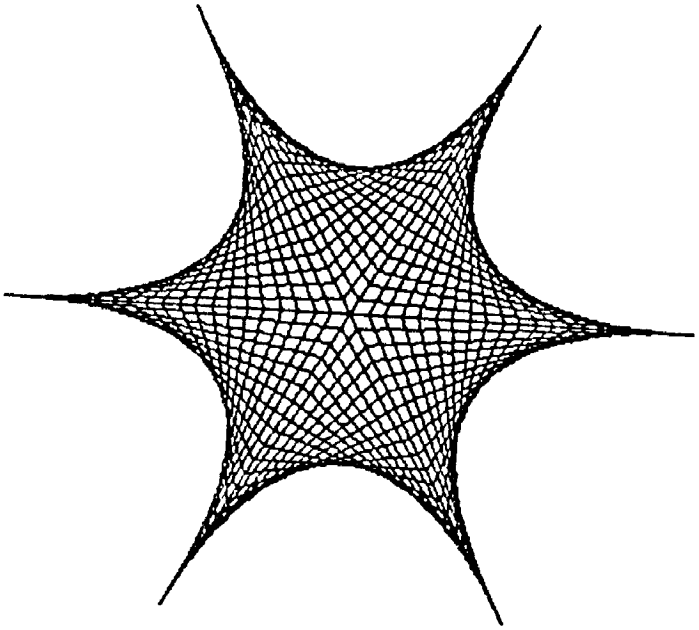
GRAFIK1(n) := VECTOR([[n-i,0],[0.5i,0.5i√3]],i,0,n)

Die folgenden Abbildungen werden durch die Befehle GRAFIK2(n), GRAFIK3(n) bzw. GRAFIK4(n) erzeugt:

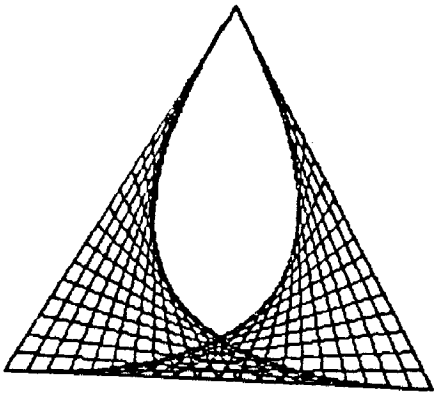
GRAFIK2(n) :=
VECTOR([[i,0],[0.5(n-i),0.5(n-i)√3],[-0.5i,0.5i√3],[-n+i,0],[-0.5i,-0.5i√3],
[0.5(n-1),0.5(-n+i)√3],[i,0]], i, 0, n)

GRAFIK3(n) := VECTOR([[0.5(n-i),0.5(n-i)√3],[i,0],[n-0.5i,0.5i√3]], i, 0, n)

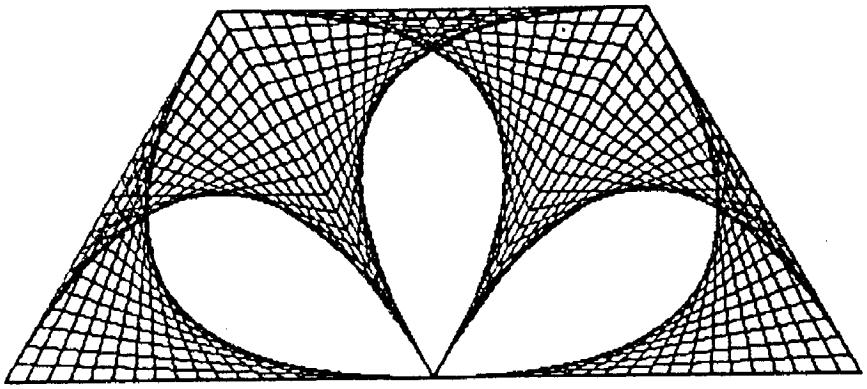
GRAFIK4(n) :=
VECTOR([[i,0],[n-0.5i,0.5i√3],[0.5(n-i),0.5(n-i)√3],[-0.5n+i,0.5n√3],
[-0.5i,0.5i√3],[-0.5(n+i),0.5(n-i)√3],[-n+i,0]], i, 0, n)



GRAFIK2(20)



GRAFIK3(20)



GRAFIK4(20)

2. Regelmäßige n-Ecke

2.1 Zeichnen von regelmäßigen n-Ecken

Die Eckpunktskoordinaten eines regelmäßigen n-Ecks mit Mittelpunkt [0,0] und Umkreisradius r sind gegeben durch

$[r*\cos(2\pi i/n), r*\sin(2\pi i/n)]$ wobei $0 \leq i \leq n-1$ ist.

In der Syntax von DERIVE kann dieses n-Eck durch die folgende Punktliste beschrieben werden:

VIELECK(n,r) := VECTOR([r.COS(2πi/n),r.SIN(2πi/n)], i, 0, n)

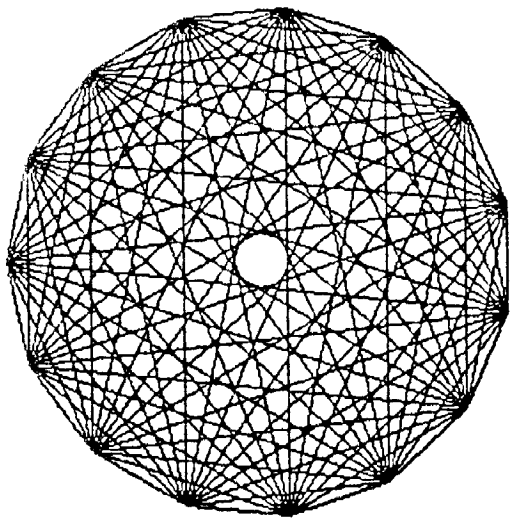
Eine andere Möglichkeit besteht darin, die Eckpunkte durch die Ausdrücke [r.COS(i),r.SIN(i)] festzulegen, wobei i die Werte von 0 bis 2π mit Schrittweite $2\pi/n$ durchläuft. Der entsprechende DERIVE-Befehl lautet:

VIELECK(n,r) := VECTOR([r COS(i), r SIN(i)], i, 0, 2π, 2π/n)

Vereinfacht man, zum Beispiel, den Ausdruck VIELECK(6,1) mit dem Menübefehl SIMPLIFY, so erhält man eine Liste aus sieben Punkten. Markiert man diese Liste und wählt dann zweimal PLOT an, so wird der Streckenzug gezeichnet, der diese Punkte fortlaufend verbindet. Am Bildschirm entsteht ein regelmäßiges Sechseck mit Mittelpunkt [0,0] und Umkreisradius 1. Wichtig ist, daß die obige Punktliste den ersten Eckpunkt [r.COS(0), r.SIN(0)] als siebenten Punkt der Liste ein zweites Mal enthält. Dadurch wird auch die letzte Verbindungsstrecke vom sechsten Eckpunkt zum ersten eingezeichnet.

2.2 Alle Diagonalen eines regelmäßigen n-Ecks

Eine reizvolle Grafik entsteht, wenn man jeden Eckpunkt eines regelmäßigen n-Ecks mit jedem anderen Eckpunkt verbindet:



Der Einfachkeit halber soll wieder der Umkreisradius des Vielecks gleich 1 gesetzt werden. Die Linie, die den i-ten Eckpunkt [COS(i), SIN(i)] mit dem j-ten Eckpunkt [COS(j), SIN(j)] verbindet, wird in DERIVE durch die Punktliste

[[COS(i), SIN(i)],[COS(j),SIN(j)]]

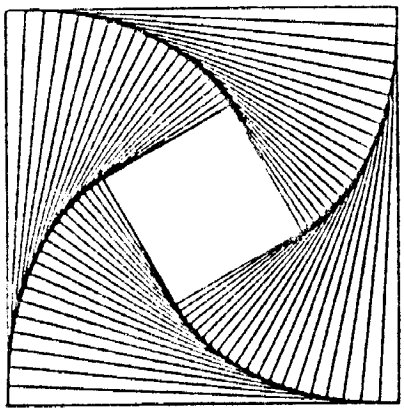
beschrieben. Um zu vermeiden, daß Linien unnötig doppelt gezeichnet werden, genügt es, den jeweils i-ten Eckpunkt nur mit dem (i+1)-ten, (i+2)-ten, ..., n-ten zu verbinden. Der folgende Befehl LINIEN(i,n) erzeugt diese vom i-ten Eckpunkt ausgehenden Linien. Um alle Verbindungsstrecken zu erhalten, muß schließlich noch i über alle n Eckpunkte des Vielecks variiert werden. Das wird durch den Befehl DIAGON(n) erreicht.

