L-Systeme (Lindenmayer-Systeme): Erweiterungen

L-Systeme:

- Verwendung hauptsächlich für Vegetationsmodelle
  aber auch: Webmuster, Gebäude, Roboter, Tiere (Vermehrung, Nahrungsaufnahme), Melodien
- 2D- und 3D-Varianten
- Emulation von iterierten Funktionssystemen möglich (siehe Prusinkiewicz & Lindenmayer 1990)
- *dynamische* Simulationen ($\Rightarrow$ Möglichkeit der Animation)
- Anbindung physikalisch oder biologisch begründeter Simulationsmodelle möglich (Kombination von Modell-Ansätzen)

Wiederholung:
Formalismus der L-Systeme:
aus der Theorie formaler Grammatiken

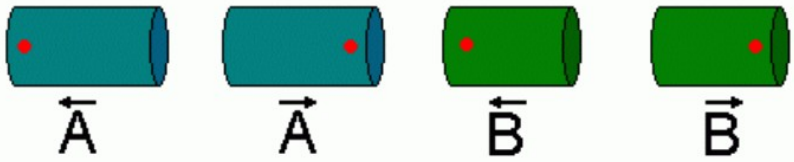
analog zu Chomsky-Grammatiken (regulär, kontextfrei, kontext-sensitiv etc.)
aber: in jedem Ableitungsschritt *parallele* Ersetzung aller Zeichen, auf die eine Regel anwendbar ist

von Aristid Lindenmayer (Botaniker) 1968 zur Modellierung des Wachstums von fadenförmigen Algen eingeführt

Fortgeschrittene Softwaresysteme zur Umsetzung:

- cpfg / LStudio
  http://algorithmicbotany.org/virtual_laboratory/
- Grogra http://www.grogra.de
- Graphtal
- LParser http://members.ziggo.nl/laurens.lapre/
- GroIMP http://www.grogra.de

Erstes Beispiel: Anabaena Catenula



Blaugrünes Bakterium, existiert in vier Varianten:

- „groß“: Rechts-A und Links-A
- „klein“: Rechts-B und Links-B

Beobachtungen:

- R-A teilt sich in L-A und R-B und R / L vertauscht
- R-B wird spontan zu R-A, L-B zu L-A

L-System für Anabaena Catenula

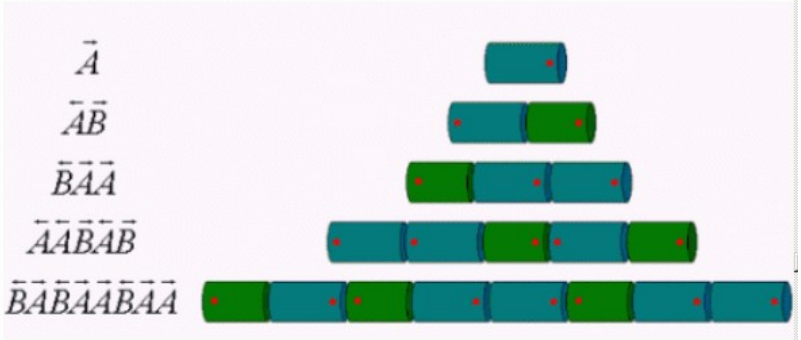
$\omega: \quad \vec{A}$

$p_1: \quad \vec{A} \rightarrow \overleftarrow{A}\vec{B}$

$p_2: \quad \overleftarrow{A} \rightarrow \overleftarrow{B}\vec{A}$

$p_3: \quad \vec{B} \rightarrow \vec{A}$

$p_4: \quad \overleftarrow{B} \rightarrow \overleftarrow{A}$



Simultane Ersetzung („parallel rewriting“)

Grundversion gut für alle Strukturen mit *lokal 1-dimensionalem Grundgerüst* (Verzweigungssysteme).

Wiederholung:
Definition:
Ein (kontextfreies, nichtparametrisches) *L-System* ist ein Tripel (Σ, α, R), darin ist

- Σ eine nichtleere Menge von Zeichen (das *Alphabet*),
- α ein Element von Σ^*, das *Startwort* oder *Axiom*,
- R eine nichtleere Teilmenge von $\Sigma \times \Sigma^*$, die Menge der *Produktionsregeln* (generative Regeln).

Ein *Ableitungsschritt* eines Wortes $\beta \in \Sigma^*$ besteht aus der Ersetzung aller Zeichen in β, die in linken Regelseiten von R vorkommen, durch die entsprechenden rechten Regelseiten. Man vereinbart: Zeichen, auf die keine Regeln anwenbar sind, werden unverändert übernommen.

"interpretierte" L-System-Abarbeitung:

$$\begin{array}{ccccccc} \alpha & \rightarrow & \sigma_1 & \rightarrow & \sigma_2 & \rightarrow & \sigma_3 \rightarrow \ldots. \\ & & \downarrow & & \downarrow & & \downarrow \\ & & S_1 & & S_2 & & S_3 \quad .... \end{array}$$

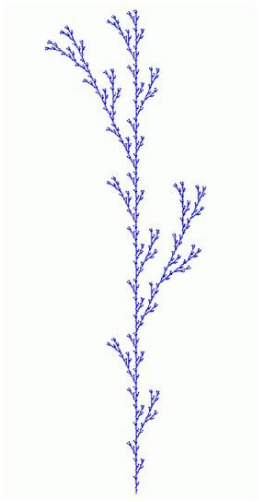
Interpretationsabbildung $\sigma_i \rightarrow S_i$: meist Turtle-Geometrie.
$S_1, S_2, S_3, \ldots$ können als Generationen oder als Entwicklungsstufen eines belebten Objekts (Pflanze, Biotop...) interpretiert werden.

Beispiele

Verzweigungsbeispiel:

```
F0 → F0 [ RU(25.7) F0 ] F0 [ RU(-25.7) F0 ] F0 ;
```

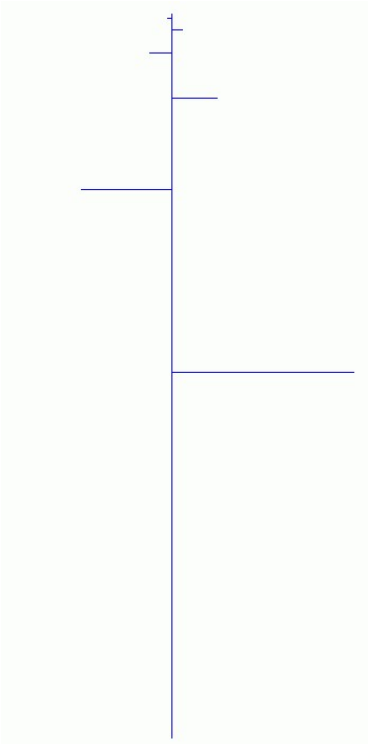
Ergebnis nach 7 Schritten:



(Startwort `L(10) F0`)

"realistischeres" Beispiel:
Verzweigung, alternierende Zweigstellung und Verkürzung.

```
Axiom ==> F0 A;
A     ==> LMul(0.5) [ RU(90) F0 ] F0 RH(180) A;
```



welche Struktur liefert

```
α → F(10) A ;

A → [ RU(-60) F(6) RH(180) A Sphere(3) ]
    [ RU(40) F(10) RH(180) A Sphere(3) ];
Sphere → Z;   ?
```

(**F(n)** liefert Linie der vorgegebenen Länge *n*,
Sphere(n) eine Kugel mit Radius *n*)

Stochastische L-Systeme: Einbau von Zufallsauswahl

Nichtdeterministische L-Systeme: ein Beispiel

$\omega: \quad F$

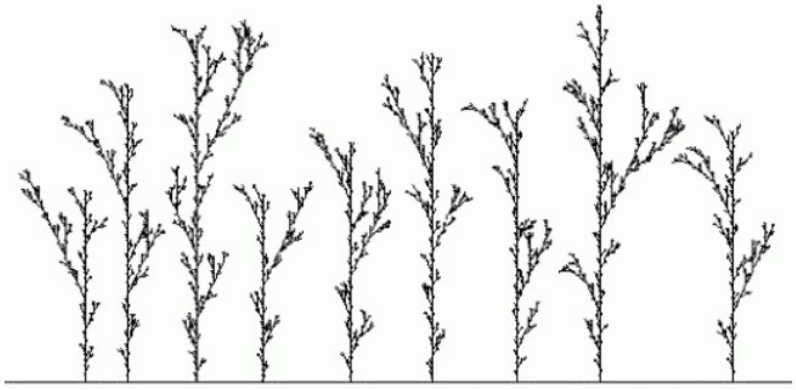
$p_1: \quad F \xrightarrow{0.33} F[+F]F[-F]F$

$p_2: \quad F \xrightarrow{0.33} F[+F]F$

$p_3: \quad F \xrightarrow{0.33} F[-F]F$

$\delta = 30°$

Regeln werden zufällig (hier mit gleichem p) ausgewählt



(in diesem Beispiel – aus Prusinkiewicz & Lindenmayer (1990) – steht "+" für "RU(δ)" und "–" für "RU(–δ)". "F" steht für "F0".)

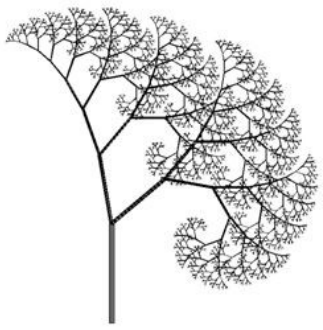
Stochastische L-Systeme

Verwendung von Pseudozufallszahlen

Beispiel:

deterministisch

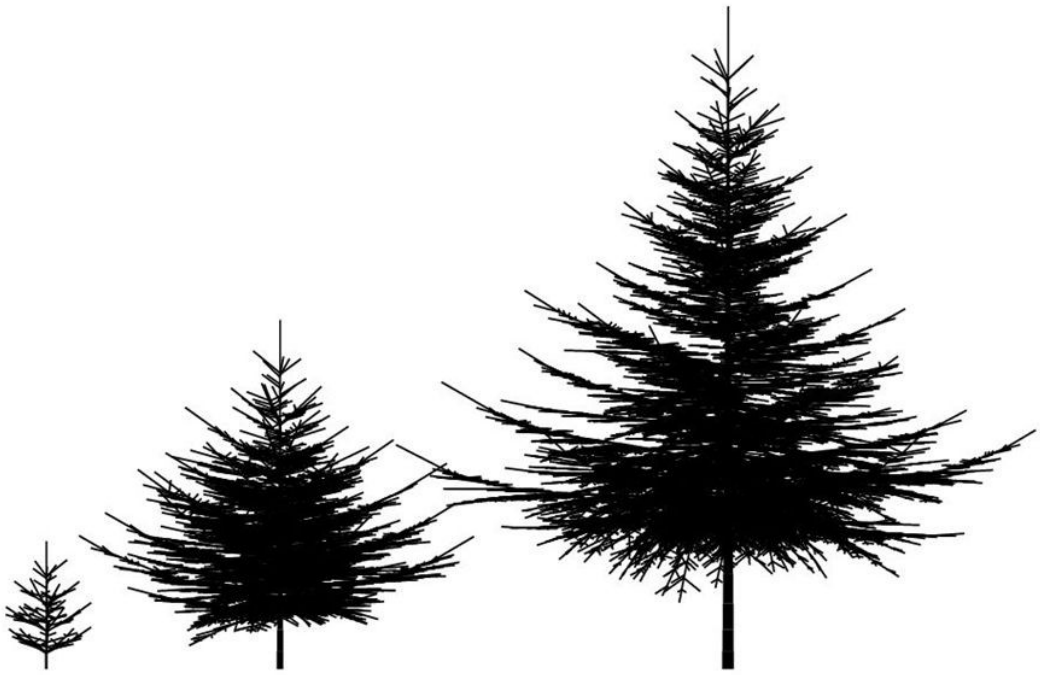
```
Start ==> L(100) D(5) A;
A ==> F0 LMul(0.7) DMul(0.7)
   [ RU(50) A ] [ RU(-10) A ];
```



stochastisch

```
Start ==> L(100) D(5) A;
A ==> F0 LMul(0.7) DMul(0.7)
  if (probability(0.5))
   ( [ RU(50) A ] [ RU(-10) A ] )
  else
   ( [ RU(-50) A ] [ RU(10) A ] );
```

