

Lecture 5:

Grammars, plants, levels

Procedural Content Generation, Autumn 2013

Julian Togelius and Noor Shaker

(some material borrowed from Gabriela Ochoa)

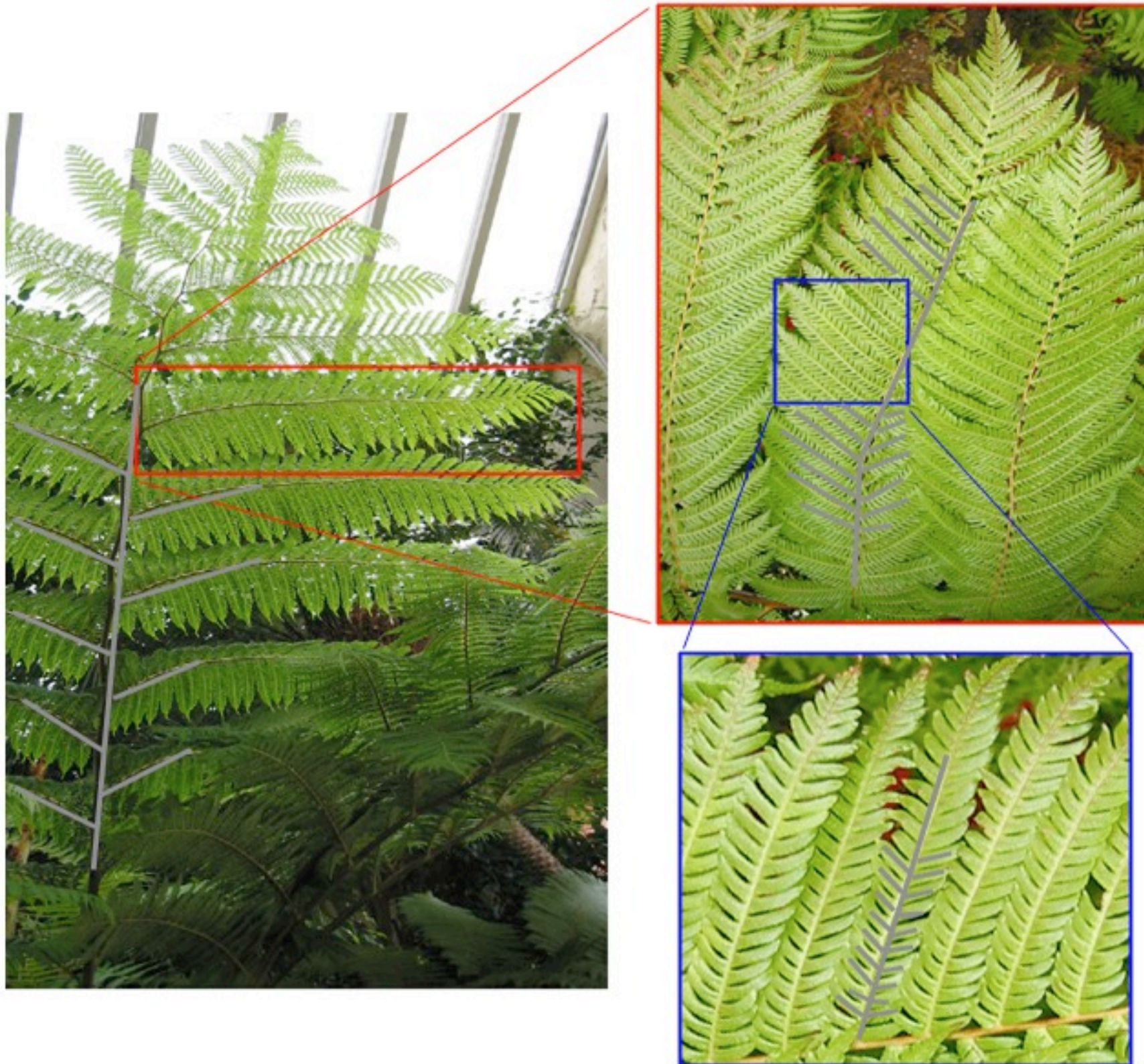
Plants?

- Core feature of the natural world... therefore of many games
- Need for believability
 - Infinitely detailed
 - Similar and recognisable, but not identical
- Need for compact representation
- Need for automatic large-scale generation

SpeedTree



Self-similarity





Self-similarity

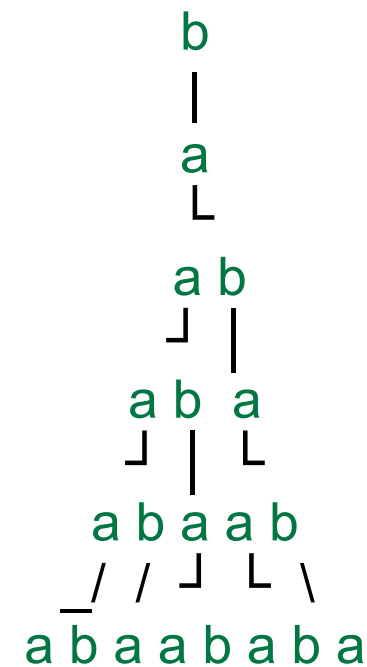
- Nature has obviously thought out some clever way of representing complex organisms using a compact description...
- ...permitting individual variation...
- ...why is this relevant for us?

L-systems

- Introduced by Aristid Lindenmeyer 1968, to model plant development
- Creates strings (text) from an *alphabet* based on a *grammar* and an *axiom*
- Closely related to Chomsky grammars (but productions carried out in parallel, not sequentially)

An example L-system

- Alphabet: $\{a, b\}$
- Production rules (grammar):
 $a \rightarrow ab$
 $b \rightarrow a$
- Axiom: b



Example of a derivation in a
DOL-System

Types of L-systems

- Context-free: production rules refer only to an individual symbol
- Context-sensitive: productions can depend on the symbol's neighbours
- Deterministic: there is exactly one production for each symbol
- Non-deterministic: several productions for a symbol

A graphical interpretation of L-systems

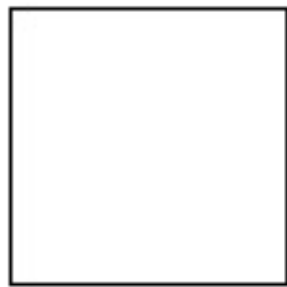
- Invented/popularized by Prusinkiewicz 1986
- Core idea: interpret generated strings as instructions for a turtle in turtle graphics
- Read the string from left to right, changing the state of the turtle (x, y, heading)

Example graphical L-system

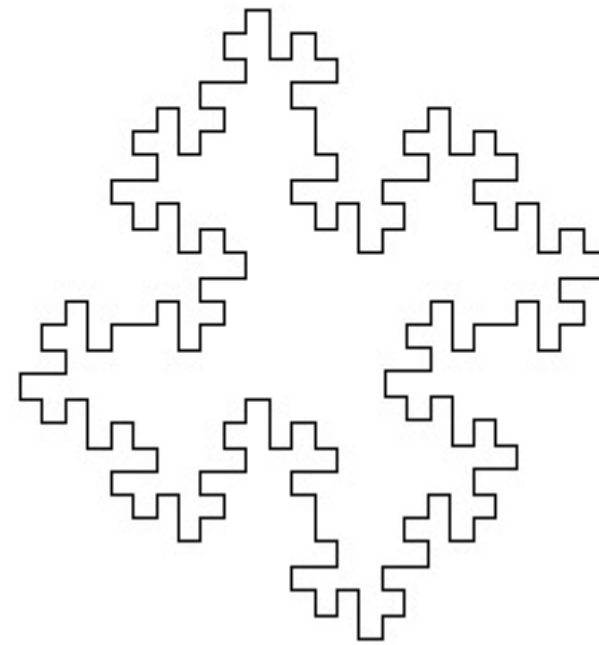
- Alphabet: $\{F, f, +, -\}$
- F: move the turtle forward (drawing a line)
- f: move the turtle forward (don't draw)
- +/-: turn right/left (by some angle)

Graphical L-system

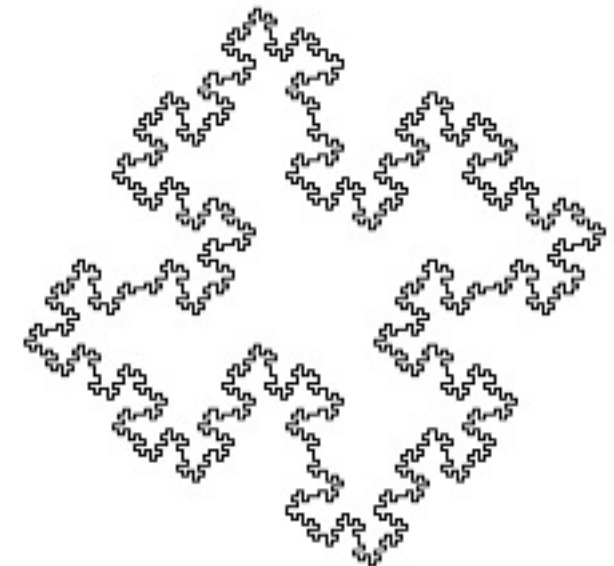
- axiom: $F+F+F+F$
- grammar:
 $F \rightarrow F+F-F-FF+F+F-F$
- Turning angle: 90°



$n=0$



$n=1$



$n=2$

Graphical L-systems

- What's the limit of these systems?

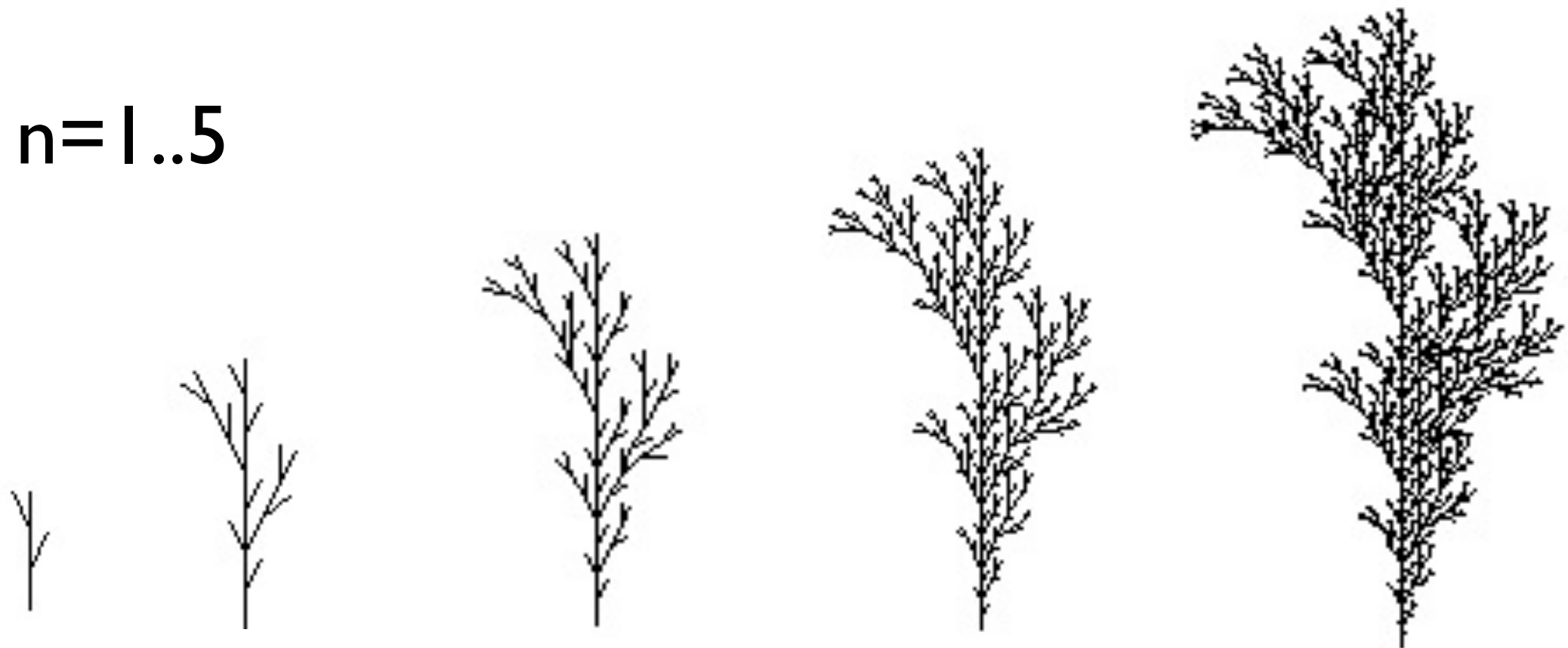
Bracketed L-systems

- Alphabet: $\{F, f, +, -, [,]\}$
- $[$: push the current state (x, y, heading of the turtle) onto a pushdown stack
- $]$: pop the current state of the turtle and *move the turtle there without drawing*
- Enables branching structures!