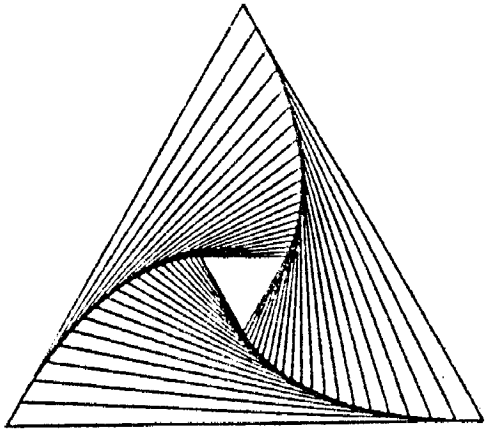
LINIEN(i,n) :=
VECTOR([[COS(i),SIN(i)],[COS(j),SIN(j)]],j, 2π(i+1)/n, 2π, 2π/n)

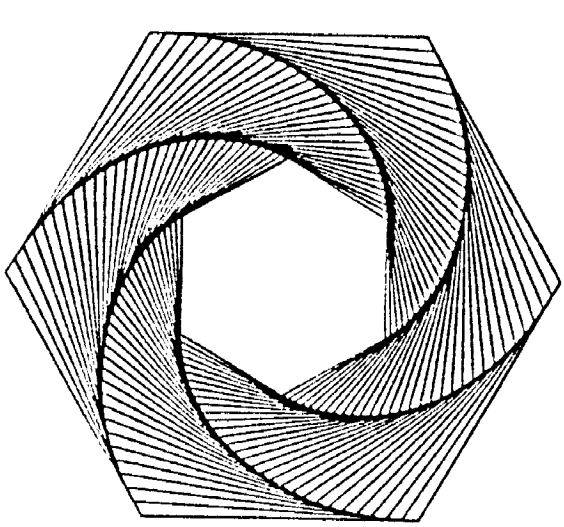
DIAGON(n) := VECTOR(LINIEN(i,n), i, n-1)

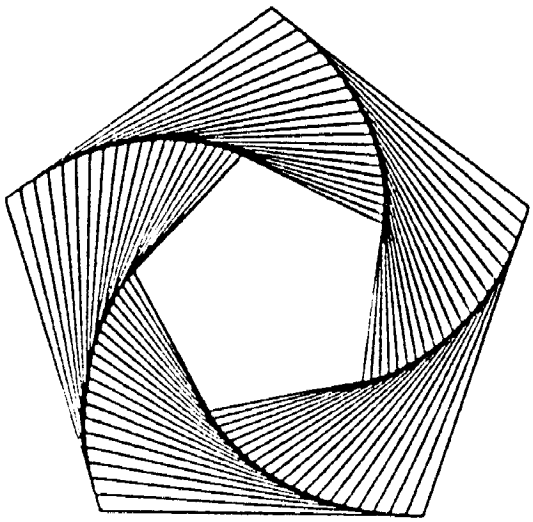
Selbstverständlich könnte man dieses aus zwei Befehlen bestehende DERIVE-Programm auch durch einen einzigen Befehl ersetzen, indem man die beiden VECTOR-Befehle ineinander schachtelt. Hier ist aus Gründen der Übersichtlichkeit die Notation mit Hilfe von zwei kürzeren Befehlen gewählt worden.

2.3 Eine Schar rotierter n-Ecke



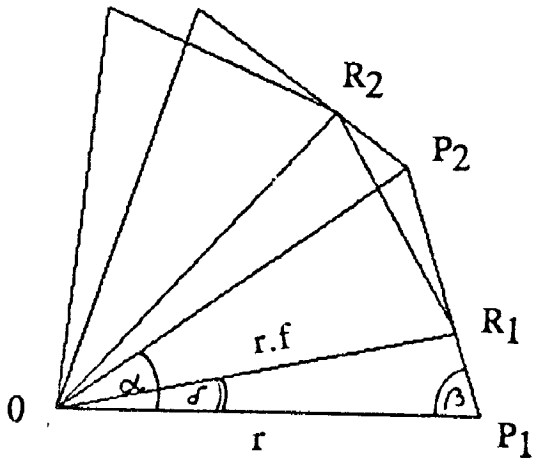






Diese Bilder zeigen Scharen von Vielecken, die dadurch entstehen, daß ein Vieleck immer wieder um einen fixen Winkel δ (zum Beispiel um $\delta=5°$) gedreht und gleichzeitig um so viel gestaucht wird, daß das neue Vieleck dem vorhergehenden eingeschrieben ist.

Um solche Bilder erzeugen zu können, muß man zunächst die Größe des Stauchfaktors f bestimmen:



Gegeben sei ein Sektor $0P_1P_2$ eines regelmäßigen n-Ecks mit Umkreisradius r. Der Zentriwinkel $\alpha = \sphericalangle P_1 0 P_2$ eines solchen Sektors beträgt $\alpha = 2\pi/n$, der Basiswinkel ß des gleichschenkeligen Dreiecks OP_1P_2 ist gegeben durch

$$ß = (180°-\alpha)/2 = \pi/2 - \pi/n.$$

Nun wird der Sektor $0P_1P_2$ um den Winkel δ gedreht und so gestaucht, daß das neue Dreieck $0R_1R_2$ dem gegebenen n-Eck eingeschrieben ist.

Den Umkreisradius $0R_1 = r.f$ des neuen n-Ecks kann man durch Anwenden des Sinussatzes im Dreieck $0P_1R_1$ bestimmen:

$$\frac{r.f}{\sin(ß)} = \frac{r}{\sin(180°-ß-\delta)}$$

Daraus folgt aber

$$f = \frac{\sin(ß)}{\sin(180°-ß-\delta)} = \frac{\sin(\pi/2-\pi/n)}{\sin(\pi/2+\pi/n-\delta)}$$

Mit Hilfe der Reduktionsformel $\sin(90°-\alpha) = \cos(\alpha)$ kann man f weiter vereinfachen zu

$$F(n,\delta) := \frac{COS(\pi/n)}{COS(\delta - \pi/n)}$$

Setzt man den Umkreisradius des Ausgangsvielecks gleich 1, so ist der Umkreisradius des k-ten gedrehten Vielecks gleich f^k. Dieses Vieleck ist gegenüber dem Ausgangsvieleck um den Winkel $k.\delta$ gedreht. Es wird daher beschrieben durch:

ROTECK(n,δ,k) :=
VECTOR([F(n,δ)^k COS(i+kδ),F(n,δ)^k SIN(i+kδ)], i, 0, 2π, 2π/n)

Eine Schar aus z solchen gedrehten n-Ecken erhält man dann durch

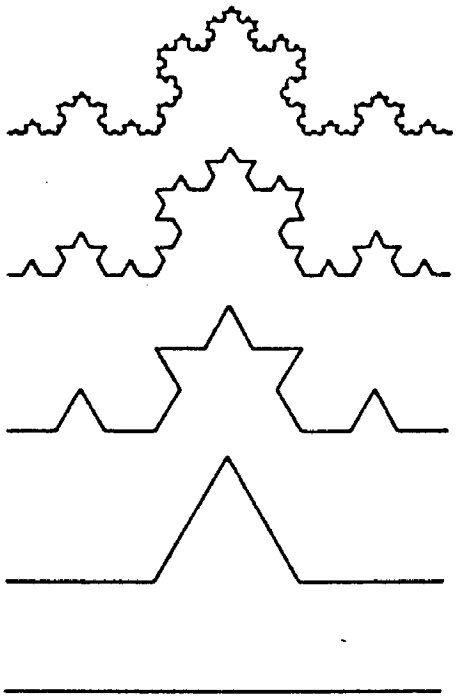
ROTSCHAR(n,δ,z) := VECTOR(ROTECK(n,δ,k), k, 0, z)

Vereinfachen des Ausdrucks ROTSCHAR(6,5°,12) liefert dann, zum Beispiel, die Eckpunktskoordinaten von insgesamt 13 regelmäßigen Sechsecken. Neben dem Ausgangssechseck werden 12 weitere um immer wieder 5° gedrehte einander eingeschriebene Sechsecke erzeugt.

3. Die Kochkurve

Die Kochkurve ist ein bekanntes Beispiel eines Fraktals. Sie wurde bereits 1904 vom schwedischen Mathematiker Helge von Koch definiert. Kochs Intention war es, ein Beispiel einer Monsterkurve anzugeben, die zwar überall stetig, aber an keiner Stelle differenzierbar ist.

