

Ludic Computing Course CO345

Lectures 5 and 6

Visual Grammars

and some Applications

Simon Colton & Alison Pease
Computational Creativity Group
Department of Computing
Imperial College, London

ccg.doc.ic.ac.uk

Lectures 5 and 6

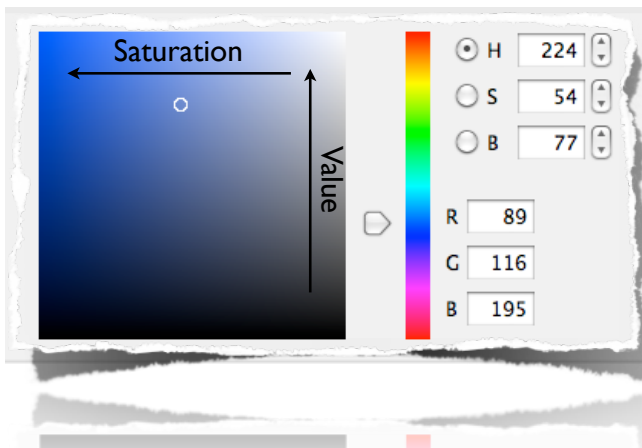
Overview

- Brief Graphics Preliminaries
- Grammars
- Shape grammars in general
- Context free design grammars
- L-systems
- Applications to the visual arts
- Applications to terrain modelling for games
- Applications to city generation for games

Graphics Preliminaries

HSV A Colour Model

- Hue, Saturation, Value (=Brightness)
 - Hue: 1-360 (which colour)
 - Saturation: 0-100 (how vibrant)
 - Value: 1-100 (how bright)
 - Alpha: 1-100 (how transparent)



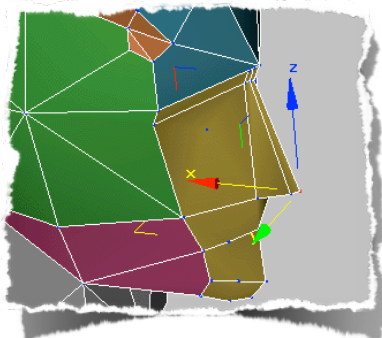
H	S	V	Colours
*	*	0	Black
*	0	50	Grey
*	0	100	White
0	100	100	Red
36	100	100	Orange
62	100	100	Yellow
116	100	100	Green
216	100	100	Blue
279	100	100	Indigo
331	100	100	Violet

Note that it's a colour *wheel*
Hue: 0 = 360

Graphics Preliminaries

3D Polygons

- Polygons are represented as three or more points in a 2D plane in 3D space ($P_1, P_2, \dots, P_n$), with each (P_i, P_{i+1}) and (P_n, P_1) connected by a straight line. A collection of polygons which share points and edges comprises a 3D surface



<http://www.webreference.com/3d/lesson71/part2.html>



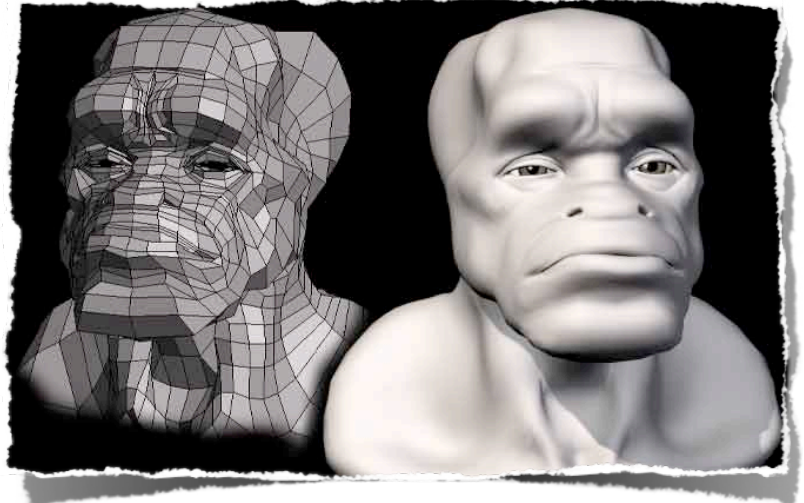
By Ken Cope

http://www.ozcot.com/mod_cat.htm

Graphics Preliminaries

3D Polygons

- Software can interpolate the polygons
- And simulate light reflecting off the polygons
- The results can be remarkably realistic...



<http://www.brilliant-creations.com/zbrushtutorial.htm>

Graphics Preliminaries

3D Polygons



By Gilles Tran - entirely modelled with POV-Ray, Radiosity, Rhino and Cinema 4D
http://en.wikipedia.org/wiki/File:Glasses_800_edit.png

Between Art and Content Generation

- We're looking at art generation techniques, which could be used for content generation in video games
- We think of content generation as producing sets of polygons
- Shape grammars can and are applied to polygon generation for games

Grammar

Definitions

- Let Σ be a finite set of terminal symbols
- Let N be a set of non-terminal symbols
- Let $\Sigma \cup N$ be the union of terminal and non-terminal symbols
- Let $(\Sigma \cup N)^*$ be the set of strings of terminal and non-terminal symbols
- Let P be a set of production rules
 - Where each production rule takes as input a string from $(\Sigma \cup N)^*$ which must contain a non-terminal symbol, and outputs another string from $(\Sigma \cup N)^*$
- Let $S \in N$ be the start symbol
- We say that the quadruple $\langle \Sigma, N, P, S \rangle$ defines a formal grammar
- We denote $x \Rightarrow_G y$ for the application of production rule $G \in P$ to $x \in (\Sigma \cup N)^*$ to produce $y \in (\Sigma \cup N)^*$. We say that G derives y from x

Grammar

Example

$$\Sigma = \{a, b, c\}$$

$$N = \{S, B\}$$

Production rules, P:

$$1. S \rightarrow aBSc$$

$$2. S \rightarrow abc$$

$$3. Ba \rightarrow aB$$

$$4. Bb \rightarrow bb$$

$$S \Rightarrow_1 aBSc \Rightarrow_2 aBabcc \Rightarrow_3 aaBbcc \Rightarrow_4 aabbcc$$

Context Free Grammars

- Special case of formal grammars
- P contains production rules for which the input is a single non-terminal from N
- “Context free” indicates that the ways in which non-terminals are re-written are not dependent on the surrounding symbols
- We can draw a parse tree which shows the strings which can be produced

Example production rules:

$$1. S \rightarrow aBSc$$

$$2. S \rightarrow abc$$

$$3. B \rightarrow aB$$

$$4. B \rightarrow bb$$

$$S \Rightarrow_1 aBSc \Rightarrow_2 aBabcc \Rightarrow_3 aaBabcc \Rightarrow_4 aabbabcc$$

$\Rightarrow_5$

$$abbabcc$$

Parse Tree

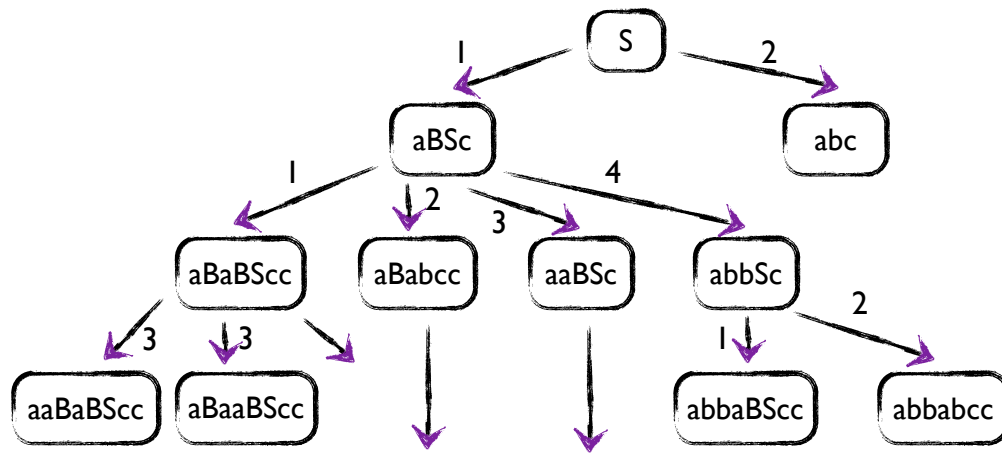
Example

1. $S \rightarrow aBSc$

2. $S \rightarrow abc$

3. $B \rightarrow aB$

4. $B \rightarrow bb$



And so on...

