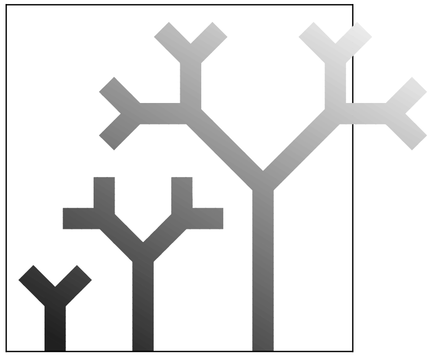
Developmental Systems



Biological systems

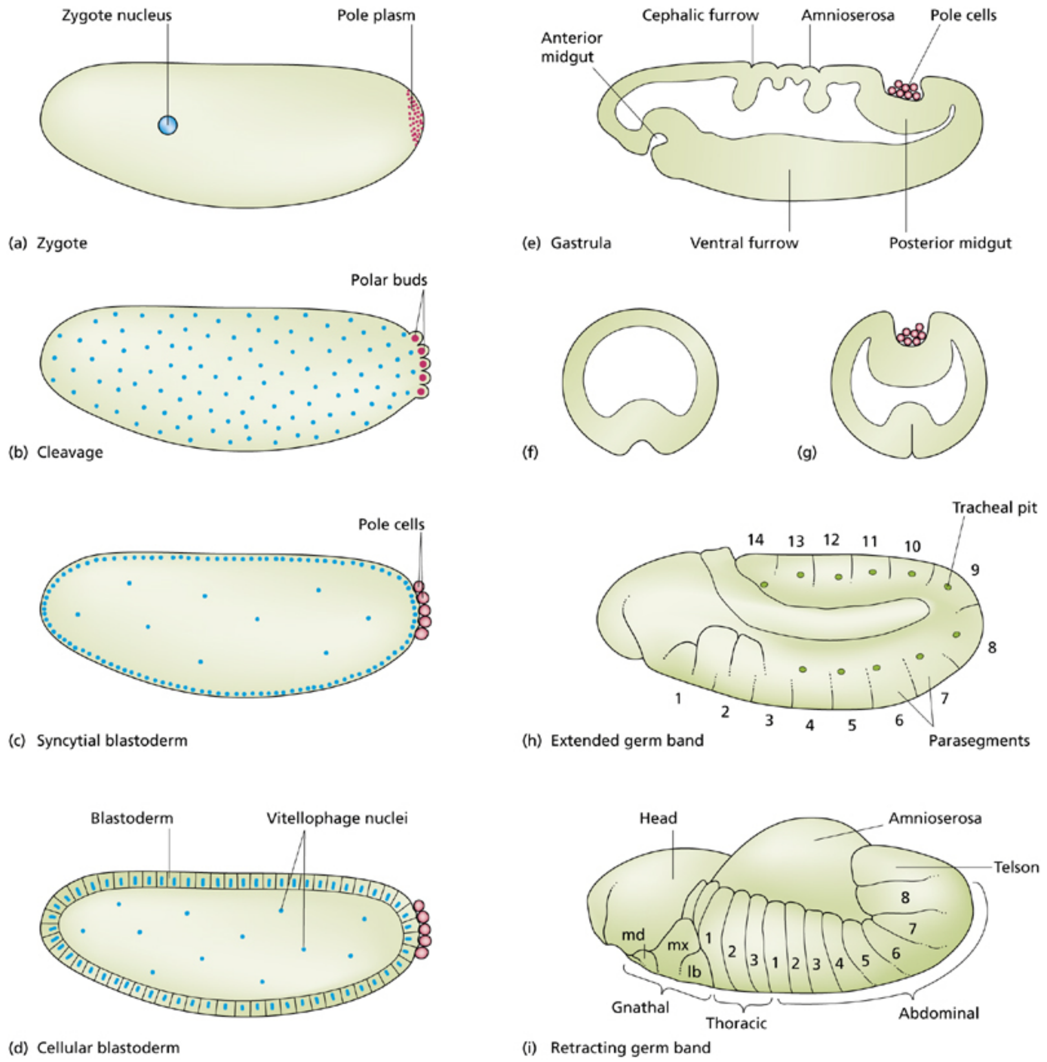
Early development of the *Drosophila* fly

http://flybase.bio.indiana.edu

dorsal view

lateral view

Biological systems



Early development of *Drosophila* [Slack 2006]

Artificial Developmental Systems

Artificial developmental systems attempt to capture mechanisms of growth of biological systems. In nature, growth is given by a process of cell duplication and differentiation. In artificial systems, it is often based on a process of iterated symbol rewriting.

Advantages of development in artificial systems:

- Complex structures can be described by few symbols and rules
- Symmetric and modular structures can be easily defined
- Development can be scalable, robust, and adaptable (sensitivity to environmental context)

Disadvantages of development in artificial systems:

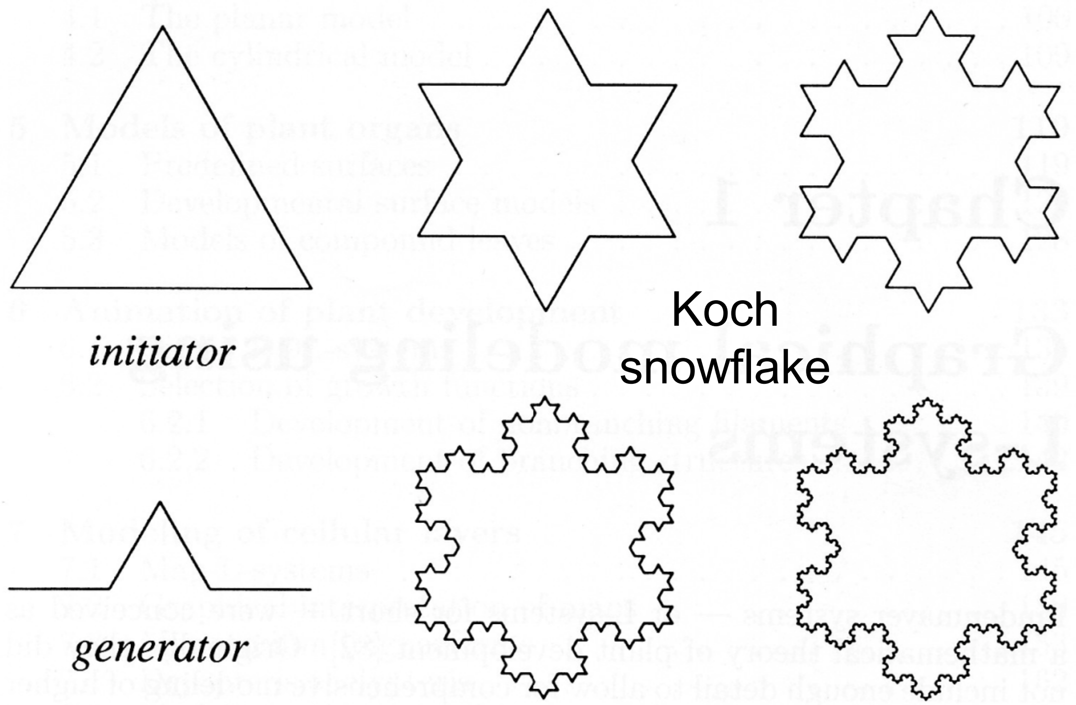
- It can be difficult to introduce irregularities and asymmetries
- It can be difficult to design rules that generate a desired structure

Rewriting systems

Introduction

Rewriting is a technique for defining complex objects by successively replacing parts of a simple initial object using a set of rewriting rules, or production rules.

Fractal curves can be generated by replacing the edges of a polygon with open polygons [von Koch, 1905]. At each iteration, the polygon is rescaled.



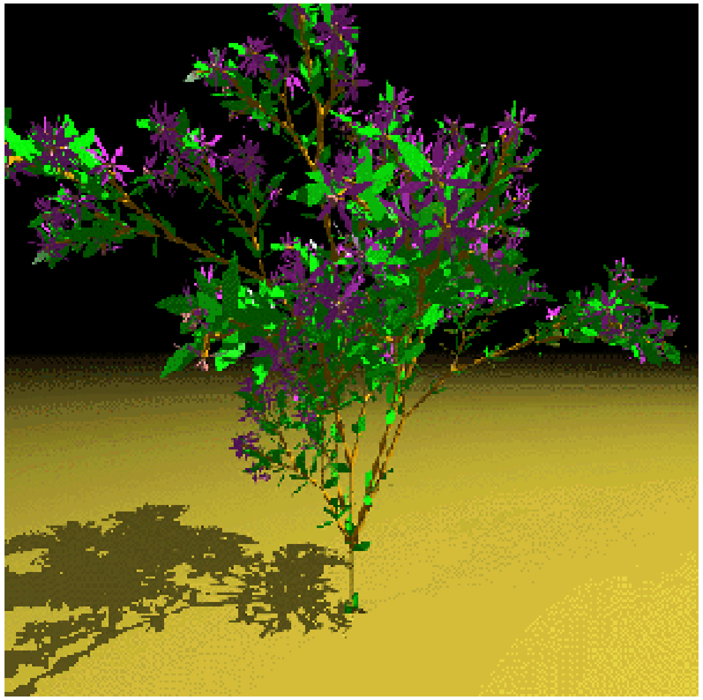
Several types of rewriting systems have been developed. These include *L-systems*, variations of *cellular automata*, and *language systems*.

L-systems

Lindenmayer systems, or L-systems for short, were conceived as rewriting systems to model organism development.