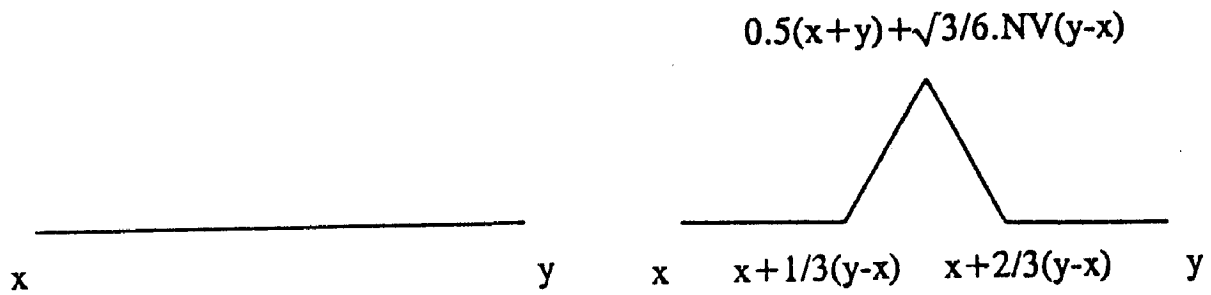
3.1 Die Konstruktion der Kochkurve



Bei der Konstruktion der Kochkurve geht man von der Einheitsstrecke aus. Entfernt man das mittlere Drittel und ersetzt es durch zwei Seiten eines über diesem Drittel errichteten gleichseitigen Dreiecks, so erhält man eine Kochkurve 1.Ordnung. Diese besteht aus vier Strecken der Länge 1/3. Wendet man auf jede dieser vier Teilstrecken dasselbe Konstruktionsprinzip an, so erhält man eine Kochkurve 2. Ordnung usw.

Die von Koch als Beispiel einer nirgends differenzierbaren Funktion definierte Kurve entsteht, wenn dieses Konstruktionsprinzip unendlich oft angewendet wird.

Mit Hilfe der DERIVE-Programmierbefehle kann eine Kochkurve k-ter Ordnung relativ einfach auf dem Bildschirm erzeugt werden. Man geht von der Einheitsstrecke als Kochkurve 0-ter Ordnung aus, und konstruiert daraus Schritt für Schritt immer wieder eine Kochkurve nächsthöherer Ordnung. Der wesentliche Konstruktionsschritt besteht darin, einer Teilstrecke mit den Endpunkten x und y den Strekkenzug GENERATOR(x,y) zuzuordnen, der den Übergang zur Kochkurve nächsthöherer Ordnung bewirkt.



Die einzige Schwierigkeit dabei ist, daß man auch die Spitze des gleichseitigen Dreiecks durch x und y ausdrücken muß. Um das zu bewerkstelligen, wird zunächst der Hilfsbefehl NV(v) definiert, der einem Vektor v der Ebene einen nach links gedrehten Normalvektor gleicher Länge zuordnet.

NV(v) := [[0,-1],[1,0]].v

Auch wenn die Beschreibung von Abbildungen der Ebene mit Hilfe von Matrizen im Unterricht nicht behandelt worden ist, kann dieser Befehl den Schülern als Black-Box zur Verfügung gestellt werden. Mit Hilfe des Befehls NV(v) kann der Generatorbefehl folgendermaßen formuliert werden:

GENERATOR(x,y) := [x, 1/3(2x+y), 0.5(x+y+√3/3.NV(y-x), 1/3(x+2y)]

Besteht die bereits ermittelte Kochkurve n-ter Ordnung aus der Punktliste v = [v_1, v_2, v_3, ..., $v_{DIM(v)}$], so erhält man die Kochkurve (n+1)-ter Ordnung, indem man auf jede Teilstrecke (v_i, v_{i+1}) den GENERATOR-Befehl anwendet. Der folgende Befehl NÄCHST(v) erzeugt diese Vektoren GENERATOR(v_i,v_{i+1}) der einzelnen Teilstrecken und reiht sie aneinander:

NÄCHST(v) := APPEND(APPEND(VECTOR(GENERATOR(v_i,v_{i+1}),
i, DIMENSION(v)), [$v_{DIMENSION(v)}$]))

Die Punktliste GENERATOR(x,y) enthält den rechten Eckpunkt y der betrachteten Strecke nicht, weil der Endpunkt der i-ten Teilstrecke immer gleichzeitig der Anfangspunkt der (i+1)-ten Teilstrecke ist. Aus diesem Grund muß im NÄCHST-Befehl der Endpunkt $v_{DIMENSON(v)}$ der allerletzten Teilstrecke hinzugefügt werden.

Ausgehend von der Einheitstrecke [[0,0],[1,0]] als Kochkurve 0-ter Ordnung erzeugt nun der folgende rekursive Befehl KOCH(k) eine Kochkurve k-ter Ordnung:

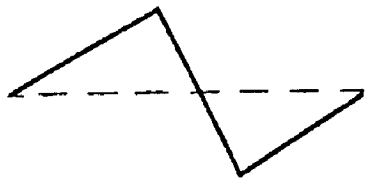
KOCH(k) := IF(k=0, [[0,0],[1,0]], NÄCHST(KOCH(k-1))

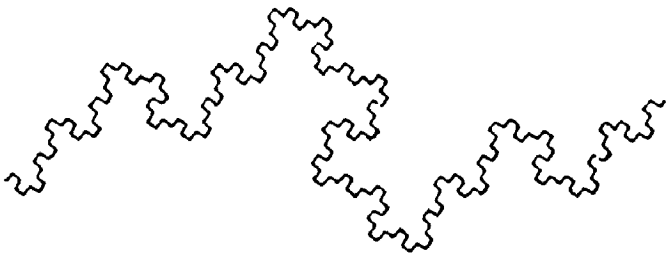
3.2 Kochähnliche Kurven

Die Kochkurve kann auf vielfältigste Weise verallgemeinert werden. Ersetzt man den in 3.1 angegebenen Befehl GENERATOR(x,y) durch einen geigneten anderen, so erzeugt der Befehl KOCH(k) die zugehörige kochähnliche Kurve k-ter Ordnung. Im folgenden werden vier Beispiele für mögliche Generatoren vorgestellt.

3.2.1 Die Koch-2-Kurve

Wählt man ausgehend von der Einheitsstrecke die Punkte [0.4,0.2] und [0.6,-0.2] als Eckpunkte des Generators, so entstehen drei gleich lange Strecken der Länge $1/\sqrt{5}$.



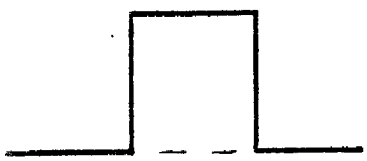


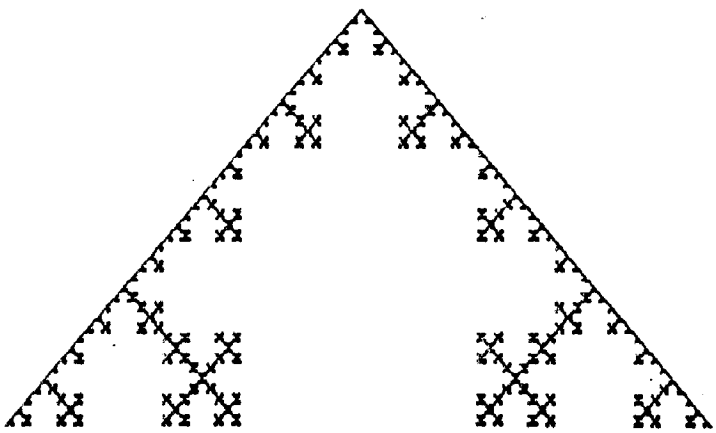
In DERIVE kann man diesen Generator folgendermaßen beschreiben:

GENERATOR(x,y) := [x, 0.6x+0.4y+0.2NV(y-x), 0.4x+0.6y-0.2NV(y-x)]

Der Befehl KOCH(5) erzeugt mit diesem Generator die obige Kurve, die in der Literatur als Koch-2-Kurve bzw. als Alternative-Koch-Kurve bezeichnet wird.

3.2.2 Die Kreuzstichkurve





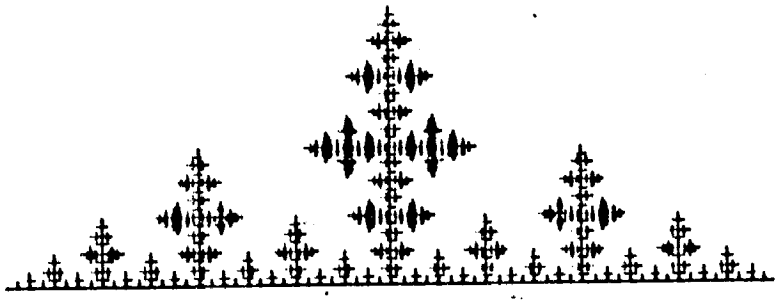
Eine andere Möglichkeit besteht darin, über das mittlere Dreieck der Einheitsstrecke anstelle des gleichseitigen Dreiecks der Kochkurve ein Quadrat zu setzen. Dieser Generator besteht aus fünf Teilstrecken der Länge 1/3.

