**Visual concepts**

My pursuit has been grasping visual concepts. What kind of visual analyses does our brain perform looking at things and identifying them as what they really are. Using algorithms to create shapes or patterns that fool us to see the desired objects.

Strictly in 2D.

Partly because I’m intrigued by the visual algorithms applied by our perception in identifying the elements of our familiar world.

Let me give you an example of what I mean.

Seeing a tree, even from a distance, you immediately recognize it, of course, partly because of some contextual information, but – I believe --, primarily because we all possess the means to quasi-automatically analyze and identify in a split second the visual constellation corresponding to a tree.

The challenge, then, was to construct an algorithmic image of a tree, using random parameters so that it produces different images each time it runs.

The process of designing the algorithm obviously involves a rather arbitrary series of decisions, as to the using of abstraction, simplification, generalization on one side and a fine sense for the specific details on the other.

In this case I set out to model a tree as a trunk and a collection of branches. A barren tree, so to speak, without leaves of roots.

My choice was to assume, that a trunk with offshoots branches out alongside is the single defining constituent of visual structure. With each branch and twigs showing the same structure as the trunk plus the primary branches (apart from its actual random parameters). This obviously defines a model of a recursive or fractal character. A tree basically consisting of smaller trees branching out from the trunk.

This, of course, may not be a very prices description of an actual tree, whose twigs are probably more intricately tangled than the of the trunk or of the main branches.

Yet I made the arbitrary (if not the least conscious) decision to take this level of truthfulness for my model.

So far, so good.

The task, then, was to draw a trunk, with randomly chosen points along it as the starting positions of further branches.

*# this produced the images, only this time with recursive function calls*

from PIL import Image, ImageDraw  
import numpy as np  
import random  
  
im = Image.new(**'L'**, (2000, 1600))  
  
draw = ImageDraw.Draw(im)  
w = 1800  
h = 1600  
n = 10  
q = 0.88 *# parameter 1: section contraction rate*random.seed()  
startx = w // 2  
starty = h - 1  
startlength = 80.0 *# parameter 2. starting section*startangle = 0.5 \* np.pi  
  
def branch(actualx, actualy, actuallength, actualangle):

*# draws a branch recursively* x = actualx  
 y = actualy  
 l = actuallength  
 angle = actualangle *# section angle changing in second order* angle2 = 0.0 *# amount of section angle change* while (l > 4.0):  
 *# length of minimal section – with 2.0 rather fuzzy, with 10 clean  
 # parameter 3: minimal section* u = int(x + l \* np.cos(angle))  
 v = int(y - l \* np.sin(angle))  
 l = int((q + 0.06 \* random.randint(0,2)) \* l)

*# parameter 4: range for section contraction rate.*

*# here: q = 0.88...0.94* curvature = 0.000035 \* (random.randint(0, 1000) - 501) \* np.pi  
 *# quasi curvature: change of angle change*

*# rather sensitive. actual value: = 0.0175 Pi = 3.15 degrees  
 # parameter 5: section curvature, actually 0.0175 Pi* angle2 += curvature  
 angle += angle2  
 draw.line(((x, y, u, v)), fill=255, width= int(0.4 \* l), joint=**"curved"**)  
 x = u  
 y = v  
 if (random.randint(0, 100) < 27) and (random.randint(0, 45) > np.log(l)):  
 *# parameter 6: branching probability: here 0.27 AND (0.9... 1,0))*

*# a strange condition  
 # parameter 7: branching angle, actual value = - Pi/4 ... Pi/4* branch(x, y, l, angle + .025 \* (random.randint(0, 20) - 10) \* np.pi)  
 return()  
  
branch(startx, starty, startlength, startangle)  
im.show()  
im.save(**"randagasfa1recursive1.jpg"**)

* How to **model a tree graphically**

Ó: <https://en.wikipedia.org/wiki/Recursion_(computer_science)>

* + (in 2 D, monochromatically, just the overall shape, or I should say, the “visual concept”)
* Prominent feature: its recursive **structure**
  + trunk / branches = branch / twigs = … (parent and child)
* 1st schematic representation: straight trunk & branches with the above-mentioned recursive structure.
  + the question arises:
    - where **new branches start**
      * + the number of branches determined at random from an interval of *1* to *3*
        + and then the exact point is determined randomly between the starting point of the previous child branch and the end point of the parent branch. this will result in a distribution of with a preference for higher starting points
    - how they connect and relate to “parent” by
      * **angle**
        + fixed angle
      * **size**
        + a smaller than 1 **contraction factor** defines the size next generation
        + with contraction factor *q* the approximate size of the tree will be
        + *trunk\_length \* (1 + q + q2 + q3 …) = 1 / (1 – q)*
        + which also explains the sensitivity of choosing *q.* With *q* approaching *1*

*1 / (1 – q)* tends to infinity, in other words, it may react by any proportion to a relatively small change of the *q* value.