They represent a powerful formalism to model plant development [Lindenmayer, 1968].

Aristid Lindenmayer


Artificially generated tree

http://local.wasp.uwa.edu.au/~pbourke

L-system: Definition

L-systems are rewriting systems that operate on symbol strings.

An L-system is composed of:

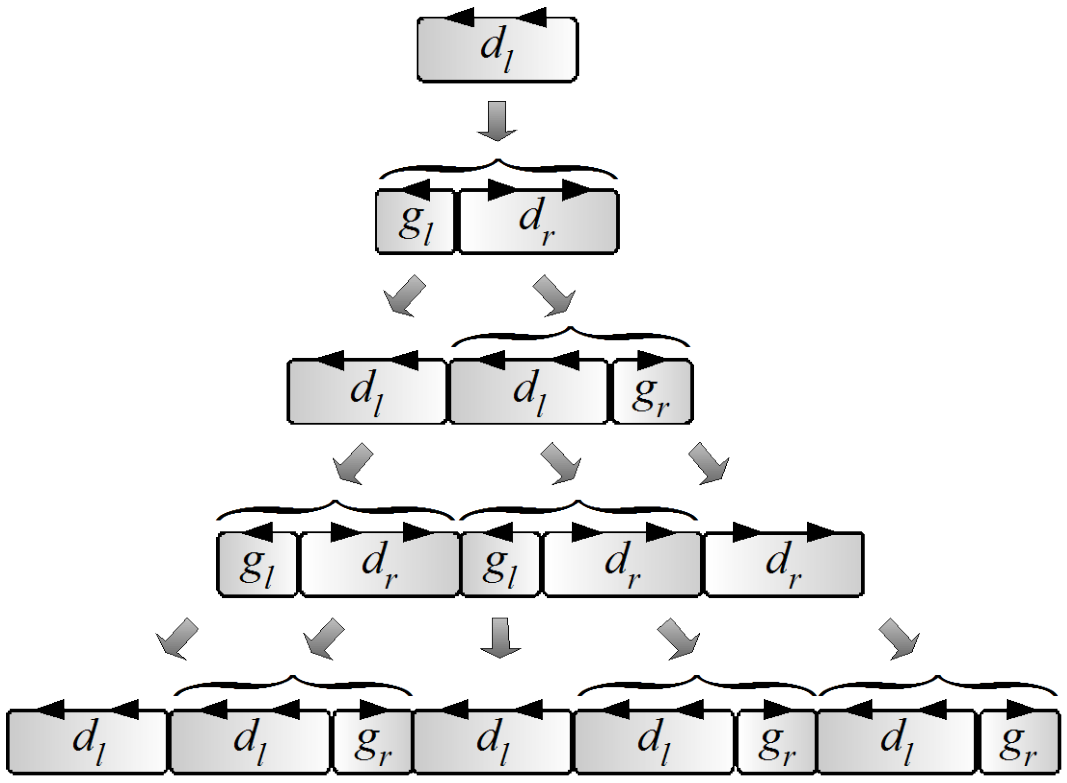
1. A set of symbols forming an *alphabet* A
2. An *axiom* ω (initial string of symbols)
3. A set $\pi = \{p_i\}$ of *production rules*.

The following assumptions hold:

1. Production rules are applied in parallel and replace recursively all symbols in the string.
2. If no production rule is specified for a symbol s, then we assume the identity production rule $p_o : s \rightarrow s$.

L-system: 1D Example

Development of a multicellular filament of blue-green bacteria *Anabaena catenula* [Lindenmayer 1968]


d_l
g_l d_r
d_l d_l g_r
g_l d_r g_l d_r d_r
d_l d_l g_r d_l d_l g_r d_l g_r

Cells can be in a "growing" state g or in a "dividing" state d with left or right polarity

$$A = \{g_r, g_l, d_r, d_l\}$$

$$\omega = d_l$$

$$p_1 = d_r \rightarrow d_l\, g_r$$

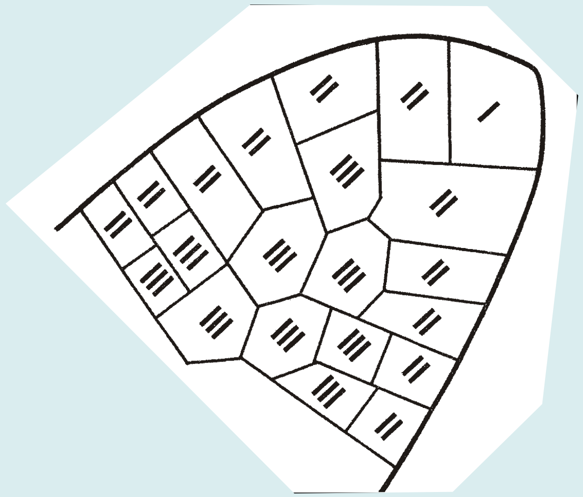
$$p_2 = d_l \rightarrow g_l\, d_r$$

$$p_3 = g_r \rightarrow d_r$$

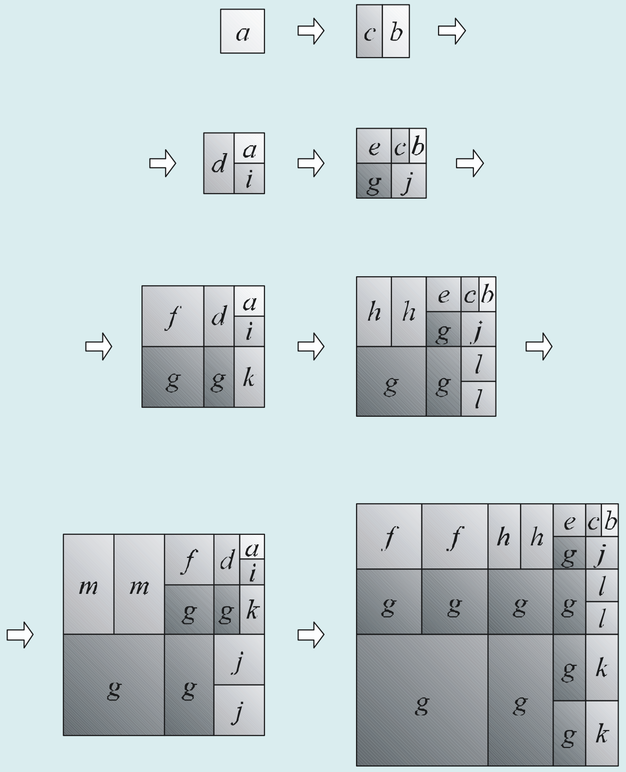
$$p_4 = g_l \rightarrow d_l$$

L-system: 2D Example

Development of moss leaves [Lindenmayer 1975]



Biological development according to Nägeli [1845], showing primary, secondary, and tertiary cells.


a
c b
d a i
e c b g j
f d a i g g k
h h e c b g j g g l l
m m f d a i g g k g g j j
f f h h e c b g j g g g g l l g g g k g k

$A = \{a, b, c, \ldots D, R\}$

$\omega = a$

$p_1 = a \rightarrow c\,R\,b$

$p_2 = b \rightarrow a\,D\,i$

$p_3 = c \rightarrow d$

$p_4 = e \rightarrow f$

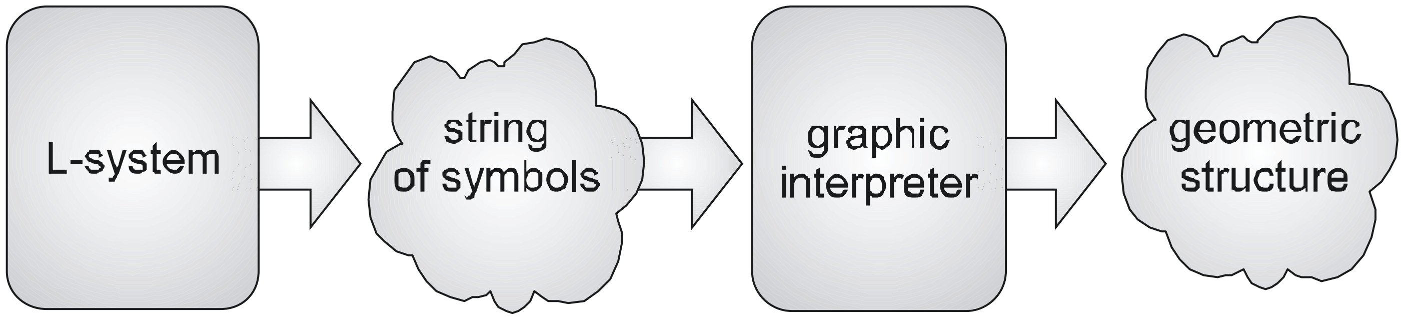
$\ldots$

$p_{13} = m \rightarrow f\,D\,g$

Lindenmayer's model

Graphics Interpretation

- Using symbols that represent directly geometric entities such as 1D or 2D cells becomes rapidly impractical.
- We can increase the graphic potential of L-systems by following the phase of production of strings of symbols with a phase of graphic interpretation of the strings



Turtle Graphics Interpretation

Prusinkiewicz [1986] gave L-systems a 2D and 3D graphic interpretation based on LOGO-style turtle geometry.

In 2D the state of the turtle is defined as a triplet (x, y, α) where the Cartesian coordinates (x, y) represent the turtle's position and the angle α, also known as heading, represents the direction in which the turtle is facing.