Random Generation

of Sentences in a Context Free Grammar

- Start with the S node, then iteratively:
 1. Randomly choose a node, D, in the tree
 2. Randomly choose a non-terminal letter, L, in the string at that node
 3. Randomly choose a production rule, R, from P which applies to L. If no such R can be found, then start again from 1
 4. Apply R to L to derive a new sentence, X, and add this as a child node of D
 5. If S contains only terminals then stop and return X, else if no termination conditions hold, then continue from 1
- Termination conditions might include the length of the string, or the branch length of a node, or something tailored to the problem (e.g., shape size)

Random Generation

of Sentences in a Context Free Grammar

- Can assign probabilities to individual production rules, to bias the random generation
- For instance, giving higher probabilities to production rules containing only terminals (or with only one non-terminal) will bias the random generation towards smaller sentences
- Make sure that the sum of probabilities of production rules with the same input non-terminal is 1

Parse Tree

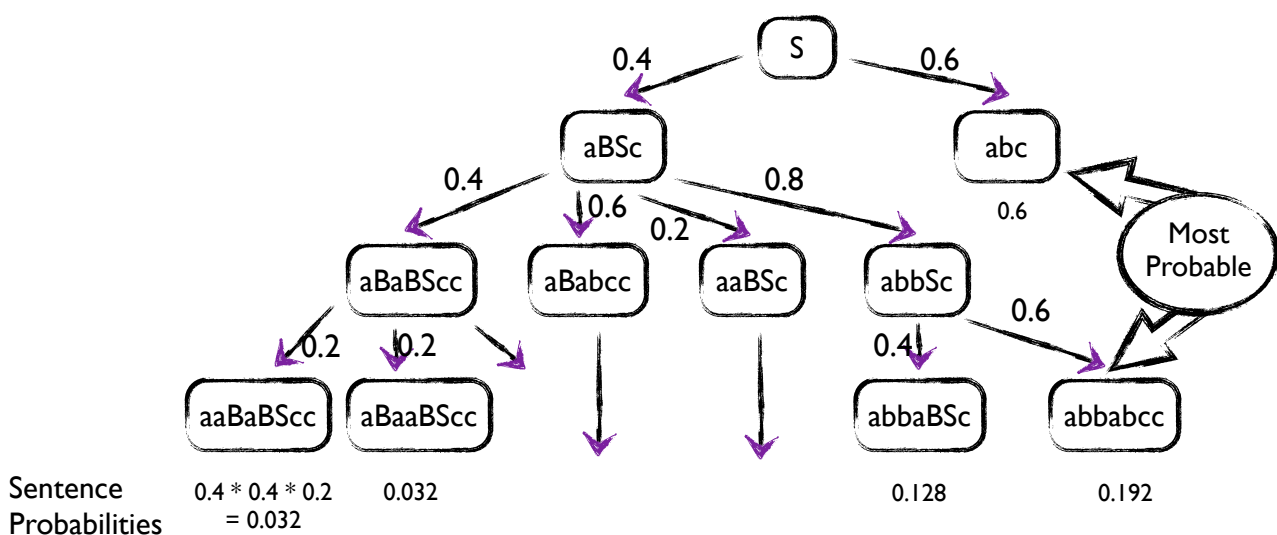
with Probabilities

1. $S \rightarrow aBSc$ (0.4)

2. $S \rightarrow abc$ (0.6)

3. $B \rightarrow aB$ (0.2)

4. $B \rightarrow bb$ (0.8)

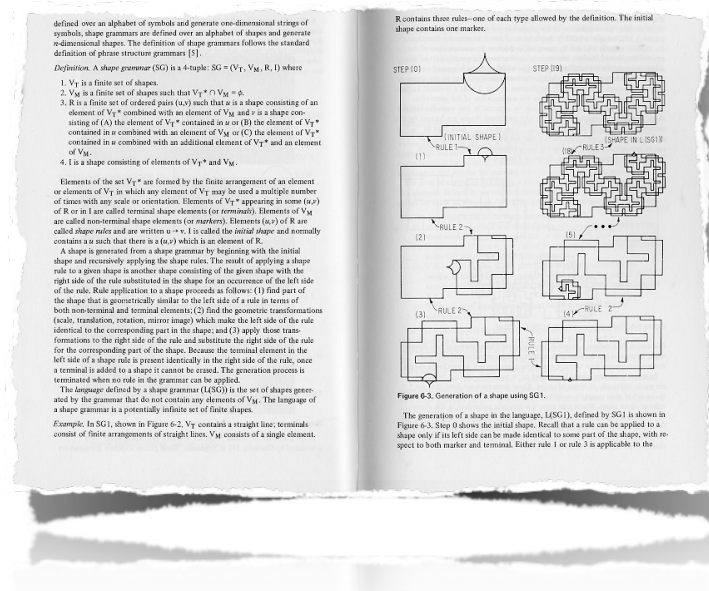


Overview of Shape Grammars

- Terminal symbols are shapes
- Grammars essentially define how to turn one sequence of shapes into another sequence of shapes
- Need a start rule, a non-empty set of transformation rules and a termination rule
- Software implementations also need to know how to place shapes onto the canvas (screen), which is not a problem in linguistic grammars
- Have been used in applications to architecture, video game design and in the visual arts

Overview of Shape Grammars

- Original paper by Stiny and Gips (“Best Computer Papers of 1971”)
- <http://www.shapegrammar.org/ifip/SGBestPapers72.pdf>



From Grammar to Shape Grammar

- First imagine that the letters from our example are replaced by shapes:
 - a = square ■, b = triangle ▲, c = circle ●
- Then imagine a sentence interpreter which places the shapes linearly next to each other on a graphics canvas
 - e.g., the screen or an image file

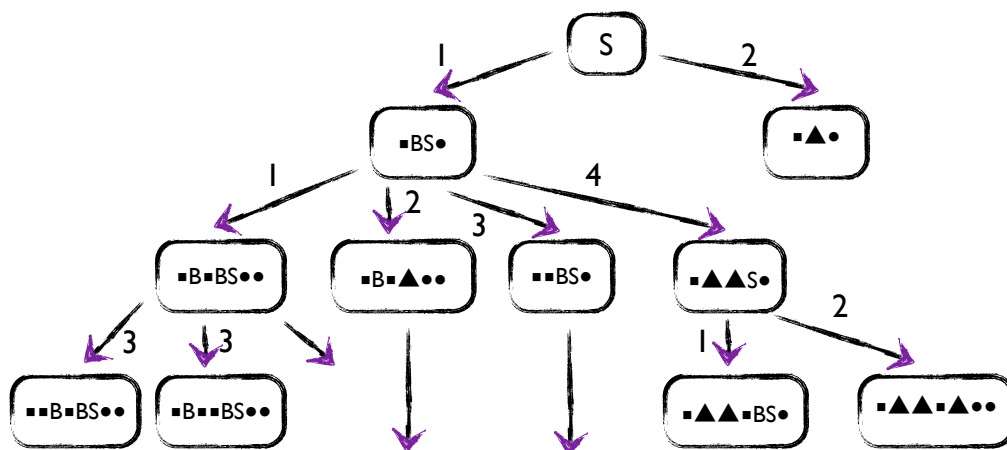
Linear Shape Grammar Example

1. $S \rightarrow \blacksquare BS \bullet$

2. $S \rightarrow \blacksquare \blacktriangle \bullet$

3. $B \rightarrow \blacksquare B$

4. $B \rightarrow \blacktriangle \blacktriangle$

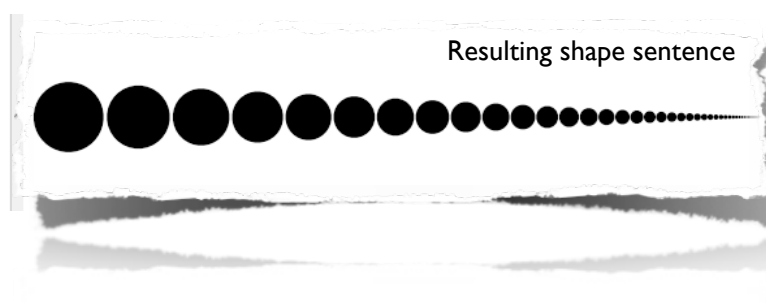


Shape Transformations

- Next, imagine that the production rules are allowed to transform any terminal symbol (shape) that they put into a sentence
 - And they make sure that the transform applies also to any terminal symbols (shapes) that are derived from the non-terminal symbols in the sentence that the rules are applied to
- These transformations replace the linear placement routine
- Transformations can include:
 - Translating, scaling, shearing, rotating, reflecting, changing HSVA

Simple Example

- Suppose we have only one production rule in the grammar: $R \rightarrow \bullet R$
- And that the rule also moves the shapes to the right and scales them to 0.9 of the original size



