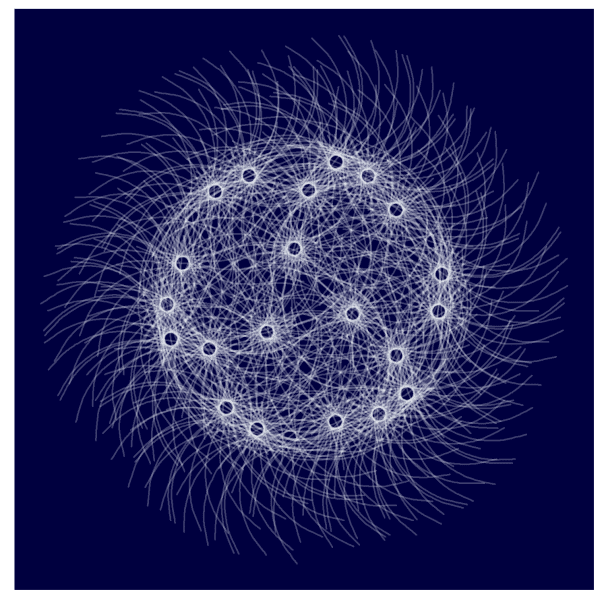
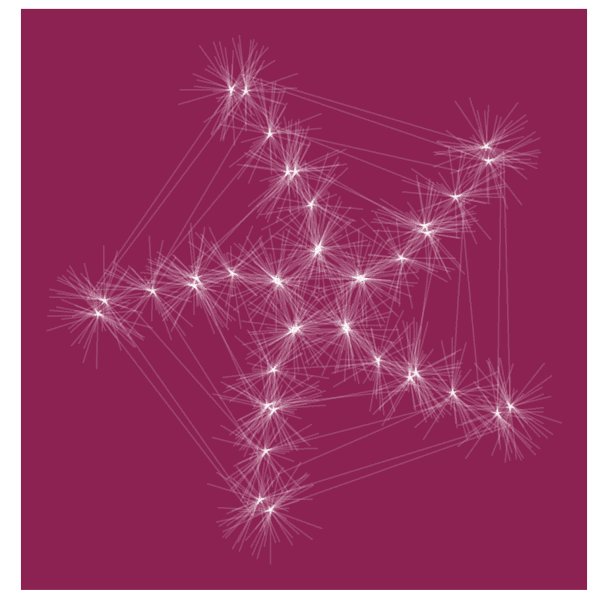
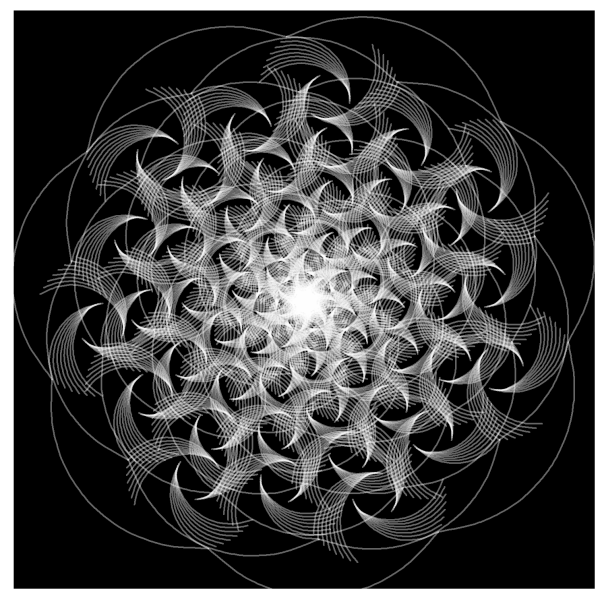
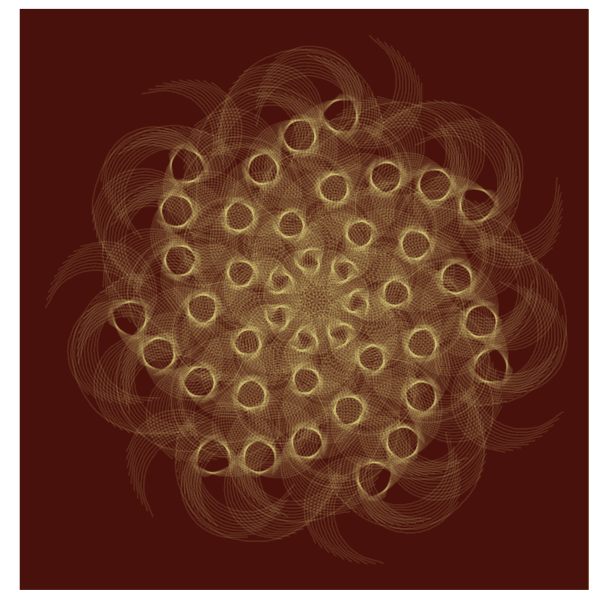
La luna es un pozo chico  
las flores no valen nada  
lo que valen son tus brazos  
cuando de noche me abrazan  
(Zorongo Gitano, Carmen Linares)

When I publish a post showing my drawings, I use to place some outputs, give some highlights about the techniques involved as well as a link to the R code that I write to generate them. That’s my typical *generative-art* post ([here](https://fronkonstin.com/2019/01/10/rcpp-camaron-de-la-isla-and-the-beauty-of-maths/) you have an example of it). I think that my audience knows to program in R and is curious enough to run and modify the code by themselves to generate their own outputs. Today I will try to be more *educational* and will explain step by step how you can obtain drawings like these:

* 
* 
* 
* 

There are two reasons for this decision:

* It can illustrate quite well my *mental journey* from a simple idea to what I think is a interesting enough experiment to publish.
* I think that this experiment is a good example of the use of accumulate, a very useful function from the *life-changing* purrr package.