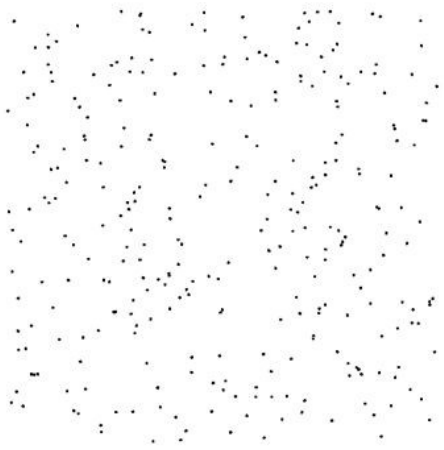
Beispiel: Fichtenmodell in 3D
mit L-System erzeugt



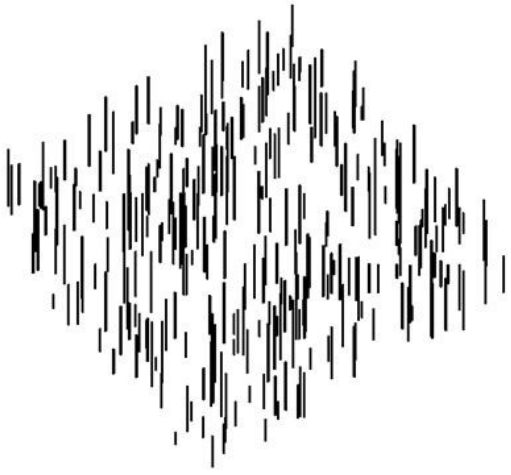
Erzeugung einer Zufallsverteilung in der Ebene:

```
Axiom ==> D(0.5) for ((1:300))
              ( [ Translate(random(0, 100), random(0, 100), 0)
                  F(random(5, 30)) ] );
```

Ansicht von oben



schräg von der Seite



Erweiterung des Symbol-Konzepts:

Lasse reellwertige Parameter nicht nur bei Turtle-Kommandos wie "`RU(45)`" und "`F(3)`" zu, sondern bei allen Zeichen

→ *parametrische L-Systeme*

beliebig lange, endliche Parameterlisten
Parameter werden bei Regel-Matching mit Werten belegt

Beispiel:

Regel `A(x, y) → F(7*x+10) B(y/2)`

vorliegendes Zeichen z.B.: `A(2, 6)`
nach der Regelanwendung: `F(24) B(3)`

Parameter können in Bedingungen abgeprüft werden
(logische Bedingungen mit Java-Syntax):

`A(x, y) (x >= 17 && y != 0) → ....`

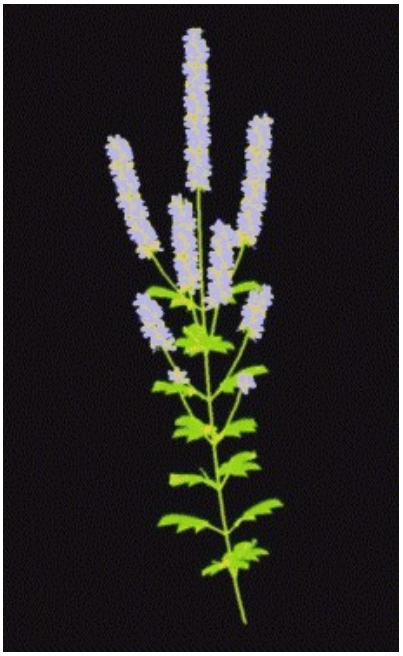
Welche Struktur wird von folgendem L-System erzeugt?

```
α    → [ RU(90) M(1) RU(90) A(1) ] A(1);
A(n) → F(n) RU(90) A(n+1);
```

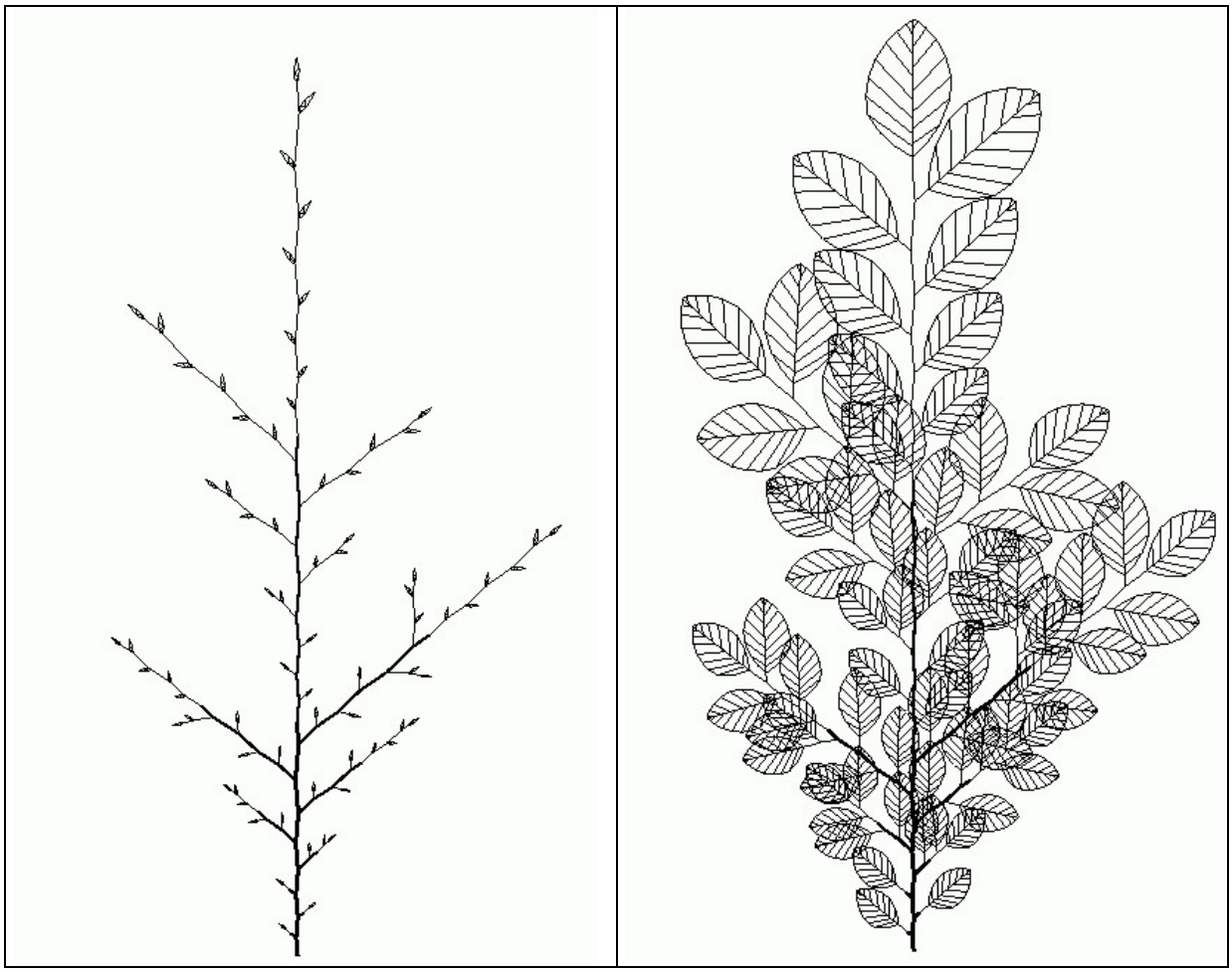
Variante:

in der zweiten Regel "`RU(90)`" etwa durch "`RU(92)`" ersetzen.

Beispiel Minze (von Prusinkiewicz & Lindenmayer):



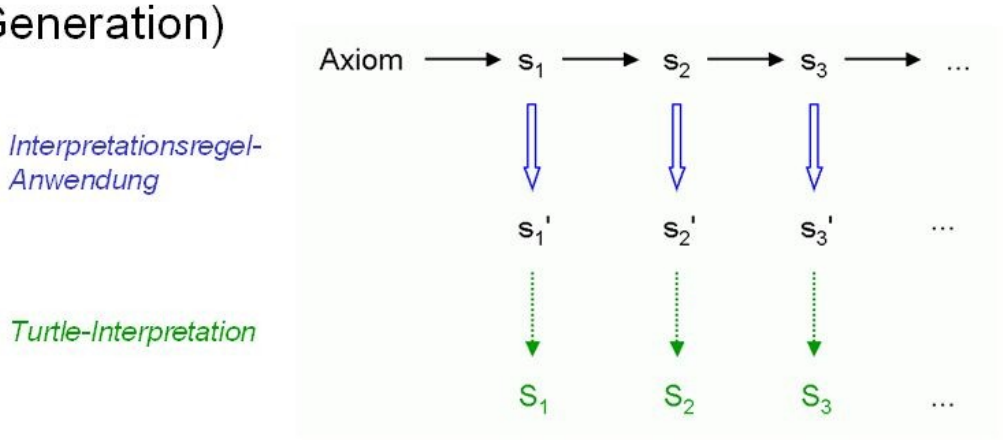
Beispiel Buchenzweige:



Nützliche Erweiterung des Formalismus:
Einführung einer zusätzlichen Regelmenge
"Interpretationsregeln"
wirken nicht auf den String der nächsten Generation
Verwendung zum Zeichnen: Vorstufe der Turtle-Interpretation
z.B. für die Knospen und Blätter beim obigen Buchenzweig

Interpretationsregeln

Einbau einer weiteren Regelanwendung unmittelbar vor der grafischen Interpretation (ohne Wirkung auf die nächste Generation)



Beispiel:

```
public void run()
{ [
   Axiom ==> A;
   A ==> Scale(0.3333) for (i:(-1:1))
                         for (j:(-1:1))
                           if ((i+1)*(j+1) != 1)
                              ( [ Translate(i, j, 0) A ] );
   ]
   applyInterpretation();
}

public void interpret()
   [
   A ==> Box;
   ]
```

```
public void run()
{ [
   Axiom ==> A;
   A ==> Scale(0.3333) for (i:(-1:1))
                         for (j:(-1:1))
                           if ((i+1)*(j+1) != 1)
                             ( [ Translate(i, j, 0) A ] );
   ]
   applyInterpretation();
}

public void interpret()
   [
   A ==> Box;
   ]
```