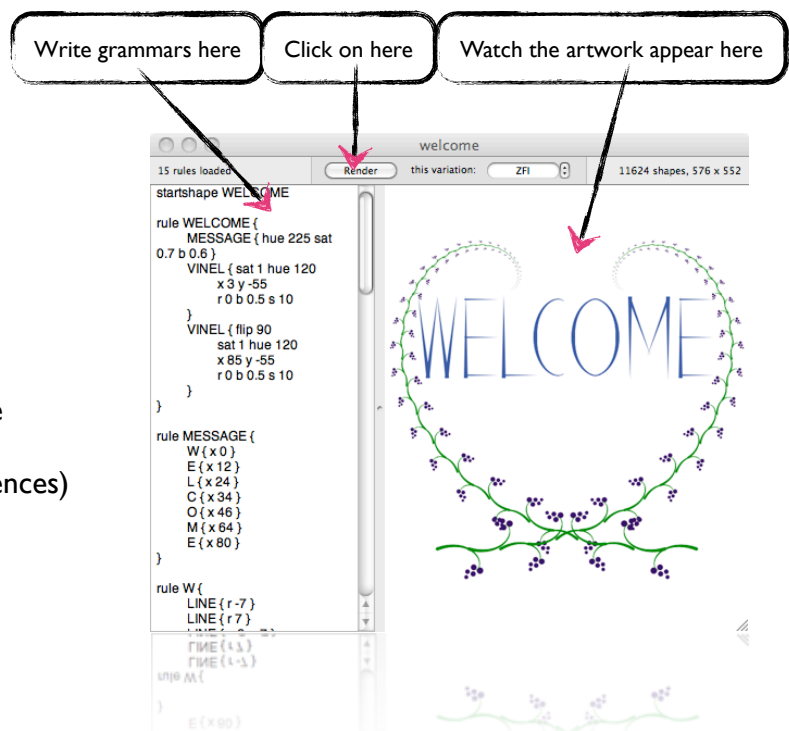
Quick Summary

- To design a shape grammar, we have available:
 - The shapes we allow as terminals
 - The rules of the grammar
 - The probabilities with which each rule is applied
 - The shape transformations that each rule performs when placing shapes on the screen

CFDG Software

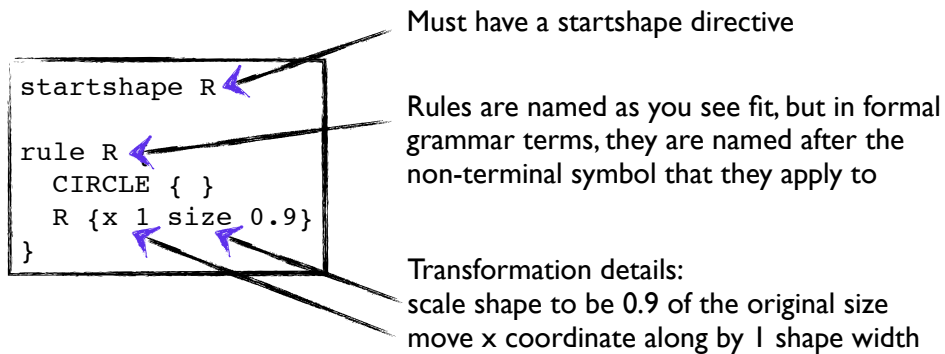
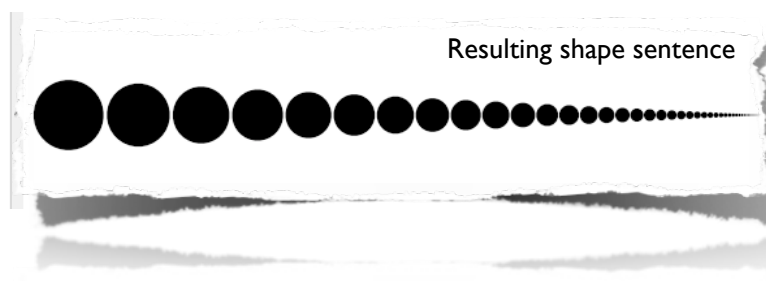
Overview and Screenshot

- Context Free Design Grammars
- Open source, available for Mac, Windows and Linux, at: www.contextfreeart.org
- Very fast implementation
- Applies similar random sentence expansion algorithm to the one described here (there are differences)
- Thousands of design grammars available at the web site, mostly produced by amateur artists



Simple Example

in CFDG Syntax



CFDG Software

Grammar Format

- Each grammar has a set of directives and a set of grammar rules
- Directives at the start of the file:
 - *startshape* *STARTING_RULE* - which rule to start with [mandatory]
 - *include* *OTHER_FILE.cfdg* - which other files to include
 - *background* { *COLOUR VALUES* } - what the background colour should be
 - *tile* { *s m n* } - tile the design on an m by n grid
- Rules are of the form: *rule* *RULENAME* { *rule code lines* }, where each code lines are
 - *SHAPE_OR_RULENAME* { *transformation parameters* }
 - *SHAPE_OR_RULENAME* [*transformation parameters*]
 - *n* * { *transformation parameters* } *SHAPE_OR_RULENAME* { *transformation parameters* }
- Where *SHAPE_OR_RULENAME* is either (a) one of the fundamental geometric shapes available: *CIRCLE*, *SQUARE*, *TRIANGLE* or (b) a path for a shape defined in the grammar, or (c) a rule name (possibly the same as *RULENAME*)

CFDG Software

Path Syntax

- Shape definition commands:
 - MOVETO, ARCTO, LINETO, CURVETO, CLOSEPOLY
 - And relative versions
- STROKE and FILL commands also available
- Stroke width is relative to the size of the shape

```
path dot {  
  MOVETO{y 1}          // required  
  ARCTO{y -1 r 1}  
  ARCTO{y 1 r 1}  
  FILL{hue 216 sat 1 b 0.7333}  
  MOVETO{y 1}          // can be left out  
  ARCTO{y -1 r 1}  
  STROKE{b -1 width 0.25}  
  MOVETO{y -1}         // can be left out  
  ARCTO{y 1 r 1}  
  STROKE{b 1 width 0.25}  
}
```



CFDG Software

Shape Adjustment Syntax

- Shape transformation options
 - *x n* : translation of n shape widths along x axis
 - *y n* : translation of n shape heights along y axis
 - *rotate n* : rotation by n degrees around the top left hand corner of the shape's bounding box
 - *size n* : scaling by factor n
 - *size m n* : scale horizontally by m and vertically by n
 - *skew m n* : shear transform in x/y plane m degrees from the y axis and n degrees from the x axis
 - *flip n* : reflect in x/y plane across a line through the origin at n degrees
 - Transformations are carried out in this order: translate/rotate/scale/skew/flip, but if you want to explicitly specify the order, use square brackets,
 - e.g., { *x 0.1 size 1.1 rotate 10* } and { *rotate 10 x 0.1 size 1.1* } are the same, but [*x 0.1 size 1.1 rotate 10*] and [*rotate 10 x 0.1 size 1.1*] are different
 - See www.contextfreeart.org/mediawiki/index.php/Shape_adjustments for further details and to check out the target colour functionality; abbreviations of transformation names; and the z axis
- Colour change options
 - *hue n* : add n modulo 360 to the shape's hue
 - *saturation n* : if (n < 0), move the saturation n% towards zero, but if (n > 0), move the saturation n% towards 1
 - *brightness n* : as saturation
 - *alpha n* : as saturation
 - For saturation, brightness and alpha, n must be between -1 and 1

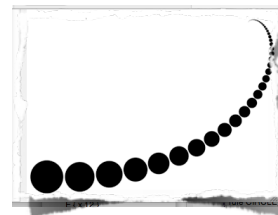
More Simple Examples

in CFDG Syntax

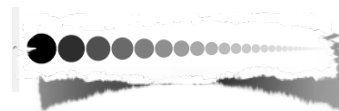
```
startshape R
rule R {
  CIRCLE { }
  R {x 1 size 0.9}
}
```



```
startshape R
rule R {
  CIRCLE { }
  R {x 1 size 0.9 rotate 5}
}
```