Bracketed L-systems

- Axiom: F
- Grammar: $F \rightarrow F[-F]F[+F][F]$
- Turning angle: 30°

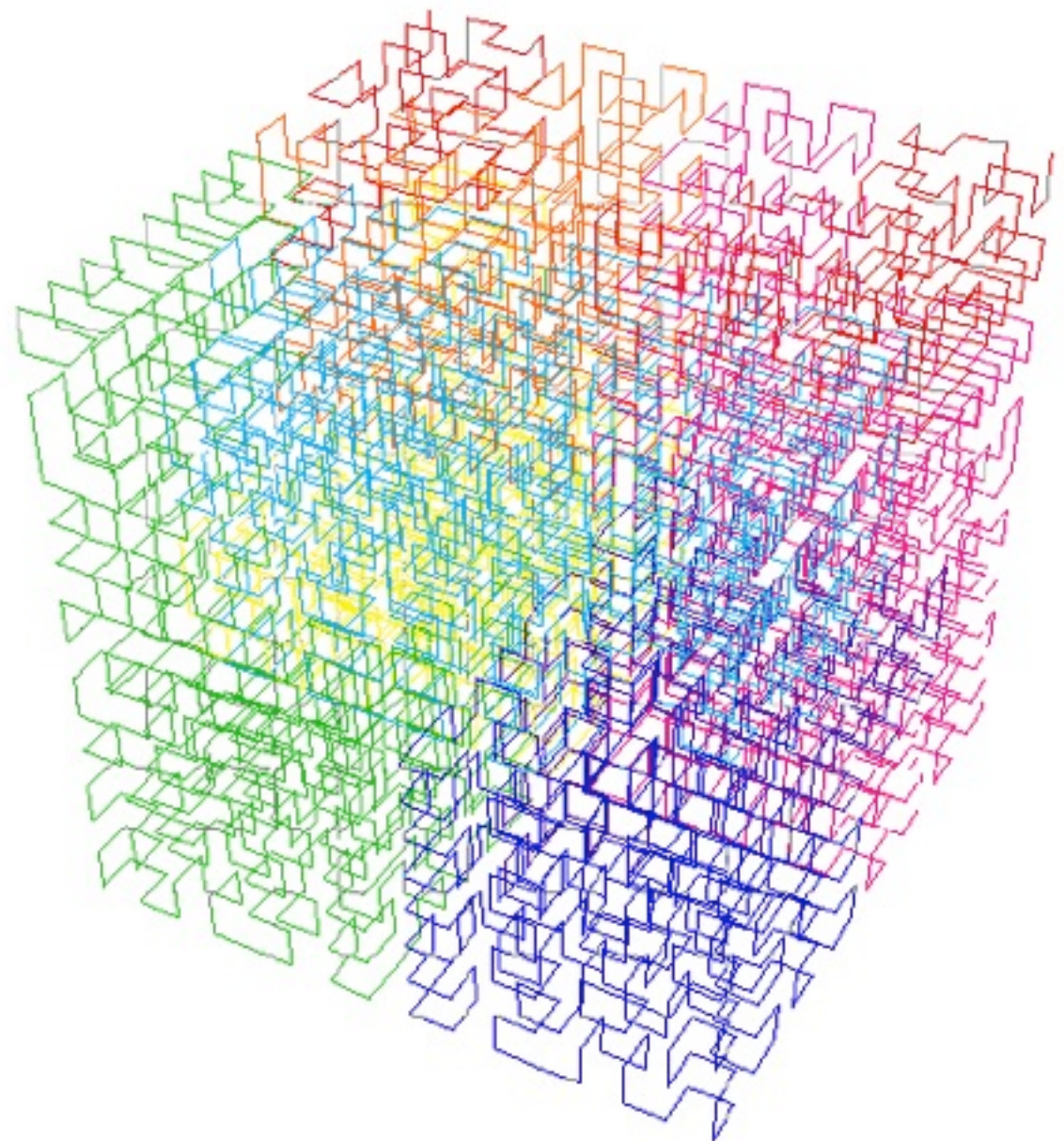
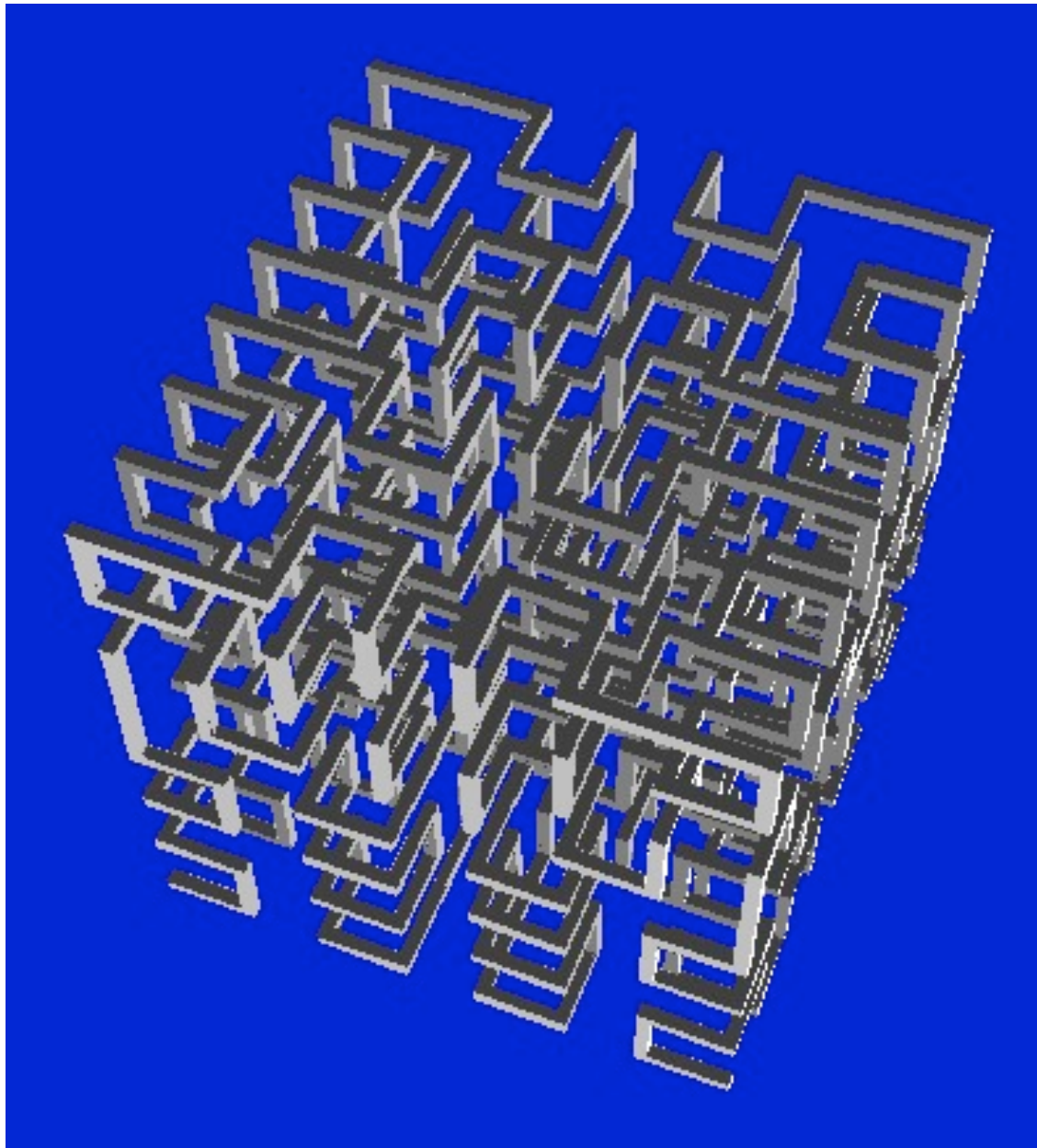
$n=1..5$



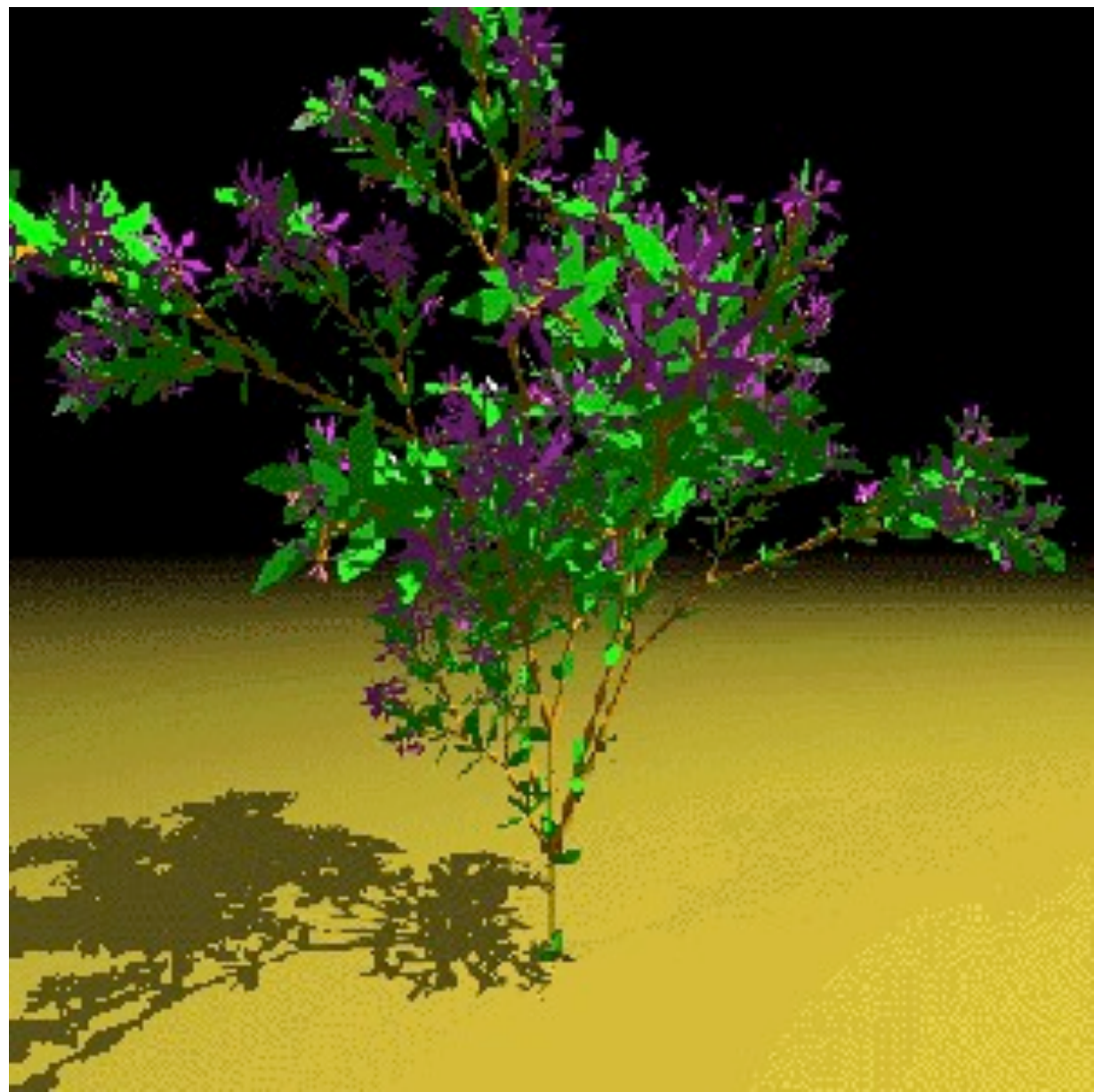
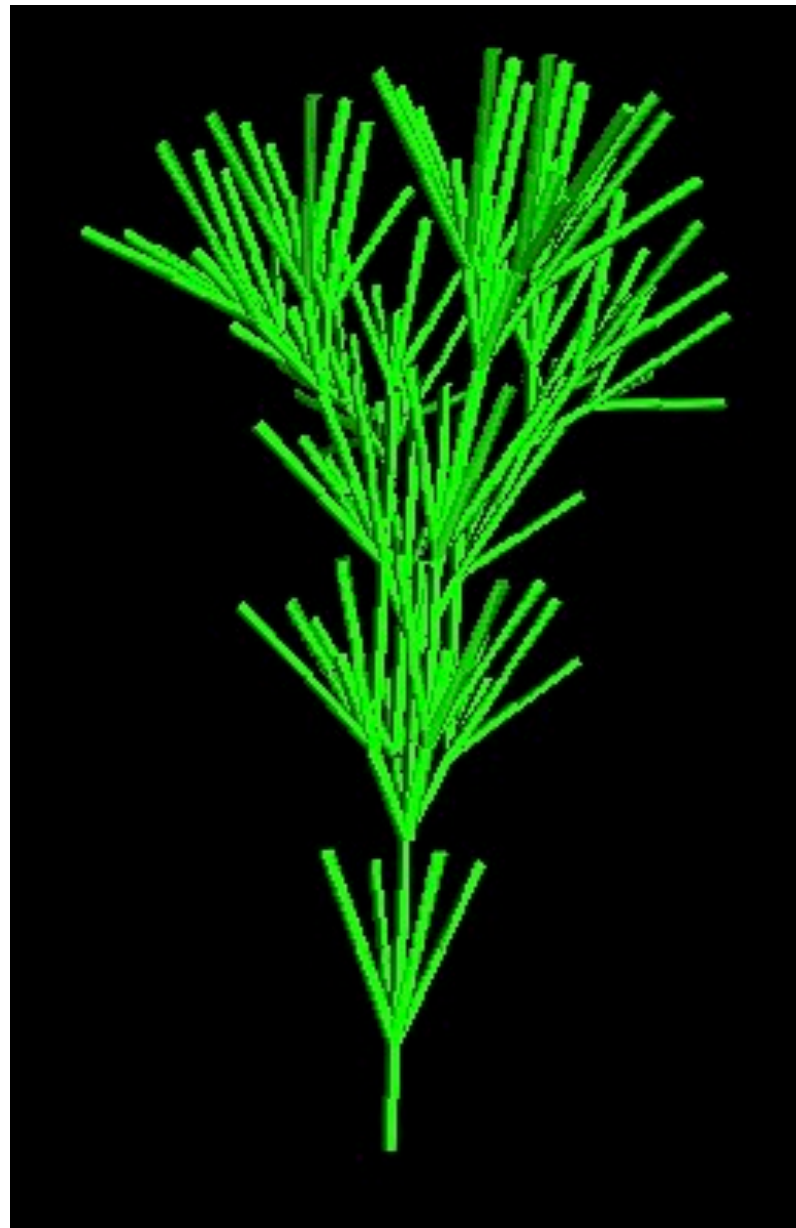
3D graphics