* + - when they **end**
      * + a natural criterion: when (through repeated contraction) branch size decreases below a limit
* The whole process of modelling proceeds by **repeated adjustment** of features we are dissatisfied with
  + straight branches can be replaced with lines curving according to different rules – we’ll return to it later
  + fixed angles – the angles a new branch starts growing relative to its parent can be chosen randomly from an interval
  + fixed size proportion (contraction) – this can lead to annoyingly regular cauliflower-like shape and can again be modified by random values instead
  + the problems of rigidly fixed values (causing too regular shapes) can be always solved by using random values from a fixed interval (with uniform distribution, normally, but sometimes even this can be changed).
* branches:
  + angle
  + contraction factor
  + lines
* random parameters
  + chosen from an interval with uniformly distribution
  + with upward turning:
    - used to force effect – controlled chance
* to produce more life-like tree shapes, we draw branches with **curving lines.** we can use a number of algorithms to draw curving lines.
  + anyway, all curving lines are made up of **sections**
    - sections are defined either by *(length, inclination angle)* pairs or *vectors (u, v).* They can be converted to each other by

*(u, v) = (length \* cos(inclination), length \* sin(inclination)) pairs).*

* + inclinations define the angle between the horizontal and the actual vector. To achieve curving lines, inclinations need to vary.
  + curving with random angles in 0-, 1- or 2-order, and with an upward turning tendency.
    - in 0-order, angles vary randomly in a chosen interval
    - in 1-order angles are incremented by values chosen randomly from an interval
    - in 2-order, angles are incremented by values which are in turn incremented by values chosen randomly from an interval
    - if we wish to introduce an upward turning tendency, in either 1-order or second order, the random values are multiplied by *sign(pi / 2 – inclination)*, *pi / 2* corresponding to the direction “up”.
    - this is an example of “controlled chance” whereby random values are used to serve our goal in modifying a given shape (instead of using fixed parameters to “force” it).
    - now the same procedure works with the *(u, v)* coordinates of vectors instead of inclination angles. in 0-order, *u* and *v* are chosen randomly, whereas in order *1* or *2* their increments or the increments of their increments are so defined.
    - upward bending, though, becomes a bit more difficult in this case. It can be handled by incrementing *v* (and, perhaps, decrementing the absolute value of *u*) with a probability proportionate to *u* (or *v / u* but in this case special care should be taken to avoid zero division).
    - to create an event at a certain probability *q*, we use the expression: *if random.uniform(0, 1) < q.*
    - A picture containing text, person, black

      Description automatically generatedwidths of the given sections can also vary to produce gradually tapering branches. The easiest thing to do is to define widths proportionate to (say 0.4 times) section lengths. Of course, this method will not cause lines to become thinner by grades instead of gradually but the difference is hardly visible and would take too much effort to correct – even if it is viable.

1. idea: to draw a tree
   1. idea: a tree = a trunk and branches growing out.
   2. idea: branches also have twigs. schematically: twigs/branches = branches/trunk (parent/child)
   3. idea: recursive structure. tree = trunk + trees branching out.

parameters: starting position, angle and length

contraction. length proportion between child / parent tree

base case: (gradually decreasing) length reaches a minimal size.

* 1. idea: fixed values replaced by uniformly distributed random values from an interval
  2. idea: random seed – to be able to reproduce “good” solutions

fig. 1. tree with straight branches

* 1. solution: recursive function call, starting position, angle, length, random position, angle, length, contraction for new copies – and minimal length for terminating condition

1. solution 1.2.2: contraction provides natural ending condition: a pre-given minimal length.
2. idea 2: the necessary parameters: starting position starting length, starting angle, contraction factor, minimal length. recursion: determining the new positions, angles, lengths (relative to parent) and drawing the new branch (i.e. sub-tree)
3. solution 2: the first straight lined tree.
4. idea 3: lines should curve.
5. solution 3: branches consist of shorter sections, also determined by position, angle, length and contraction (now applied to sections instead of branches, but also resulting in the contraction of branches
6. idea 3.1: starting angles (inherited for the first sections of a new branch) incremented (from section to section) by values that in turn increase by small random values – thereby causing branches bend “smoothly”.
7. solution 3.1:
8. *angle (inherited).*
9. *angle1 += random.uniform(a, b).*
10. *angle += angle1.*
11. idea 3.2: drawing sections with widths proportionate to (say 0.4 times) their lengths.
12. idea 3.3: branches could/should also tend to bend upwards
13. solutions 3.3
14. *angle (inherited).*
15. *angle1 += sign(pi / 2 – angle) \* random.uniform(a, b).*
16. *angle += angle1.*
17. further idea 3.1: other types of curves used for branch lines.
18. solutions 3.1.1: bending in second order, using yet another angle2
19. *angle (inherited).*
20. *angle2 += random.uniform(a, b).*
21. *angle1 += angle2*
22. *angle += angle1.*
23. solutions 3.1.2: instead of *(length, angle)* pairs, using *(u, v) = (length \* cos(angle), length \* sin angle)* vectors, with their coordinates incremented between sections in the same manner.
24. further idea 4: how to draw an oak with twisting and turning thick branches?