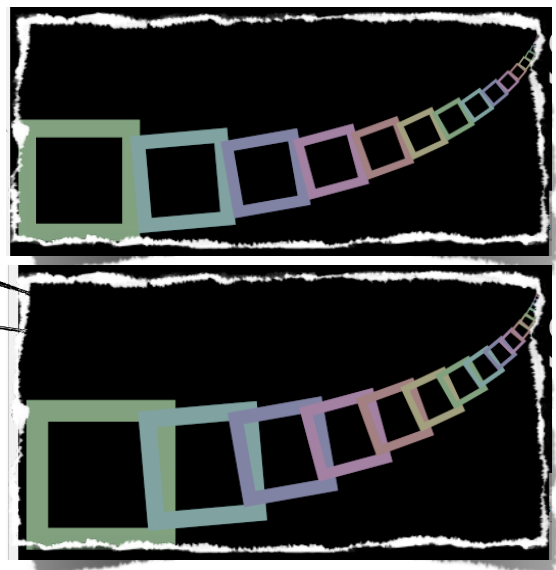
```
startshape R
rule R {
  CIRCLE { }
  R {x 1 size 0.9 brightness 0.9}
}
```



Less Simple Example

in CFDG Syntax

```
startshape BOX_TRAIL
background { b -1 }
rule BOX_TRAIL {
  box { hue 117 saturation 0.3 brightness 0.5 }
  BOX_TRAIL { size 0.8 rotate 5 x 1 hue 60 }
}
path box {
  MOVETO { x 0.5 y 0.5 }
  LINETO { x -0.5 y 0.5 }
  LINETO { x -0.5 y -0.5 }
  LINETO { x 0.5 y -0.5 }
  CLOSEPOLY { }
  STROKE { width 0.17 }
}
```



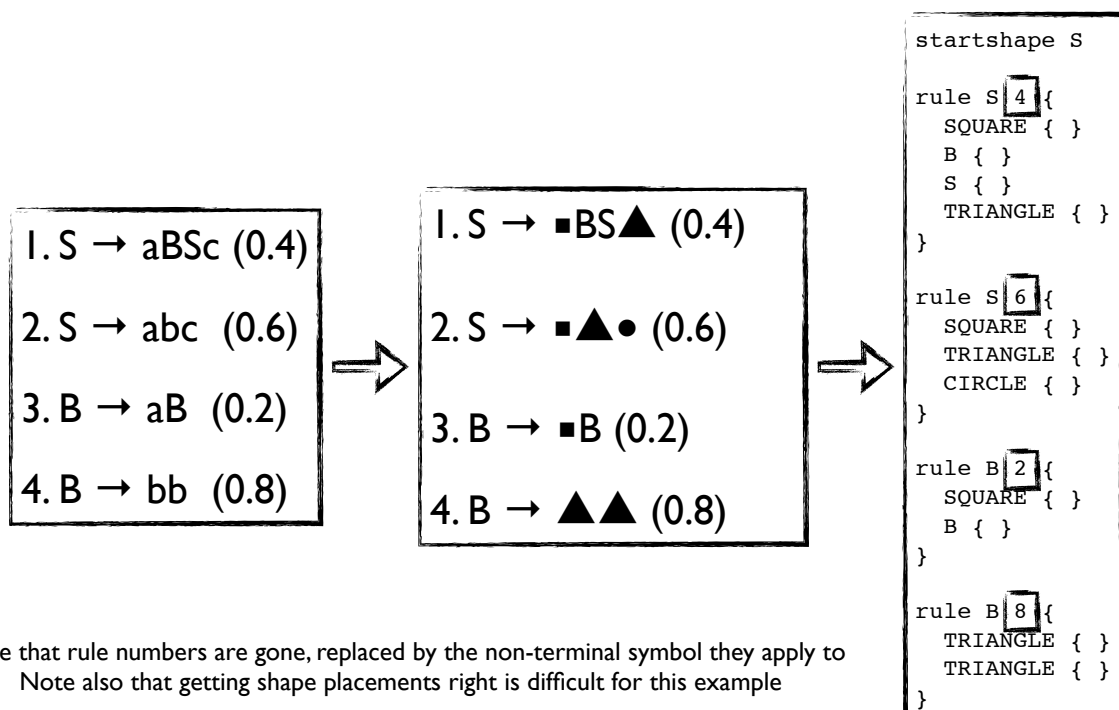
Note: meaningful rule names; change of brackets; paths

Introducing Randomness

- Each grammar rule applies to a single non-terminal, which the rule is given as a name
 - This is possible because we are using context free grammars
- Rules can each be assigned a probability
 - Expressed as a number which is taken relative to the numbers for the other rules with the same name
 - Hence no need to make numbers add up to 1
 - Default number (if missing) is: 1
- CFDG applies each rule according to the probability distribution
- Software allows you to explicitly choose the random seed, from AAA to ZZZ, so you can reproduce designs exactly

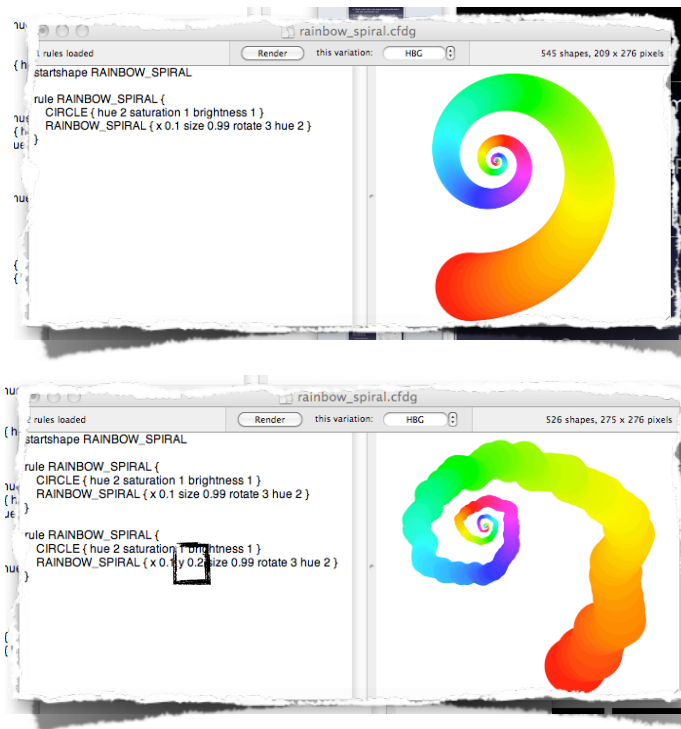
Introducing Randomness

Example



Introducing Randomness

Another Example



Using Repetition



- CFG also allows you to specify the number of times in a row you apply each rule
- Syntax:
$$n * \{ \text{transformations} \} \text{SHAPE_OR_RULE} \{ \text{transformations} \}$$
- For example, this code gives the example above:
$$6 * \{ \text{rotate } 30 \text{ brightness } 0.2 \} \text{CIRCLE} \{ \text{size } 1 \text{ } 0.1 \text{ brightness } 0.1 \text{ saturation } 1 \}$$

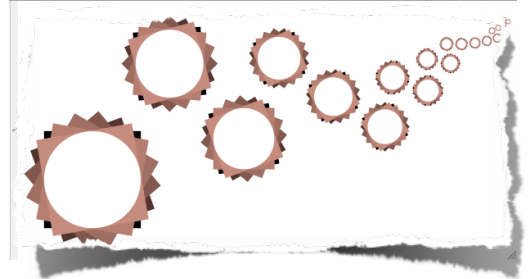
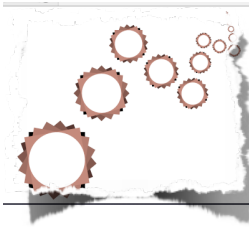
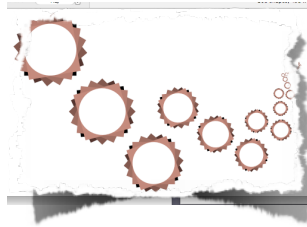
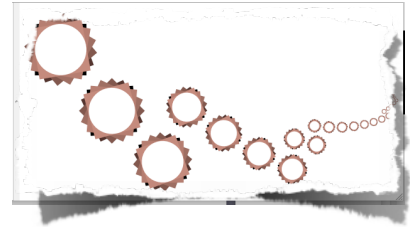
Final Toy Example

```
startshape LOOP_TEST
```

```
rule LOOP_TEST {  
  BROWN_SQUARES { }  
  LOOP_TEST { x 0.8 size 0.7 rotate 5 y 1.2 }  
}
```

```
rule LOOP_TEST {  
  BROWN_SQUARES { }  
  LOOP_TEST { x 1 size 0.9 rotate 5 y -1.2 }  
}
```

```
rule BROWN_SQUARES {  
  6 * { rotate 20 brightness 0.2 } SQUARE { hue 10 saturation 0.5 brightness 0 }  
  CIRCLE { brightness 1 }  
}
```



Rendering and Termination

- CFDG renders the shapes into the given window area, but whenever it has to render a shape outside the area, it translates the shapes and zooms out, i.e., it reduces the sizes of all the shapes on screen
- The process will stop if there are no non-terminals in the sentences to derive new sentences from (as any grammar)
- The process also stops when the size of any shape becomes less than one pixel
 - The user can define the minimum shape size also