- Turtle graphics L-system interpretation can be extended to 3D space:
- Represent state as x, y, z and pitch, roll, yaw
- $+, -$: turn (yaw) left/right
- $\&, ^$: pitch down/up
- $\backslash, /$: roll left/right (counterclockwise/clockwise)

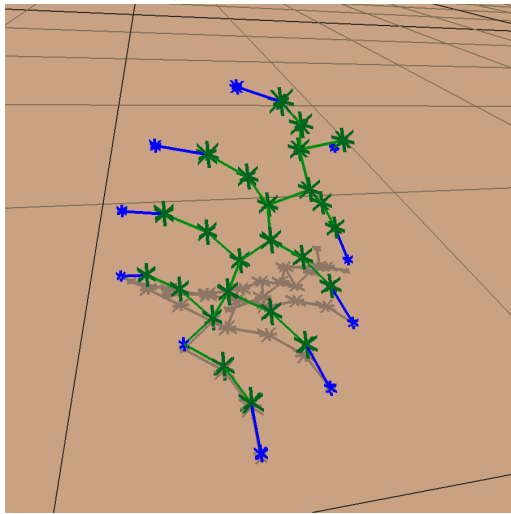
3D interpretation of L-systems



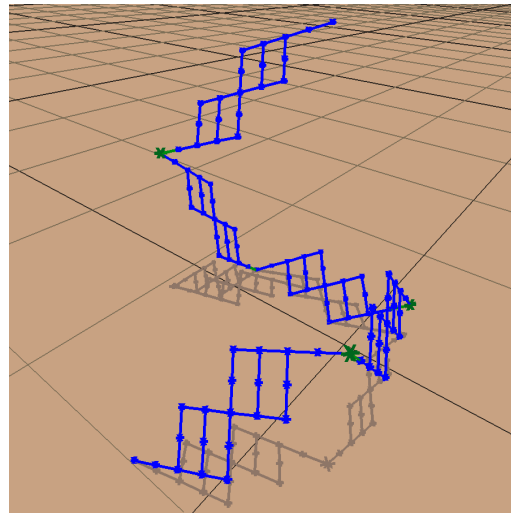
3D interpretation of bracketed L-systems



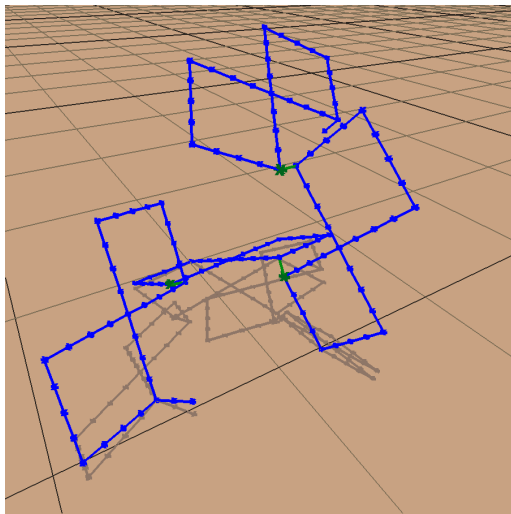
L-system animal morphology



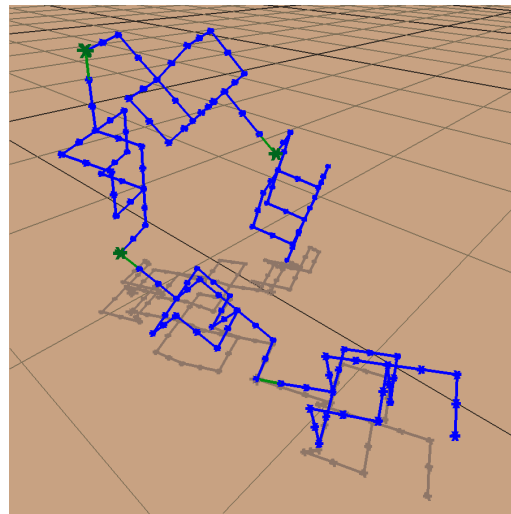
(a)



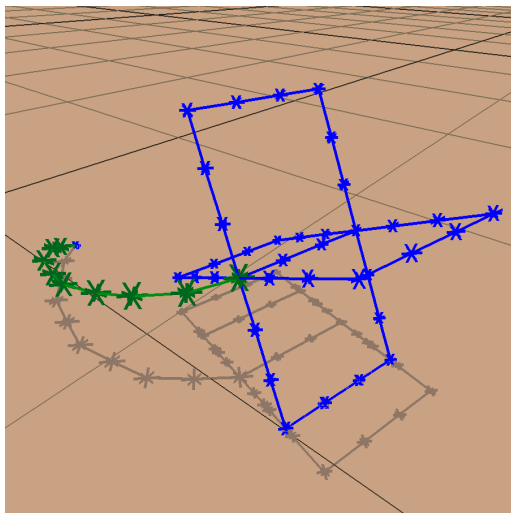
(b)



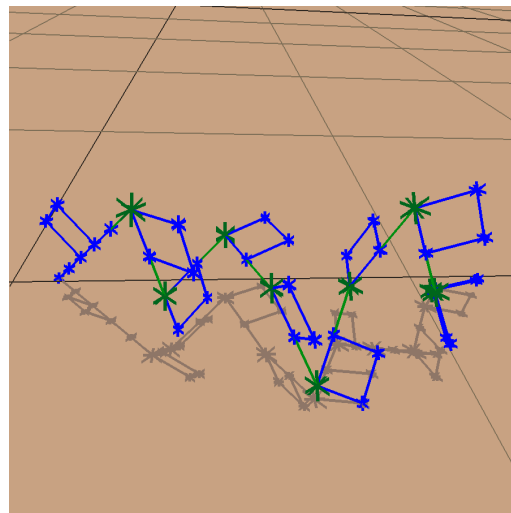
(c)



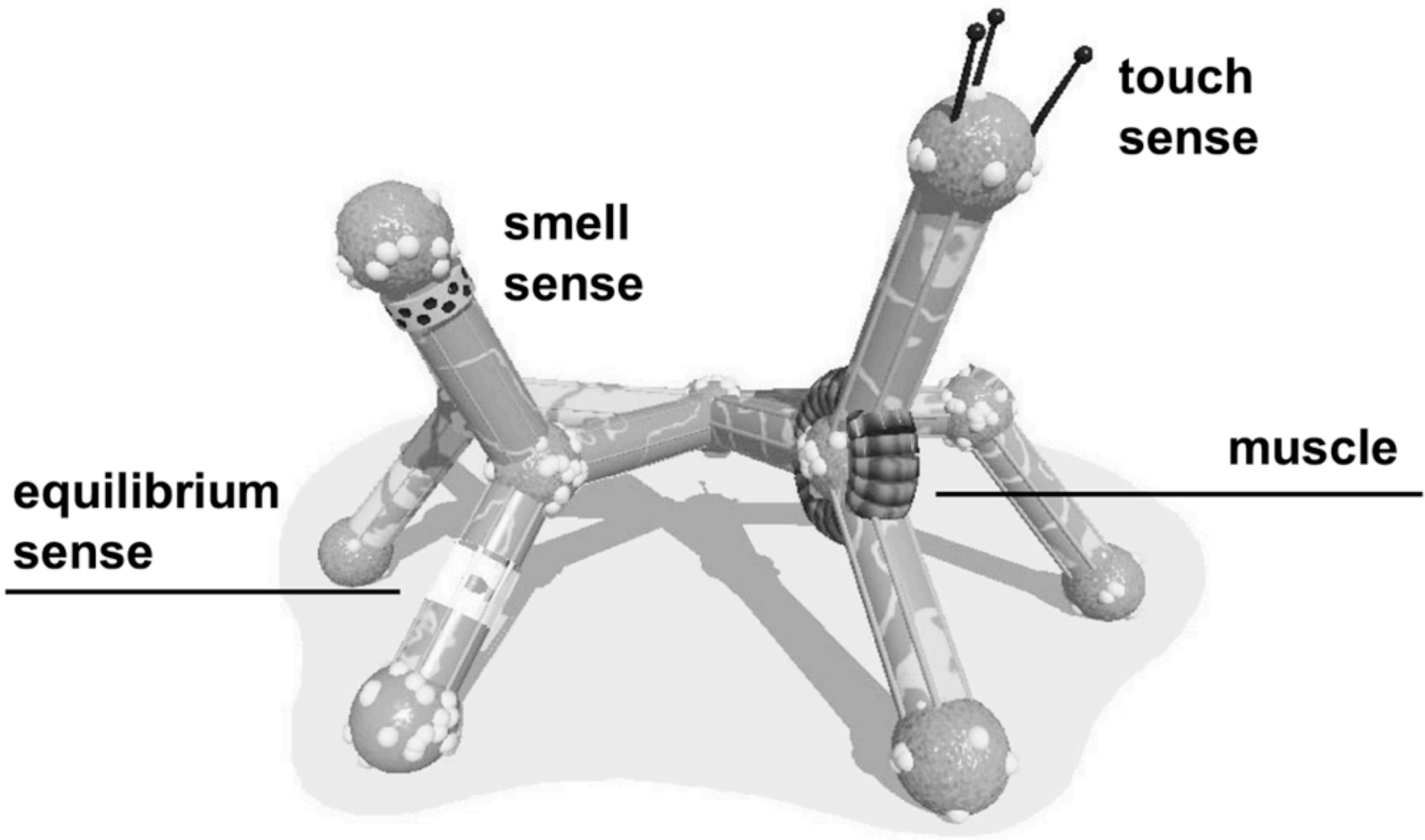
(d)



(e)



(f)



How do we design
these L-systems?

Evolving L-systems

- How can we combine L-systems with evolutionary computation?

Evolving L-systems

- Evolving the axiom
- Evolving the grammar:
 - change the shape of one or more production rules, or
 - add/remove/replace productions
 - counter limits
- Evolving the interpretation:
 - Evolve production probabilities
 - Evolve other aspects (e.g. turning angles)

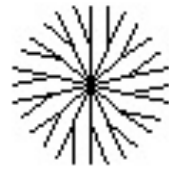
Evolving L-systems

- One example: Ochoa evolved the consequent of a single production rule
 - starting from $F \rightarrow F[-F]F[+F]F$
- Mutation: replace single symbols, or blocks of a few symbols
- Crossover: swap complete “sub-trees” (like in genetic programming)

Fitness functions

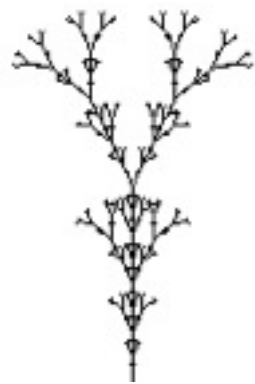
- Phototropism
- Bilateral symmetry
- Proportion of branching points

Evolved L-systems



Branching
points

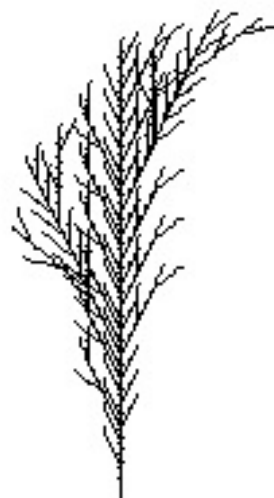
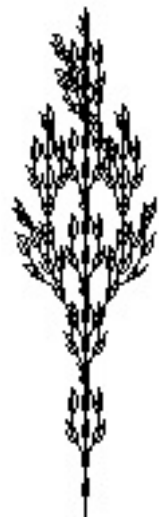
Symmetry