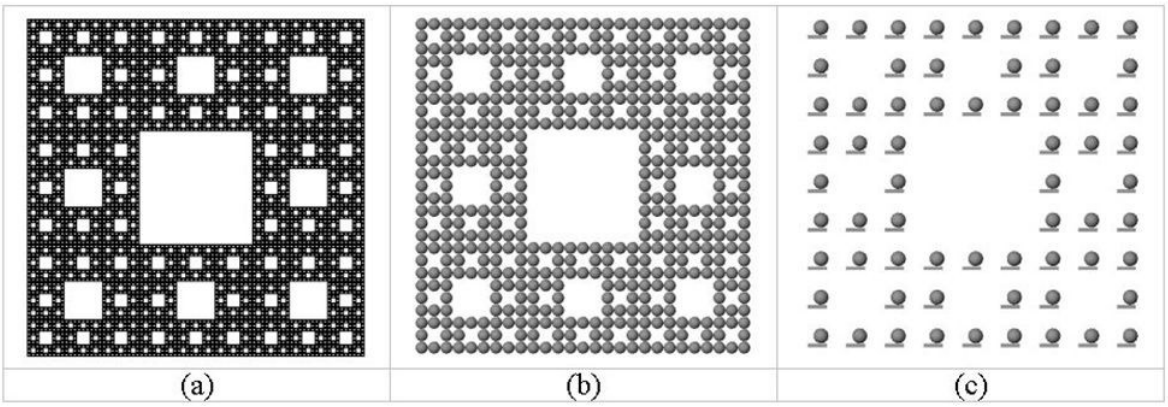
(a)

(b)

```
A ==> Sphere(0.5);
```

(c)

```
A ==> Box(0.1, 0.5, 0.1)
      Translate(0.1, 0.25, 0) Sphere(0.2);
```



(a) (b) (c)

was wird durch dieses Beispiel erzeugt?

```
public void run()
{
   [
   Axiom ==> [ A(0, 0.5) D(0.7) F(60) ] A(0, 6) F(100);
   A(t, speed) ==> A(t+1, speed);
   ]
   applyInterpretation();
}
public void interpret()
   [
   A(t, speed) ==> RU(speed*t);
   ]
```

Nachteil der bisher vorgestellten L-Systeme:
Kontrolle nur durch Vorgänger-Symbol ("lineage control") oder stochastisch

⇒ fehlende Interaktion innerhalb des modellierten Objekts oder mit der Umwelt

⇒ Determinismus oder stochastische Modelle ohne kausale Komponenten

Abhilfe: Einführung von *Sensitivität* bei der Regelanwendung.

(a) *Kontext*sensitivität

(schon altes Konzept, Beispiele bereits bei Lindenmayer...):

Abhängigkeit einer Regelanwendung vom linken und / oder rechten Kontext im String:
leftcontext < a > *rightcontext* → β.
stringbasiert!

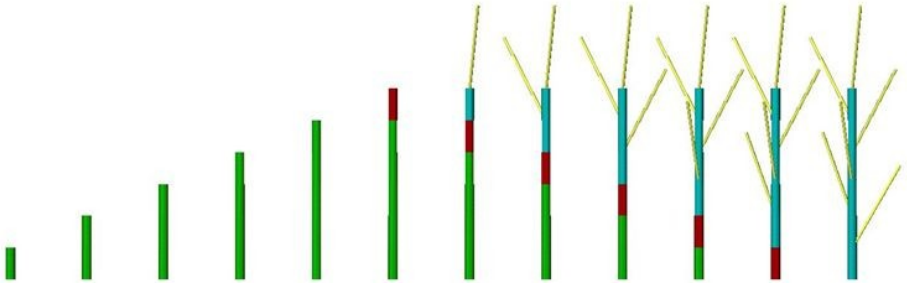
Kontextsensitivität

Abfrage eines Kontexts, der vorhanden sein muss, damit eine Regel anwendbar ist

Angabe des Kontexts in (* .... *)

Beispiel:

```
module A(int age);
module B(super.length, super.color) extends F(length, 3, color);
Axiom ==> A(0);
A(t), (t < 5) ==> B(10, 2) A(t+1);
A(t), (t == 5) ==> B(10, 4);
B(s, 2) (* B(r, 4) *) ==> B(s, 4);
B(s, 4) ==> B(s, 3) [ RH(random(0, 360)) RU(30) F(30, 1, 14) ];
```



Verwendung:

- Weiterleitung von Signalen innerhalb der modellierten Struktur

  s < a > → a s,

  s → , /* leeres Wort */
- Konzentration von Substanzen (Hormonen)
- Bewegung von Objekten (z.B. Insekten auf der Pflanze)

Beispiel: Entwicklung von Blütenständen häufig hormonal gesteuert

häufig: Zusammenwirken von 2 Signalen (von unten und von oben)

- das Modell kann Aufschluss geben, ob für eine in der Natur beobachtete morphologische Sequenz der Blütenbildung 1, 2 oder mehr Pflanzenhormone notwendig sind.

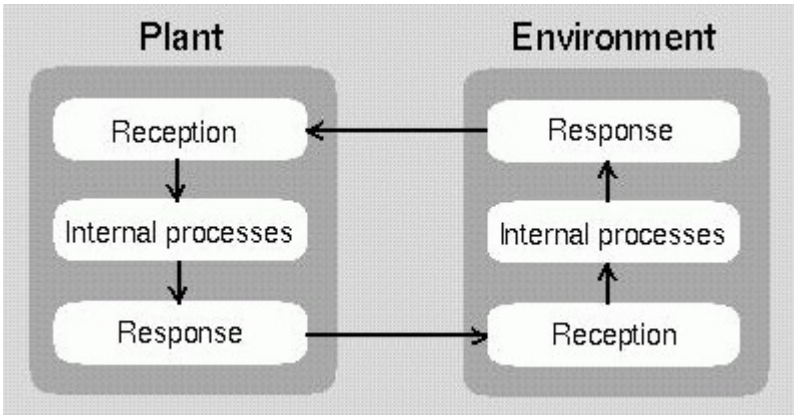
weiteres Beispiel von Prusinkiewicz & Lindenmayer (1990): *Lychnis coronaria*

(b) globale Sensitivität
(auch: *environmentally sensitive L-systems*)

- Kommunikation mit der Umgebung über spezielle Kommunikationsmodule oder über sensitive Funktionen
- Regelanwendung hängt (potentiell) von der gesamten, aktuell vorhandenen Struktur im Objektraum (und von eventuellen externen Eingriffen) ab (nicht nur von der String-repräsentation)
- Schnittstelle zu physikalisch oder biologisch basierten Simulationsmodellen (man spricht dann von *open L-systems*)

2 Herangehensweisen:

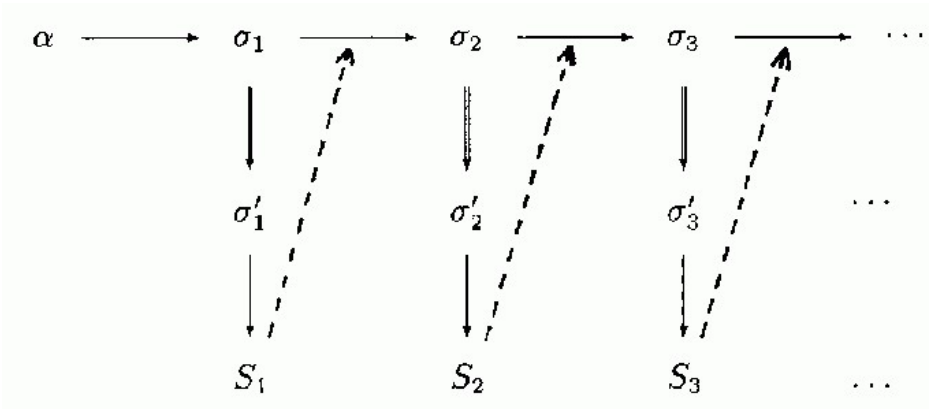
(a) scharfe Trennung von Organismus und Umgebung; der lebende Organismus wird mittels L-System modelliert, die Umgebung mittels anderer Modelle, Implementation als parallele, kommunizierende Prozesse



(Prusinkiewicz)

(b) durch Erweiterung des L-System-Formalismus Aspekte der Umgebung ins L-System "hineinholen"

Prinzip der Anwendung global sensitiver Funktionen:



gestrichelte Pfeile: Informationsfluss von der erzeugten Objektstruktur zur Regelanwendung (Regelauswahl, Parametrisierung) auf die Strings.

$\Rightarrow$ konzeptionell Annäherung an Graph-Grammatiken (dort würden nur noch S_1, S_2 usw. existieren, keine Strings mehr).

Beachte:

- kontextsensitive L-Systeme entsprechen den "selbst-regulierten Mustern" mit endogener Kontrolle,
- global sensitive Systeme den "sehenden Mustern" mit exogener Kontrolle (in der Systematik der Morphogenese-Modelle nach Adrian D. Bell).

einfaches Beispiel:

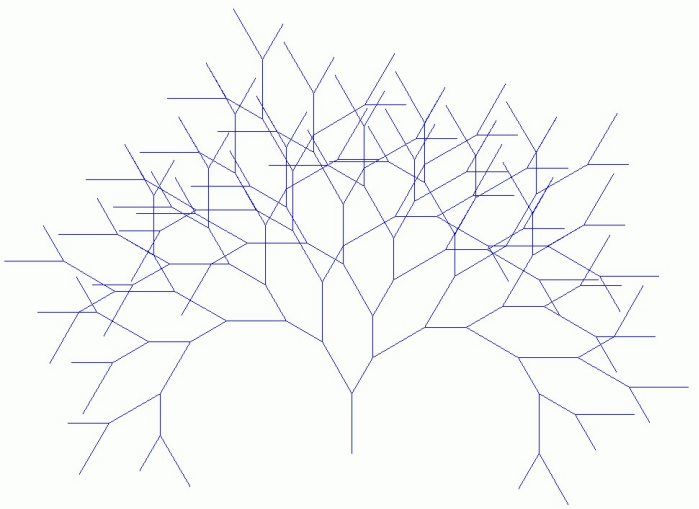
Nichtsensitives und global sensitives L-System im Vergleich

nichtsensitive, dichotome Verzweigung (Syntax von GROGRA):

```
\axiom a 1–8,
\angle 30,
a → RH180 F100 [ – b ] + a,
b → RH180 F70 [ – b ] + a
```

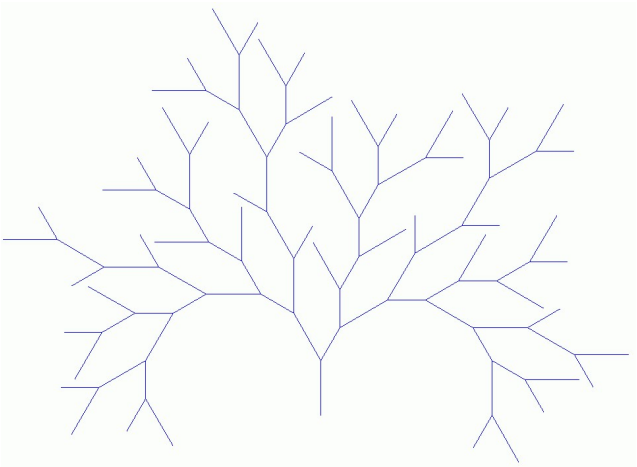
(Beispiel wird später noch einmal mit XL-Syntax aufgegriffen!)