Video of Rendering



L-Systems

- Originally used to describe growth mechanisms of organisms such as yeast
 - Seminal work by Aristide (L)indenmeyer in 1968
- Used in Artificial Life simulations and in fractal image generators
- Formal grammar with simplifications and one specialisation:
 - As many production rules as possible are applied at each iteration

L-Systems Definition

- In a simplified view, an L-system consists of three components:
 - An alphabet of a finite set of letters
 - An initiator, which is a string of letters (word) from the alphabet
 - Also called the axiom
 - A set of productions which map one letter from the alphabet to a word
- At each iteration, every letter is looked at, and each production rule is tried in turn, to derive a new word to replace the letter, until one is found
- Randomness can be achieved by either (a) randomising the order of the productions at each iteration, or (b) giving each production only a random chance - over some probability distribution - of being applied at each iteration
- We call these stochastic L-Systems

Fibonacci Example

- Alphabet: {A, B}, Initiator: A
- Productions: { $P_1: A \rightarrow B$, $P_2: B \rightarrow AB$ }
- Iterations:

Start with the Initiator: A

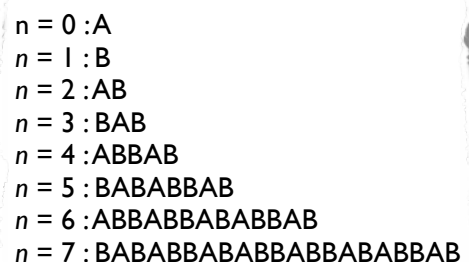
Can use P_1 to replace A with B, giving: B

Cannot use P_1 on B, but can use P_2 , giving: AB

Can replace the A with B using P_1 , and replace the B using P_2 , giving: BAB

Use P_2 to replace the first B with AB, then use P_1 to replace the A with B, then use P_2 again to replace the final B with AB, giving: ABBAB

And so on...

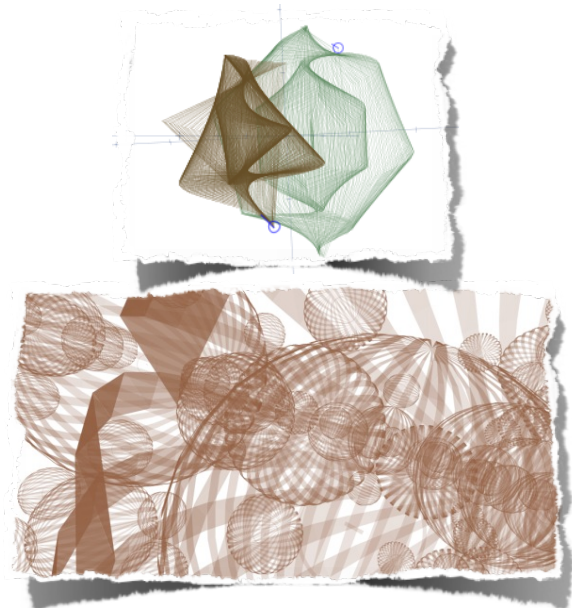


$n = 0 : A$
 $n = 1 : B$
 $n = 2 : AB$
 $n = 3 : BAB$
 $n = 4 : ABBAB$
 $n = 5 : BABABBBAB$
 $n = 6 : ABBABBBABBBAB$
 $n = 7 : BABABBBABBBABBBAB$

String sizes: 1, 1, 2, 3, 5, 8, 13, 21, ...
(Fibonacci series)

L-Systems Using Turtle Graphics

- Can stop the iteration at a certain point, and interpret the final word as a set of instructions for drawing lines, as per turtle graphics commands
- Commands are relative to the turtle's position, e.g., go forward 10, turn left 90 degrees, etc.
- Use an alphabet such as {F,B,R,L} for forward, backward, right, left, etc., then generate a long word using the L-system, and plot it



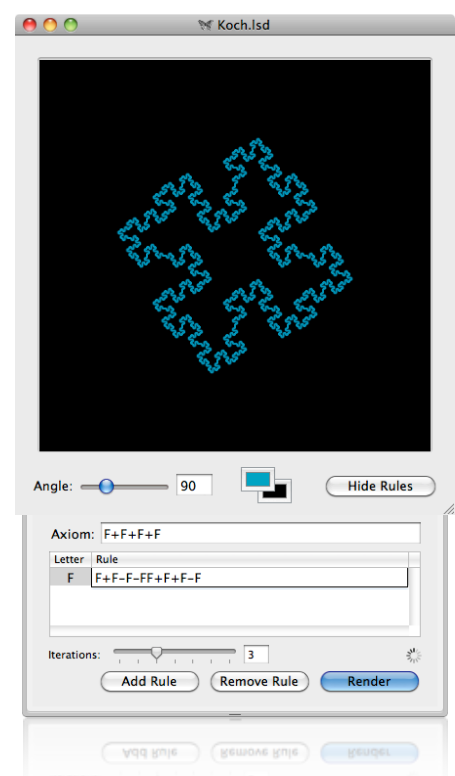
Produced by the Java 3D Turtle Graphics library here:
<http://spencertipping.com/#section=cheloniidae>

LSystem Software

- Freeware available for Mac OS X at:
 - <http://www.spiderland.org/lssystem/>
- Alphabet: {F, B, +, -, <, >, [,], @n, !}
- Interpreted as below
- And other non-terminal symbols
- Software enables you to choose the line colour, background colour, and angles

```

F      Go forward
B      Go backwards
+      Rotate right (by the user-defined angle)
-      Rotate left (by the user-defined angle)
<      3D Rotate up (by the user-defined angle)
>      3D Rotate down (by the user-defined angle)
[      Start a new branch
]      End the branch previously started with '['
@n     Scale the line length by a factor of n
!      Swap the meaning of "left" and "right" (multiplies the "angle" by -1)
    
```



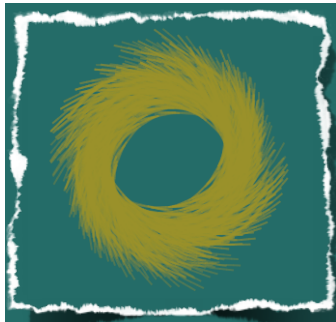
LSystem

Example

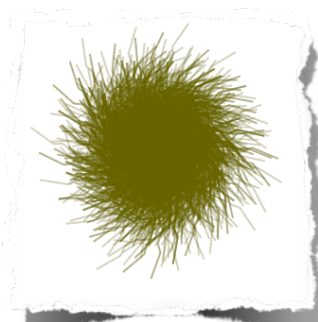
Alphabet: {F, +, -, [,]}

Initiator: F

Rule: $F \rightarrow F\text{-FF+}[-\text{FF}[\text{+F-FFF}]]\text{--}[\text{+F-FF}]$



4 Iterations
angle = 7



4 Iterations
angle = 308



5 Iterations
angle = 308

LSystem

More Examples

Alphabet: {A, B, F, -, and {C}}

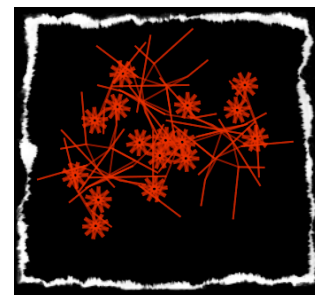
Initiator: A

Rule 1: $A \rightarrow AAB\text{BFF-B-FAB}$

Rule 2: $B \rightarrow C+C+C+C+C$

Rule 3: $C \rightarrow F+F+F$

Not seen
in output



4 Iterations

Alphabet: {A, F, +, -}

Initiator: $[F\text{-F+AF}][\text{+FA}]$

Rule 1: $A \rightarrow FF+A\text{-F}$

Rule 2: $F \rightarrow F[-\text{AFF+FA+F-BBB}][-\text{FF}]$



3 Iterations

Visual Grammar Systems

Considerations

- When building your own visual grammar systems...
- Main considerations: what types of grammar do you allow, and how will they be interpreted as visualisations?
 - Shapes (CFDG); Line drawings (LSystem); others
- Other considerations:
 - What transformations to allow
 - How to enable randomness and repeatability
 - What additional programmatic facilities to allow, e.g., loops, and what additional shape functionality to allow, e.g., paths

Visual Grammar Systems

Desirable Ludic Qualities

- We want simple grammars to lead to complex images/models
 - Language should be fairly expressive, but simple to program in, e.g., context free grammars
- Small tweaking should be stable, but larger tweaking of parameters should produce novel and surprising results
- Fast implementation
 - To enable lots and lots of trial and error