All 3

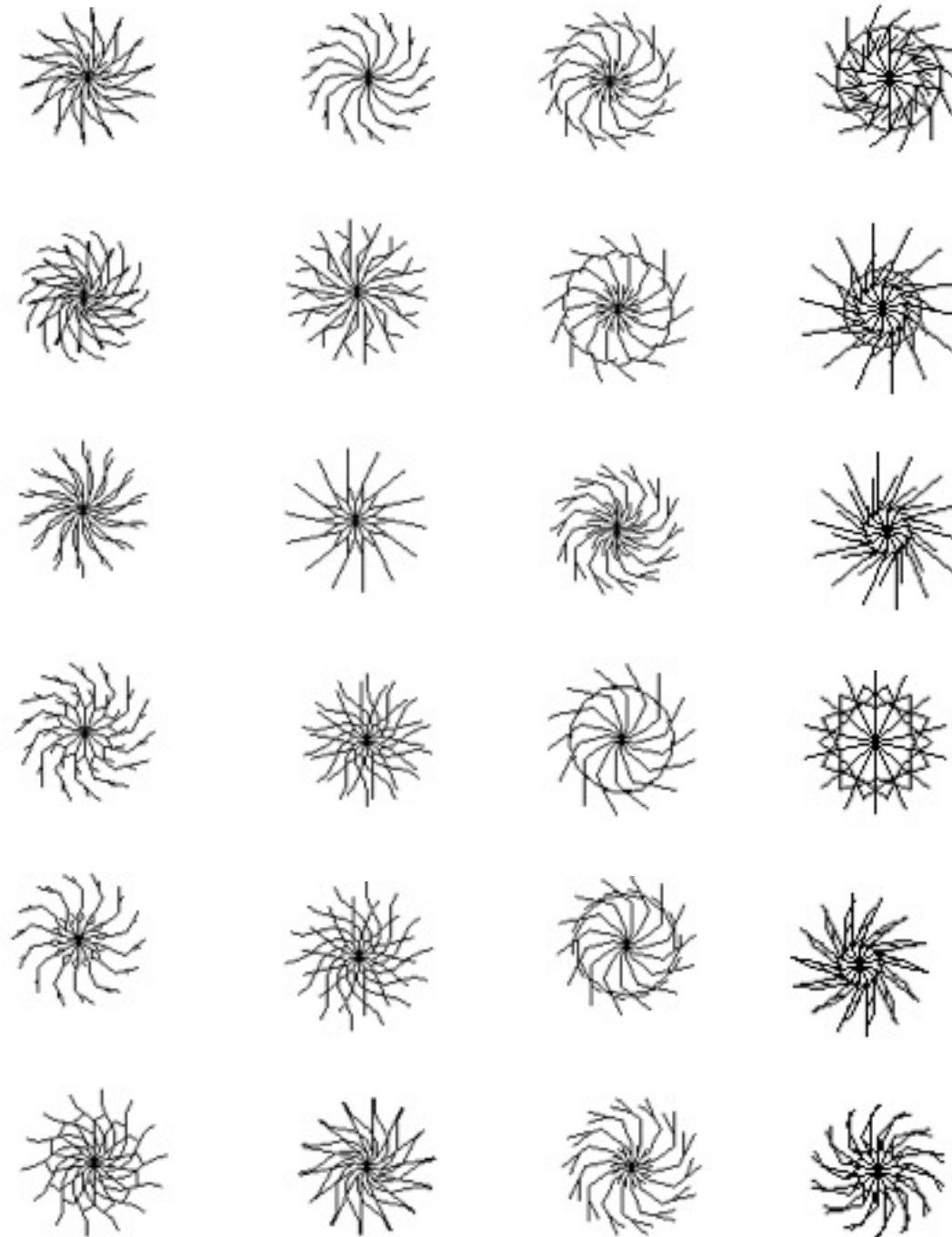


Phototropism

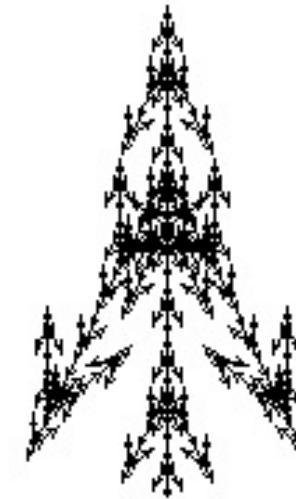
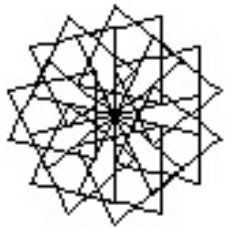
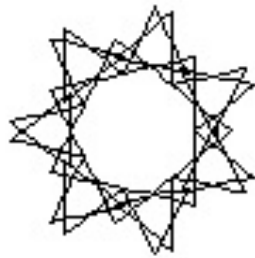
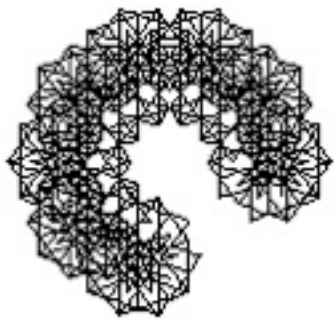


Phototropism +
Symmetry

Evolved L-systems



Evolved L-systems



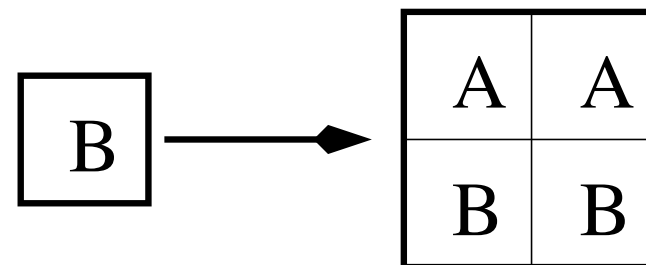
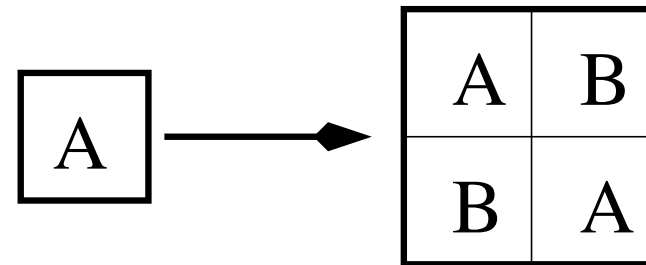
...and this was an
extremely simple L-
system!

2D L-systems

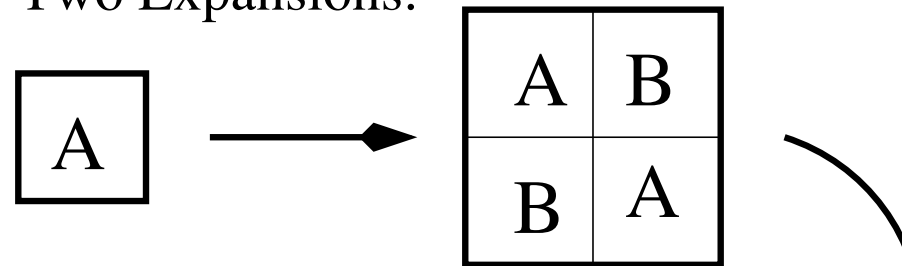
Axiom:

A

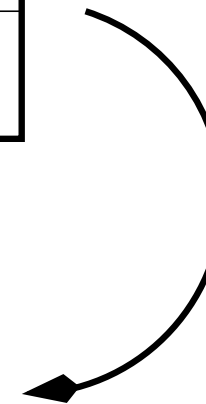
Rules:



Two Expansions:

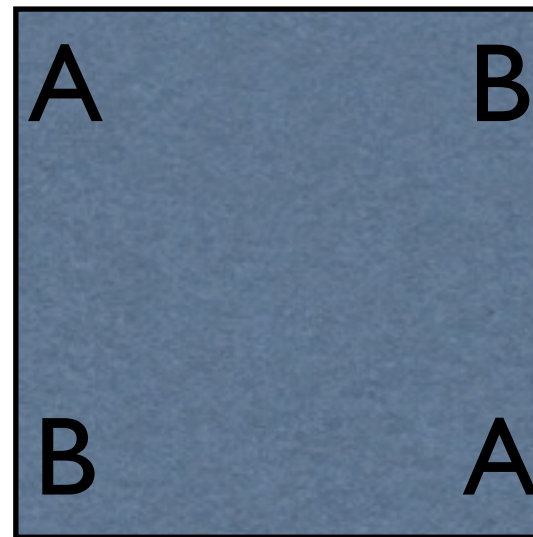


A	B	A	A
B	A	B	B
A	A	A	B
B	B	B	A



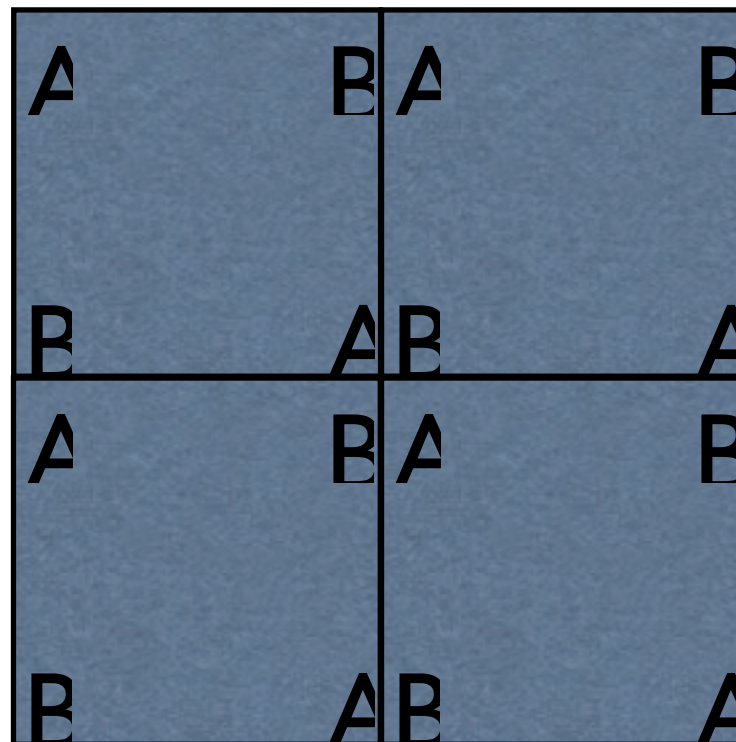
Terrain interpretation of 2D L-systems

- Each group of four letters is interpreted as instructions for lowering or raising the corners of a square
- e.g. $A=+0.5$, $B=-0.5$

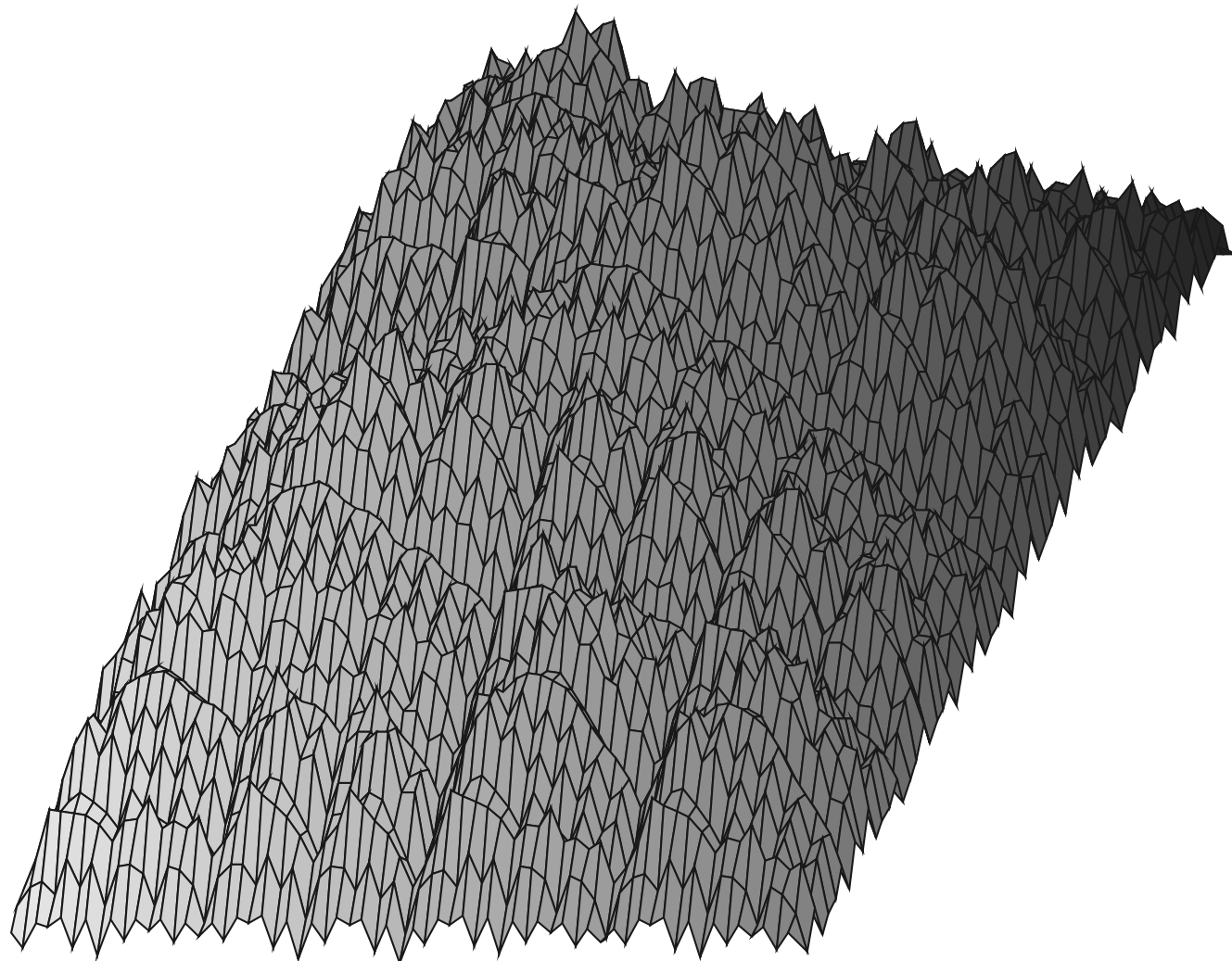


Terrain interpretation of 2D L-systems

- In next iteration, the 2D L-system is rewritten once, and each square is divided into two
- “Doubling the resolution”