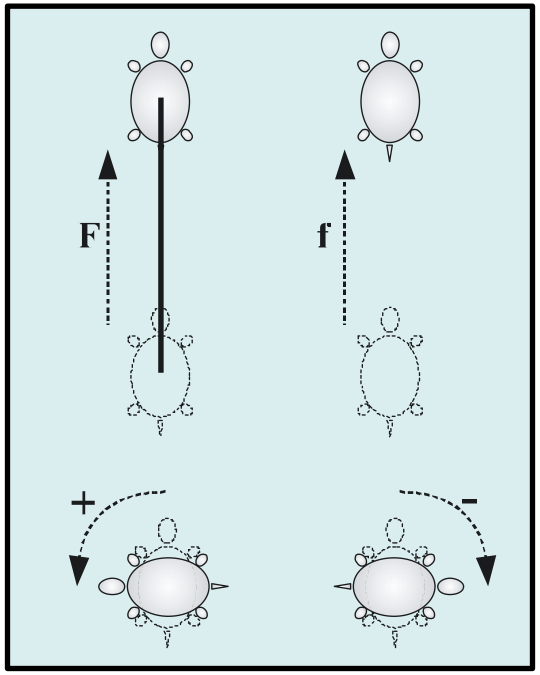
Given the step size d and the angle increment δ, the turtle can respond to the following commands:

F : move forward by a step while drawing a line.

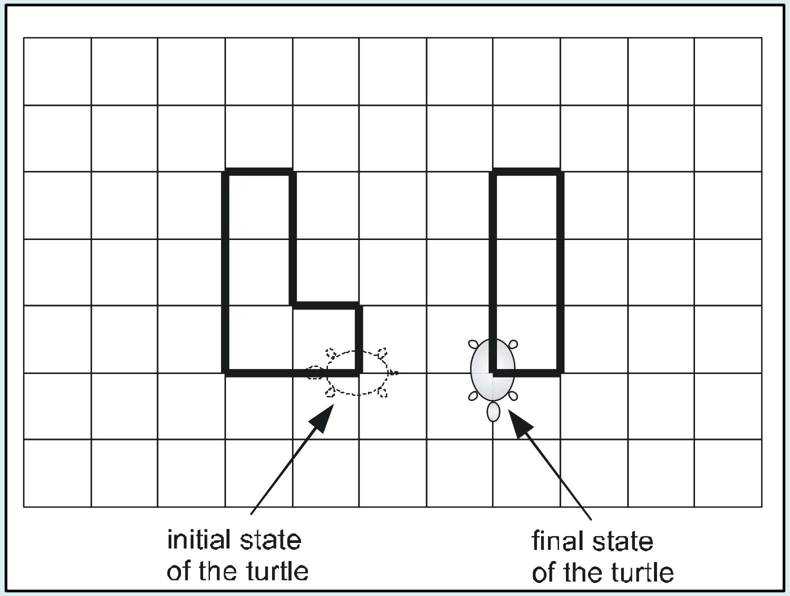
f : move forward by a step without drawing a line.

+ : turn left (counterclockwise) by angle δ.

– : turn right (clockwise) by angle δ.

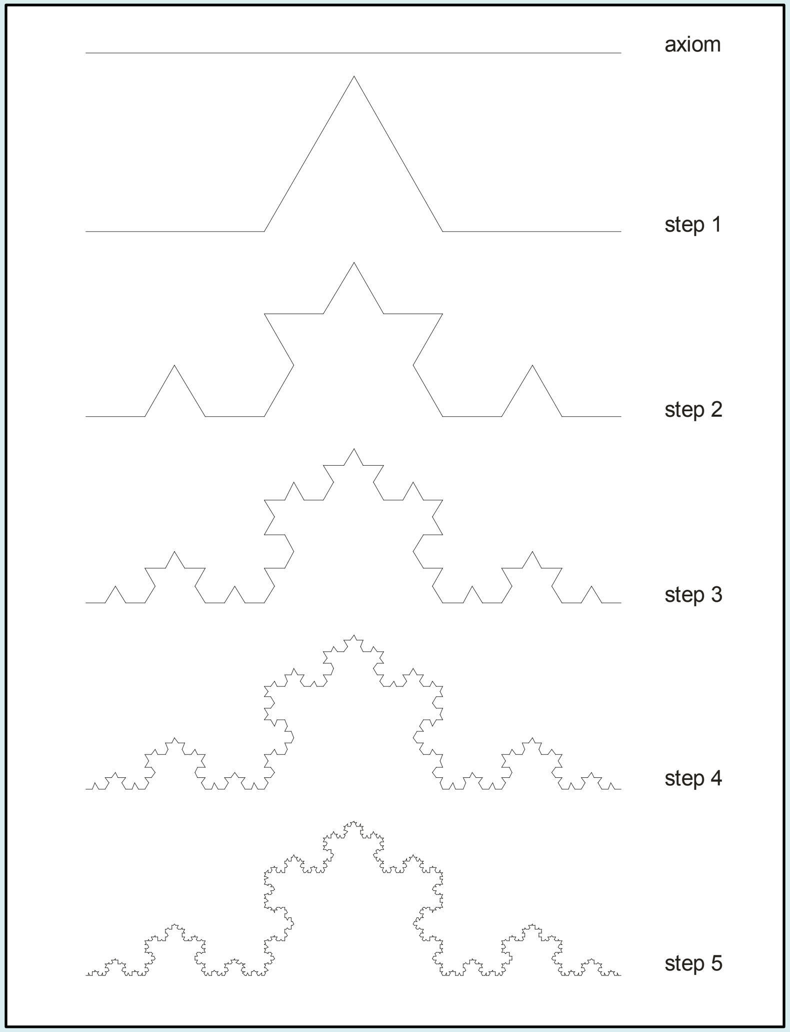


Turtle at Work



$\delta = 90°$, $A = \{ F, f, +, - \}$

$\omega = FF - FFF - F - FF + F - F + f f F + FFF + F + FFF$

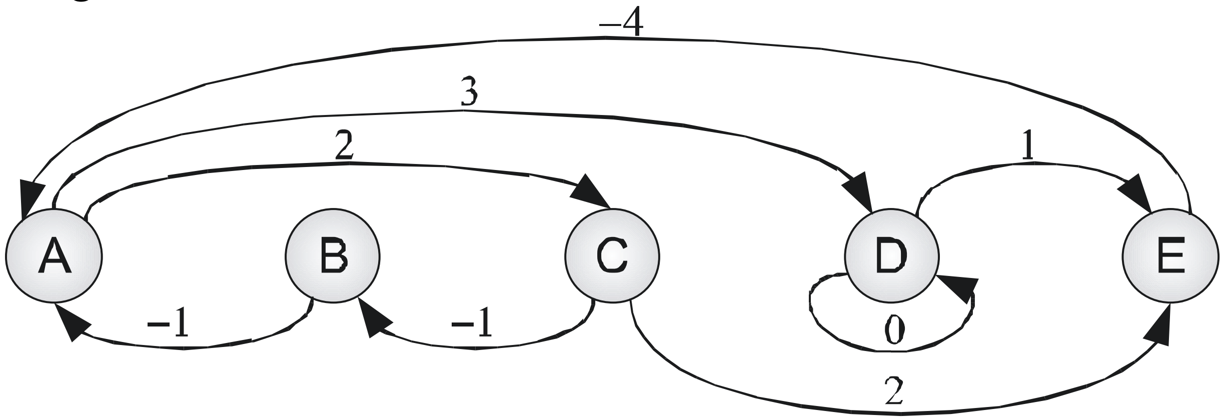


$\delta = 60°$, $A = \{ F, f, +, - \}$, $\omega = F$

$p = F \rightarrow F+F--F+F$

Graph Interpretation

- Turtle graphics interpretation is not well suited to the definition of circuits and networks (the paths must be closed “manually”).
- Boers and Sprinkhuizen-Kuyper [2001] proposed the *graph interpretation*, where the L-system alphabet contains symbols for nodes N (typically, characters) and symbols for links L (typically, integers)
- Example: the string *“A 2 3 B −1 C −1 2 D 0 1 E −4”* becomes the following network

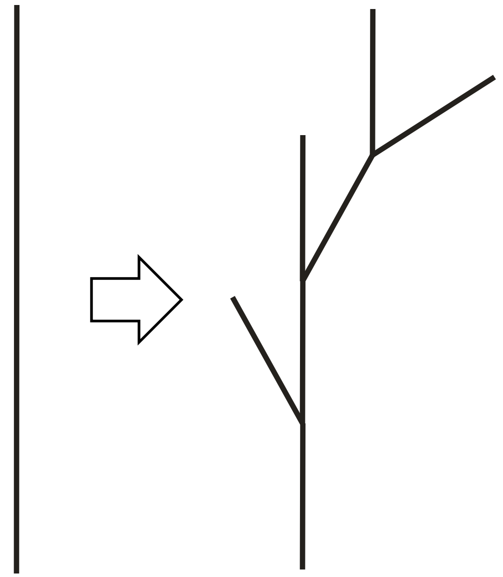


Bracketed L-systems

- In drawing branching structures using the turtle interpreter it is necessary to reposition the turtle at the base of a branch after the drawing of the branch itself
- Bracketed L-systems facilitate this task
- Two new symbols are defined

[	Save current state of the turtle (position, orientation, color, thickness, etc.).
]	Restore the state of the turtle using the last saved state (no line is drawn).