Applications

- Quick look at visual arts
- Quick look at terrain modelling
- More in-depth look at city generation

www.contextfreeart.org

Gallery

Stairway by Kipling



```
startshape S
size{ s 1.2 }

rule S{
  G{h 200 sat .5 b .1 z -999 }
  SW{y .55 h 30}}

rule G{
  240*{ y .005 b .01 sat -.01}
  CIRCLE{s 1.25 y -.5}}

rule SW{
  A{x -.15 y -.66 b .4 sat .5 a -1}
  SW{f 90 s .93 .96 b .05 sat -.05 h .2 z .04}}

rule A{
  B{ x .3 s .1 l r -45 skew 0 -2 z -2 }
  A{ s .995 .99 .997 y -.001 b .005 sat -.01 a .
    01}}

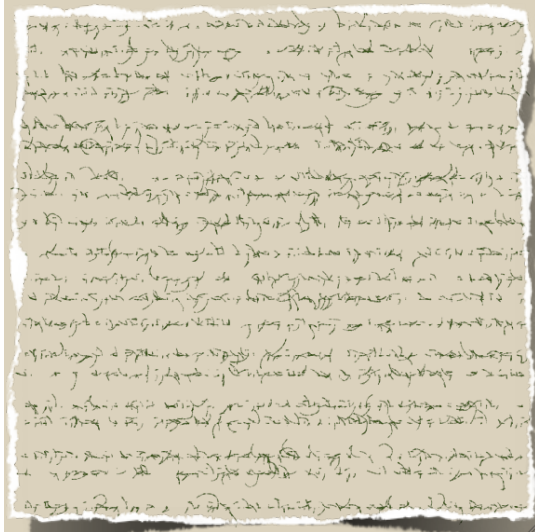
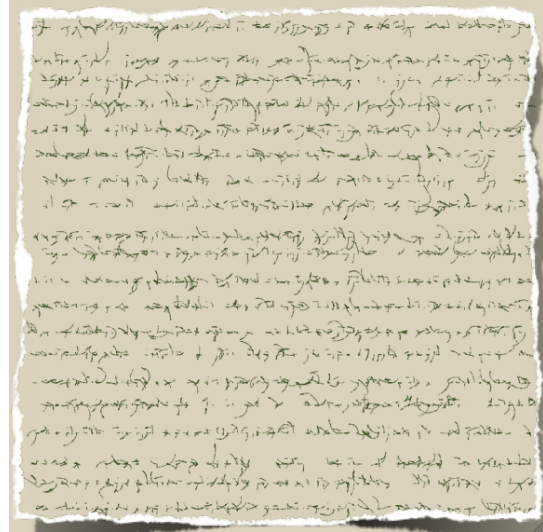
rule B{
  SQUARE{}
  CIRCLE{b -.08 s 1 .03 y -.5}}
```

```
GIKCFE{P -0.08 s 1 .03 l -2}
SQUARE{}
CIRCLE{b -.08 s 1 .03 y -.5}}
```

www.contextfreeart.org

Gallery

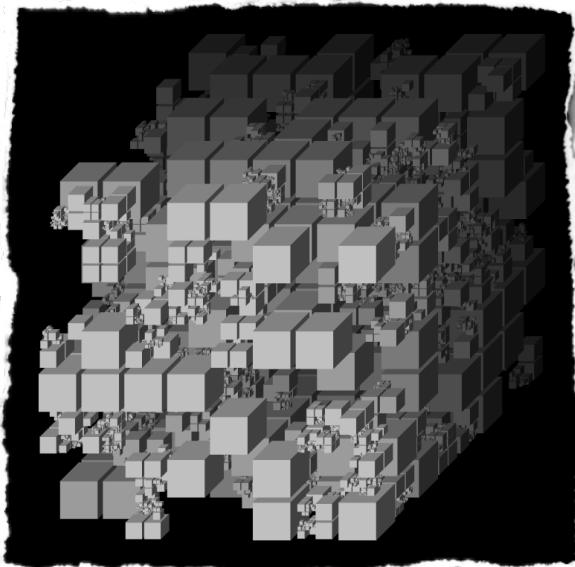
Ancient Scripts by leafnode



www.contextfreeart.org

Gallery

Shadow Box by adno



```
startshape ShadowBox
background { b -1 }
rule ShadowBox{
  8 * { x -0.5 y -0.5 z 128}{
    8 * { y 1 z 8 }{
      8 * { x 1 z 1 } Maybox { }
    }
    SQUARE { s 12 12 x 6 y 6 a -0.8 z -64 }
  }
}
rule Maybox{
}
rule Maybox{
  Twobox{ s 0.4375 0.4375 0.125 }
}
rule Maybox{
  Box{ s 0.21875 }
}
rule Twobox{
  2 * { x -0.5 y -0.5 z 4 }{
    2 * { y 1 z 2 }{
      2 * { x 1 z 1 } Maybox { }
    }
  }
}
rule Box{
  SQUARE{ s 4 4 b 0.75 }
  SQUARE[ x 1 y 3 skew 45 0 s 4 2 b 0.5 ]
  SQUARE[ x 3 y 1 skew 0 45 s 2 4 b 0.25 ]
}
}

gblume[ x 3 y 1 skew 0 45 s 3 3 p 0.52 ]
gblume[ x 1 y 3 skew 45 0 s 3 3 p 0.5 ]
gblume[ s 3 3 p 0.52 ]
```


www.contextfreeart.org

Gallery

hidden by Guigui



```
startshape o

background { b -.9 sat .8 hue 210 }
tile { s 20 }

rule O{
  45*(s .99)Vb{sat 1}
  CIRCLE{s 9 b .6 sat .6 hue 180 z 90 alpha -.75}
}

rule Va 10{Vb{}}
rule Va {Va{hue 30}}

rule Vb 1000{SQ{Vb{x 1 y .05 r 3 s .99 b .001 z 1}}
rule Vb 100{Vb{Vb{x -9 s .3 -.3 b .05}}
rule Vb 10{3*(r -9)Vb{s .6 b .1 sat -.3}}
rule Vb {Vb{flip 0}}

rule SQ { S{ } }
rule SQ 10{SQ{s 1.03} }
rule SQ 10{SQ{r 7} }
rule SQ 10{SQ{y .1} }

rule S{SQUARE{b .1 hue 180 alpha -.8}}
rule S 2{20{Vb{p '1 hue 180 sat .8}}
rule S 20 10{20{λ '1} }
rule S 20 10{20{r 1} }
```

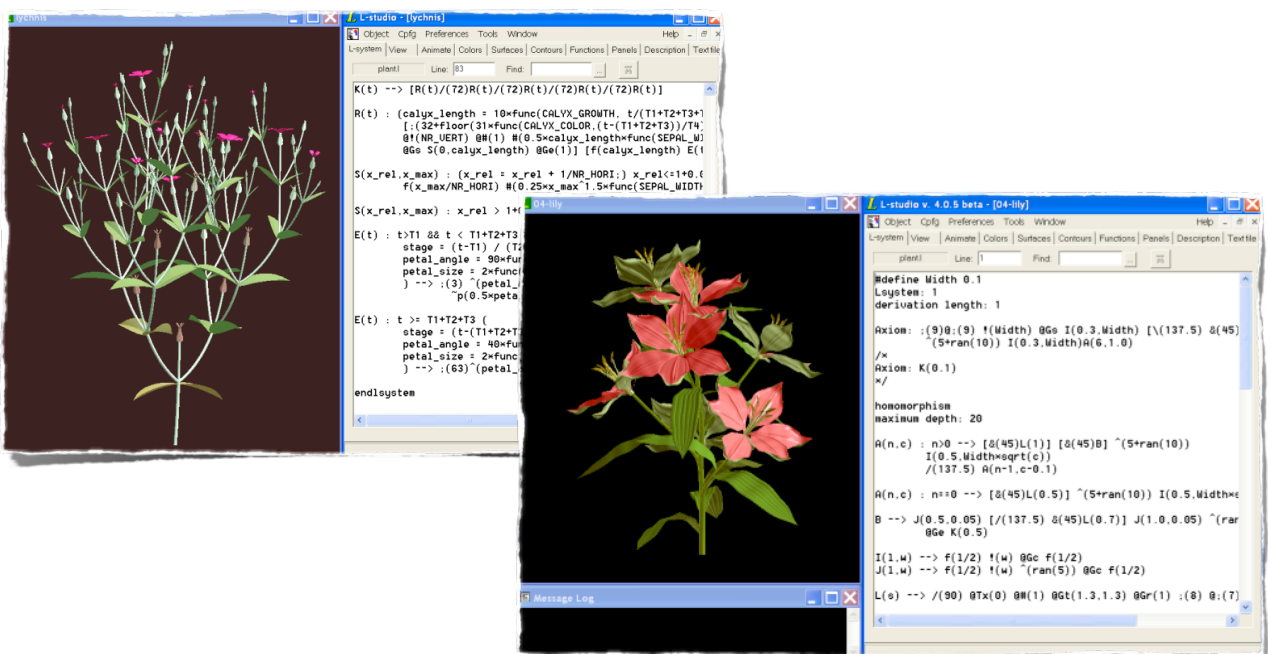
Terrain Modelling



Terrain Modelling

- Video game designers need procedural generation of vegetation and various terrains, because it is too laborious to create themselves
- Designers have “spray on” graphics systems for trees, grass and other vegetation
- Plant ecosystem modelling
 - For a good paper, see Deussen et. al:
 - <http://graphics.stanford.edu/papers/ecosys/ecosys.pdf>
 - Requires individual plant generation, which uses the cpfg L-system plant generator and produced the image in the previous slide