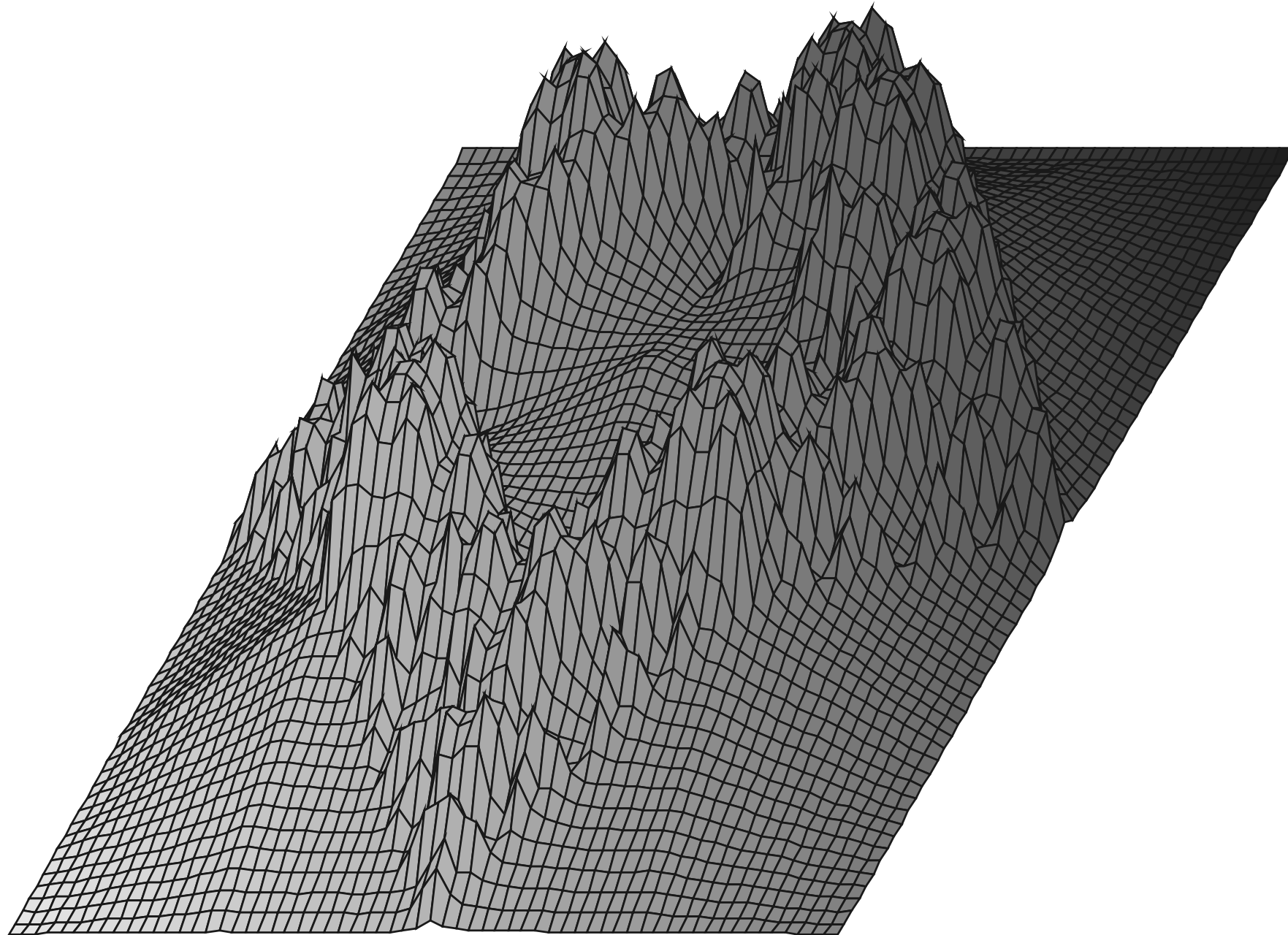


Terrain interpretation of 2D L-systems

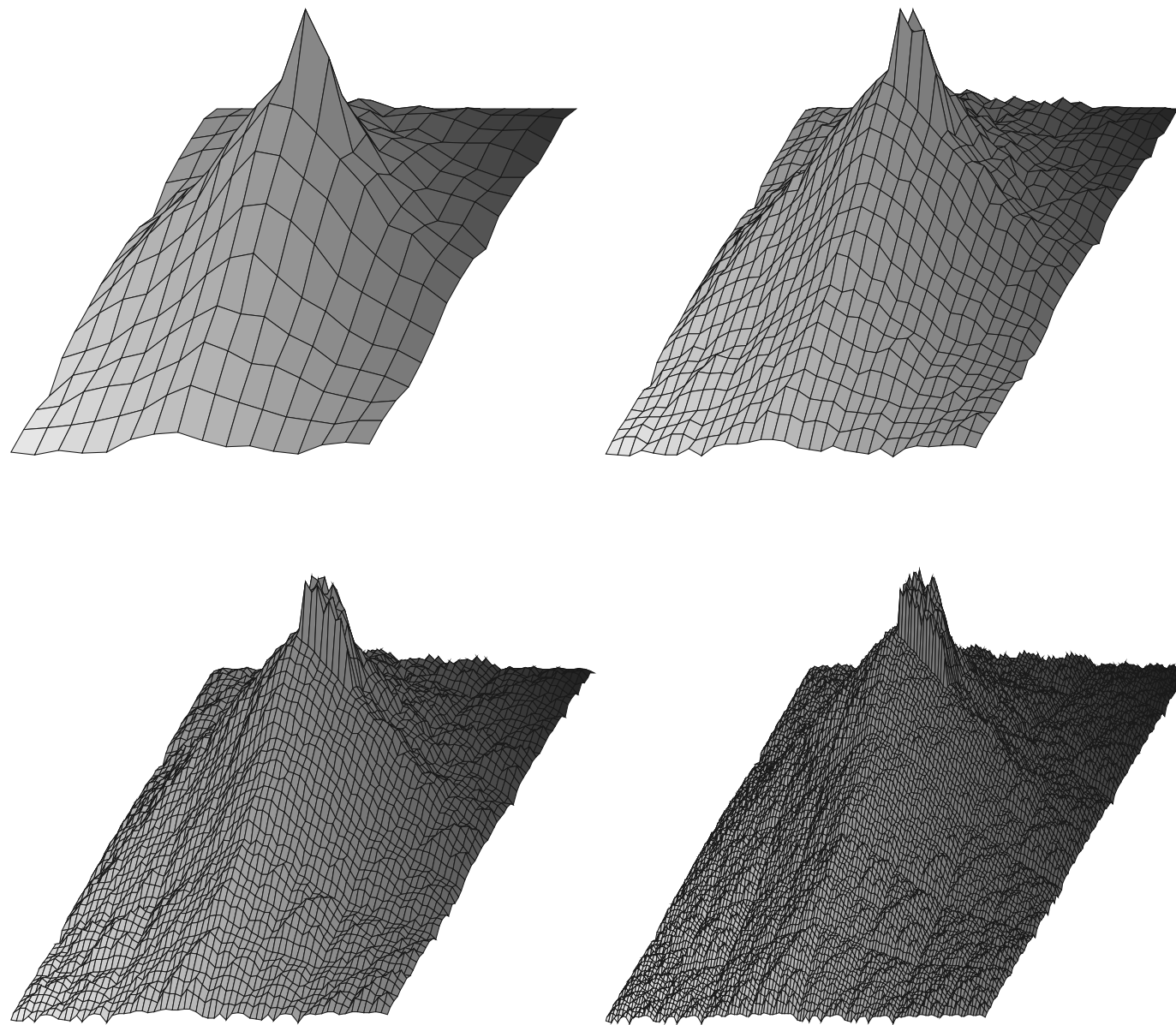


Six rewritings of $A \rightarrow ABBA$, $B \rightarrow AAB B$

Evolved 2D L-system terrains



Evolved 2D L-system terrains



Very short specification, yet infinite resolution!

Grammars for adventure level design

1. Dungeon \rightarrow Obstacle + treasure
2. Obstacle \rightarrow key + Obstacle + lock + Obstacle
3. Obstacle \rightarrow monster + Obstacle
4. Obstacle \rightarrow room

could give...

key + monster + room + lock + monster + room + treasure

key + monster + key + room + lock + monster + room + lock +
room + treasure

room + treasure

monster + monster + monster + monster + room + treasure

Graph grammars

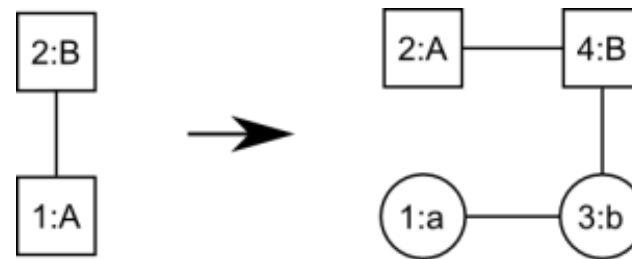


Figure 2. An example of a graph grammar rule

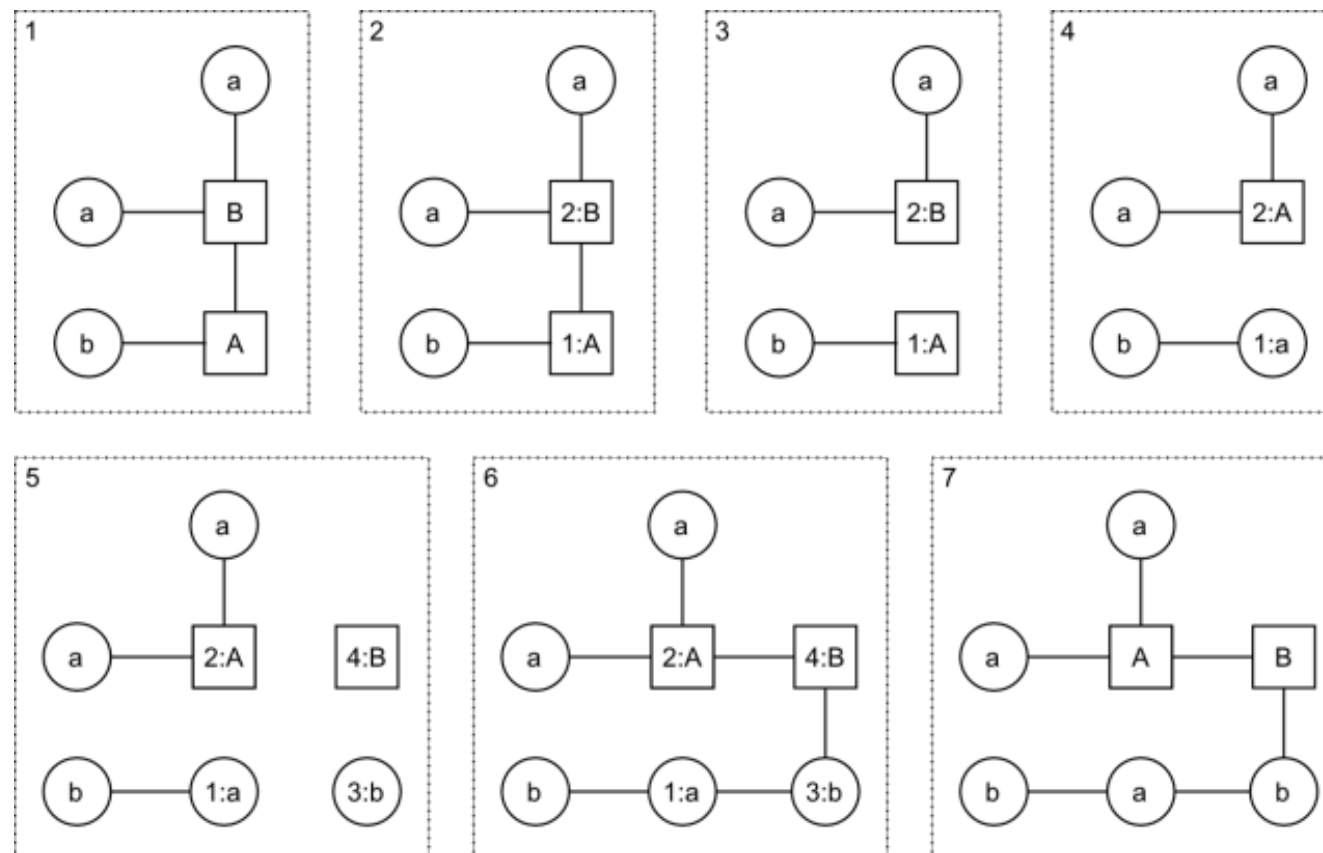
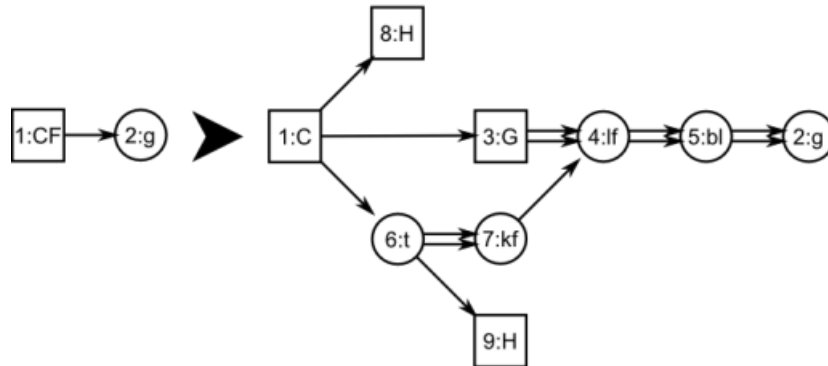


Figure 3. The replacement operations according the rules from figure 2.