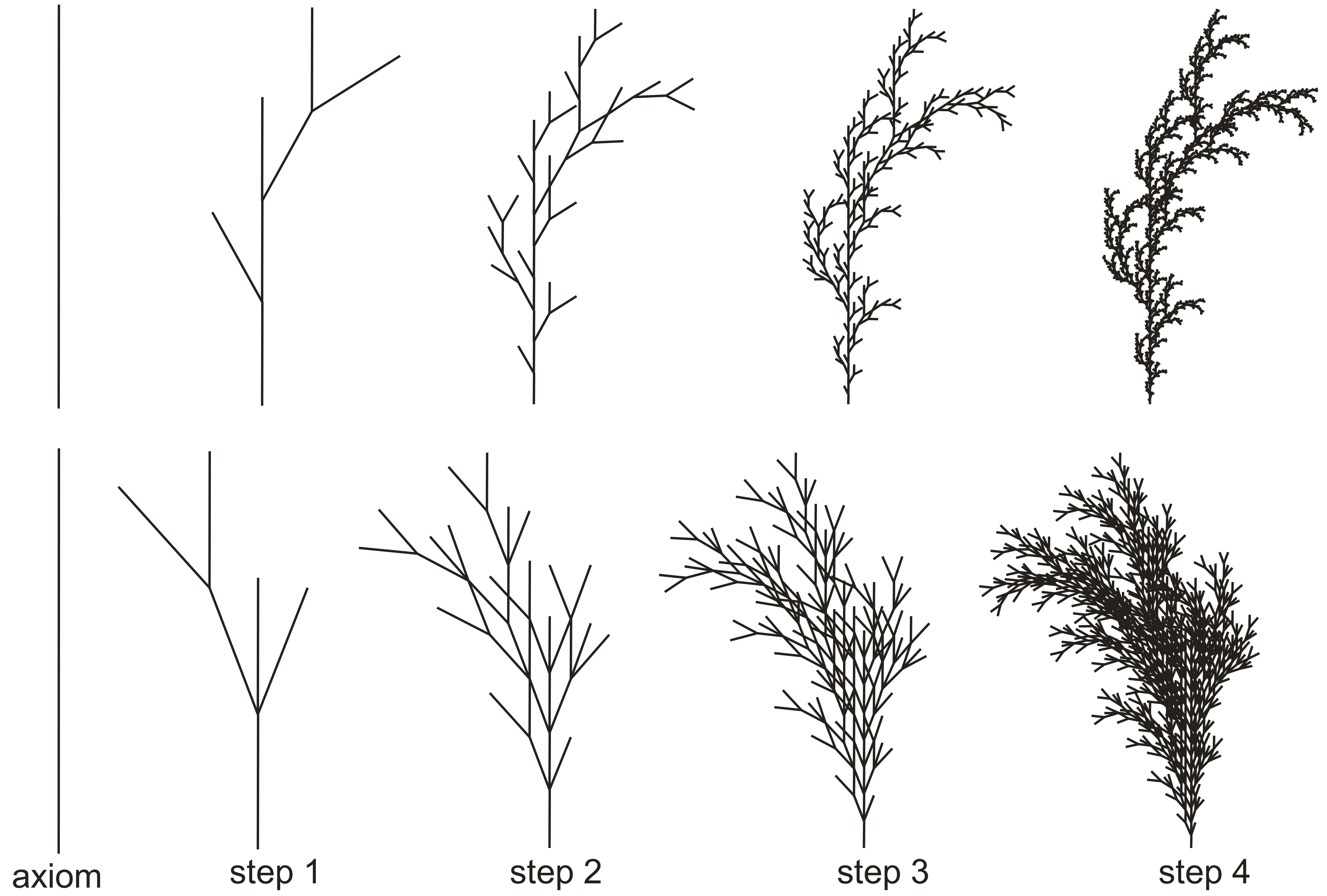
- Bracketed L-systems are also useful to define hierarchical networks using the graph interpreter



$\delta = 29°$, $A = \{ F, +, -, [,] \}$

$\omega = F$

$p = F \rightarrow F\,[+F]F\,[-F\,[+F][-F]]F$


axiom
step 1
step 2
step 3
step 4

Stochastic L-systems

- All plants generated by the same L-system are identical, but in nature individuals are not identical.
- Specimen-to-specimen variation can be modeled by introducing production probabilities. For every symbol, there is one or more production rules with an associated probability. The sum of all probabilities over the same symbol must be 1

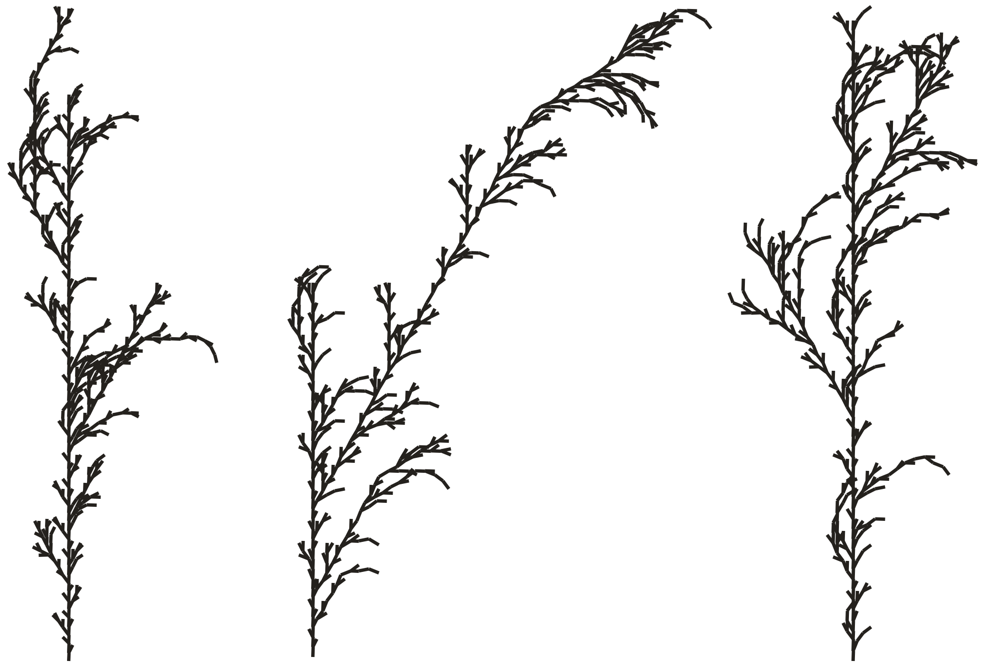
$\delta = 29°,\ A = \{ F, +, -, [,] \}$

$\omega = F$

$p_1 = F \xrightarrow{1/3} F[+F]F[-F]F$

$p_2 = F \xrightarrow{1/3} F[-F]F[+F]F$

$p_3 = F \xrightarrow{1/3} F[-FF-F]F$



Context Sensitive L-systems

- L-systems considered so far were context-free because the production rules were applied to symbols independently of their context. But the context affects differentiation in biological systems (hormone concentration, chemical signaling, etc.).
- Context-sensitive L-systems apply a production rule only if the symbol is preceded and/or followed by specific symbols.
- Symbol   delimits the *left context,* and symbol   delimits the *right context*.

Context free example: $p = b \quad a \quad c \rightarrow g$

Context-sensitive example: $p = b \quad a \quad c\ [d]e \rightarrow g$

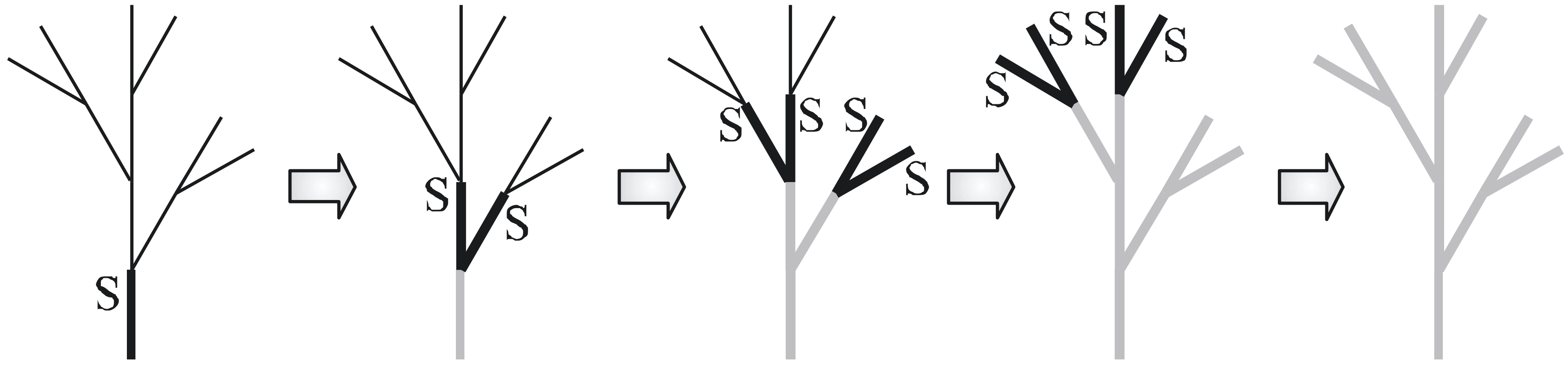
(applies to: $u\ \mathbf{b}\ [v[w\ x]]\mathbf{a}\ \mathbf{c}\ [d\ y]\mathbf{e}\ z$)

Signal Propagation

- Context-sensitive L-systems permit the exchange of signals between adjacent elements of a structure.
- Concatenating a series of local exchanges one can obtain the long-range propagation of signals.