Ergebnis:



sensitive Verzweigung mit Abhängigkeit vom Abstand zum nächsten Nachbar-Element (im Objektraum!):

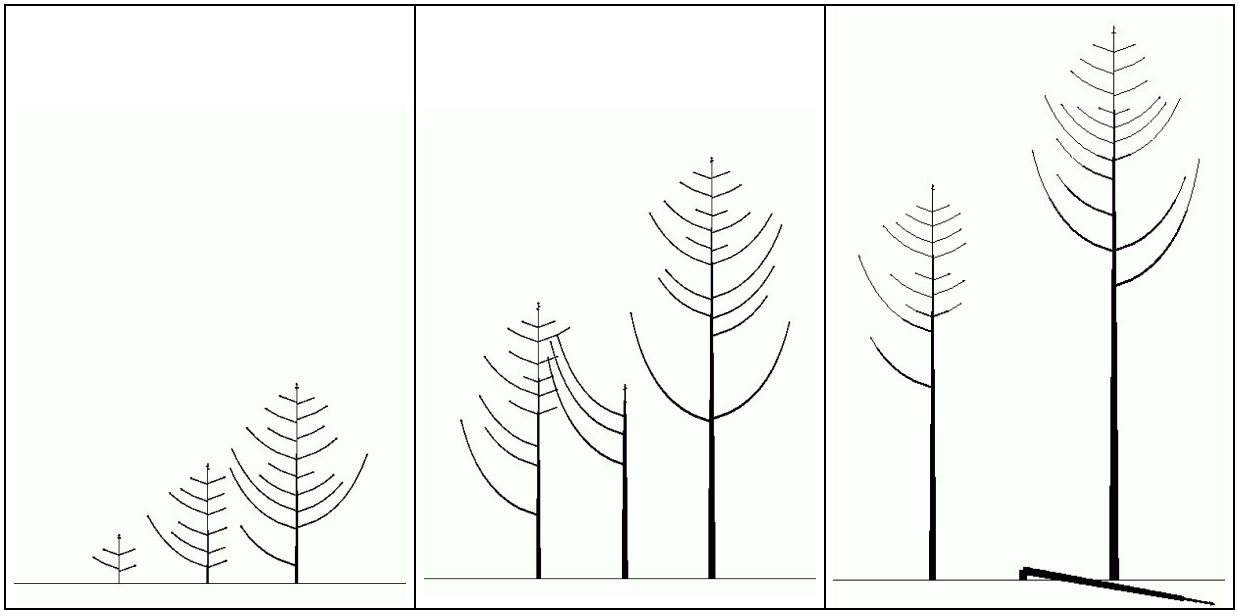
```
\axiom a 1–8,
\angle 30,
\var f function 2 1,
(f(1) > 60) a → RH180 F100 [ – b ] + a,
(f(1) > 60) b → RH180 F70 [ – b ] + a
```



Anwendung in der Pflanzenmodellierung:

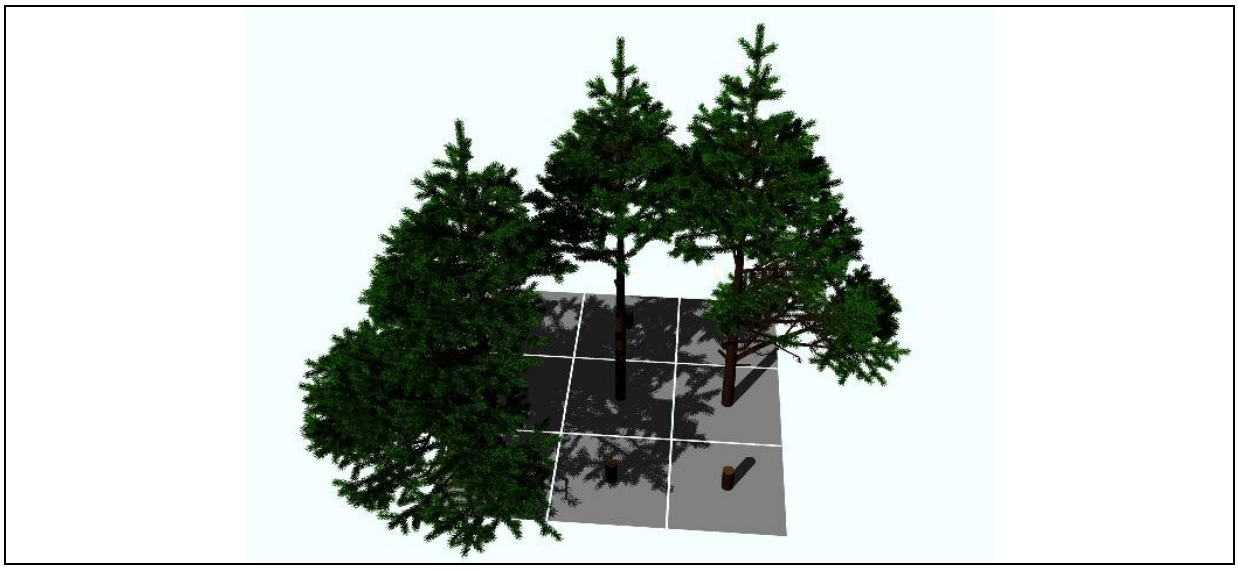
- Dichteabhängigkeit des Wachstums
- Einfluss des Neigungswinkels eines Astes auf den Neuaustrieb
- Einfluss der Beschattung
- Wechselwirkung mit Herbivoren (Tiere)

Beispiel: einfaches Überschattungsmodell (2D)



hier ist das Wachstum in jedem Apex vom "Offen-sein" eines nach oben geöffneten Kegels (bzw. Winkelfeldes) abhängig; zusätzlich werden Äste, die mehrere Schritte nicht gewachsen sind, als "tot" entfernt, und der Baum kippt um, wenn alle seine Äste tot sind.

Wachstum von Bäumen unter Konkurrenzbedingungen, mit *open L-system* realisiert
(P. Prusinkiewicz et al.)



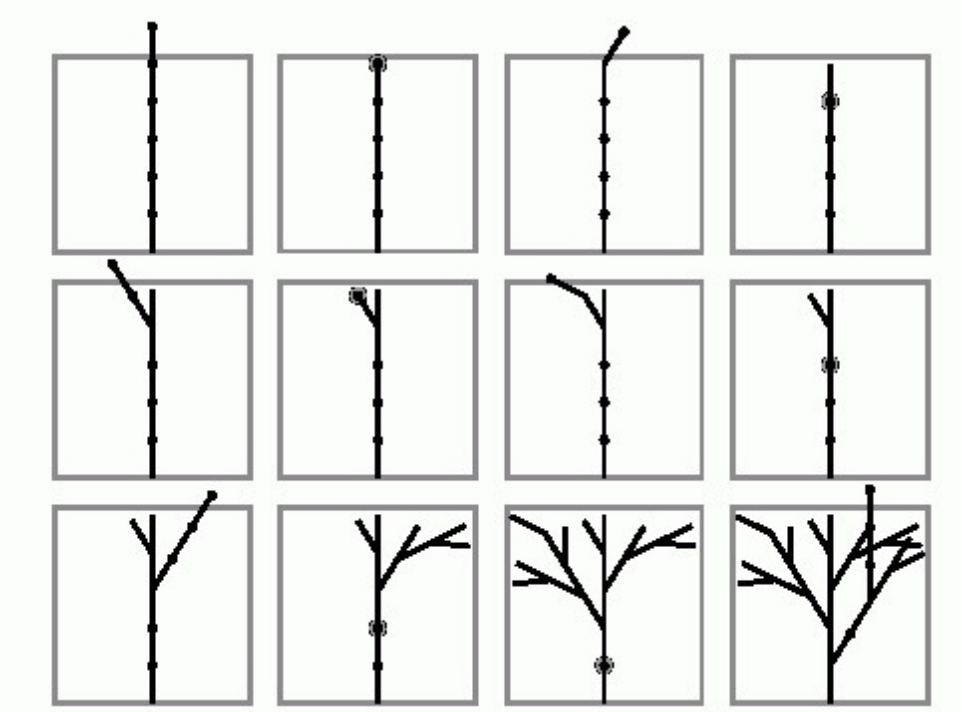
Selbstauslichtung (Astabwurf im Bestandesschatten) und asymmetrische Kronenformen als emergente morphologische Erscheinungen

weitere Beispiele später.

äußere Eingriffe: Beschneiden von Gehölzpflanzen, z.B. im Gartenbau, um eine bestimmte Form zu erzielen

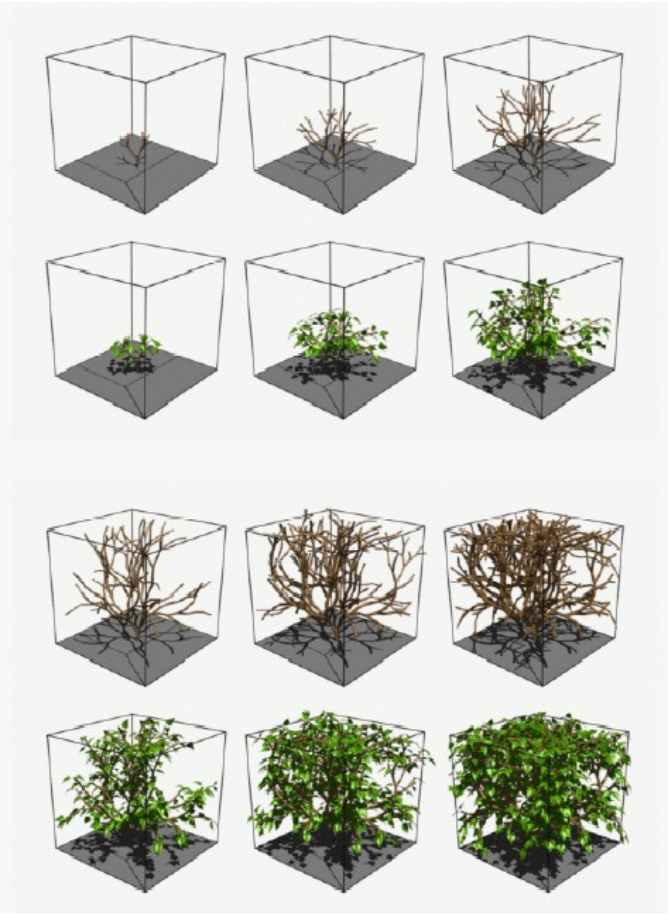
- botanischer Mechanismus: Induktion der Bildung von Ersatztrieben
- Modell: sensitives L-System, das Überschreiten einer räumlich vorgegebenen Grenze registriert und wachstumsanregendes Signal induziert, dieses bringt schlafende Knospen zum Austrieb

aus Prusinkiewicz et al. (1994):



(kleine Kreise: schlafende Knospen, großer Kreis: Wachstumssignal – wandert von oben nach unten bis zur nächsten schlafenden Knospe)

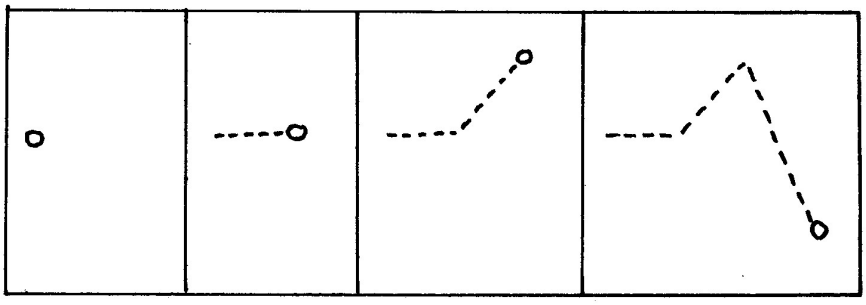
3D-Version, gerendert:



Moseley Old Hall Garden:

Beschränkung der L-System-Modelle auf ortsfeste Organismen: wie kann diese überwunden werden?