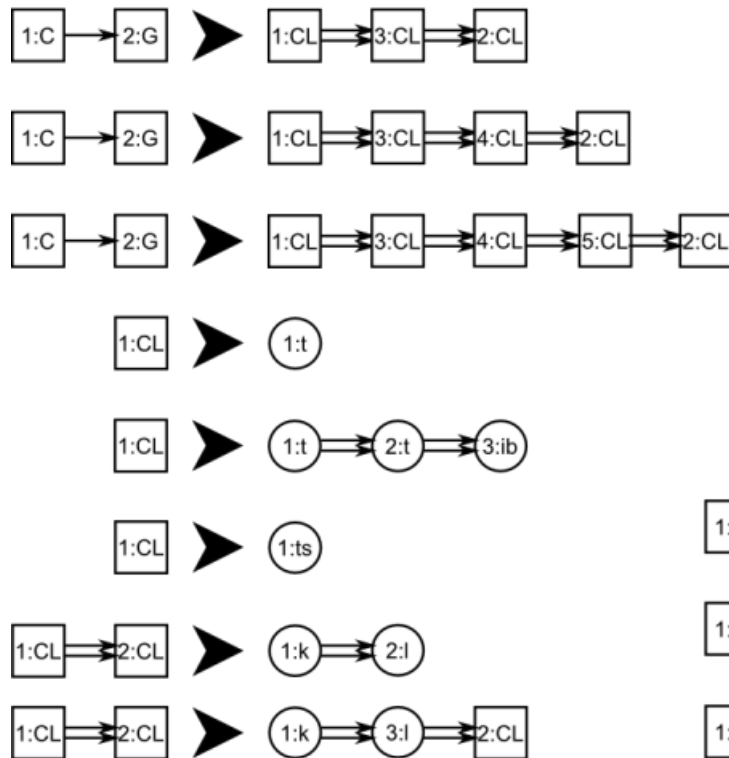
Start rule:



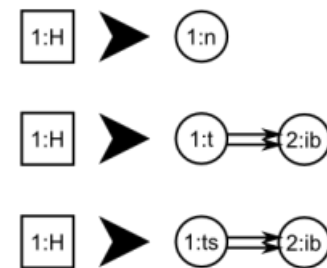
Create Final Chain:



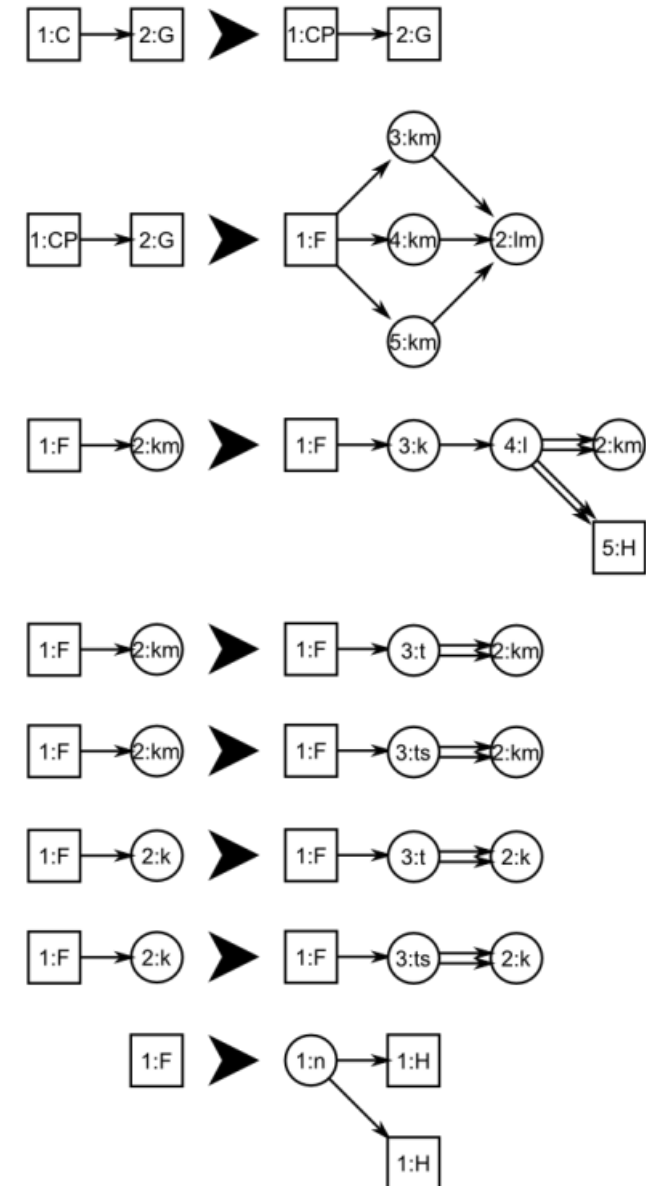
Create Linear Chain:



Resolve hooks:



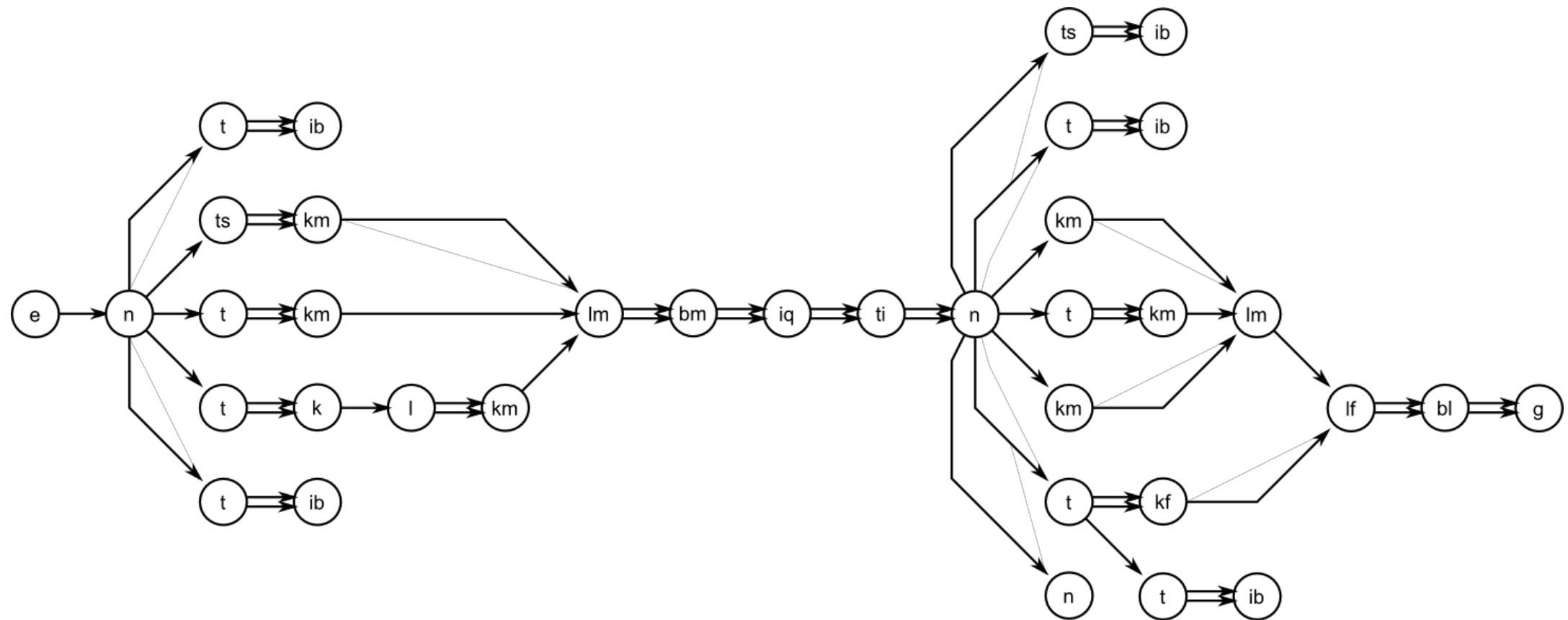
Create Parallel Chain:



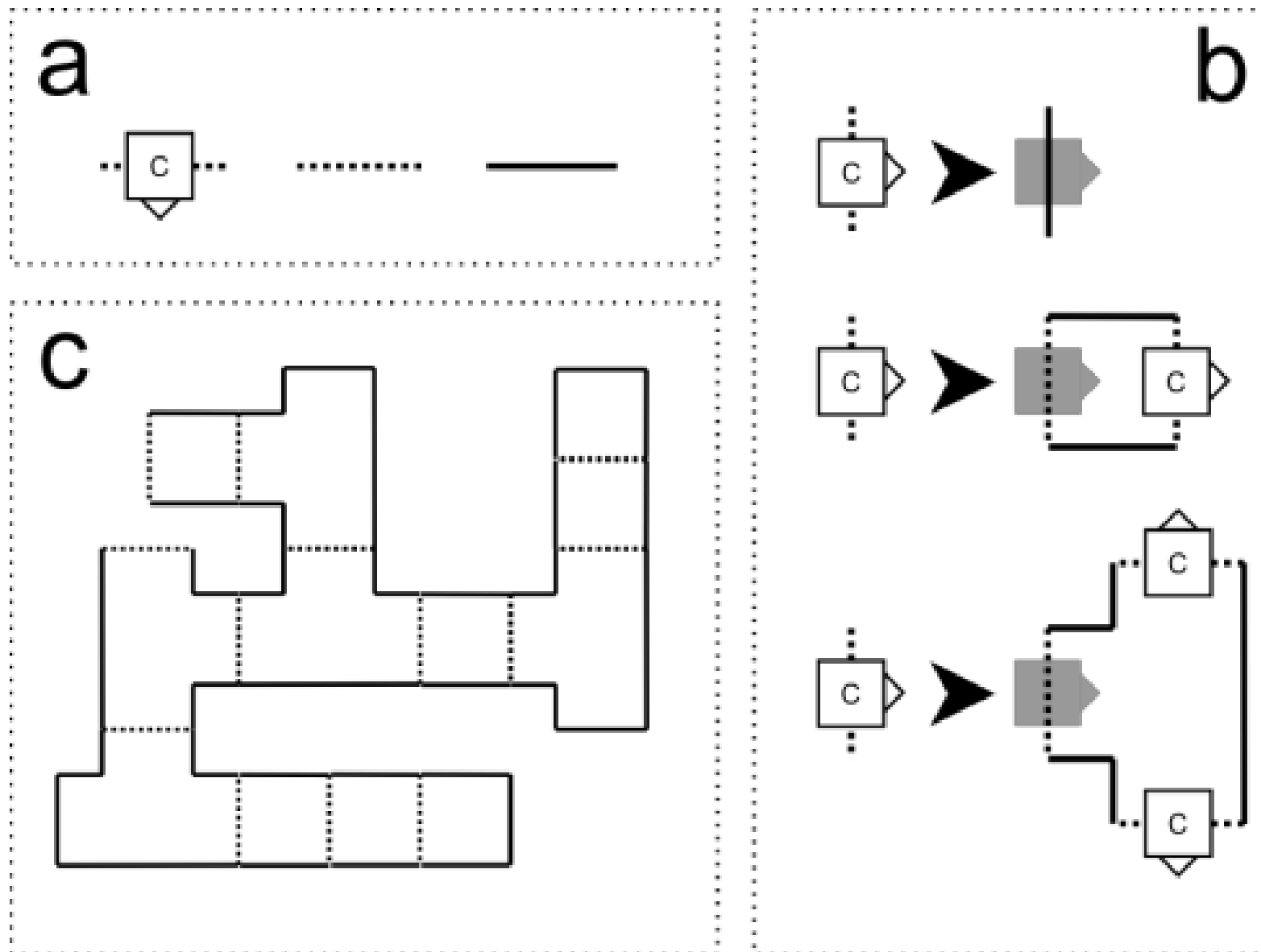
Alphabet:

bl = boss (level)	G = Gate	I = lock
bm = boss (mini)	g = goal	lf = lock (final)
C = Chain	H = Hook	lm = lock (multi)
CF = Chain (Final)	ib = item (bonus)	n = nothing/exploration
CL = Chain (Linear)	iq = item (quest)	S = Start
CP = Chain (Parallel)	k = key	t = test
e = entrance	kf = key (final)	ti = test (item)
F = Fork	km = key (multi piece)	ts = test (secret)