GENERATOR(x,y) :=
[x, 1/3(2x+y), 1/3(2x+y+NV(y-x)), 1/3(x+2y+NV(y-x)), 1/3(x+2y)]

Der Name Kreuzstichkurve kommt daher, weil das Aussehen der zugehörigen Kochkurve an ein Kreuzstichmuster erinnert. Die Abbildung zeigt KOCH(5).

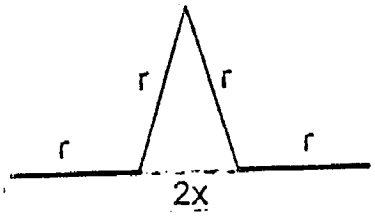
3.2.3 Die Eiskurve

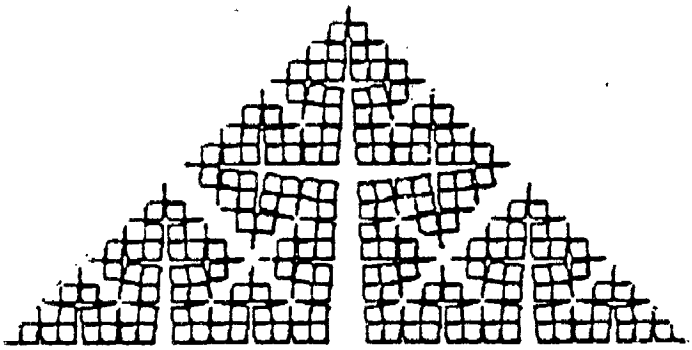
Eine Kochkurve, die an Eisnadeln denken läßt, erhält man mit dem folgenden Generator: Über dem Mittelpunkt der Einheitsstrecke wird eine Lotstrecke der Länge 1/3 errichtet.



GENERATOR(x,y) := [x, 0.5(x+y), 0.5(x+y)+1/3NV(y-x), 0.5(x+y)]

3.2.4 Die Cesárokurve





Hier wird der Einheitsstrecke anstelle des gleichseitigen Dreiecks der Kochkurve ein gleichschenkeliges Dreieck mit vorgegebenem Basiswinkel α aufgesetzt. Die Spitze dieses Dreiecks liegt über dem Mittelpunkt der Einheitsstrecke, seine Schenkellänge ist so gewählt, daß der Generator aus vier gleich langen Strecken besteht.

Die Länge r der vier Teilstrecken hängt von der Größe des Winkels α ab. Sie kann folgendermaßen bestimmt werden:
Aus $\cos\alpha = x/r$ und $2r+2x=1$ folgt, daß $r = 1/(2(1+\cos(\alpha)))$ ist.

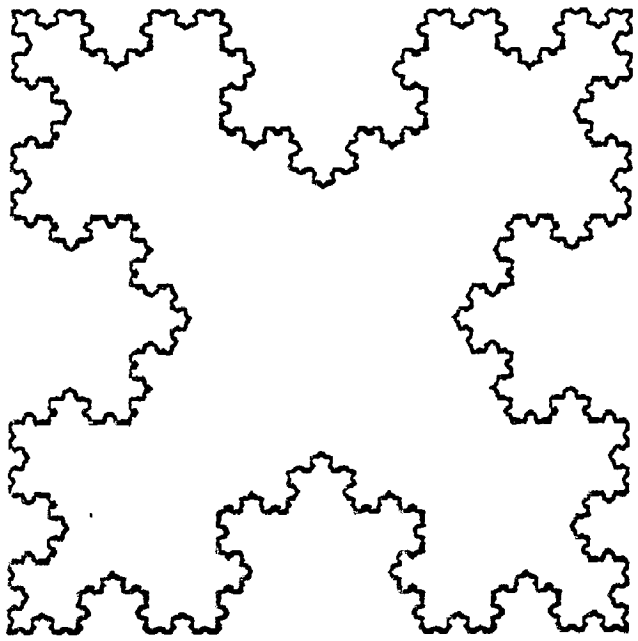
In der Syntax von DERIVE kann man den Generator daher folgendermaßen beschreiben:

$$[\alpha := 80°,\ r := \frac{1}{2(1+\mathrm{COS}(\alpha))},$$

GENERATOR(x,y) := [x, x+r(y-x), 0.5(x+y)+r.SIN(α).NV(y-x), y-r(y-x)]]

3.3 Kochkurven in einem Quadrat

Aus vier Kochkurven kann man das unten abebildete Kochquadrat zusammensetzen.



[0,0] [1,0]

Sei [0,0] der linke untere Eckpunkt des Quadrats und v die Kochkurve, aus der die zwischen [0,0] und [1,0] liegende untere "Quadratseite" besteht.

Dann erhält man die rechte Quadratseite QUR(v) durch Linksdrehung von v um 90° um den Nullpunkt (d.h. durch Anwenden des Befehls NV) und anschließendes Verschieben um den Vektor [1,0].

QUR(v) := VECTOR(NV(v_i) + [1,0], i, DIMENSION(v))

Die obere Quadratseite QUO(v) erhält man durch Drehen von v um 180° um den Nullpunkt (d.h. $v_i \longrightarrow -v_i$) und anschließendes Verschieben um den Vektor [1,1].

QUO(v) := VECTOR(-v_i + [1,1], i, DIMENSION(v))

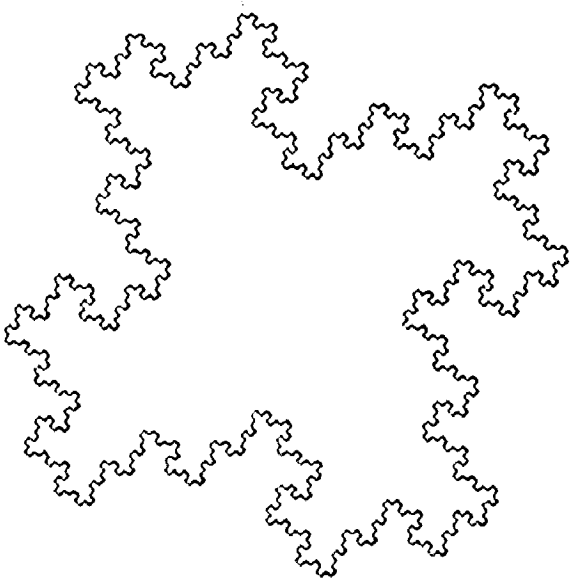
Die linke Quadratseite wird beschrieben durch QUO(QUR(v)). Das gesamte Quadrat QUADRAT(v) erhält man durch Aneinanderreihen dieser vier Punktlisten:

QUADRAT(v) := APPEND(v, QUR(v), QUO(v), QUO(QUR(v)))

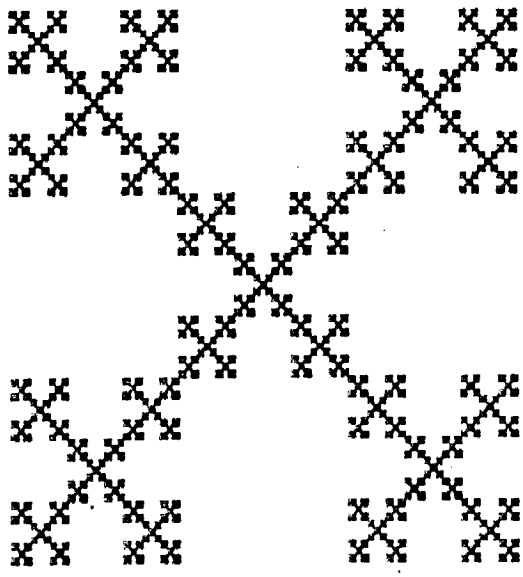
Wendet man diesen Befehl auf den Vektor KOCH(k) an, so erhält man das Kochquadrat k-ter Ordnung:

KOCHQUADRAT(k) := QUADRAT(KOCH(k))