Verwendung des `f`-Befehls für Bewegungen:



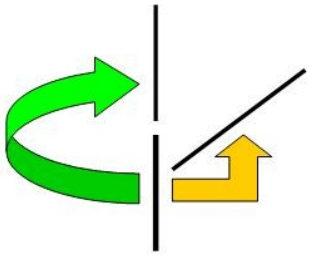
Nachteil: die Folge von "`f`"s wird im String immer mitgeführt

Abhilfe: Verwendung von "Metaregeln", die in diesem Fall z.B. zwei aufeinanderfolgende, parametrisierte `f`-Kommandos durch ihre Vektorsumme ersetzen
(bisher noch nicht in L-System-Software realisiert!)

Der Schritt zu relationalen Wachstumsgrammatiken

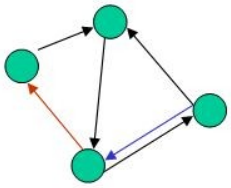
Nachteil von L-Systemen:

- in L-Systemen mit Verzweigungen (über Turtle-Kommandos) nur 2 mögliche Relationen zwischen Objekten: "direkter Nachfolger" und "Verzweigung"



Erweiterungen:

- Zulassen weiterer Relationstypen (beliebig wählbar)
- Zulassen von Zyklen (→ Graph-Grammatik)

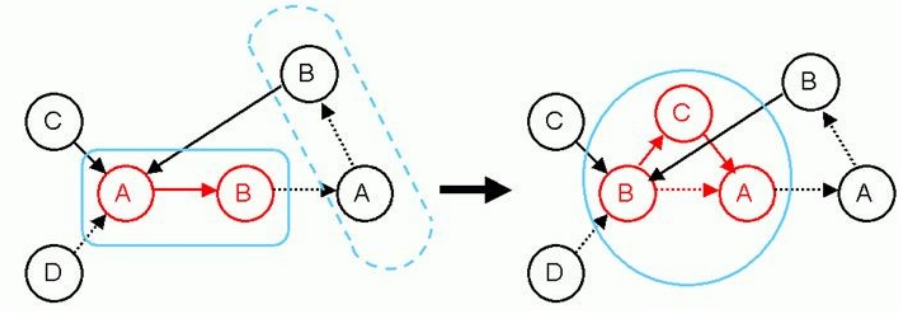


ebenfalls regelbasierter Mechanismus:

Graph-Grammatiken

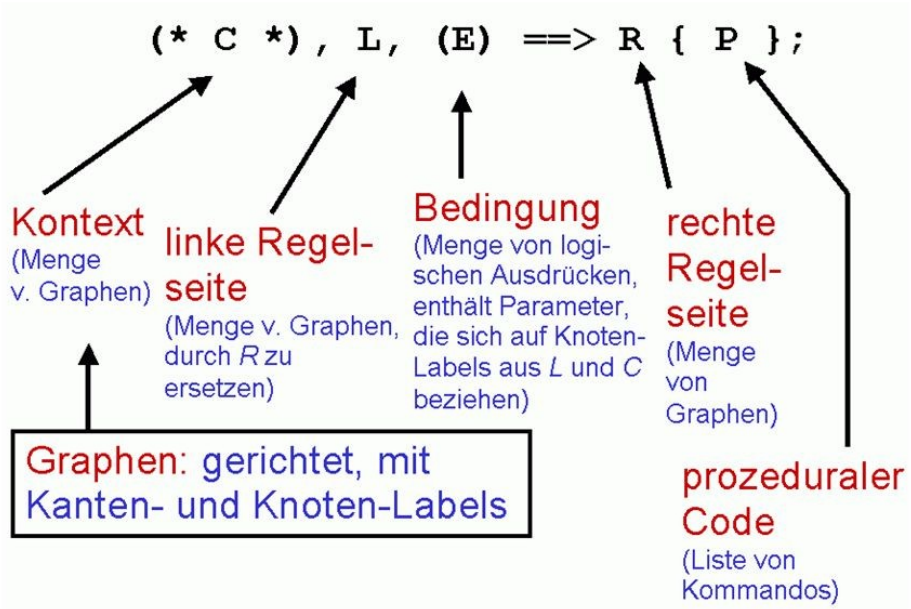
Regel:

Anwendung:



RELATIONALE WACHSTUMSGRAMMATIKEN
(RGG: Relational Growth Grammars, parallele Graph-Gramm.)

Aufbau einer Regel einer RGG:



Kanten-Markierungen repräsentieren verschiedene Arten von Relationen:

- ist Nachbar von
- enthält
- trägt
- codiert (genetisch)
- ist gepaart mit
- (...)

→auch möglich: Darstellung von multiskalierten Strukturen

Standard-Kantentypen:

successor (> oder *blank*), branch (+> oder erste Kante bei Klammern [...]), refinement (/>)

RGG als Verallgemeinerungen von L-Systemen:

Zeichenketten entsprechen speziellen Graphen

In Textform schreiben wir allgemeine (selbstdefinierte) Kanten als `-kantensorte->`

Kanten des speziellen Typs "Nachfolger" werden meist als Leerzeichen geschrieben (statt `-successor->`)

Sonderformen von RGG-Regeln:

Aktualisierungsregeln (Regelpfeil `::>`): es werden nur Parameter verändert

Beispiel: `s:Sphere ::> s[radius] += increment;`

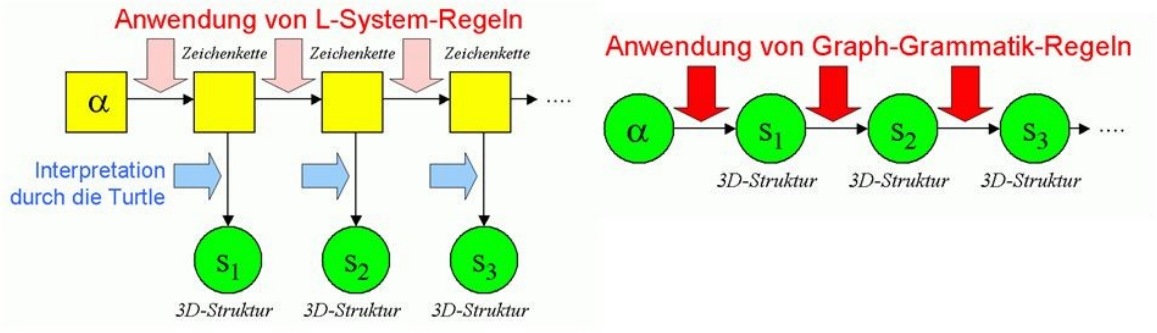
Instanzierungsregeln: einzelne Zeichen werden in Substrukturen aufgelöst, ohne Einfluss auf den nächsten Entwicklungsschritt
(Regel muss dann direkt in der Moduldeklaration stehen)

Beispiel:

```
module S(float value) extends Null ==>
   { float x = value;}
   Sphere(0.1).(setShader(new RGBAShader(1,1-x,1-x)));
```

Charakteristika von RGGs:

• Grammatik modifiziert direkt den Graphen, Umweg über String-Codierung entfällt (bzw. wird nur noch für Regel-Input gebraucht)



außerdem Nachteil der Turtle-Interpretation von L-Systemen: Segmente sind nur Zylinder, keine Objekte im Sinne der OOP

→ Erweiterungen:

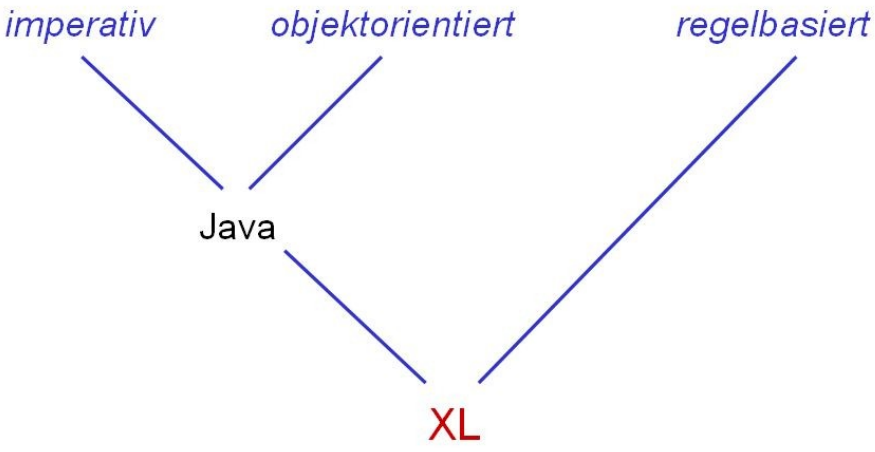
• Knoten des Graphen können beliebige Objekte sein (auch Grafikobjekte)

• Einbettung von Code einer höheren, imperativen oder objektorientierten Programmiersprache in die Regeln (für uns: Java)

Programmiersprache zur Implementation von RGG: XL

„eXtended L-system language“

Programmiersprache, die parallele Graph-Grammatiken (RGG) einfach verfügbar macht



Die Sprache XL

Sprachspezifikation: Kniemeyer (2007/08)

(Dissertation erscheint in Kürze)

- Erweiterung von Java
- erlaubt zugleich Spezifikation von L-Systemen und RGG in intuitiv verständlicher Regelschreibweise

prozedurale Blöcke, ähnlich Java: { ... }

regelorientierte Blöcke (RGG-Teil): [...]

Eigenschaften der Sprache XL:

- Knoten der Graphen sind Java-Objekte, auch Geometrie-Objekte

Beispiel: XL-Programm für die Koch'sche Kurve

```
public void derivation()
  [
  Axiom ==> RU(90) F(10);
  F(x) ==> F(x/3) RU(-60) F(x/3) RU(120) F(x/3) RU(-60) F(x/3);
  ]
```

Knoten des Graphen

Kanten (Typ „Nachfolger“)

Eigenschaften der Sprache XL:

- Knoten der Graphen sind Java-Objekte, auch Geometrie-Objekte

Spezielle Knoten:

Geometrieobjekte

Box, Sphere, Cylinder, Cone, Frustum, Parallelogram...