$A = \{ F, S, Q, +, -, [,] \}$
$\omega = S [-F [-F] F] F [+F [+F] F] F [-F] F$

ignore $+, -$
$p_1 = S \quad F \rightarrow S$
$p_2 = S \rightarrow Q$



Applications to computer graphics

Synthesis of Developmental Systems

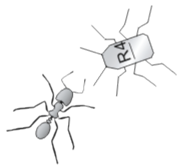
Synthesis of Rewriting Systems

- Can be done by hand
  - When the rewriting rules are explicitly given (e.g., fractal curve)
  - When the rewriting rules can be easily deduced from the description of the developmental process (e.g., development of bacteria filaments and moss leaves)
  - When the geometric specifications of the end result are not strict (e.g., plant-like appearance)
- Requires automated search method
  - When the target of the synthesis is a non-trivial developmental outcome (e.g., a neural network controlling a robot)

Typically, the *inverse problem* (finding the developmental process that realized a given outcome) is difficult and admits no general systematic solution (from global to local)

Evolution and Development

- The use of artificial evolution facilitates the automatic synthesis of developmental systems (not only of rewriting systems)
- The use of a developmental representation facilitates the definition of more evolvable and more powerful artificial evolutionary processes
- We will consider examples of
  1. Evolutionary rewriting systems (rewriting rules evolved; modality of application of rules predefined)
  2. Evolutionary developmental programs ("rewriting" rules predefined; modality of application of rules evolved)
  3. Evolutionary developmental processes ("rewriting" rules *and* modality of application of rules evolved)



Matrix rewriting

- Matrix rewriting is an *evolutionary rewriting system* devised by Kitano [1990] to synthesize neural network architectures.
- Only the presence/absence of connections are evolved. Each network is then trained with backpropagation.

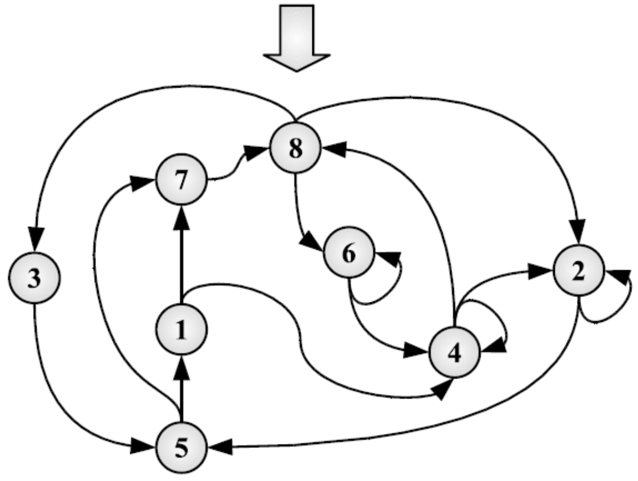
$$\mathcal{E} \rightarrow \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

$$A \rightarrow \begin{bmatrix} a & d \\ a & a \end{bmatrix} \quad B \rightarrow \begin{bmatrix} c & b \\ b & a \end{bmatrix} \quad C \rightarrow \begin{bmatrix} b & a \\ a & c \end{bmatrix} \quad D \rightarrow \begin{bmatrix} a & b \\ a & d \end{bmatrix}$$

$$a \rightarrow \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \quad b \rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \quad c \rightarrow \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \quad d \rightarrow \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

$$\mathcal{E} \Rightarrow \begin{bmatrix} A & B \\ C & D \end{bmatrix} \Rightarrow \left[\begin{array}{cc:cc} a & d & c & b \\ a & a & b & a \\ \hdashline b & a & a & b \\ a & c & a & d \end{array}\right] \Rightarrow \left[\begin{array}{cc:cc:cc:cc} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ \hdashline 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \end{array}\right]$$

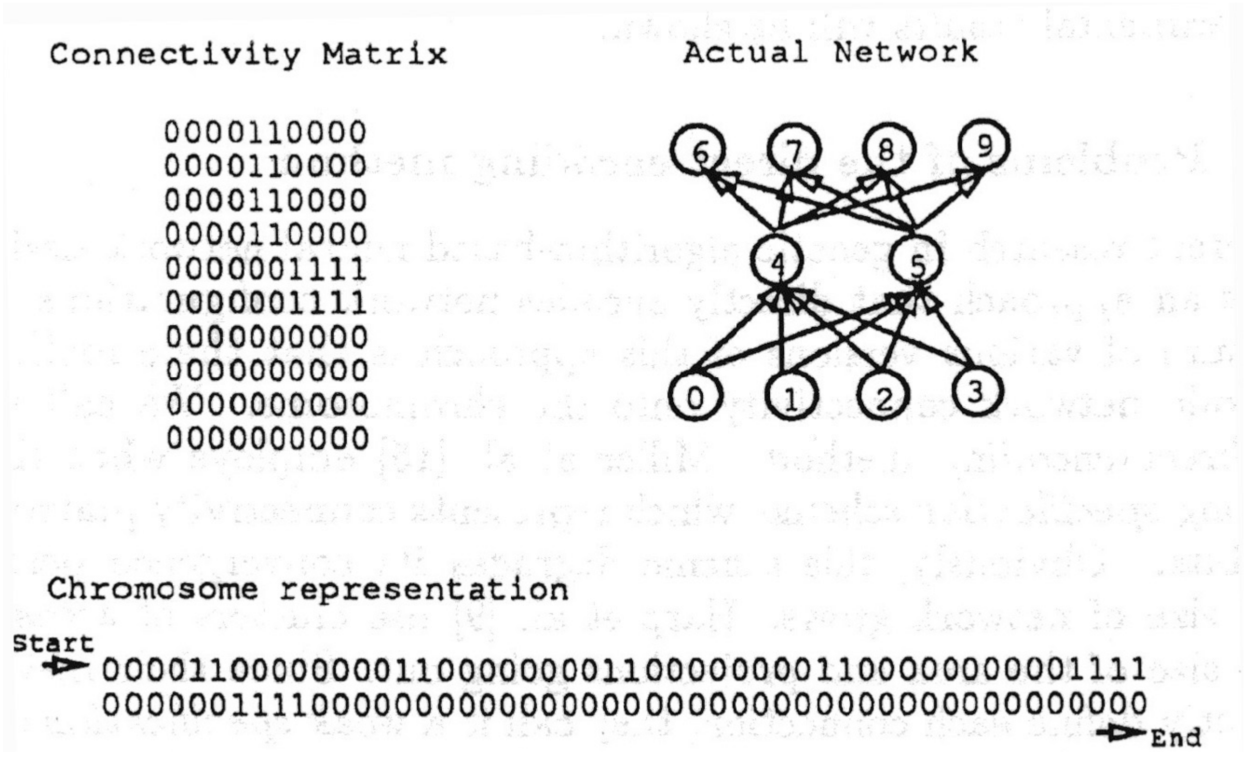
$$\begin{array}{c|cccccccc} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} \\ \hline \mathbf{1} & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ \mathbf{2} & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ \mathbf{3} & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \mathbf{4} & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ \mathbf{5} & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \mathbf{6} & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ \mathbf{7} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \mathbf{8} & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \end{array}$$



Genome → *ABCD adaa cbba baac abad 0001 1000 0010 0100*

Comparison with Direct Encoding

Direct encoding techniques for neural networks suffer from scalability and lack of regularity in the resulting architecture.



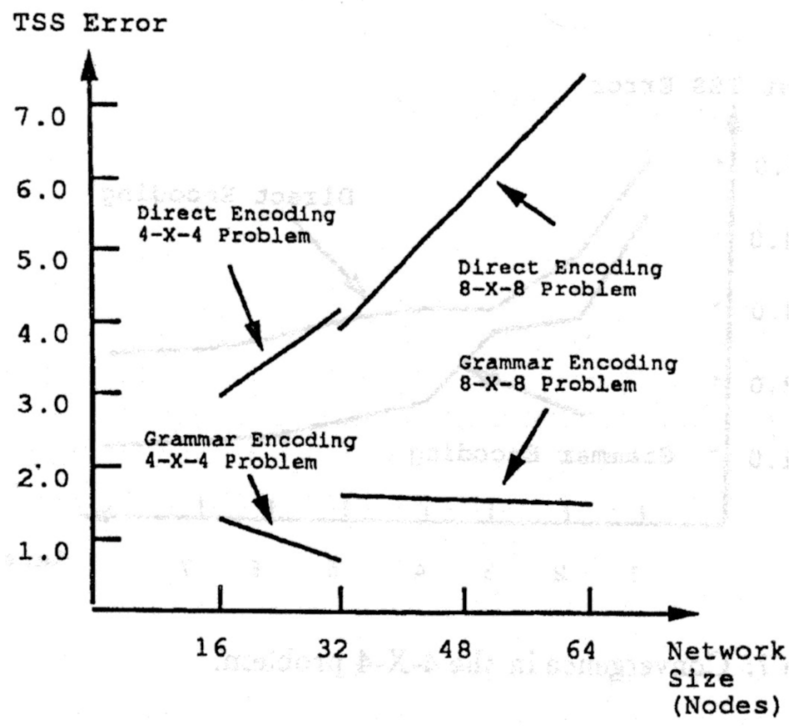
```
0000110000
0000110000
0000110000
0000110000
0000001111
0000001111
0000000000
0000000000
0000000000
0000000000
```

```
000011000000001100000000110000000011000000000001111
0000001111000000000000000000000000000000000000000
```

