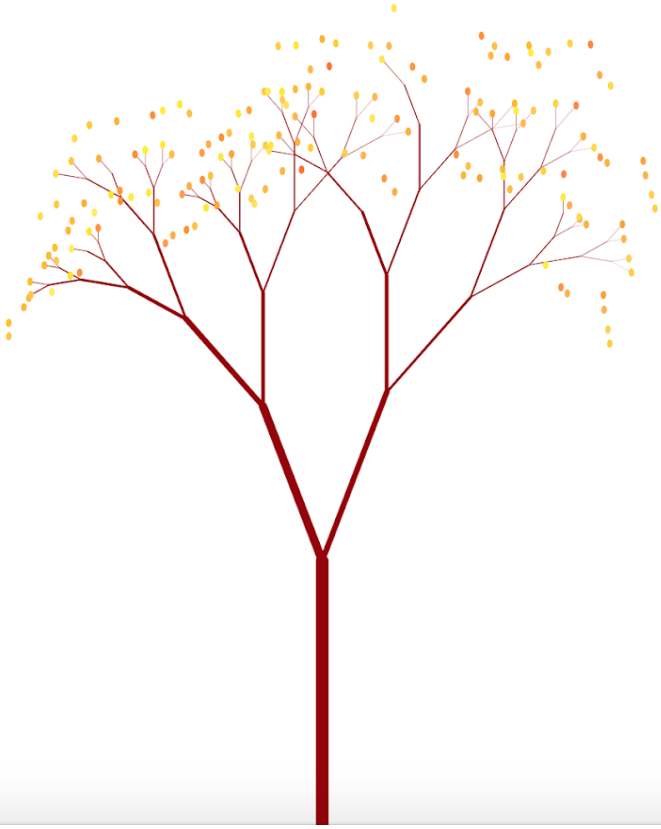
Aarhus University

Fractal tree



Thomas Thomsen, Sille Kock, Rasmus Skov & Nanna Størup
08-05-2018

The software

Our software is a generative program that simulates the growth of a tree - also called fractal trees. Whenever the program loads, a tree will grow from a random placement on the bottom x-axis.

The program is based on 4 simple rules:

1. Draw a root for the tree from a random point on the x-axis at the bottom of the canvas
2. Draw either 1 or 2 branches from the endpoint of the root
3. Continue to draw either 1 or 2 branches from each of the branches' endpoint
4. Draw 3 leaves on every endpoint of the last generation of branches

To make the branches grow both to the left and right we have created a class with two different types of branches, branch A and branch B - branch A grows to the right whereas branch B grows to the left. The same rotational value is used for both branches, but in branch A it is a positive (+) rotation and in branch B it is negative (-). What makes the branches differ is the thickness and length of the branches. For every generation of branches the thickness will be scaled down by 2/3 by returning the thickness of the previous branch. Furthermore, the lengths of the branches are shortened for every generation by multiplying the length of the previous branch with a random number between 0.5 and 0.8. 1 out of 13 times the branch will not split into two branches but only one. We chose to do this in order to make the tree look as lifelike as possible. To make it appear as if the tree is growing from the bottom, we lowered the frameRate. By doing this, it appears as if branch grows from the previous one.

The tree will stop branching after a counter hits 200 and the leaves will show. To control this, we have used conditional statements and a counter that is added +1 each time the program runs the draw function. If the user does not like the tree that has been generated, a new one will be generated when 'R' is pressed on the keyboard. When the user is happy about the tree on the canvas, it can be saved on the user's computer by pressing 'S'.

With this program we wished to explore how we could use randomness to simulate nature and make it look natural or lifelike and how the use of simple rules can generate complex outcomes.

Generativity

Our program is about generativity. We were captured by the constant change and randomness generativity could provide us with and wanted to make a structured program by using classes and objects. We believe our generative tree was a perfect example of this and we could implement different features to the program.

As mentioned, our program consists of a tree that gets randomly generated for every press of the "R" key. This means that the tree will change for every reload there is made. Winnie Soon states: *"A generative program may consist of many combinations of different rules and conditions which emerge autonomously."* (Soon 2016, 87)**.** Rules are a crucial part of making a generative artwork. This was why we decided to make a set of rules for the program; to create structure and make it understandable. These rules created a base and starting point for our program. By making small but constant changes to the outcome of our program, we were able to create a new piece of generative art for every reload.

When using randomness in code, you cannot predict the visual outcome of the program. Generative art is not always perfect, but it adds an unexpectedness to the programs understanding which in many ways has a positive impact on the outcome of the program. Every changing aspect also creates a never-ending element to our program. It does not have a defined ending, which means it can go on for eternity. The blend of eternity and randomness in our program is in our opinion a great visualization of generative art.