Zugriff auf Properties über Parameterliste (Konstruktor):

```
Box(x, y, z)
```

oder mit setter-Methoden:

```
Box(...).(setColor(0x007700))
```

Transformationsknoten

```
Translate(x, y, z), Scale(cx, cy, cz), Scale(c),
Rotate(a, b, c), RU(a), RL(a), RH(a), RV(c), RG, ...
```

Lichtquellen

PointLight, DirectionalLight, SpotLight, AmbientLight

Eigenschaften der Sprache XL:

- Knoten der Graphen sind Java-Objekte, auch Geometrie-Objekte
- Regeln in Blöcken [...] organisierbar, Steuerung der Anwendung durch Kontrollstrukturen

Beispiel: Regeln für den stochastischen Baum

```
Start ==> L(100) D(5) A;

A ==> F0 LMul(0.7) DMul(0.7)
  if (probability(0.5))
   ( [ RU(50) A ] [ RU(-10) A ] )
  else
   ( [ RU(-50) A ] [ RU(10) A ] );
```

Eigenschaften der Sprache XL:

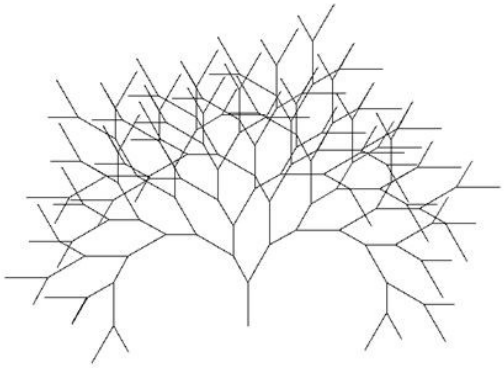
- Knoten der Graphen sind Java-Objekte, auch Geometrie-Objekte
- Regeln in Blöcken [...] organisierbar, Steuerung der Anwendung durch Kontrollstrukturen
- parallele Regelanwendung
- parallele Ausführung von Zuweisungen möglich
- Operatorüberladung (z.B. „+“ für Zahlen wie für Vektoren)
- mengenwertige Ausdrücke (genauer: Producer statt Mengen)
- Graph-Abfragen (queries) zur Analyse der aktuellen Struktur

Beispiel für Graph-query:

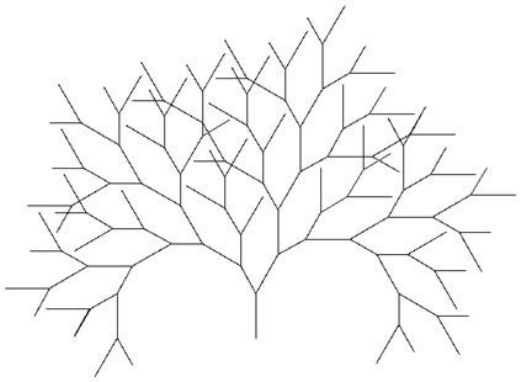
Binärer Baum, Wachstum soll nur erfolgen, wenn genügender Abstand zu anderen F-Objekten

```
Axiom ==> F(100) [ RU(-30) A(70) ] RU(30) A(100);
a:A(s) ==> if ( forall(distance(a, (* F *)) > 60) )
           ( RH(180) F(s) [ RU(-30) A(70) ] RU(30) A(100) )
```

ohne die if-Bedingung



mit der if-Bedingung



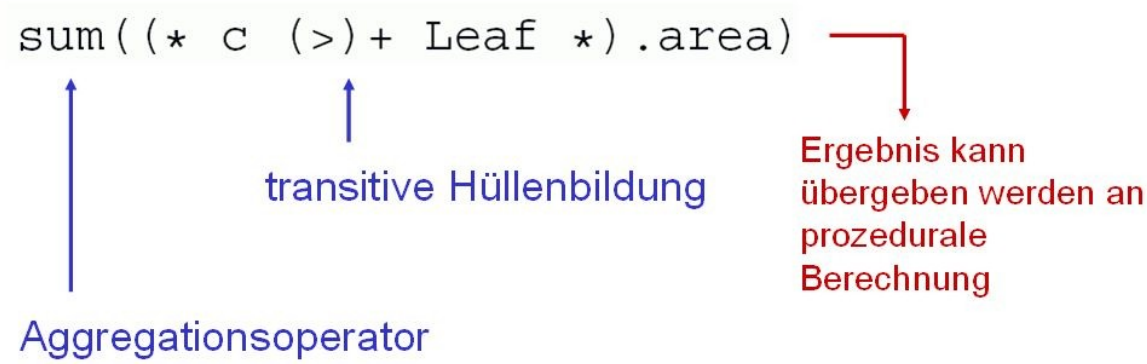
Eigenschaften der Sprache XL:

- Knoten der Graphen sind Java-Objekte, auch Geometrie-Objekte
- Regeln in Blöcken [...] organisierbar, Steuerung der Anwendung durch Kontrollstrukturen
- parallele Regelanwendung
- parallele Ausführung von Zuweisungen möglich
- Operatorüberladung (z.B. „+“ für Zahlen wie für Vektoren)
- mengenwertige Ausdrücke (genauer: Producer statt Mengen)
- Graph-Abfragen (queries) zur Analyse der aktuellen Struktur
- aggregierende Operatoren (z.B. „sum“, „mean“, „forall“, „selectWhereMin“)

Anfragen (queries) in den erzeugten Graphen

Möglichkeit der Verbindung von Struktur und Funktion

Beispiel: suche alle Blätter, die Nachfolger des Knotens `c` sind, und summiere deren Fläche



Query in einem Pflanzen- / Tier-Modell:

```
p:Plant,
(* a:Animal, (distance(a,p) < p[radius]) *)
```

sucht alle Tiere innerhalb des Radius von p

was ist von der in XL erzeugten Graph-Struktur sichtbar?

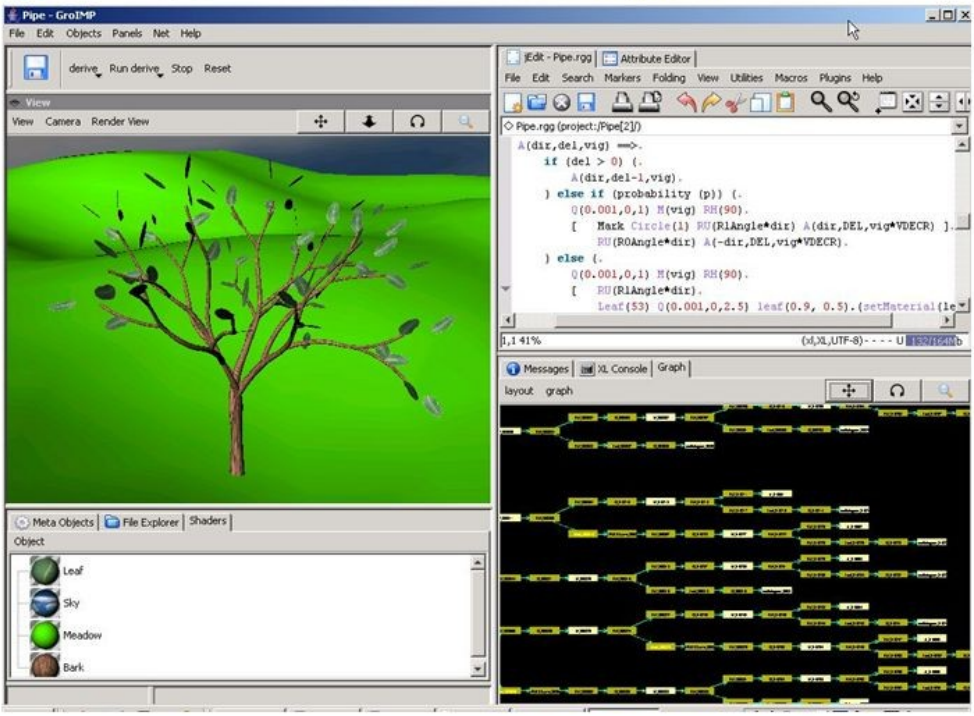
alle Geometrieknoten, die von der Wurzel (Zeichen: ^) des Graphen über genau einen Pfad, der nur aus "successor"- und "branch"-Kanten besteht, erreichbar sind.

Erzwingen, dass ein Objekt auf jeden Fall sichtbar ist:

```
==>> ^ Objekt
```

zum Arbeiten mit XL:

Interaktive 3D-Plattform ***GroIMP*** (Growth-grammar related Interactive Modelling Platform) mit XL-Compiler



- GroIMP ist ein Open Source-Projekt

http://www.grogra.de

GroIMP ist eine Kombination von:

- Graph-Grammatiken- (XL-) Interpreter
- Entwicklungsumgebung für XL
- 3D-Modeller
- 3D-Renderer (mehrere Varianten)
- 2D-Graphen-Visualisierer
- Editor für 3D-Objekte und Attribute
- Texturerzeugungswerkzeug
- X3D- (VRML-) Viewer *(in Arbeit)*

Anwendungsbeispiel: Modellierung von Parklandschaften

(Rogge & Moschner 2007, für Stiftung Branitzer Park, Cottbus)

mit GroIMP generierte Erle in VRML-Welt

Erweiterungen des L-System-Formalismus, die bisher nicht (direkt) in die Sprache XL integriert sind:

Timed L-systems (zeitverstetigte L-Systeme)

Überwindung der Beschränkung auf diskrete Zeit:

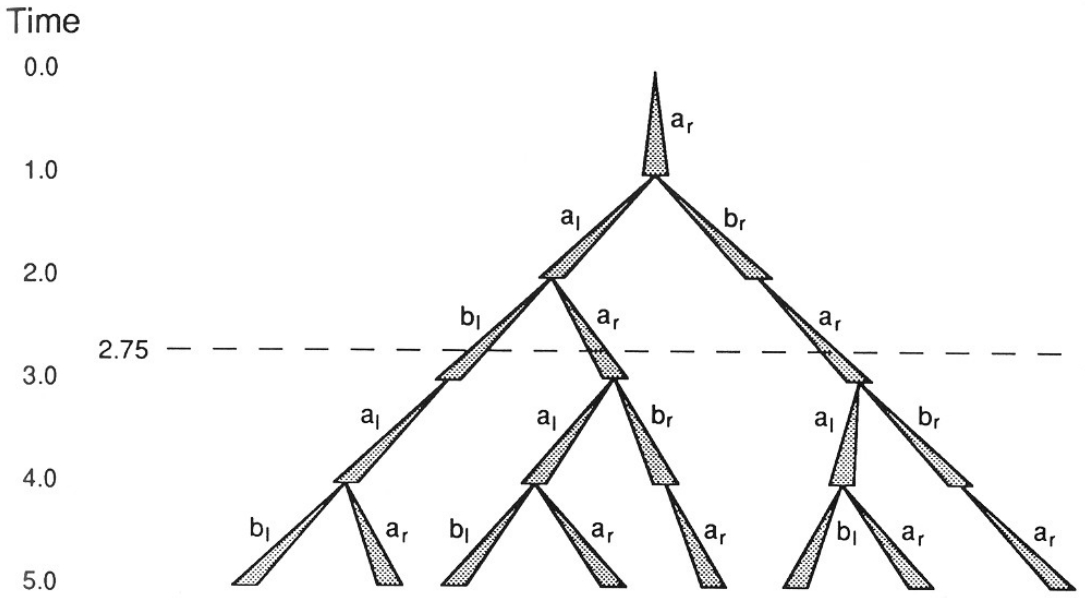
- jedes Symbol erhält als (zusätzlichen) Parameter einen Zeit-Wert
- dieser wird im Verlauf der Simulation stetig "hochgezählt"
- Produktionsregeln werden ausgelöst, wenn das "terminale Alter" eines Symbols erreicht ist

formale Def. siehe Prusinkiewicz & Lindenmayer (1990)

Beispiel:

$$
\begin{array}{lll}
\omega: & (a_r, 0) & \\
p_1: & (a_r, 1) & \rightarrow (a_l, 0)(b_r, 0) \\
p_2: & (a_l, 1) & \rightarrow (b_l, 0)(a_r, 0) \\
p_3: & (b_r, 1) & \rightarrow (a_r, 0) \\
p_4: & (b_l, 1) & \rightarrow (a_l, 0)
\end{array}
$$

(ω = Startsymbol, p_1, ..., p_4 = Produktionsregeln)



Nachteil: Simulation wird aufwändiger, ein Scheduler muss die Synchronisation überwachen

Überwindung der Beschränkung auf lokal 1-dimensionale Topologien (Verzweigungssysteme):