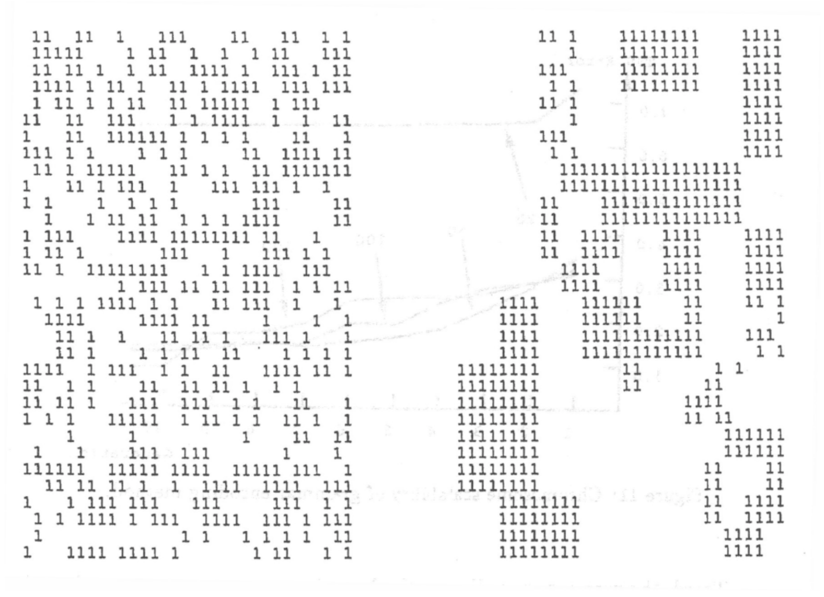
Comparative results

The performance of developmental networks evolved using matrix rewriting (grammar encoding) does not suffer from network size, as direct encoding networks do, and resulting architectures are much more regular.

Performance comparison



Architecture comparison



Direct encoding

Grammar encoding

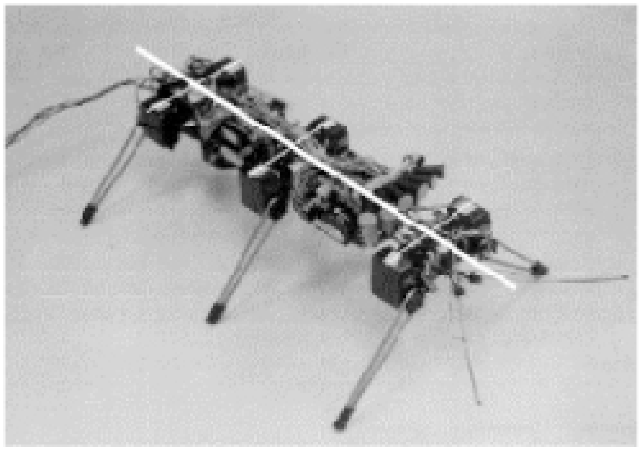
Cellular Encoding

Gruau [1994] suggested a method to encode cell division and differentiation by means of *evolutionary developmental programs*. The genome is composed of a tree that encode a series of instructions to duplicate a cell, apply modification, and connect to previous cells. The tree is read from top to bottom in order to build a network from a single mother cell.

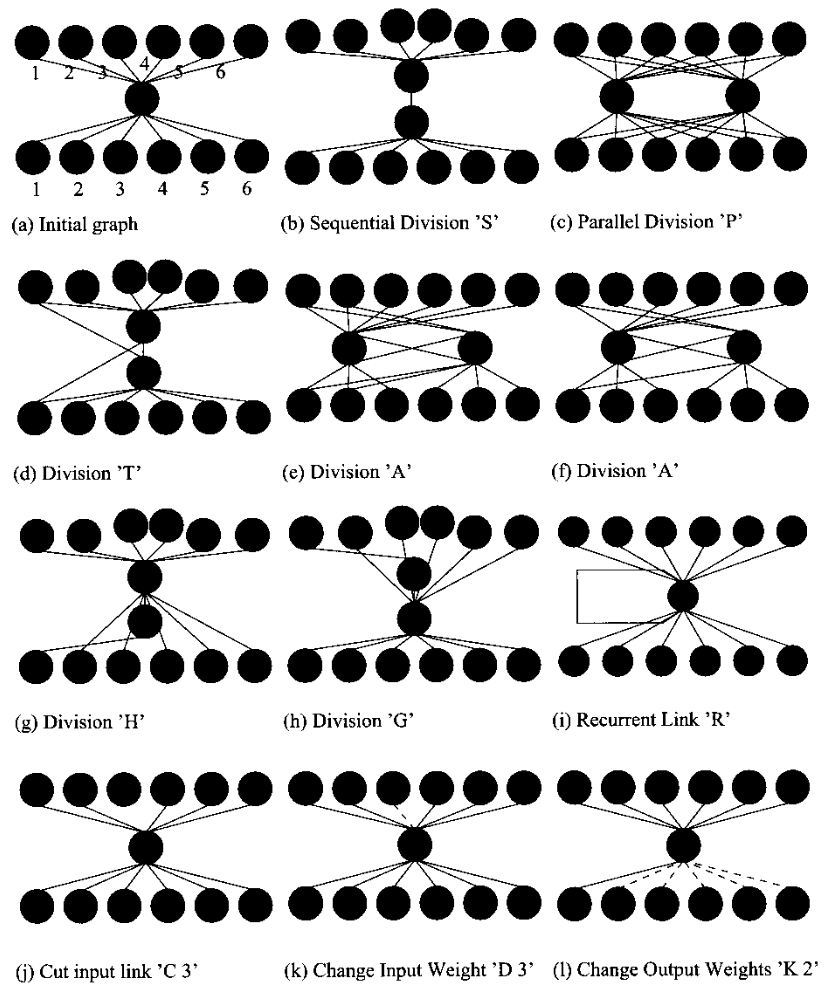
A variation consists of evolving several trees in parallel where the terminals of one tree can point to the root of another tree. This allows reuse of existing structures, simpler codes, and generation of modular architectures.

The method was applied to the generation of a walking neural controller for an insect-like robot [Gruau, 1994].

Multiple trees obtained better performance, generated simpler networks, and displayed regular modularity.

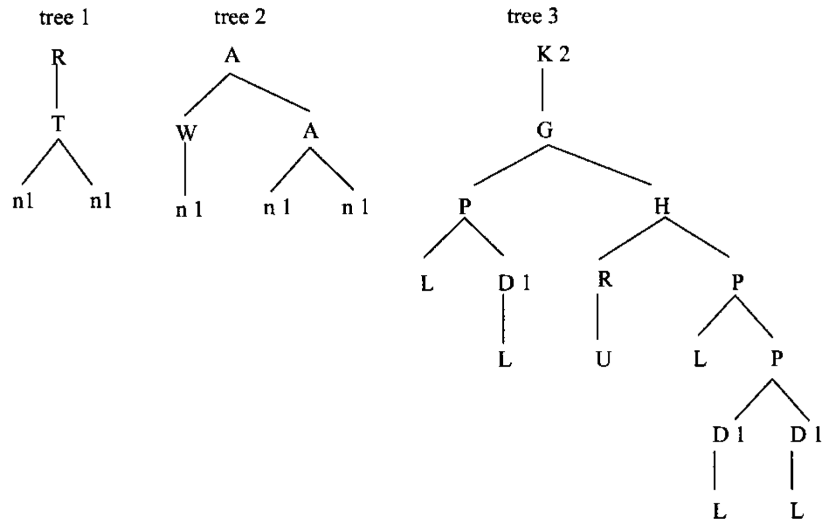


Cellular Encoding graph grammar



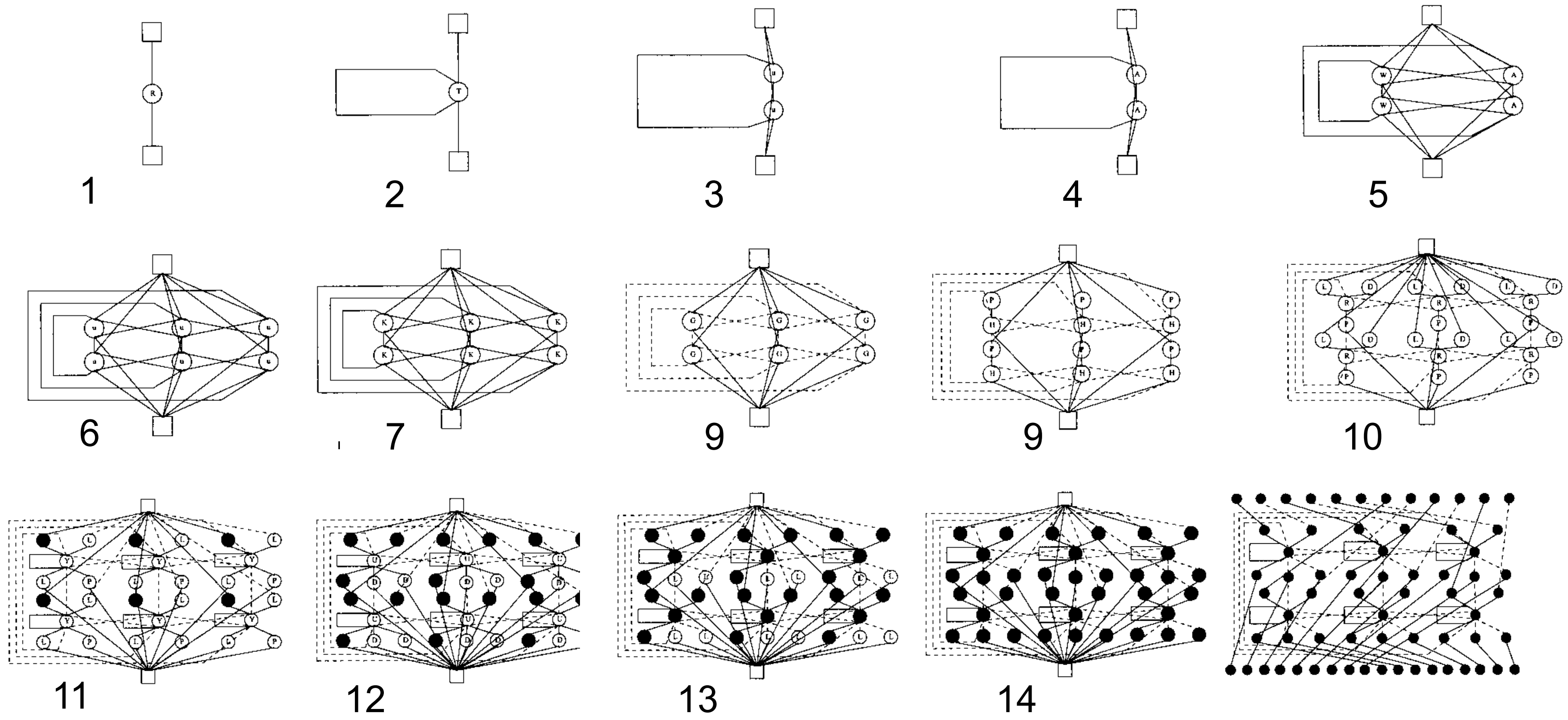
The designer defines a set of local graph transformations or *graph rewriting rules* that can appear in the trees defining the development.