**Random tree (new)**

1. **An algorithm to draw the graphic model of a tree** (or its visual concept) can be built using the following strategy.
2. **Trunk and branches.** A tree is made up of a trunk and a set of branches. The relationship between trunk and branches is like that of parent and child and it repeats itself for further generations. This recursive structure and is naturally programmed with a recursive algorithm. Offshoots connect to their parent branches under a certain angle and are contracted by a given factor. Both the angle and the contraction can be either fixed or vary randomly. The recursive process can end where branches shrink below a given size.
3. **Random parameters:** an algorithm using random parameters allows for some diversity, creating “similar” but actually different outcomes each it is carried. By assigning random seeds, however, identical instances can be produced.
4. **Recursive structure:** the tree then consists of a trunk with smaller copies of similarly structured trees branching out from it. The starting positions of new branches can be given by a uniform probability**.**

The first attempt along these lines created this image:

A picture containing fireworks, outdoor object

Description automatically generated**random tree with straight branches, fixed angles, fixed contraction**

This example also illustrates the way visual modelling happens in subsequent turns of raising questions and finding answers provoke further issues.

1. **Sections and inflections.** Branches are made up of a series of sections each with given inflections relative to the horizontal. Consecutive sections are shortened by a (random) contraction factor and can be drawn at a width proportionate to their length.
2. **Curving lines:** Branches might bend for greater lifelikeness. This is achieved by section inflections diverting somewhat from those of the previous ones. Figure 1. uses straight lines with constant inflections. Inflections can be controlled by random values in various ways. If the actual inflections are taken randomly (order 0) sections will connect in zig-zags. Alternatively, minute random differences can be given for subsequent section inflections (order1) or even smaller changes of these differences (order 2). Especially in the latter cases the overall image is quite sensitive to the actual values of increments, requiring careful fine-tuning.

A picture containing black

Description automatically generated

**order 0**

**order 1**

**order 2**

**upward bending**

1. **Angles.** As opposed to inflections angles are used to determine the starting directions of new branches relative to their parents. Of course, they can have definite values or chosen randomly.
2. **Random contraction:** The process of drawing is governed by a contraction factor for the consecutive sections of the branches. A fixed contraction can result in too regular “cauliflower” shapes, which can be handled with random values. The fact that the shape of the tree greatly depends on the careful choice of the value is easy to explain. The overall size has to do with the value

*h \* (1 + q + q2 + q3 +…) = h / (1 – q)*

with *h* denoting the first section (or trunk). Note that with *q* close to *1,* a small adjustment *delta* of its value can alter *1 - q* and therefore the whole fraction by a significantly larger proportion. (Say, at *q = 0.95,* the difference is 20-fold.)

1. **Random branching:** The recursive structure of the tree is created by the branches having outgrowths at irregular intervals. The new branch diverts from the old one by an angle chosen randomly with uniform distribution between -0.25pi and 0.25pi. And the positions of the new outgrowth are determined by an overall probability. I chose it to be 27%.
2. **Upward bending.** In a more sophisticated version we can multiply the second order angle difference with the sign of (*pi/2* minus the actual branch angle), which will cause each branch to eventually bend upwards towards the “sky”.

A picture containing outdoor object, fireworks

Description automatically generated

**random tree with second order curving branches and random contraction**

An algorithm to draw a 2D tree-shape.

A tree consists of a trunk and branches with twigs. (Roots not represented.) This suggests a recursive structure. Trunk : branch = branch : twigs… etc.

We need to determine the **starting positions** of branches growing out from the trunk, the **angles** with which they turn away, and the **size proportion** between them. Each can be chosen randomly from a given range.

In the simplest case I drew trunks, branches and further twigs with straight lines. fig. 1.

The trunk has a fix length.

lines

sections

contraction factor

width

inclinations

curving in (0), (1), (2) order, or with an upward turning tendency

– random parameters

uniformly distributed from an intervall

with upward turning:

used to force effects