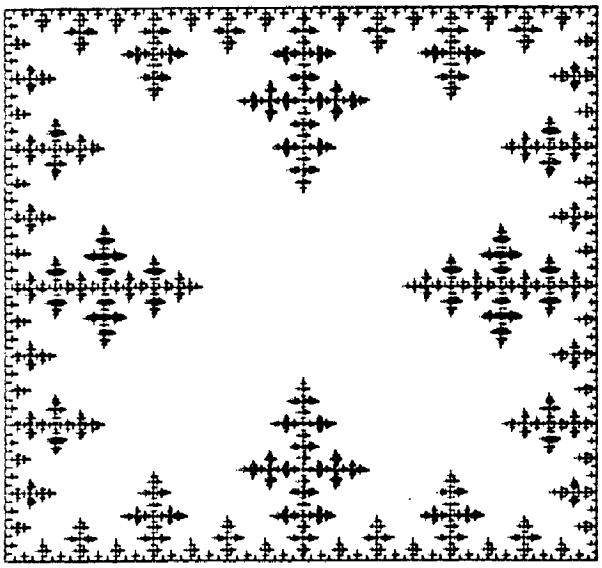
Je nach Wahl des Generators erhält man die folgenden Bilder:



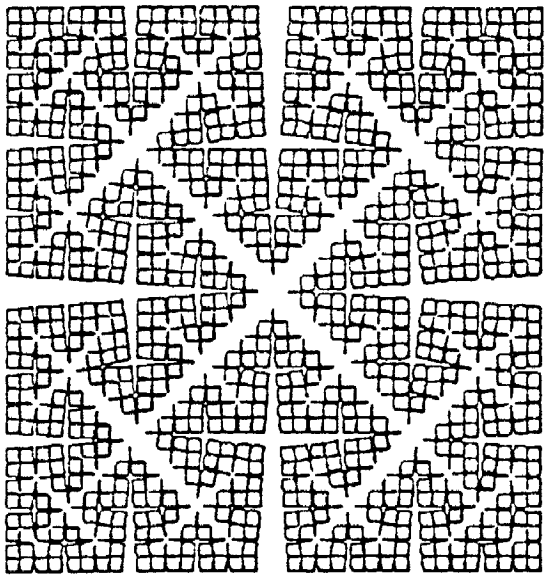
Koch-2-Quadrat



Kreuzstichkurve



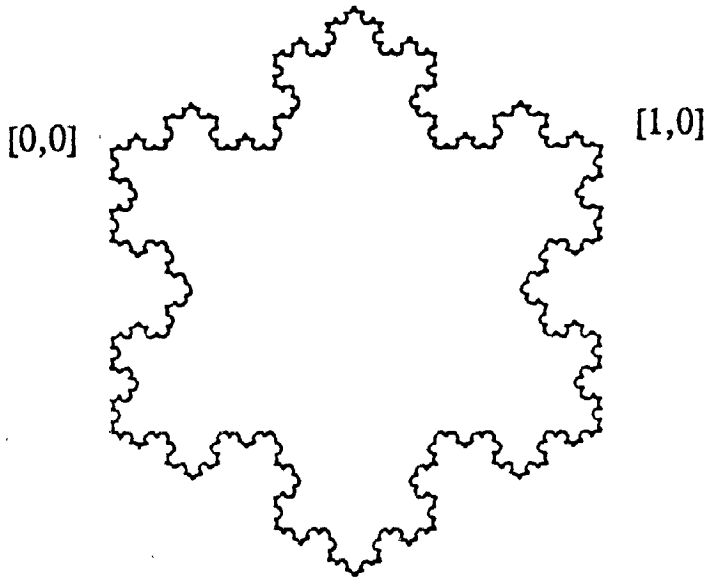
Eisquadrat



Cesaroquadrat

3.4 Die Schneeflockenkurve

Setzt man drei Kochkurven zu einem gleichseitigen Dreieck zusammen, so erhält man eine sternförmige Kurve, die als Schneeflockenkurve bezeichnet wird.



Sei [0,0] der linke obere Eckpunkt des gleichseitigen Dreiecks, und v die Kochkurve, aus der die zwischen [0,0] und [1,0] liegende obere "Dreiecksseite" besteht.

Die rechte Dreiecksseite R(v) erhält man durch Drehung von v um den Nullpunkt um den Winkel -120° und anschließendes Verschieben um den Vektor [1,0].

Genauso kann man die linke Dreiecksseite L(v) durch Drehung von v um +120° um den Nullpunkt und anschließendes Verschieben um den Vektor $[0.5,-0.5\sqrt{3}]$ erzeugen.

Die erforderlichen Drehungen kann man mit Hilfe des folgenden Befehls DREH(v,α) durchführen, der allgemein das Ergebnis einer Drehung eines Punktes v um den Winkel +α um den Nullpunkt liefert.
Das gesamte DERIVE-Programm FLOCKE(k) zum Erzeugen der Schneeflockenkurve zu einer gegebenen Kochkurve k-ter Ordnung KOCH(k) besteht aus insgesamt fünf Befehlen:

DREH(v,α) := [[COS(α), -SIN(α)],[SIN(α), COS(α)]].v

R(v) := VECTOR(DREH(v_i,-120°)+[1,0], i DIMENSION(v))

L(v) := VECTOR(DREH(v_i,120°)+[0.5,-0.5√3], i DIMENSION(v))

FLOCKE1(v) := APPEND(v, R(v), L(v))

FLOCKE(k) := FLOCKE1(KOCH(k))

4. Das Sierpinski-Dreieck

Auch das Sierpinski-Dreieck ist ein klassisches Beispiel eines Fraktals. Es wurde 1916 vom polnischen Mathematiker Waclaw Sierpinski definiert.

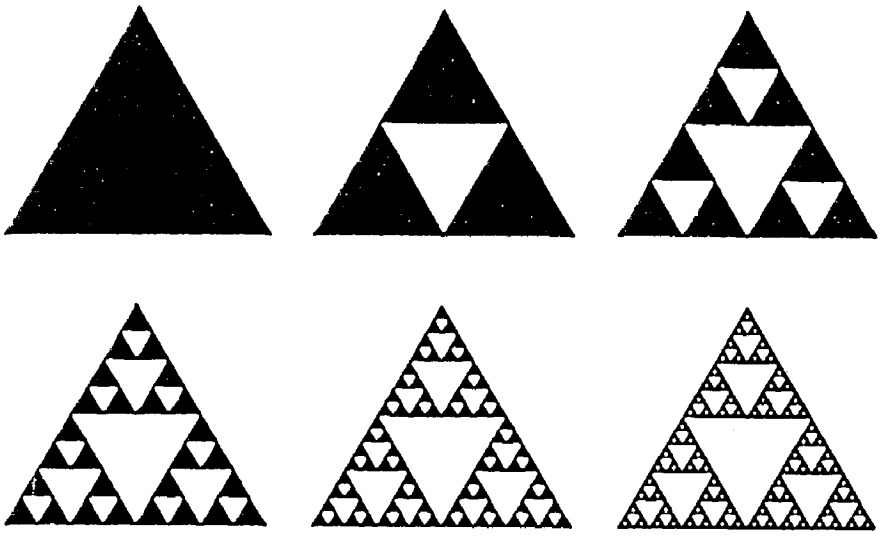
4.1 Die Konstruktion des Sierpinski-Dreiecks

Bei der Konstruktion des Sierpinski-Dreiecks geht man von einem gleichseitigen Dreieck aus, und entfernt das Mitteldreieck. Das dadurch entstehende Sierpinski-Dreieck erster Ordnung besteht aus drei kongruenten gleichseitigen Dreiecken, deren Seiten halb so lang wie die des ursprünglichen Dreiecks sind. Wendet man das-

selbe Konstruktionsprinzip auf jedes dieser Dreiecke an, so erhält man das aus neun gleichseitigen Dreiecken bestehende Sierpinski-Dreieck zweiter Ordnung usw.

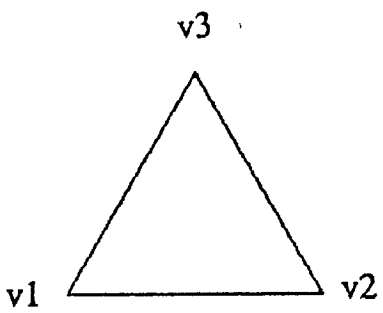
Bei jedem Konstruktionsschritt wird die Zahl der Dreiecke verdreifacht und gleichzeitig die Seitenlänge der Dreiecke halbiert, so daß das Sierpinski-Dreieck k-ter Ordnung aus 3^k Dreiecken der Seitenlänge $a_0*0.5^k$ besteht.