Additional rules specify how the reading head moves between trees (e.g., n1 moves to the next tree)



Example

Sequence of development steps obtained with Cellular Encoding

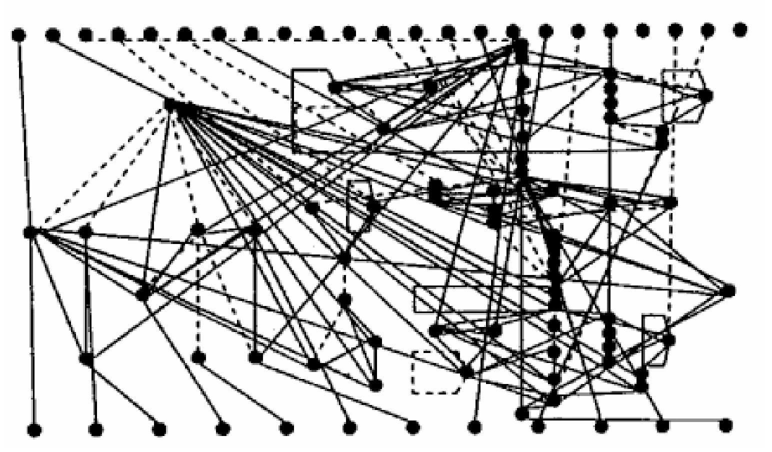


After development the network is connected to the input and output nodes

Walking Architecture

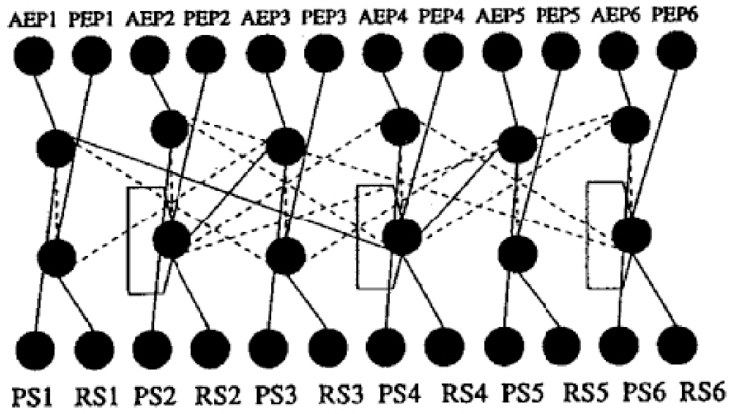
Single tree

```
single tree:
A(A(U-3)(S(L3(M3(M9)))(P(T(L4(M1))(D-6(G(D2(P(L-8(F6(F1)))(S(L8(D-1(M3)))(I-5(M1
(F1))))))(L4(M3(M9))))))(A(U7(M2))(A(H(L-8(M1(L2(M3))))(S(W(M1))(U-5)))(C-3(T(L-
1(M1))(U-5))))))))(R(T(S(L-2(M1(L4(L))))(T(D-8(W(F2)))(T(T(S(R(A(A(S(A(L-1)(L8))
(R(L3(M3))))(S(G(L-9(M1))(D7(M3(F1))))(L6)))(W(S(A(S(D-5(T(I9(U-1))(I-7(L(F1))))
)(S(L-6(M2))(U2())))(R(L2(M1))))(T(L3(L-5))(L-7(M1)))))))(F1))(L-3(M2)))(L-3))))
(H(I8(P(S(H(P(L-2(M2(M1)))(R(U9(M1))))(S(U4)(U5(U-2(M1)))))(A(M1)(T(L5)(I-5(U-1)
))))(A(T(T(C(A(P(G(C(T(C-4(A(P(G(C6(L4(U-5(M1))))(W(U-8)))(I3(L-2)))(U-7)))(F1))
)(M1(L-6(L(M3)))))(U6))(U9)))(G(T(P(D1(L2))(M1))(T(D-2(L-6(M1)))(S(L8)(U-4(M1)))
))(R(T(M1)(C-3(L8))))))(W(W(T(A(A(S(A(L-3)(L-7))(R(L1(M3))))(S(G(L(M1))(D7(M3(F1
))))(L6)))(W(S(A(S(D(T(I-9(U-1))(I-7(L-3(F1)))))(S(L-6(M1))(U2(S(G(L(M1))(D7(M3(
F1))))(L6)))))(R(L2(M1))))(T(L3(L5))(L7(M1))))))(L-9(M1(M3)))))))(G(H(G(L2(M1(C9
(F1))))(D-2(L-1)))(D-1(D7(S(T(I(L6))(H(L8(L-2(M1(L-4(L-5)))))(L-5)))(U-9)))))(U2
)))))(G(U6)(M2)))))
```



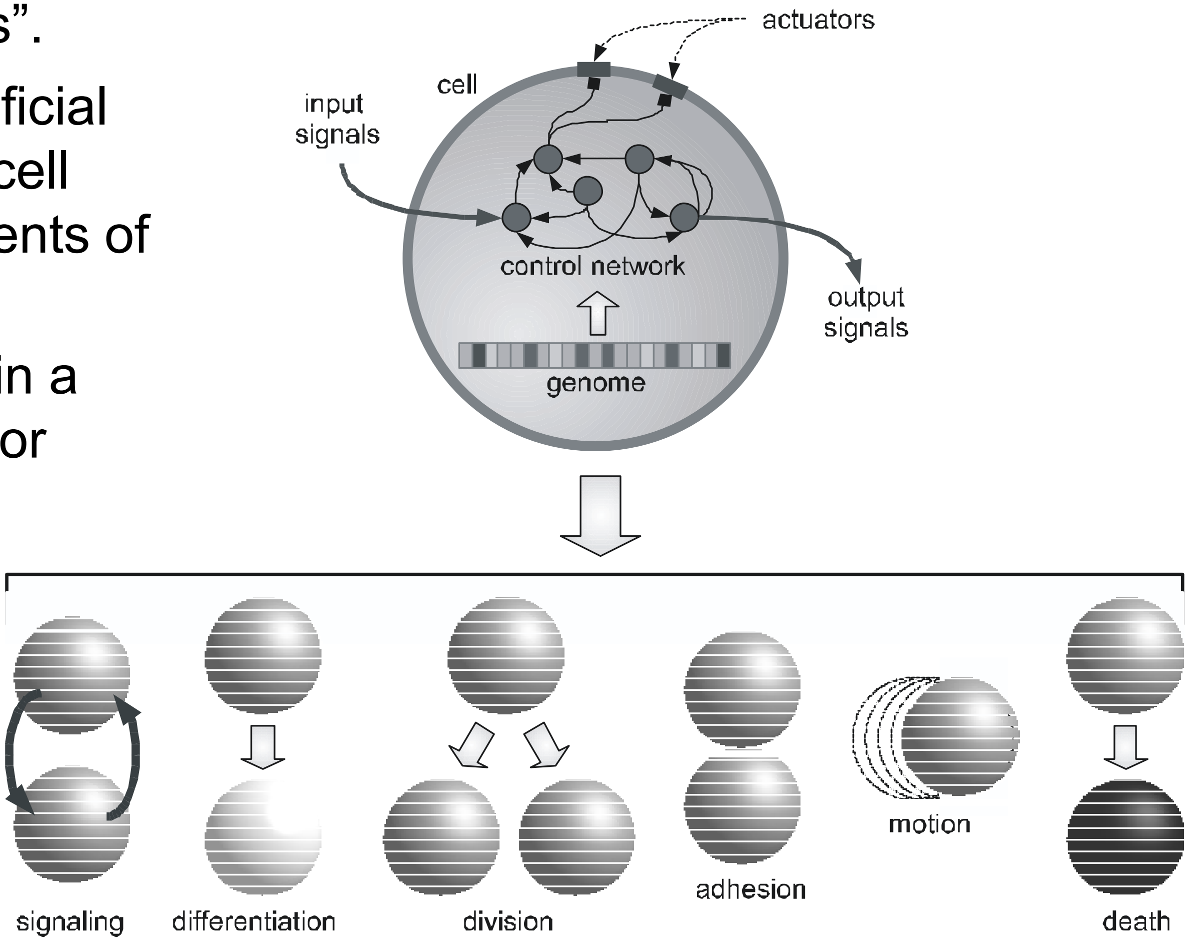
Multiple trees

```
tree 1: A(A(n2)(n2))(n2)
tree 2: not used
tree 3:
P(T(C-7(C-7(D5(M3(M2(C-7(C-7(D5(M3(M2(C-9(I2(U-5(C2(I8(U1(M2(F2(M2(I-1(F2(M4(U))
))))))))))))))))))))(D5(D5(M3(M2(I2(M2(I4(F4(F3))))))))))(T(C-7(C-7(C-7(D5(M3(M
2(I7(W(F2(M3(M2(I4(F2(F4))))))))))))))(R(U6(M2)))
```