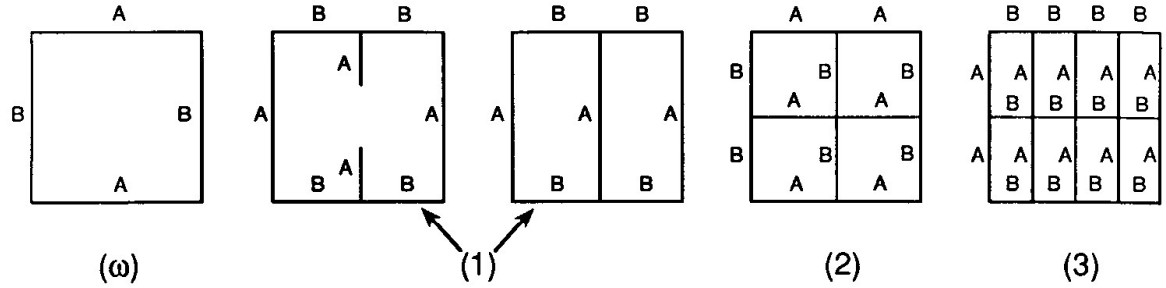
map L-systems, cellwork L-systems

verschiedene Varianten möglich
Beispiel:
Binary propagating map 0L-systems with markers (mBPM0L-systems; Prusinkiewicz & Lindenmayer 1990)

binary: eine Region kann sich in höchstens 2 Tochter-Regionen teilen
propagating: es können keine Kanten gelöscht werden
Marker: legen die Positionen neuer Kanten fest, die eine Region teilen

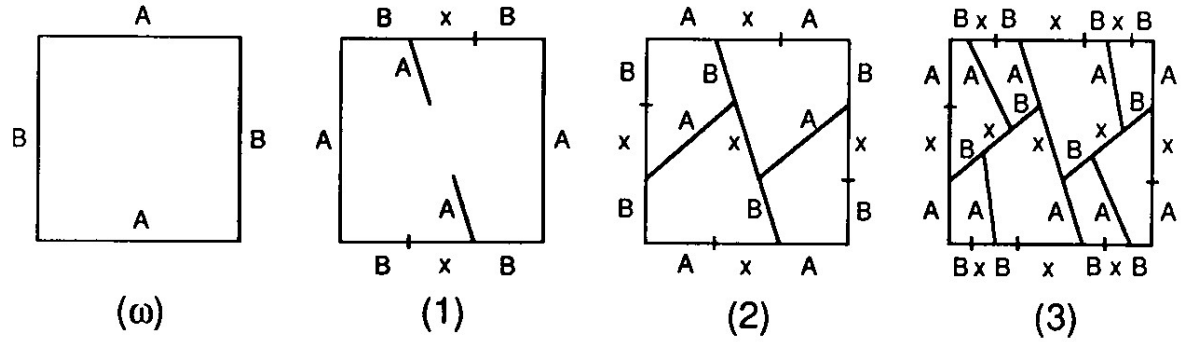
Symbole stehen für markierte, gerichtete Kanten (*edge labels*)
ein Ableitungsschritt besteht aus 2 Phasen: Ersetzung aller Kanten durch Nachfolgekanten und ggf. Marker entsprechend den Produktionsregeln, und Verbinden von Markern, die "matchen", zu neuen Kanten.

Beispiel:
Startwort ABAB,
Regeln A → B [− A] [+ A] B und B → A



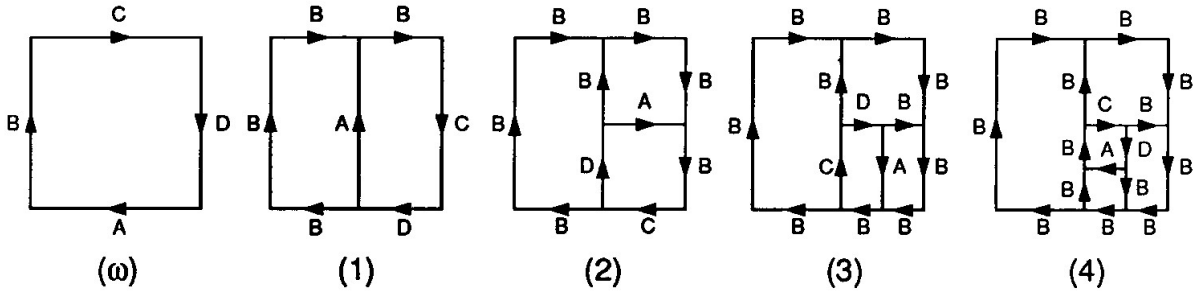
ω : ABAB
p_1 : A → B[-A][+A]B
p_2 : B → A

2. Beispiel:



ω : ABAB
p_1 : A → B[-A]x[+A]B
p_2 : B → A

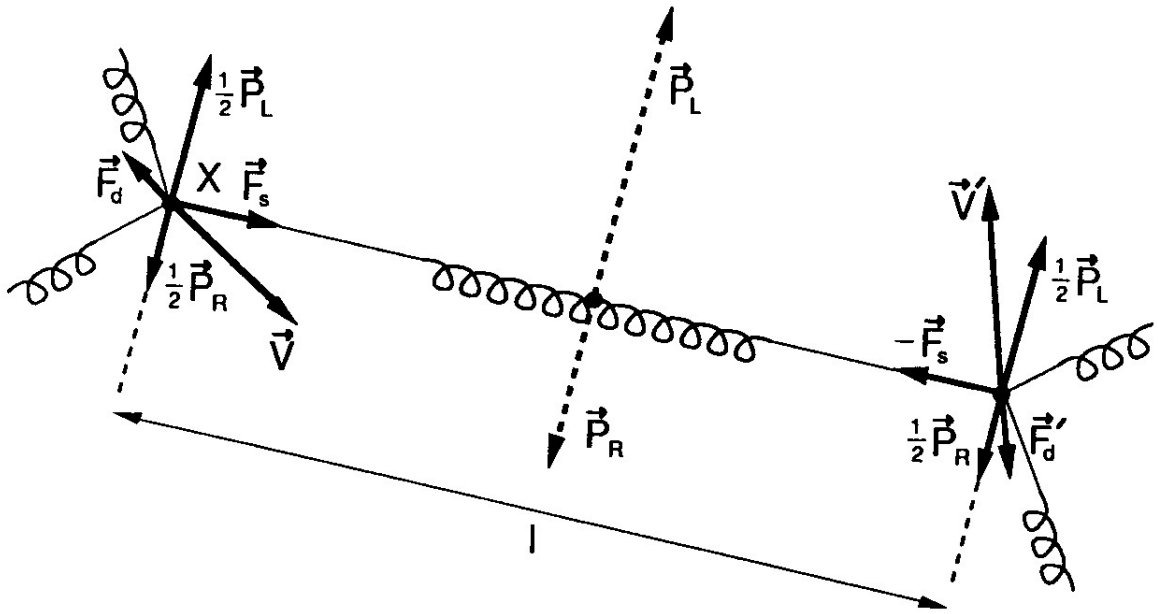
3. Beispiel (mit gerichteten Kanten):



ω : $\vec{A}\vec{B}\vec{C}\vec{D}$
p_1 : $\vec{A}$ → $\vec{D}[-\vec{A}]\vec{B}$
p_2 : $\vec{B}$ → $\vec{B}$
p_3 : $\vec{C}$ → $\vec{B}[-\overleftarrow{A}]\vec{B}$
p_4 : $\vec{D}$ → $\vec{C}$

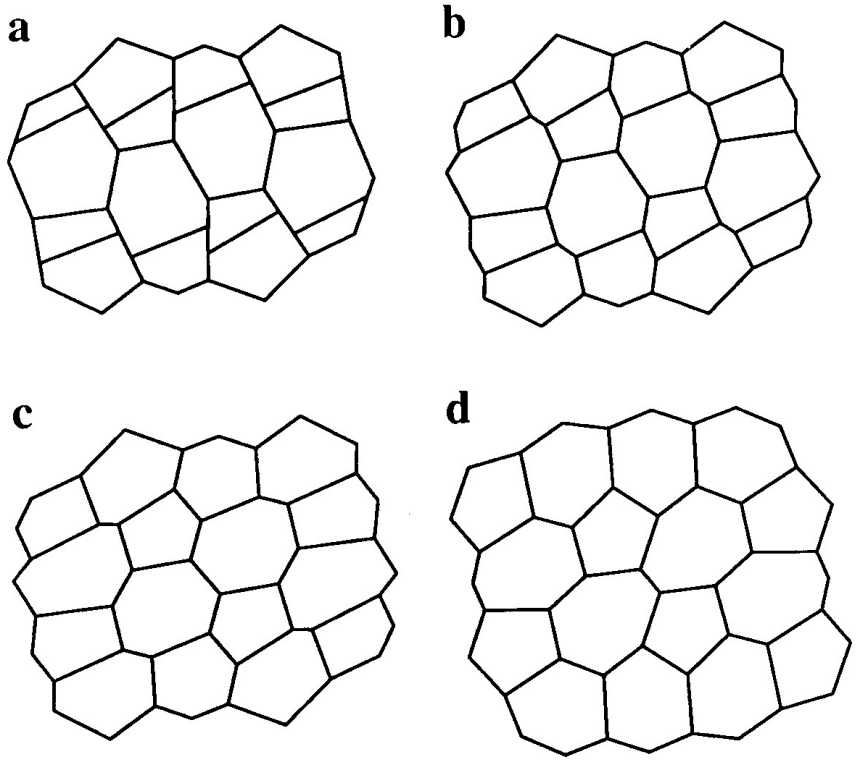
Problem: Geometrie der erzeugten Strukturen ist durch die Regeln nicht festgelegt
(anders als bei gewöhnlichen L-Systemen mit Turtle-Interpretation)

ein möglicher Ansatz: dynamisches Modell, das Kräfte annimmt, die jede Zelle auf ihre Wände (Kanten) ausübt