L-Studio Using cpfg from algorithmicbotanybay.org



Example Grammar

From “Genetic L-System Programming: Breeding and Evolving Artificial Flowers with Mathematica” by C. Jacob, 1995

$G_{\text{ArtFlower}} = (\Sigma, P = \{p_1, \dots, p_9\}, \alpha)$
 $\Sigma = \{f, \text{pd}, \text{pu}, \text{sprout}, \text{stalk}, \text{leaf}, \text{Leaf}, \text{bloom}\}$
 $\alpha: \text{sprout}(4)$
P: Sprout developing leaves and flower:
 $p_1: \text{sprout}(4) \rightarrow f \text{ stalk}(2) [\text{pd}(60) \text{ leaf}(0)]$
 $\text{pu}(20) [\text{pu}(25) \text{ sprout}(0)]$
 $[\text{pd}(60) \text{ leaf}(0)] \text{pd}(20)$
 $[\text{pu}(25) \text{ sprout}(2)]$
 $f \text{ stalk}(1) \text{ bloom}(0)$
Ripening sprout:
 $p_2: \text{sprout}(t < 4) \rightarrow \text{sprout}(t+1)$
Stalk elongation:
 $p_3: \text{stalk}(t > 0) \rightarrow f f \text{ stalk}(t-1)$
Changing leaf sizes:
 $p_4: \text{leaf}(t) \rightarrow \text{leaf}(t + 1.5)$
 $p_5: \text{leaf}(t > 7) \rightarrow \text{Leaf}(7)$
 $p_6: \text{Leaf}(t) \rightarrow \text{Leaf}(t - 1.5)$
 $p_7: \text{Leaf}(t < 2) \rightarrow \text{leaf}(0)$
Growing bloom:
 $p_8: \text{bloom}(t) \rightarrow \text{bloom}(t + 1)$
 $p_9: \text{bloom}(7) \rightarrow \text{bloom}(1)$

Figure 1: Example of a parametrized DOL-system modelling flowering stages of an artificial plantlike structure (adapted from [6], p. 83ff.)

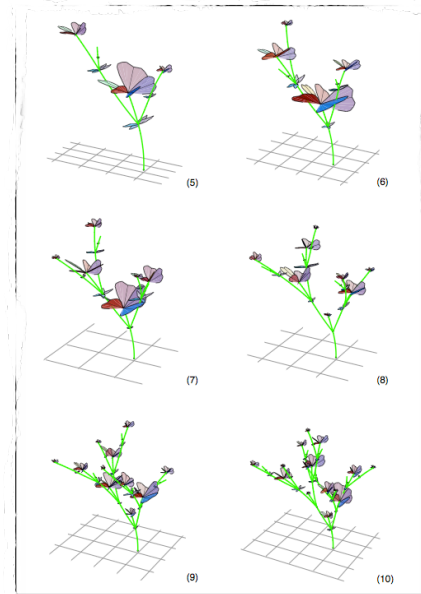


Figure 2: Visualizing growth stages of an artificial plant modeled by the DOL-system of fig. 1, the numbers in brackets mark the number of production iterations starting from the axiom.

Cellular Development Model

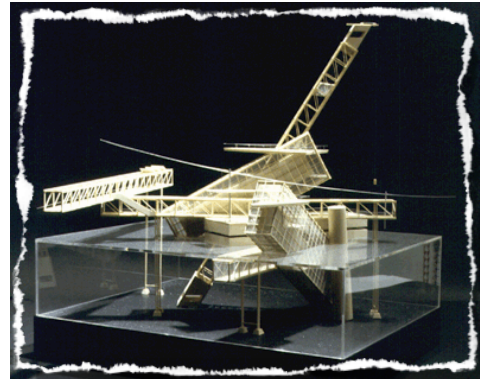
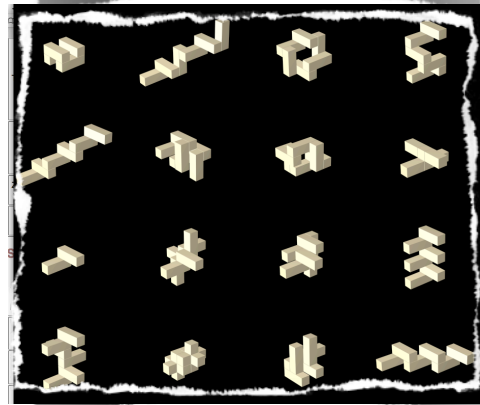
Jon McCormack

- Uses extensions of L-Systems: parametric, timed and differential
- Represents cells as simulations of biological entities within a simulated world
- Each cell is like an L-System, enhanced with a way of being interpreted in the world
- He evolves these systems using evolutionary design techniques



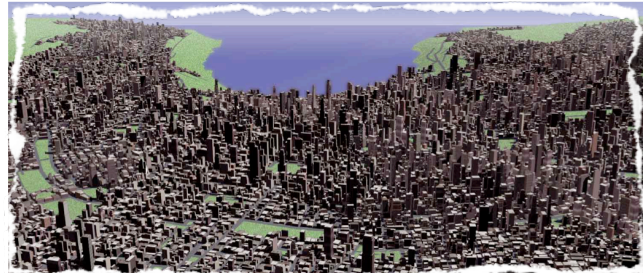
City Generation

- As with terrains, video game designers (and film makers) need automated city generation to cope with the demand for ever more extensive game worlds
- Possibly more difficult than terrain generation, as we have higher standards for buildings and there is perhaps greater variety in buildings and cityscapes than in vegetation and landscapes
- Visual grammars have been used a lot for architectural projects, for both creation and analysis of building designs
 - For instance, see:
<http://www.mit.edu/~tknight/IJDC/>
- We concentrate on the seminal work of Pascal Müller and colleagues, and Introversion Software's city generator for Subversion



CityEngine

by Parish and Müller

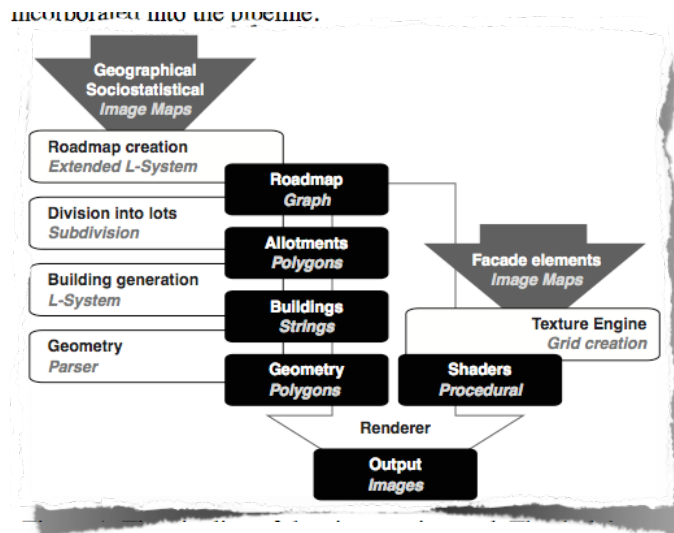


Original SIGGRAPH paper: <http://www.vision.ee.ethz.ch/~pmueller/wiki/CityEngine/PaperCities>
Commercialisation of CityEngine: <http://www.procedural.com/>

CityEngine

by Parish and Müller

- System Architecture



CityEngine - Roadmap creation

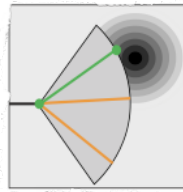
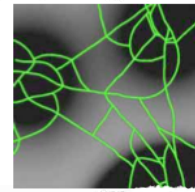
Creating the Street Map

- In overview, an L-system is used to produce a string of modules, each of which is interpreted as a command for city generation
 - Each command is parameterised, but the parameterisations are not handled by the L-system. Instead it produces a generic template which is called the ideal successor
- Whenever an ideal successor is produced, then
 1. The parameters for each module in the ideal successor are set by appealing to a globalGoals function
 2. The parameters are checked against a localConstraints function. Parameter values are adjusted if possible, if not then the module is deleted
- The final strings of parameterised modules define road and branch creation, road length, angles, road types, etc.