We knew we wanted to give the user the opportunity to save their favorite random generated tree; we added a "save" function. By pressing the "S" key, the user will automatically get a saved image of the desired tree on their computer. This feature raises the discussion about authorship, which we will elaborate more on later in the readme.

Aesthetics of Code

LOOP

The for-loop can be seen as the cannon fodder of our program, the hard worker, the busy ant. Without the for-loop there would be no trees. In code, the simple for-loop conjures the being "tree" into living. Before this operation, the tree only exists as mere skeletal beings, as boxes without content. Take for example these simple lines:

```
for (var i = 0; i < tree.length; i++) {
                tree[i].display();
                count++;
                }
```

This line loops through all branches in the array tree[i] and displays them. Without the for-loop, the branches and the trees would exist only as a figment of a thought - something there, but not quite there. The for-loop command the tree into being and represents a crucial aspect of our code.

But the for-loop does not only command trees into being. The for-loop is responsible for constructing the tree - not just displaying the tree. Take these lines:

```
 if (count<200){
                for (var i = tree.length - 1; i >= 0; i--) {
                                if (!tree[i].finished) {
                                tree.push(tree[i].branchB());
                                if (random(12)>1){
                                tree.push(tree[i].branchA());
                                }
                                }
                tree[i].finished = true;
                }
}
```

The for-loop is attaches the branches to the tree, constructing and building the tree from the ground up as the process goes along.

The for-loop, and loops in general, is a way thinking, of dealing with certain problems that requires multiple iterations, of something that needs to be defined upon a past existence of itself. As Wilfred Hou Je Bek has put it: *"[...] the sheer light- footedness of looping allows you to run away with the problem with more ease."* (Je Bek 2008, 180).

Random as a chance: if (random(12)>1)

This is a chance calculation. If this is true, the tree splits in two when adding the next generation of branches. This means, that there is a 1/13 chance that the program doesn't split the next generation of branches in two. A tree does not grow symmetrically which is why we added this chance element – and we consider it to have a huge impact on the aesthetics of our fractal trees (and the program as a whole) just like Nick Montfort argues that "10 Print" would not have achieved the aesthetic end result without its random function: *"First, using randomness is aesthetically necessary in this program; there is no other way to achieve a similar effect."* (Montfort 2012, 142). Nick Montfort further describes the random chance element as a natural aspect of life, adding to the execution of liveness and the simulative qualities of generativity: *"Life itself is full of randomness and the inexplicable, and it is no small wonder that children and adults alike consciously incorporate chance into their daily lives, as if to tame it."* (Montfort 2012, 121)

saveCanvas( ); & window.location.reload

While small parts of the code, they have a big conceptual impact on the program. The dilemma of authorship and infinite materiality in generativity and generative art can be observed through this function. The program runs with little to no influence from the user, yet the user has the power to tell the program when they are satisfied with the outcome (saveCanvas)or if they are unsatisfied with the outcome (window.location.reload). So while the code does not allow the program to generate infinitely, the programs runs pseudo-infinitely as it is not done generating trees before the user decides so.

If the user is in control of the use of the program, and the program only runs from a set of computational guidelines – paired with the fact that the judgement of the outcome is put into the hands of the user. Is it not the user and his/her creativity that have made these trees? As Galanter states:

> *"Related issues emerge with the question whether computers can be truly creative agents, and generative art requires more subtle variations to that question. In the case of Latham or Sims, for example, the artist and audience can make choices, but only among alternatives created by the computer. To what extent does such selection confer authorship?"* (Galanter, Generative Art Theory 2016, 167)

Random intervals: Rules and Freedom

While the branches are constructed through objects and classes, and they are added to arrays through loops, it is the random function through which the program generates the visual representation of the trees. These random calculations are made with random intervals and are

both the rules the program follow, as well as the range wherein the program runs freely. Roman Frigg argues randomness is unpredictable in the context of dynamical systems and explains that: *"[...] an event is random if there is no way to predict its occurrence with certainty. Likewise, a random process is one for which we are not able to predict what happens next"* (Frigg 2004, 430)**.** This freedom inside the rules set by the code, are what constitutes generativity in art. The autonomy of the program comes from these randomized intervals: as we set up these intervals as guidelines, we – as programmers – cede the control and let the program finish the job. Soon has described generative art as: *"Generative art refers to any art practice in which the artist cedes control to a system with functional autonomy that contributes to, or results in, a completed work of art."* (Galanter, What is Complexism? Generative Art and the Culture of Science and the Humantities 2008, 154)**.**

The thickness of the branches is an example of a fixed rule set used in the program. Every time a new generation of branches are added, the strokeWeight is multiplied with 0.67. The program has no interval to operate – a piece of the aesthetic that we remain in control over. In contrast, the length of the vector as well as the amount of rotation it projects is multiplied by a random number between 0.5 and 0.8. In other words, the program 'decides' itself how far the tree branches out each generation and 'decides' how short the next generation should be in relation to the previous generation (between 20 and 50 %).