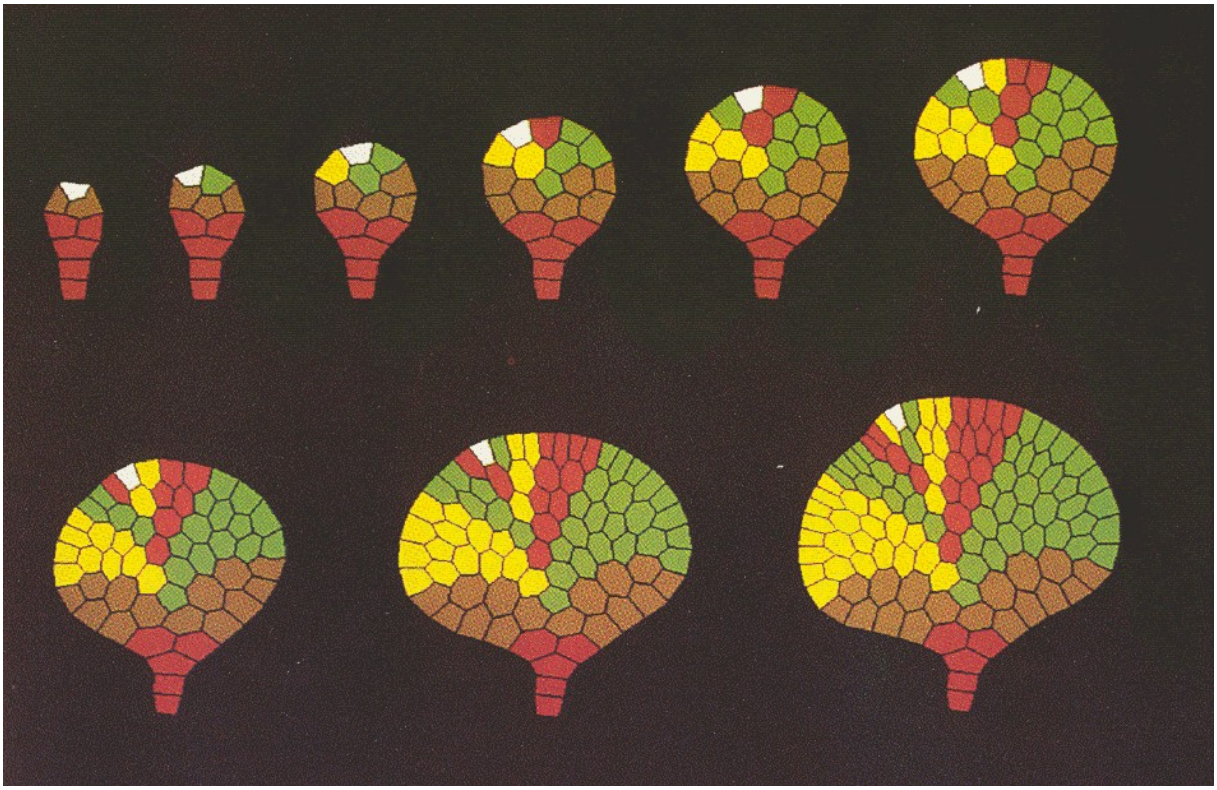
$\Rightarrow$ führt auf System von Differentialgleichungen,
Lösung führt auf Gleichgewichtszustand

Beispiel (Iterationsschritte bei Lösung):

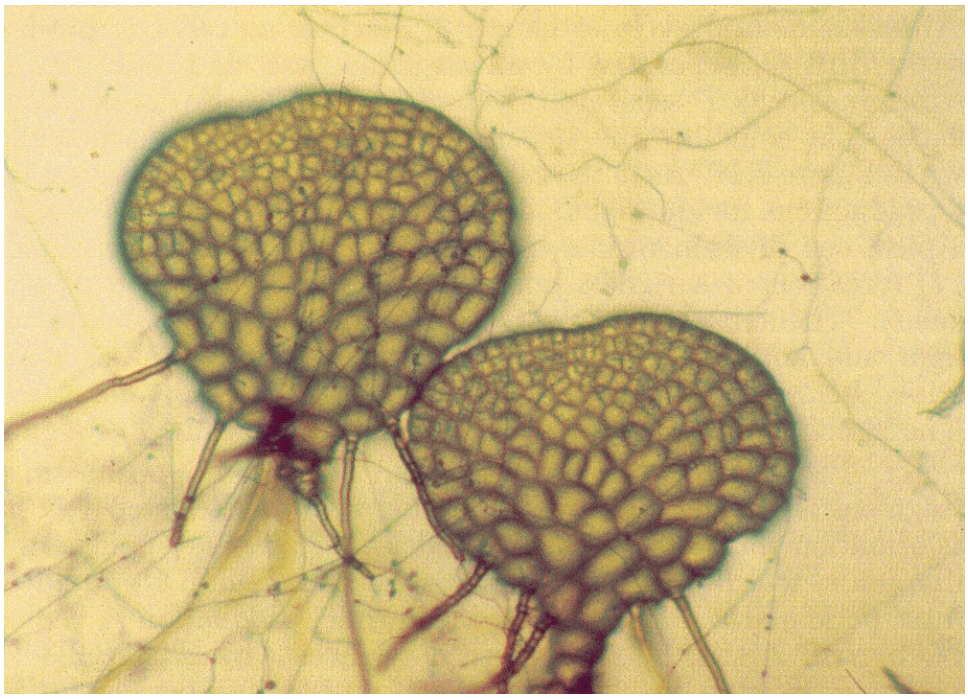


Anwendung: Simulation von Mikroorganismen
hier: *Microsorium linguaeforme* (Farn-Gametophyt; Modell beruht auf botanischen Daten von de Boer; s. Prusinkiewicz & Lindenmayer 1990)

simulierte Entwicklung:

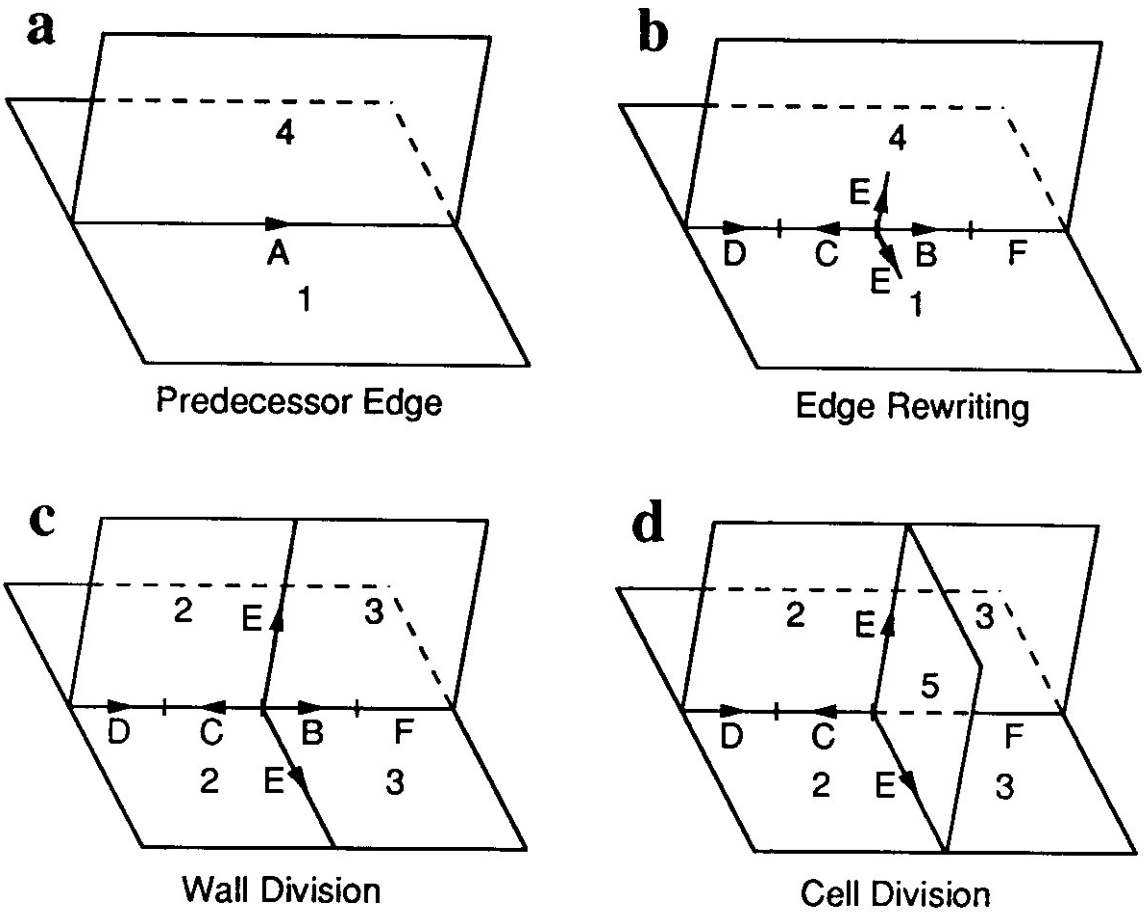


Mikrofotografie der realen Organismen:

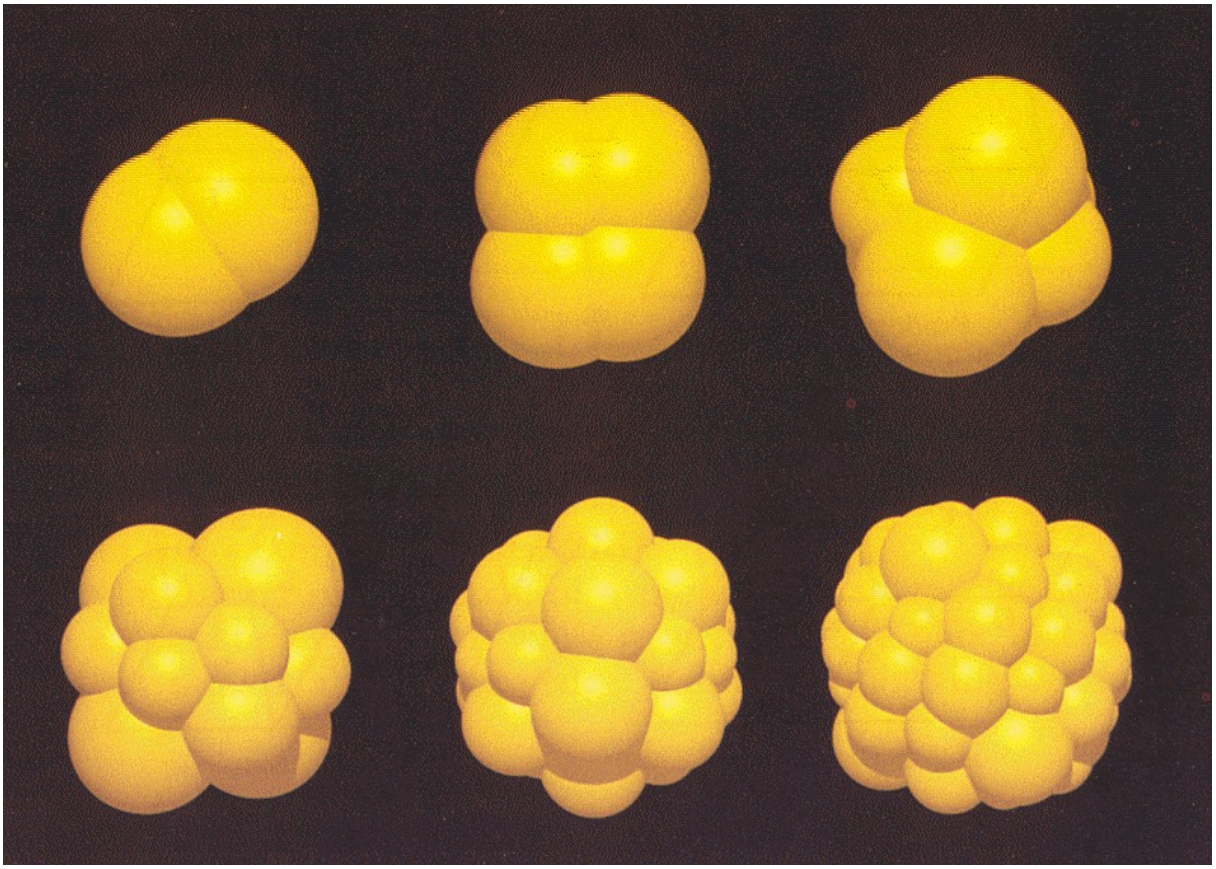


3D-Variante: *Cellwork L-systems*

- Produktionsregeln wirken ebenfalls auf die Kanten
- 3 Phasen: Rewriting, Teilung der Zellwände, Teilung der Zellen (jeweils muss *matching* geprüft werden)



Beispiel: Modell des Schnecken-Embryos *Patella vulgata*



zum Vergleich: elektronenmikroskopische Aufnahme des realen Embryos