CityEngine - Roadmap creation

Global Goals

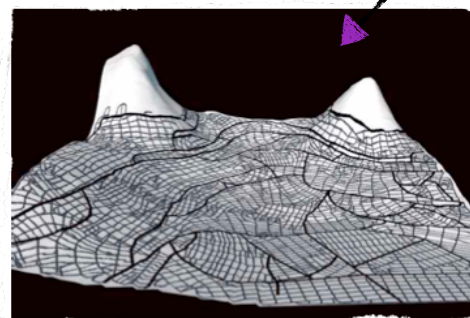
- The global city considerations are (i) population density and (ii) road patterns, which are handled by globalGoals
- Two road types
 - Highways must connect centres of high population, as determined by the geographical and statistical information provided
 - Streets must develop into the residential areas
- Highway road ends shoot out rays to find areas of high population density, and the sum of populations on each ray is used to choose the best one
- When a street reaches an area of no population, it ends
- Four types of road patterns: none, New York (checkers), Paris (radial), San Francisco (elevation)
 - Patterns for each of these are applied to the street maps



3 miles

Pattern name	Pattern	Example
Basic	No superimposed pattern.	
New York	Rectangular Raster	
Paris	Radial to center	
San Francisco	Elevation min or max	see figure 6

Table 1. An overview of the street pattern used in the



CityEngine - Roadmap creation

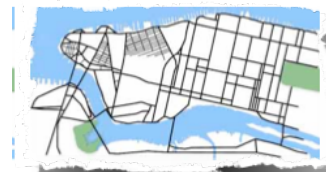
Local Constraints

- localConstraints checks whether:
 - Road ends are inside water, a park or another illegal area
- Fixes problems if possible by:
 - Pruning and rotating streets, and allowing highways some overlap
- Using self-sensitive L-Systems, roads are:
 - Pruned so that crossings from one road to another can be made, or
 - Extended, to reach another street
- Final process: road smoothing to get rid of long straight roads (unless desired)

28 steps



142 steps



Final



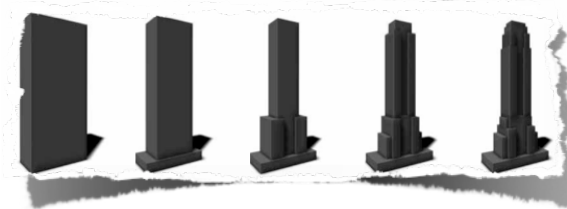
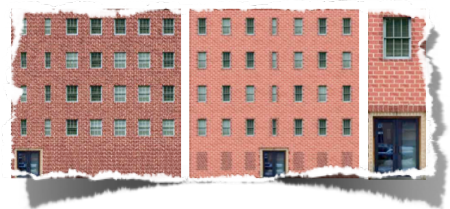
Real Manhattan



CityEngine

Building Generation

- Spaces between streets are cut up into blocks, then into allotments, and populated with a building per allotment
- A parametric stochastic L-system is used to generate strings of modules
 - Where modules can be transformations (scale and move), extrusion, termination and they can define geometric templates, e.g., for roofs and antennae
 - The initiator is the bounding box of the building site (i.e., floor plan area + allowed height). Note that the user can set building height regulations to simulate city planning directives
- Buildings are then coated in a texture
 - Either procedurally generated or stretched on from pictures of actual buildings



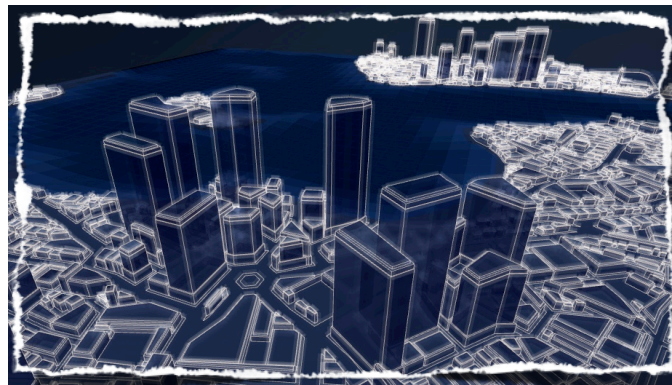
City Generation

Parish and Müller Video



Subversion City Generator

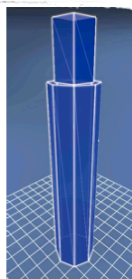
- Subversion is the next title which will be released by Introversion Software Ltd. Hmmmm.... is this still true??
- Espionage game set in a procedurally generated city - players cannot wait, so it has to be quickly generated
- See blog at: <http://www.introversion.co.uk/subversion/>



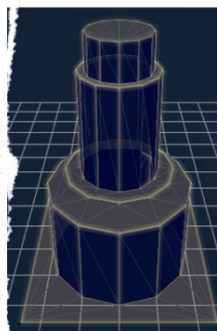
Subversion City Generator Building Generation

```
BEGIN CirclePart 1
  BEGIN Creation
    BEGIN Circle
      radius 15
      nodes 8
    END
  END
  BEGIN Sectors
    SetHeight 130
    BEGIN CreateChild
      type SquarePart
    END
  END
END
```

```
BEGIN SquarePart 1
  BEGIN Creation
    BEGIN Square
      length 15
    END
  END
  BEGIN Sectors
    SetHeight 30
  END
END
```



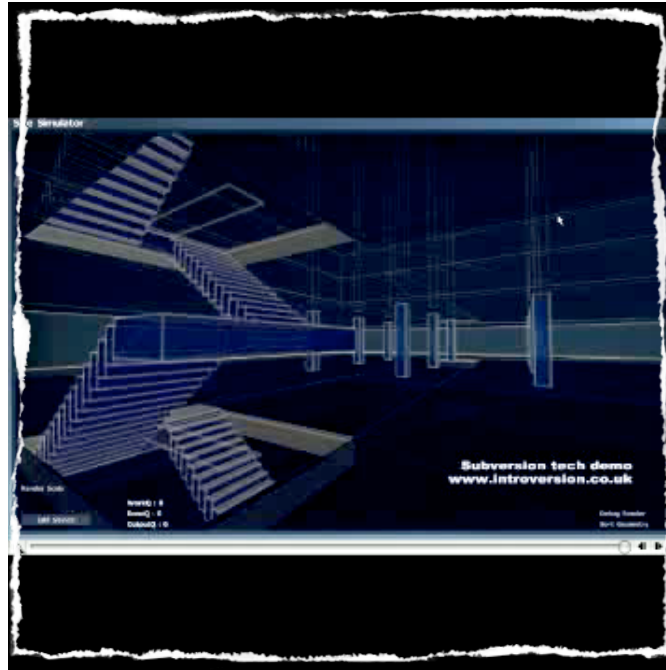
```
BEGIN parent1 1
  BEGIN Creation
    BEGIN circle
      radius 20
      nodes 12
    END
  END
  BEGIN Sectors
    SetHeight 25
    BEGIN Contract
      amount 5
      BEGIN ContractSuccess
        BEGIN NewSectors
          BEGIN MoveSectors
            offset 5
          END
        END
        BEGIN Contract
          amount 4
          BEGIN ContractSuccess
            BEGIN NewSectors
              SetHeight 29
            END
            BEGIN Contract
              amount 2
              BEGIN ContractSuccess
                BEGIN NewSectors
                  SetHeight 10
                END
              END
            END
          END
        END
      END
    END
  END
END
```



- Roadmap generation as per Müller et. al.
- Script based approach to building generation
 - Scripts generate node grids, which generate polygons

Subversion City Engine

Video



Summary

- Visual grammars are out there and being used for creation of artworks and game content
- Context free design grammars and L-Systems are most commonly used
- Game applications include terrain modelling and city generation

More Links

and CFDG Pictures

- <http://aigamedev.com/open/interviews/galactic-arms-race/>
- http://www.gamecareerguide.com/features/336/procedural_content_.php
- <http://www.shapegrammar.org/biblio.html>
- <http://graphics.tudelft.nl/~ruben/RMSmelik3AMIGAS09.pdf>
- <http://pcg.wikidot.com/>