Critical aspects

The notion of authorship

The critical aspect to our software is the notion of authorship. We want to question whether it is the program itself, the programmer or the user of the program who the artist behind the artwork is which the program generates. Due to the 'save canvas' function we have built into our program, the user can save a somewhat unique piece of art made in our program. This further raises the question about authorship in the program. The user decides when the program, and thereby the artwork, ends. He or she sets the boundaries for the result, stopping the program from generating new trees. By doing this the user appears to claim authorship of the the artwork. However, the program works autonomously, and the user can only affect the internal processes, the who one that truly creates something, by reloading it. In that sense, one could argue that the program itself is the author or artist. Perhaps will be better to talk of ownership - the one who owns the final picture of a tree. One example of the difference between ownership and authorship can be seen in the same way one can be commissioned to write a book by another person. Here the one who writes the book take on the role of author, where as the one paying for can claim ownership. Another aspect to consider is that we - as the programmers - has defined the rules for the program. Does that make us the artists? Although, we would never be able to comprehend all the different scenarios or even visualize completely how the program would look like before running it. The randomness makes this impossible; there are too many possible outcomes.

> *"Generative artworks "must be well defined and self-contained enough to operate autonomously" (Galanter, 2003, n.p). The use of the term 'self' does not point at any programmer/artist but to the core of the self-organising processes of a machine"* (Soon 2016, 85)

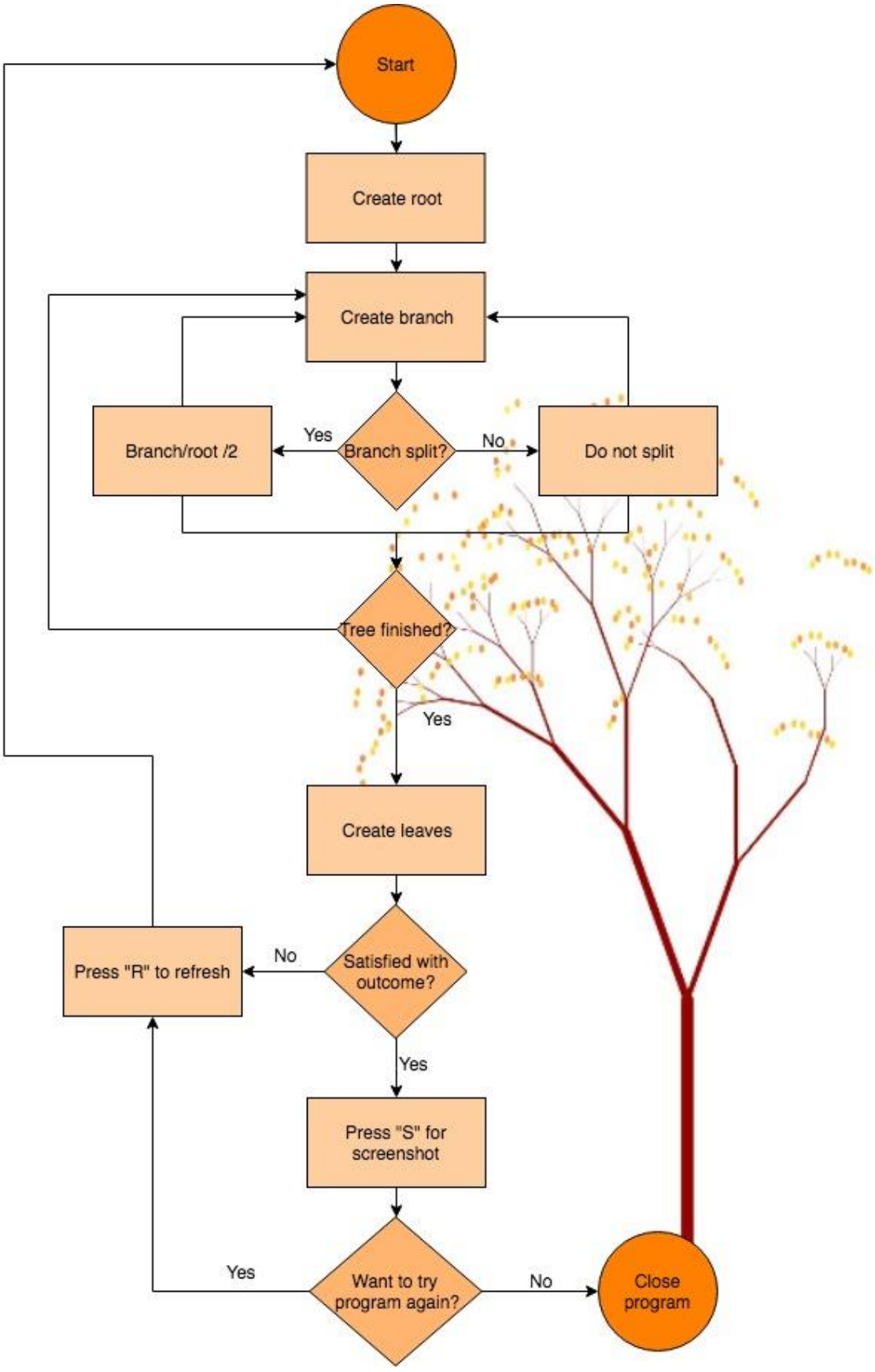
The program itself performs the artwork and the user is left with not many possible actions. When entering the page, the program does not require any user interactivity - it runs fully autonomous. This creates a bigger discussion of the true authorship and ownership over the program. If the program runs autonomously, how can there ever be another owner or author than the computer/program itself? A counter argument could state; the program cannot run without the press of the "R" key. Our program is currently setup to need a physical press of the key "R" to execute the program. Therefore, does our program only run autonomously the first time opening the program. *"Generative art refers to any art practice in which the artist cedes control to a system with functional autonomy that contributes to, or results in, a completed work of art."* (Galanter, What is Complexism? Generative Art and the Culture of Science and the Humantities 2008, 154). Following this line of thought, the artist gives away the control and lets the program run autonomously. In that sense, the program has control over the outcome, and the user just runs it.