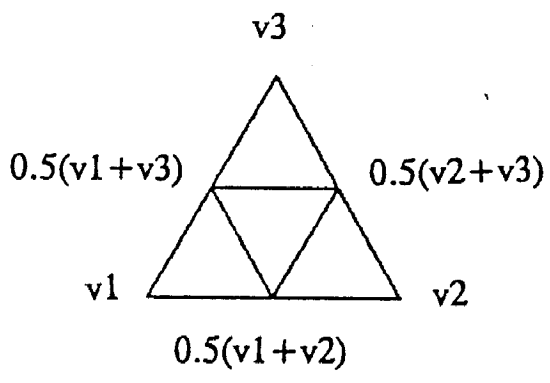
Das Sierpinski-Dreieck ist die Menge jener Punkte der Ebene, die übrig bleiben, wenn man dieses Konstruktionsprinzip unendlich oft ausführt.



Um ein DERIVE-Programm SIERP(k) zum Zeichnen eines Sierpinski-Dreiecks k-ter Ordnung zu erhalten, geht man der Einfachkeit halber vom folgenden gleichseitigen Dreieck mit Seitenlänge 1 aus:

v0 := [[0,0],[1,0],[0.5,0.5√3],[0,0]]





Der wesentliche Konstruktionsschritt besteht darin, einem bereits ermittelten Dreieck $v = [v_1, v_2, v_3, v_1]$ seine drei Teildreiecke

$[v_1, 0.5(v_1+v_2), 0.5(v_1+v_3), v_1]$
$[v_2, 0.5(v_2+v_3), 0.5(v_2+v_1), v_2]$
$[v_3, 0.5(v_3+v_1), 0.5(v_3+v_2), v_3]$

zuzuordnen. Man sieht sofort, daß diese Teildreiecke durch den folgenden Befehl TEIL(v,i) mit i=1,2,3 beschrieben werden können:

TEIL(v,i) := VECTOR(0.5(v_i+v_j), j, 4)

Beim Übergang von SIERP(k) zu SIERP(k+1) muß nun jedem der 3^k Dreiecke $v_1, v_2, v_3, \ldots, v_3\hat{}k$ von SIERP(k) sein linkes, sein rechtes und sein oberes Teildreieck zugeordnet werden. Die Menge aller 3^k linken Teildreiecke wird durch den folgenden Befehl L(v) erzeugt, analog dazu erzeugt R(v) die Menge aller rechten Teildreiecke von v und O(v) die Menge aller oberen Teildreiecke:

L(v) := VECTOR(TEIL(v_i,1), i DIMENSION(v))
R(v) := VECTOR(TEIL(v_i,2), i DIMENSION(v))
O(v) := VECTOR(TEIL(v_i,3), i DIMENSION(v))

Der Befehl APPEND(L(v),R(v),O(v)) faßt somit alle Dreiecke der nächsten Generation in einem Vektor zusammen, und der Befehl SIERP(k) erzeugt nun das gewünschte Sierpinski-Dreieck k-ter Ordnung:

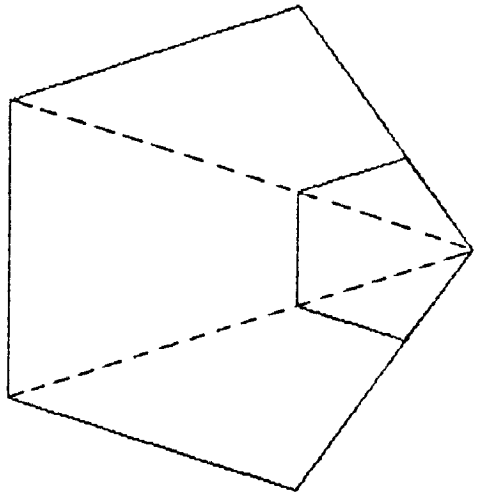
SIERP(k) := ITERATE(APPEND(L(v),R(v),O(v)),v, [v0], k)

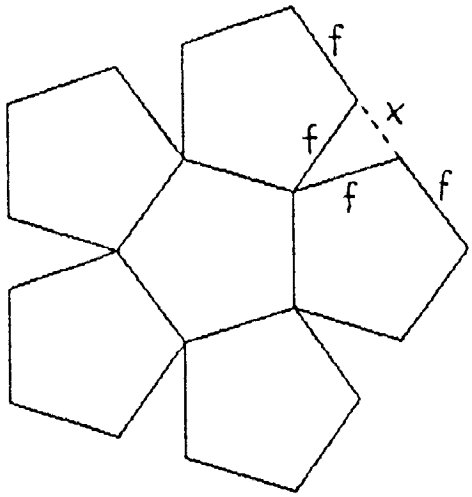
4.2 Sierpinski-Vielecke

Das Konstruktionsprinzip des Sierpinski-Dreiecks kann auch folgendermaßen erklärt werden:

Ein gleichseitiges Dreieck wird durch drei zueinander kongruente gleichseitige Dreiecke ersetzt, die
*) parallel zu den Seiten des ursprünglichen Dreiecks liegen,
*) jeweils einen Eckpunkt mit dem ursprünglichen Dreieck gemeinsam haben,
*) so lang sind, daß zwei benachbarte Teildreiecke genau einen Punkt gemeinsam haben.

Dieses Konstruktionsprinzip läßt sich ohne weiteres auf regelmäßige n-Ecke mit $n \geq 5$ übertragen.





Die kleinen n-Ecke der nächsten Generation kann man sich durch geeignete Streckungen mit jeweils einem Eckpunkt des Ausgangsvielecks als Streckungszentrum entstanden denken. Die Größe des Streckungsfaktors f hängt von n ab. Im Fall des Sierpinskidreiecks werden beim Übergang zur nächsthöheren Ordnung drei Streckungen mit Faktor f=0.5 ausgeführt, beim regelmäßigen Sechseck sechs Streckungen mit f=1/3 etc.

Geht man von einem regelmäßigen n-Eck mit Seitenlänge 1 aus, so beträgt die Seitenlänge des nächstkleineren n-Ecks f. Jede Seite des Ausgangsvielecks wird durch das kleine Vieleck in zwei Abschnitte der Länge f und einen mittleren Abschnitt der Länge x zerlegt. Diese Mittelstrecke x ist Basis eines gleichschekeligen Dreiecks mit Basiswinkel $2\pi/n$ und Schenkellänge f. Aus $\cos(2\pi/n) = x/(2f)$ und $2f+x=1$ folgt nun, daß

$$f = \frac{1}{2(1+\cos(2\pi/n)}$$

Das zugehörige DERIVE-Programm kann folgendermaßen formuliert werden. Zunächst legt man die Größen n und f sowie die Koordinaten des Ausgangsvielecks fest:

[n:=6, f:= 1/(2+2.COS(2π/n)), v0:=VECTOR([COS(i),SIN(i)],i,0,2π,2π/n)]

Wie beim Sierpinskidreieck kann man das i-te Teilvieleck eines bereits ermittelten Vielecks v durch den analogen Befehl TEIL(v,i) beschreiben:

TEIL(v,i) := VECTOR((1-f).v_i + f.v_j, j, 1, n+1)

Der Befehl TEILALL(v) ordnet dem Vieleck v alle seine n Teilvielecke der nächsten Generation zu:

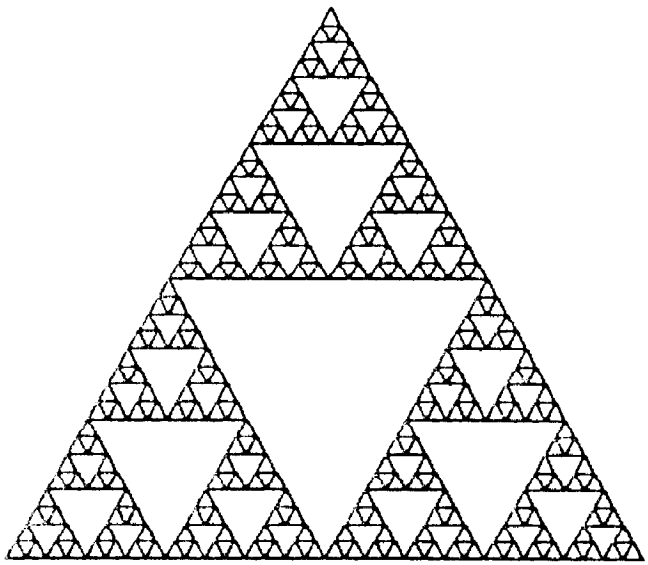
TEILALL(v) := VECTOR(TEIL(v,i), i, n)

Besteht das bereits ermittelte Sierpinskivieleck k-ter Ordnung v = SIERP(k) aus n^k Vielecken v_1, v_2, v_3, ..., $v_{DIM(v)}$, so muß beim Übergang zu SIERP(k+1) auf jedes dieser Vielecke v_i der Befehl TEILALL(v_i) angewendet werden:

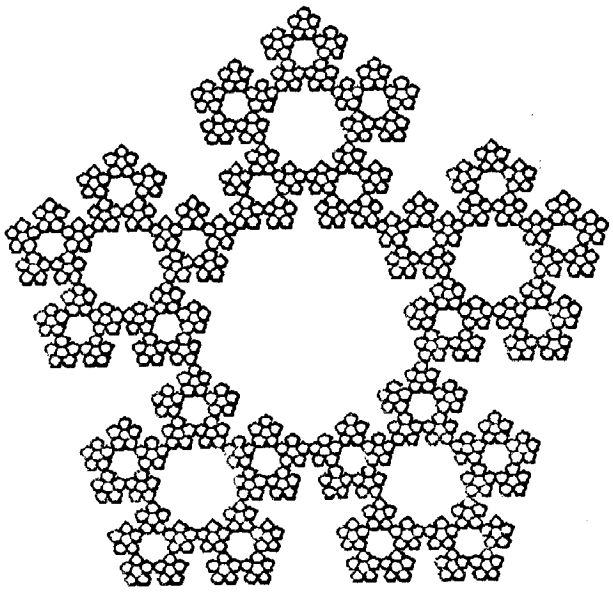
NÄCHST(v) :=
IF(DIMENSION(v)=1,TEILALL(v_1),APPEND(NÄCHST(DELETE_ELEMENT (v,DIMENSION(v)), TEILALL($v_{DIMENSION(v)}$)))

Ausgehend von v0 als Sierpinskivieleck 0-ter Ordnung erzeugt nun der rekursive Befehl SIERP(k) ein Sierpinskivieleck k-ter Ordnung:

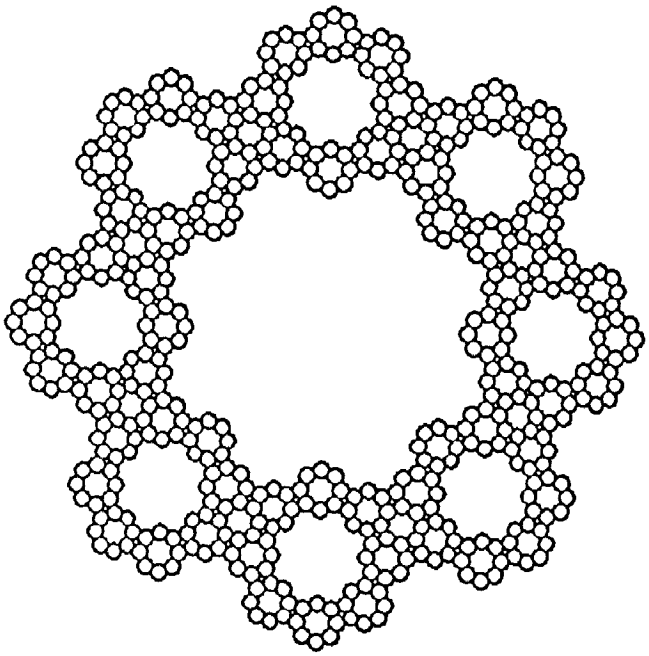
SIERP(k) := IF(k=0, [v0], NÄCHST(SIERP(k-1)))



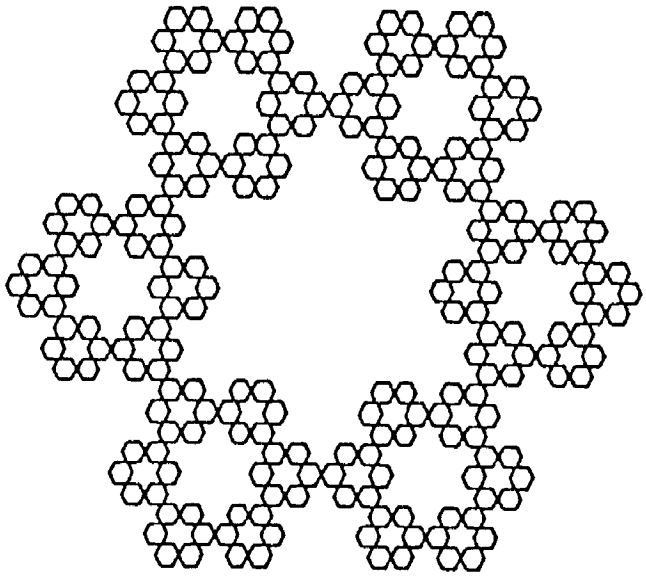
Sierpinskidreieck 5.Ordnung



Sierpinskifünfeck 4.Ordnung



Sierpinskiachteck 3.Ordnung



Sierpinskisechseck 3.Ordnung

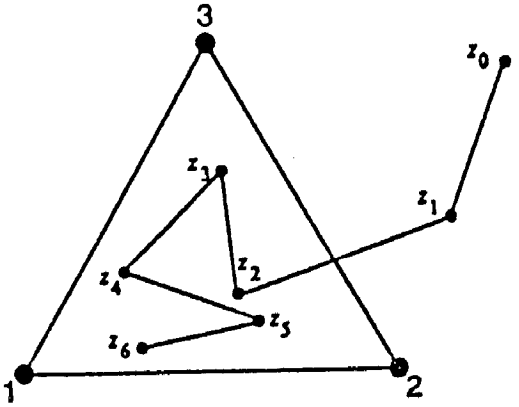
5. Das Chaosspiel

1985 hat der englische Mathematiker Michael Barnsley eine neue interessante Methode vorgestellt, fraktale Bilder zu erzeugen. Er hat diese Methode als Chaosspiel bezeichnet.

5.1 Das Chaosspiel für das Sierpinski-Dreieck

Das einfachste Beispiel eines Chaosspiels funktioniert folgendermaßen: Man geht von einem gleichseitigen Dreieck mit den Eckpunkten v1, v2 und v3 aus und wählt einen beliebigen Punkt z_0 der Ebene als Startpunkt des Spiels. Als nächstes wird ein Eckpunkt des Dreiecks mit Zufallsprinzip ausgewählt, zum Beispiel v2. Der Mittelpunkt der Strecke von z_0 zu diesem Eckpunkt v2 wird als nächster Spielpunkt z_1 festgesetzt und auf dem Bildschirm markiert. Dann wird ein neuer Dreieckseckpunkt ausgelost, zum Beispiel v1. Der Mittelpunkt der Strecke von z_1 zu diesem Eckpunkt v1 liefert den nächsten Spielpunkt z_2. Dieses Verfahren wird immer wieder durchgeführt: Jeweils vom zuletzt markierten Punkt z_n wird der Mit-

telpunkt zu einem zufällig gewählten Dreieckseckpunkt als nächster Spielpunkt z_{n+1} gewählt.



In DERIVE kann man diesen Algorithmus mit Hilfe der vordefinierten RANDOM-Funktion beschreiben. Für eine positive natürliche Zahl n liefert der Befehl RANDOM(n) ein zufällig gewähltes Element der Menge {0, 1, 2, ...,n-1}. Insbesondere liefert daher der Befehl 1+RANDOM(3) ein zufällig gewähltes Element der Menge {1,2,3}. Das Bildungsgesetz der Folge der Spielpunkte des Chaosspiels lautet:

$$z_{n+1} = 0.5(z_n + v_{1+\text{RANDOM}(3)})$$

Das zugehörige DERIVE-Programm besteht aus zwei Zeilen und sieht folgendermaßen aus: Zunächst legt man den Vektor v = [v1, v2, v3] mit den Eckpunktskoordinaten des Ausgangsdreieck und den Startpunkt z0 fest. Dann wird der Befehl CHAOS(n) formuliert, der die Koordinaten der ersten n+1 Punkte des Chaosspiels erzeugt:

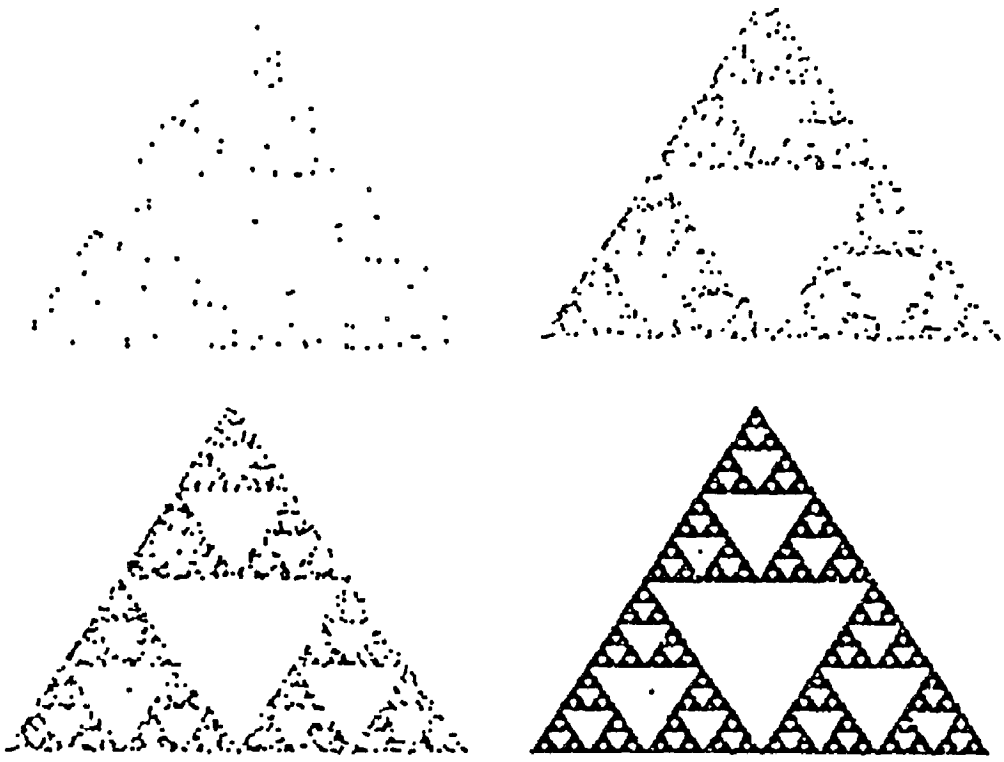
[v:= [[0,0], [1,0],[0.5,0.5√3]], z0:= [1,1]]

CHAOS(n):= ITERATES($0.5(z+v_{1+\text{RANDOM}(3)})$, z, z0, n)

Wichtig ist, daß man vor dem Zeichnen der Punkte beim Graphikmenüpunkt OPTIONS STATE den Modus DISCRETE einstellt, damit die Punkte der Folge tatsächlich als einzelne Punkte dargestellt und nicht durch Strecken am Bildschirm miteinander verbunden werden.

Interessant ist nun die Frage, wie das Ergebnis dieses Spiels für großes n aussieht. Aufgrund des Bildungsgesetzes der Folge ist es klar, daß für einen Spielpunkt, der

innerhalb des Ausgangsdreiecks liegt, alle weiteren Spielpunkte ebenfalls innerhalb dieser konvexen Menge liegen müssen. Startet man aber außerhalb des Dreiecks, so nähern sich die weiteren Punkte der Folge Schritt für Schritt dem Dreieck an, und es ist zu erwarten, daß früher oder später ein Spielpunkt (und damit auch alle weiteren) innerhalb des Dreiecks zu liegen kommt. Da die einzelnen Punkte der Folge durch ein Zufallsprinzip bestimmt werden, würde man also für großes n eine Zufallsverteilung von Punkten im Bereich des Ausgangsdreiecks erwarten.



Stellt man aber eine große Zahl von Spielpunkten dar, die Abbildungen zeigen 100, 500, 1000 bzw. 10000 Punkte, so macht man eine überraschende Entdeckung. Obwohl die einzelnen Punkte zufällig verteilt sind, und man innerhalb des Erzeugungsprozesses nie voraussagen kann, wo genau der nächste Punkt zu liegen kommt, bildet die Gesamtheit der Punkte ein ganz bestimmtes, absolut vorhersagbares Muster. Unabhängig von der Wahl des Startpunkts erscheint für genügend großes n stets deutlich das Bild eines Sierpinskidreiecks.

5.2 Weitere Chaosspiele

Dieses unerwartete Ergebnis des Chaosspiels wirft die Frage auf, inwiefern der betrachtete Zufallsprozeß mit speziellen Eigenschaften des Sierpinskidreiecks zu tun

hat, und ob es möglich ist, auch andere fraktale Bilder durch geeignete Zufallsprozesse zu beschreiben.

Durch das Chaosspiel wird einem bereits ermittelten Spielpunkt z_n der Mittelpunkt der Strecke von z_n zu einem der Dreieckseckpunkte zugeordnet, das heißt auf den Punkt z_n wird eine zentrische Streckung mit Faktor 0.5 und einem der Dreieckseckpunkte als Streckungszentrum ausgeübt. Diese drei Ähnlichkeitsabbildungen können aber als erzeugende Transformationen des Sierpinskidreiecks aufgefaßt werden. Wendet man auf ein gleichseitiges Dreieck diese drei zentrischen Streckungen an, so erhält man als Ergebnis ein Sierpinskidreieck erster Ordnung. Nochmaliges Anwenden der drei Streckungen ergibt ein Sierpinskidreieck zweiter Ordnung usw. Das Sierpinskidreieck schließlich enststeht als Grenzprozeß dieser Konstruktion. Es besteht aus allen jenen Punkten, die übrig bleiben, wenn man diese drei Ähnlichkeitsabbildungen unendlich oft anwendet.

Startet man daher das Chaosspiel in einem Punkt des Sierpinskidreiecks, zum Beispiel in [0.5,0], so gehören automatisch auch alle weiteren Punkte der Folge dem Sierpinskidreieck an.

Wählt man als Startpunkt einen Punkt innerhalb des Dreiecks v1v2v3, der nicht dem Sierpinskidreieck angehört, so liegen aufgrund des Konstruktionsprinzips der Folge auch alle weiteren Punkte nicht im Sierpinskidreieck. Die Folgenpunkte nähern sich aber Schritt für Schritt immer stärker dem Sierpinskidreieck an, und erzeugen in ihrer Gesamtheit ebenfalls dieses Bild.

Wesentlich für das "Funktionieren" des Chaosspiels ist (siehe [Peitgen]), daß sich die Folge des Chaosspiels jedem Punkt des Sierpinskidreiecks beliebig nähert: Ist x ein beliebiger Punkt des Sierpinskidreiecks und $\delta > 0$, so gibt es ein Element der Folge, das in der δ-Umgebung von x liegt.

In 4.2 ist das Erzeugungsprinzip des Sierpinskidreiecks auf regelmäßige n- Ecke mit $n \geq 5$ übertragen worden. Ein Sierpinskivieleck kann durch wiederholtes Anwenden von n zentrischen Streckungen mit Streckungsfaktor $f = 1/(2+2\cos(2\pi/n))$ erzeugt werden.

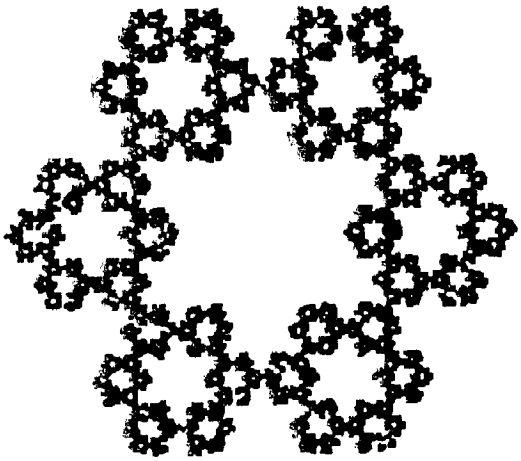
Man kann daher genauso auch Chaosspiele für Sierpinskivielecke angeben. In diesem Fall ist der bisherige Streckungsfaktor 0.5 des Dreiecks durch f zu ersetzen, und es wird jeweils einer der n Eckpunkte des n-Ecks zufällig gewählt. Das entsprechende Bildungsgesetz lautet:

$$z_{n+1} = (1-f)*z_n + f*v_{1+\mathrm{RANDOM}(n)}$$

Für ein regelmäßiges Sechseck sieht das zugehörige DERIVE-Programm folgendermaßen aus:

[v6:= VECTOR([COS(i),SIN(i)], i, 0, 5π/3, π/3), f:= 1/3, z0:= [1,1]]

CHAOS6(n):= ITERATES((1-f).z + f.$v6_{1+RANDOM(6)}$, z, z0, n)



CHAOS6(6000)

Literatur:

Endl, K.: Kreative Computergrafik. VDI-Verlag, Düsseldorf 1986.

Herrmann, D.: Algorithmen für Chaos und Fraktale. Addison-Wesley, Bonn 1994.

Koth, M.: Einführung in das Arbeiten mit DERIVE 2.58. Skriptum des PI-Wien, Wien 1994.

Peitgen, H.O. u.a.: Fractals for the Classroom, Part One. Springer-Verlag, New York 1992.

Rich, A. u.a.: Handbuch DERIVE, Version 3, Der Mathematik-Assistent für Ihren Personal Computer. Soft Warehouse GmbH Europe, Hagenberg 1994.