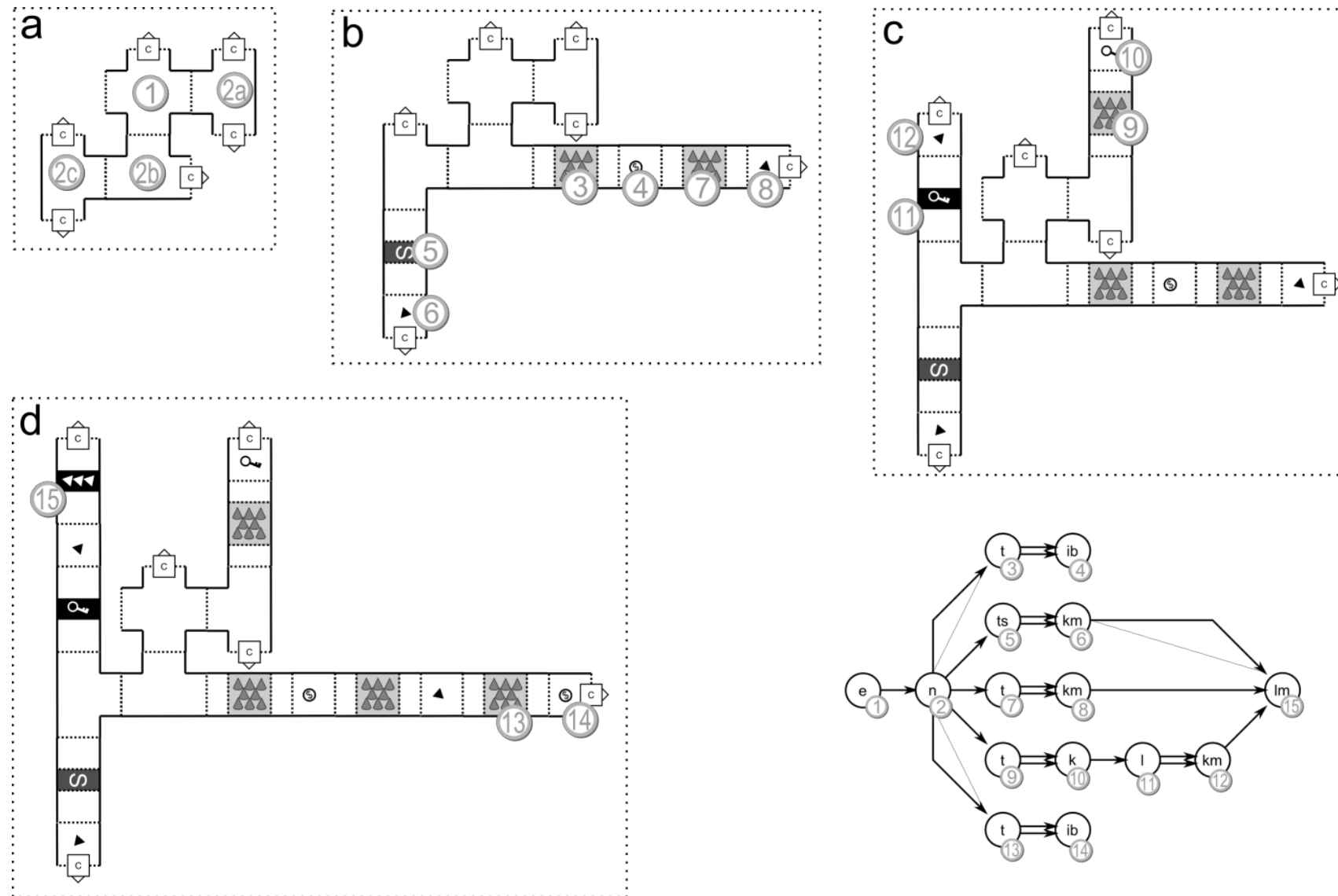
Generated mission



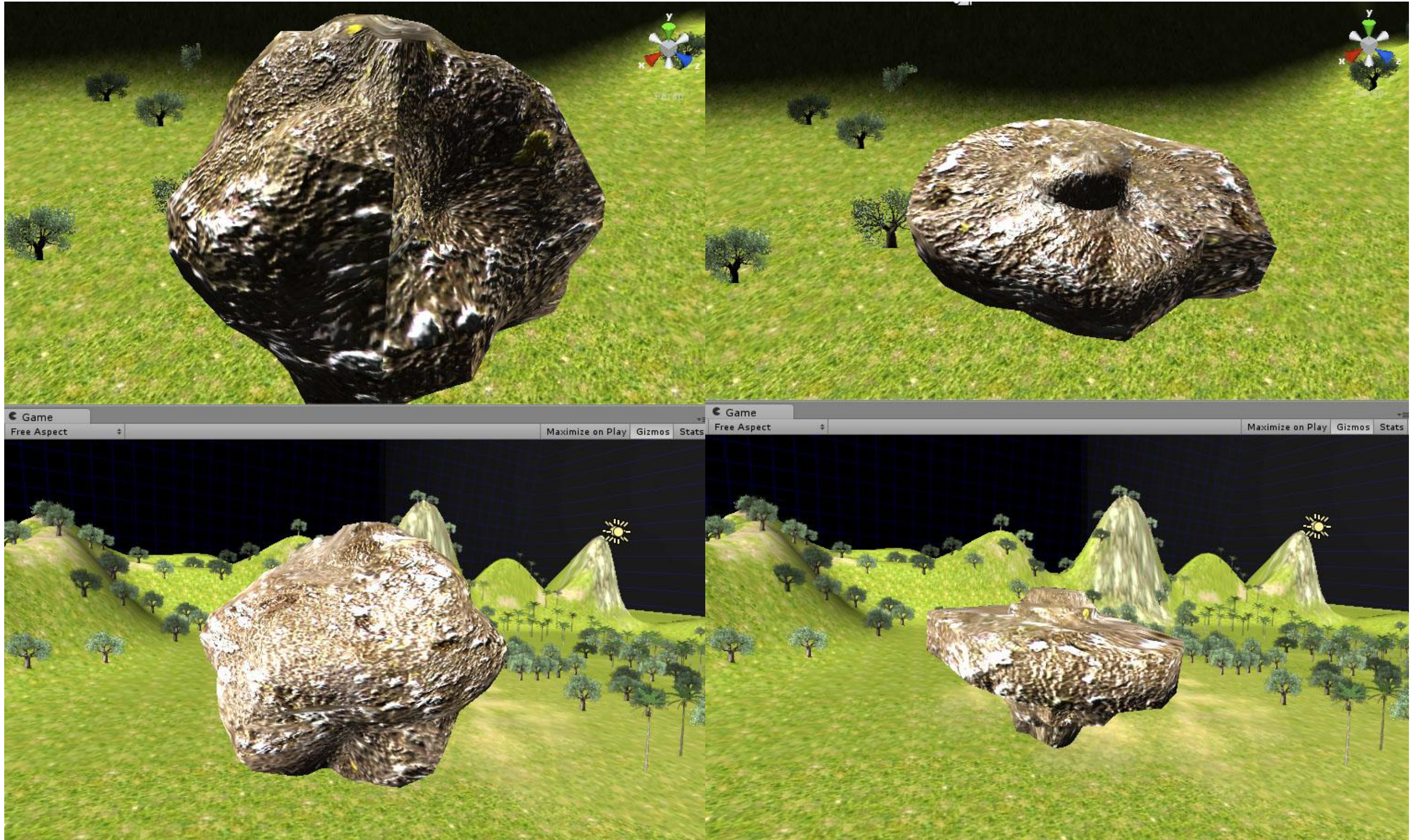
Shape grammars



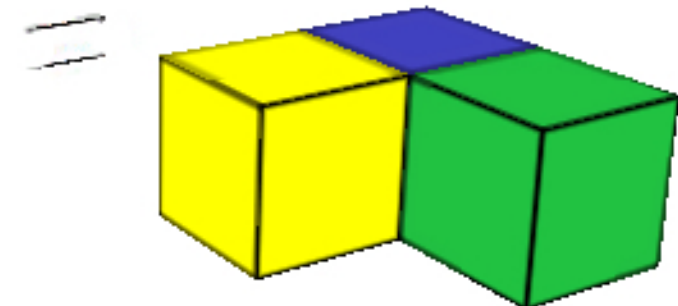
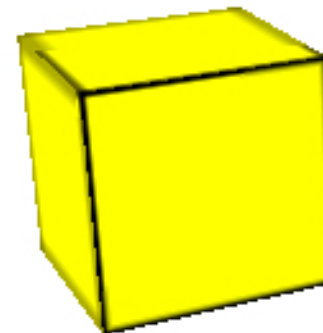
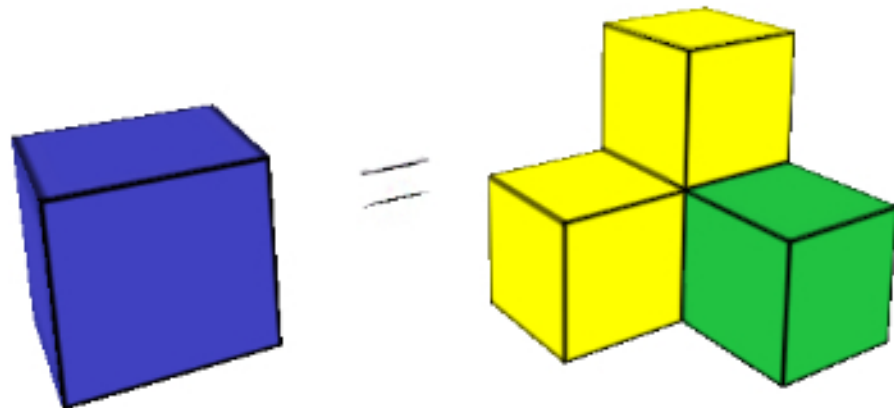
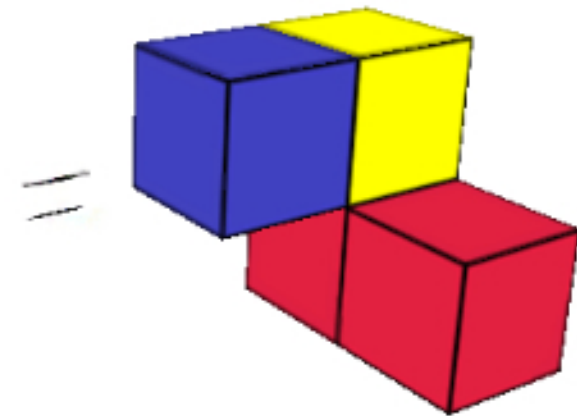
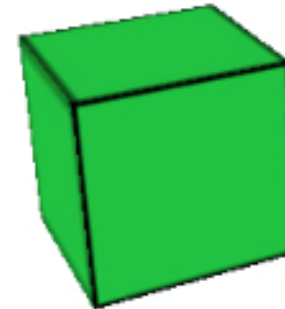
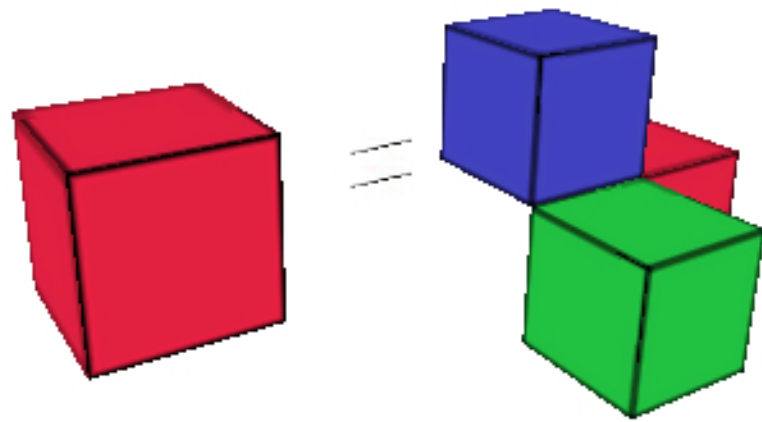
Shape grammars for level generation