Process or end result as art

There is a tendency when looking at art, at least for the everyday layman, to focus more on the end result - the dried-up paint, the neatly shaped figure, the finished video and so on.
Regarding the problem of locality, code and malleability in generativity, Galanter discusses the ontological status of generative art. "*For some people, generative art is like all other art, and, to the extent there is an object or event, the latter determines where the art resides*" (Galanter, Generative Art Theory 2016, 171)**.** However, this focus, a tunnel vision almost, ignores another critical aspect of art; the process of making what we consider art. It is not the trees themselves that are interesting in our program. They are beautiful and interesting to look at, but that is not the point. The point is the process - how the program generates trees, how the different parts are intertwined, how they affect each other and how a few changes can have a profound impact on how the tree grows.

In our artwork the viewer should not concern themselves with how beautiful or ugly the trees are. One should look at the code, the process, the system itself. Our artwork challenges to look at these things. The real beauty lies in there.

Flowchart


Start
Create root
Create branch
Branch/root /2
Yes
Branch split?
No
Do not split
Tree finished?
Yes
Create leaves
Press "R" to refresh
No
Satisfied with outcome?
Yes
Press "S" for screenshot
Yes
Want to try program again?
No
Close program

Links

Run Me:
https://rawgit.com/true-rasmus/AP2018/master/Group6_FractalTree/empty-example/index.html

Video of program:
https://youtu.be/Yx9ymZwv0Lo

Bibliography

Frigg, Roman. *In What Sense is the Kolmogorov-Sinai Entropy a Measure for Chaotic Behaviour?—Bridging the Gap Between Dynamical Systems Theory and Communication Theory'.* Oxford: British Journal for the Philosophy of Science, 2004.

Galanter, Philip. »Generative Art Theory.« I *A Companion to Digital Art*, edited by Christiane Paul, 146-180. Hoboken, NJ: John Wiley & Sons, Inc., 2016.

—. »What is Complexism? Generative Art and the Culture of Science and the Humantities.« *GA2008, 11th Generative Art Conference.* Milan: Generative Design Lab, 2008.

Je Bek, Wilfred Hou . »LOOP.« In *Software Studies*, af Matthew Fuller, 179-183. Cambridge, MA: The MIT Press, 2008.

Montfort, Nick. »Randomness.« I *10 PRINT CHR$(205.5+RND(1)); : GOTO 10*, by Nick Montfort, et al., 119-146. Cambridge, MA: MIT Press, 2012.

Soon, Winnie. *Executing Liveness.* Aarhus: Aarhus University, 2016.