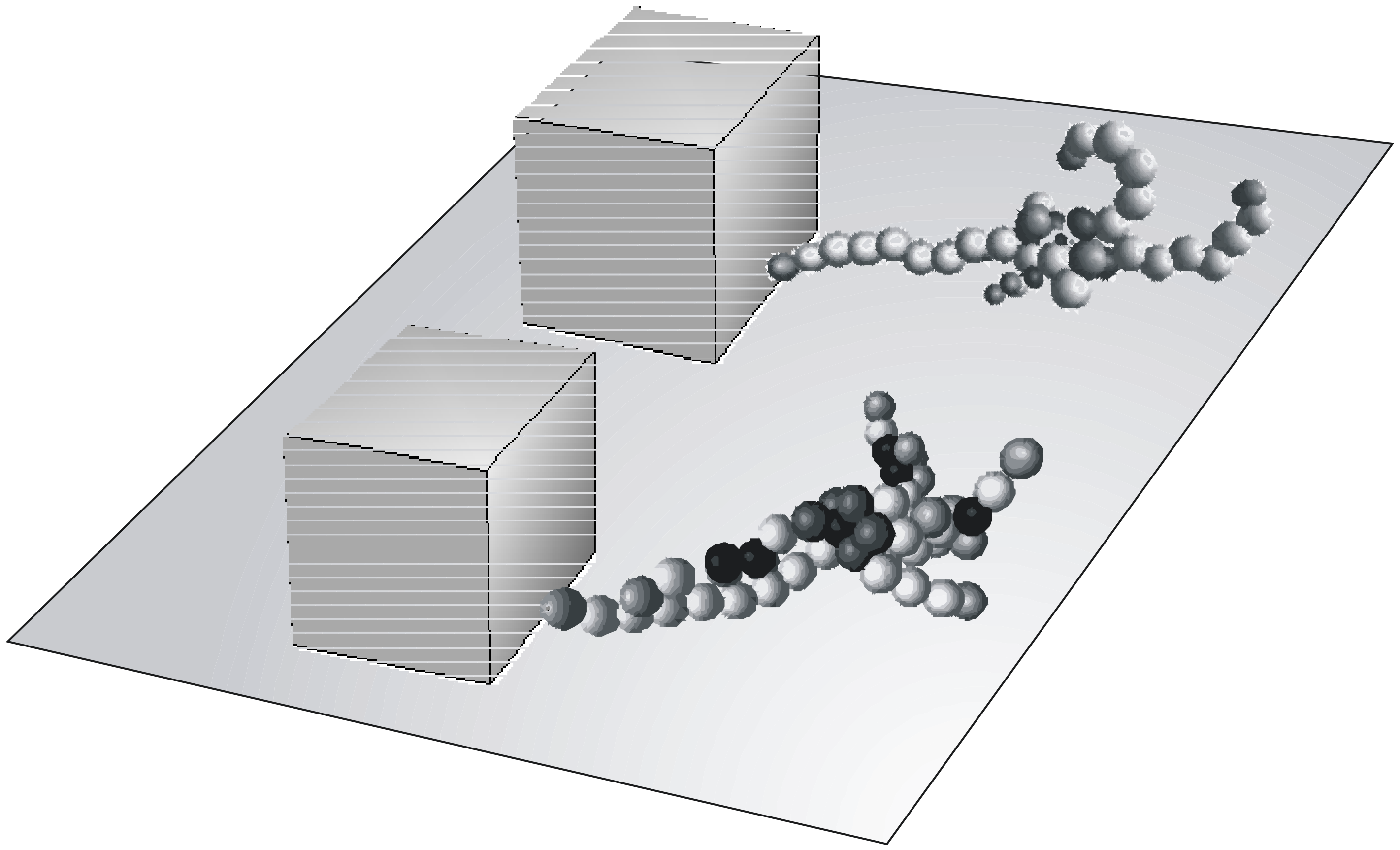


Artificial Ontogeny

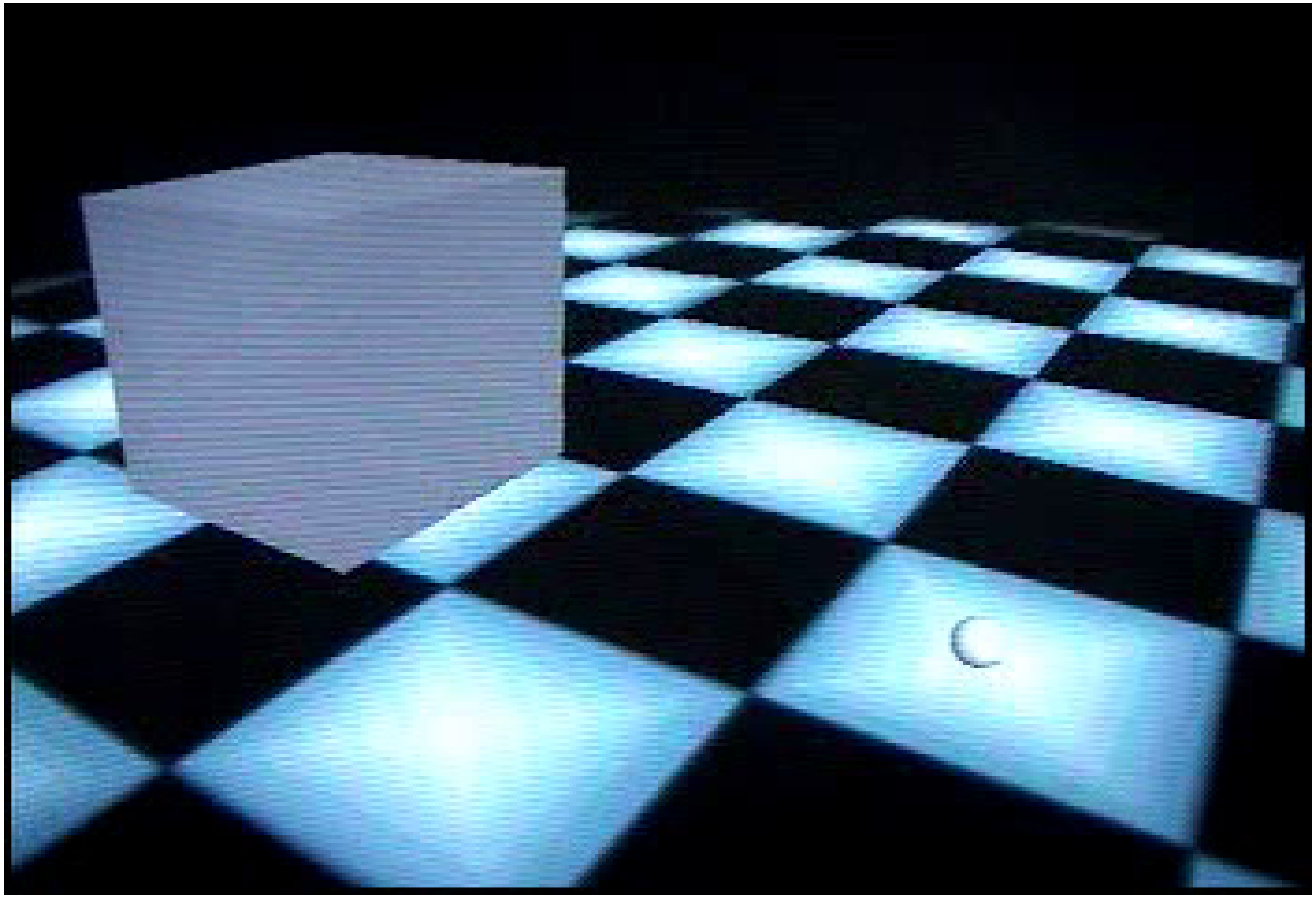
- Artificial Ontogeny (AO) is an *evolutionary developmental process* devised by Bongard and Pfeifer [2001] to synthesize artificial multicellular "creatures".
- AO is based on an artificial cell model, defining a cell genome and the elements of a virtual "cell physics".
- Evolution takes place in a physics-based simulator



Example: Evolution of Block Pushing



Example: Evolution of Block Pushing



Closing Remarks

- L-systems are interesting for computer graphics and the exploration of the space of topological relationships in developing biological systems.
- They represented a major step into modeling of plant-like structures.
- It is difficult to find the set of rules and symbols behind the developmental generation of complex structure with desired characteristics.
- Biological developmental processes are a major source of inspiration for the definition of artificial developmental systems
- It is not easy to select the aspects of the biological developmental processes that are responsible for its favorable properties (in particular, evolvability).
- The definition of artificial developmental processes is still a largely unexplored field. Interactions between evolution, development, and learning remain to be explored.