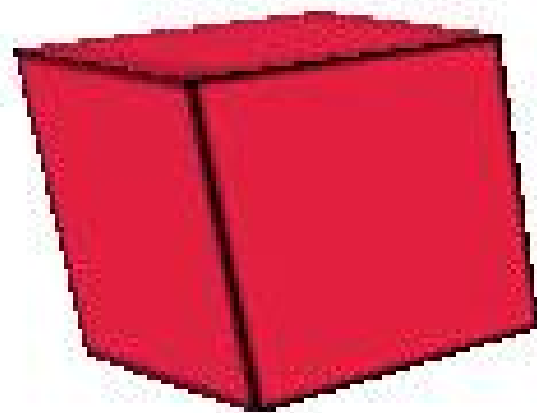
SpeedRock



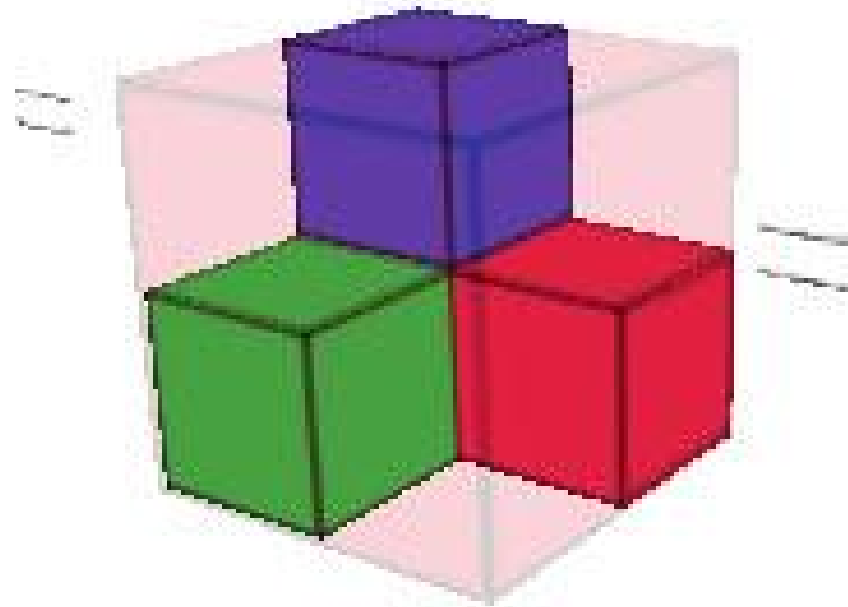
3D L-systems



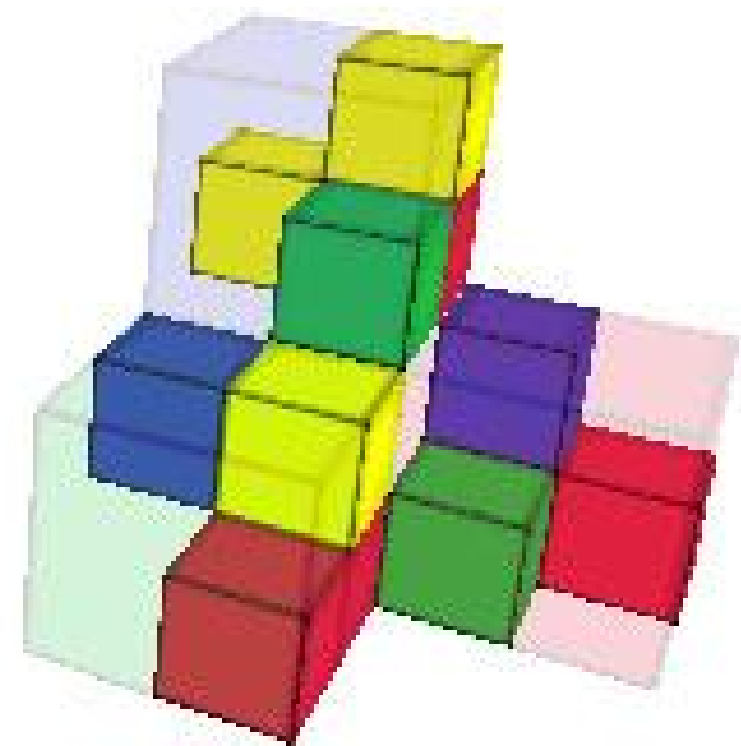
3D L-systems



Expansion 0

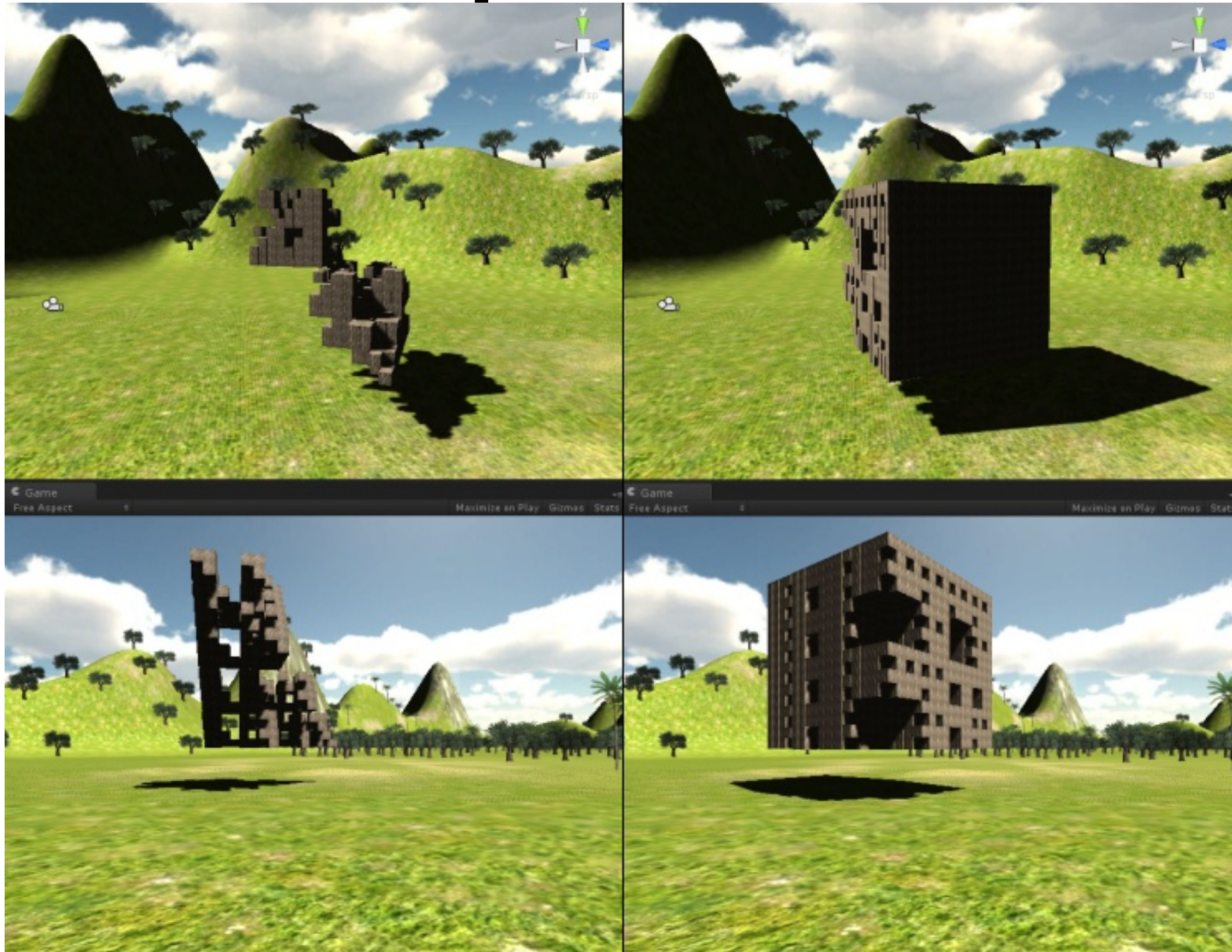


Expansion 1



Expansion 2

Random ruleset expansion



SpeedRock

- 3D L-system subdivision
- Implosion
- Erosion
- Vacuum sealing
- Optimization by evolutionary computation
- Dart, De Rossi, Togelius, FDG workshop on PCG 2011

A taxonomy of PCG

- Online/Offline
- Necessary/Optional
- Random seeds/Parameter vectors
- Stochastic/Deterministic
- Constructive/Generate-and-test

Issues in search-based PCG

- Content representation and search space
 - Direct or indirect?
- Fitness function
 - Direct, simulation-based, interactive?

Evolving Levels for Super Mario Bros Using Grammatical Evolution

Noor Shaker, Miguel Nicolau, Georgios N. Yannakakis, Julian Togelius, and Michael O'Neill

IT University of Copenhagen, University College Dublin

Grammatical Evolution

- Evolutionary algorithm + grammatical representation
 - Exploring wide space
 - Simple representation
- Design grammar
 - simple way of describing structure
 - Backus–Naur Form (BNF)

Backus Naur Form

- $G = \{N, T, P, S\}$
 - T: terminals
 - N: non-terminals
 - P: production rules
 - S: start symbol

Example

```
<exp> ::= <exp> <op> <exp>
        | ( <exp> <op> <exp> )
        | <var>
<op>  ::= + | - | * | /
<var> ::= X
```

Example

- genotype = (4,5,8,11)
- $4\%3 = 1 \rightarrow$ second production
 - $\langle \text{exp} \rangle ::= \langle \text{exp} \rangle \langle \text{op} \rangle \langle \text{exp} \rangle$
- $5\%3 = 2 \rightarrow$ third production
 - $\langle \text{exp} \rangle ::= \langle \text{var} \rangle ::= X$
- $8\%4 = 0 \rightarrow$ first production
 - $\langle \text{op} \rangle ::= +$
 - ...

$$\begin{aligned} \langle \text{exp} \rangle &::= \langle \text{exp} \rangle \langle \text{op} \rangle \langle \text{exp} \rangle \\ &\quad | (\langle \text{exp} \rangle \langle \text{op} \rangle \langle \text{exp} \rangle) \\ &\quad | \langle \text{var} \rangle \\ \langle \text{op} \rangle &::= + \mid - \mid * \mid / \\ \langle \text{var} \rangle &::= X \end{aligned}$$

Design grammar in Super Mario Bros

- Level = list of chunks
- chunk = gap, hill, platform, enemy, box, ...
- each chunk has a set of properties

Level generation

- Place the chunks in the level map according to the grammar
- constraints the chunks' properties so that the final level is playable

Design grammar

```
<level> ::= <chunks> <enemy>
<chunks> ::= <chunk> | <chunk> <chunks>
<chunk> ::= gap(<x>, <y>, <wgbeforeafterbeforeafterbeforeaftercbeforeafterbeforeafter2 | ...
| <box_type> (<x>, <y>)6

<box_type> ::= blockcoin | blockpowerup
| rockcoin | rockempty

<enemy> ::= (koopas | goompas) (<x>)2 | ...
| (koopas | goompas) (<x>)10
<x> ::= [5..95] <y> ::= [3..5]
```


Conflict Resolution

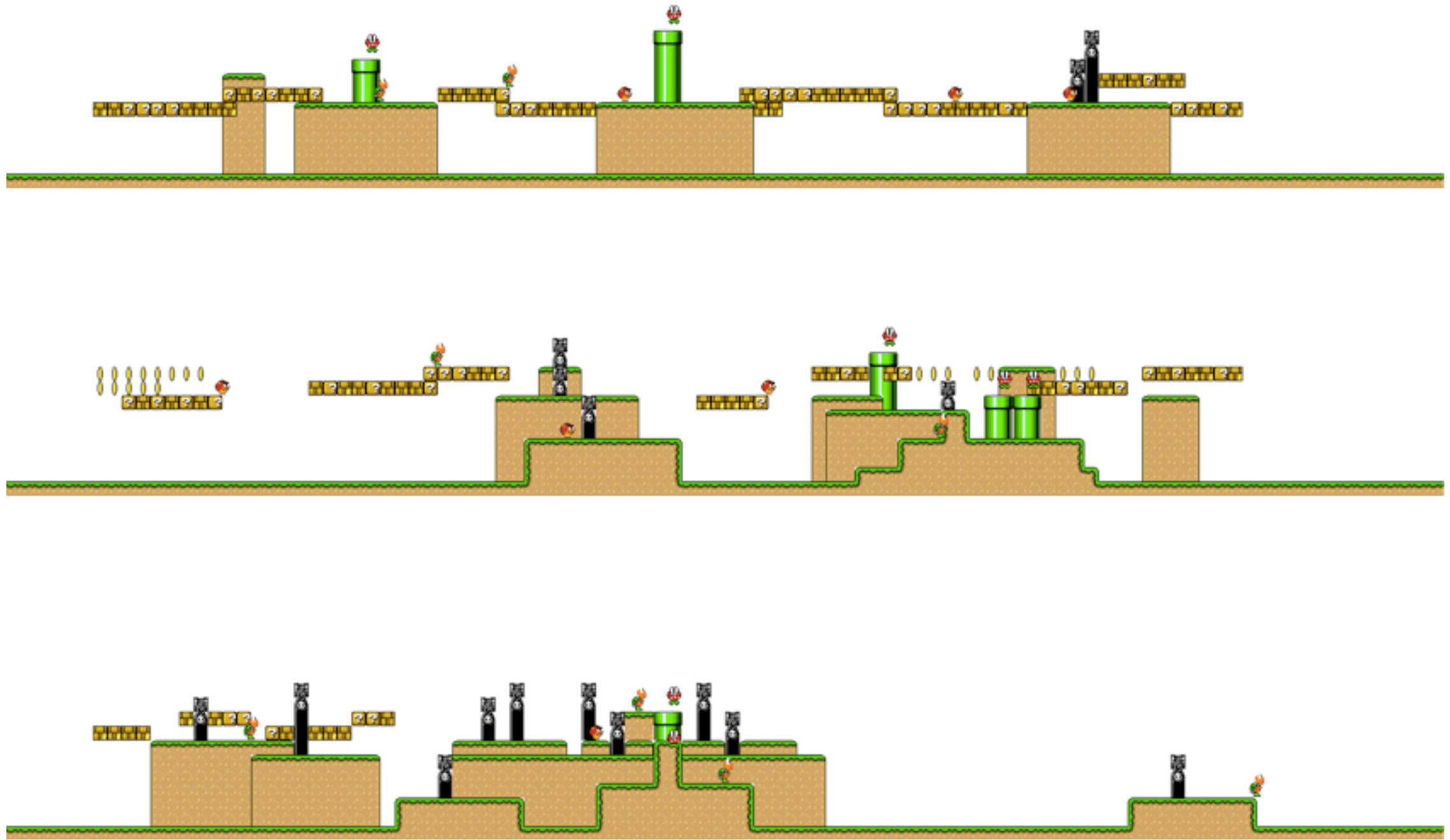
- Overlapping coordinates
 - Define priority for each element
 - When conflicting, the element with the higher priority is maintained and the other is removed
- Some of the chunks are allowed to overlap
 - Hills of different height
 - coins or boxes with hills



Fitness Function

- The more elements the better
- The less conflicting elements the better

Examples



Lab exercise

- Implement a bracketed L-system + turtle graphics program that draws a simple plant
- Arbitrary n (number of rewrites)
- Axiom: F
- Grammar: $F \rightarrow F[-F]F[+F][F]$
- Turning angle: 30°
- Tip: implement rewriting first, then graphics interpretation!