Here we go: there are many ways of drawing a pentagon in R. Following you will find a piece of code that does it using accumulate function from purrr package. I will use only two libraries for this experiment: ggplot2 and purrr so I will just load in the tidyverse (both libraries take part of it):

library(tidyverse)

pentagon <- tibble(

x = accumulate(1:4, ~.x+cos(.y\*2\*pi/5), .init = 0),

y = accumulate(1:4, ~.x+sin(.y\*2\*pi/5), .init = 0),

xend = accumulate(2:5, ~.x+cos(.y\*2\*pi/5), .init = cos(2\*pi/5)),

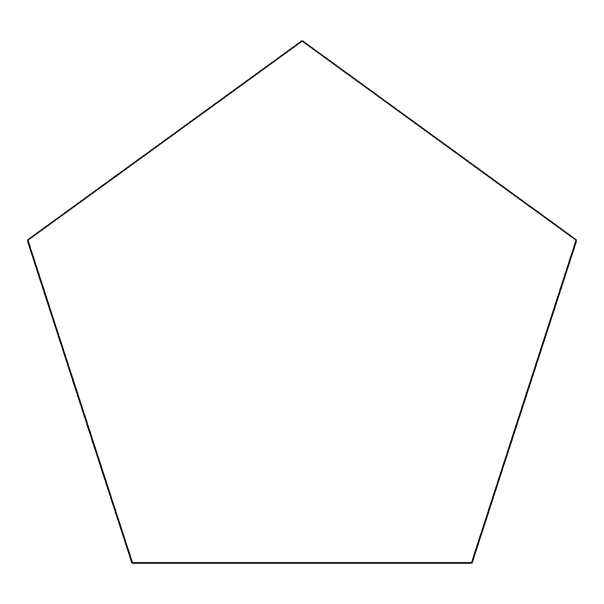
yend = accumulate(2:5, ~.x+sin(.y\*2\*pi/5), .init = sin(2\*pi/5)))

ggplot(pentagon)+

geom\_segment(aes(x=x, y=y, xend=xend, yend=yend))+

coord\_equal()+

theme\_void()



The function accumulate applies sequentially some function a number of times storing all the intermediate results. When I say sequentially I mean that the input of any step is the output of the prevoius one. The accumulate function uses internally two important arguments called .x and .y: my own way to understand its significance is that .x is the previous value of the output vector and .y is the previous value of the one which controls the iteration. Let’s see a example: imagine that I want to create a vector with the first 10 natural numbers. This is an option:

> accumulate(1:10, ~.y)

[1] 1 2 3 4 5 6 7 8 9 10

The vector which controls the iteration in this case is 1:10 and .y are the values of it so I just have to define a function wich returns that values and this is as simple as ~.y. The first iteration takes the first element of that vector. This is another way to do it:

> accumulate(1:10, ~.x+1)

[1] 1 2 3 4 5 6 7 8 9 10

To replicate the same output with .x I have to change a bit the function to ~.x+1 because if not, it will always return 1. Remember that .x is the previous output of the function and it is initialized with 1 (the first value of the vector 1:10). Intead of initializing .x with the first value of the vector of the first argument of accumulate, you can define exactly its first value using .init:

accumulate(2:10, ~.y, .init = 1)

accumulate(1:9, ~.x+1, .init = 1)

Note that using .init I have to change the vector to reproduce the same output as before. I hope now you will understand how I generated the initial and ending points of the previous pentagon. Some points to help you if not:

* I generate a tibble with 5 rows, each of one defines a different segment of the pentagon
* First segments starts at (0,0)
* The rotating angle is equal to 2\*pi/5
* The ending point of each segment becomes the starting point of the following one

The next step is to *encapsulate* this into a function to draw regular polygons with any given number of edges. I only have to generalize the number of steps and the rotating angle of accumulate:

polygon <- function(n) {

tibble(

x = accumulate(1:(n-1), ~.x+cos(.y\*2\*pi/n), .init = 0),

y = accumulate(1:(n-1), ~.x+sin(.y\*2\*pi/n), .init = 0),

xend = accumulate(2:n, ~.x+cos(.y\*2\*pi/n), .init = cos(2\*pi/n)),

yend = accumulate(2:n, ~.x+sin(.y\*2\*pi/n), .init = sin(2\*pi/n)))

}

ggplot(polygon(6))+

geom\_segment(aes(x=x, y=y, xend=xend, yend=yend))+

coord\_equal()+

theme\_void()

ggplot(polygon(7))+

geom\_segment(aes(x=x, y=y, xend=xend, yend=yend))+

coord\_equal()+

theme\_void()

ggplot(polygon(8))+

geom\_segment(aes(x=x, y=y, xend=xend, yend=yend))+

coord\_equal()+

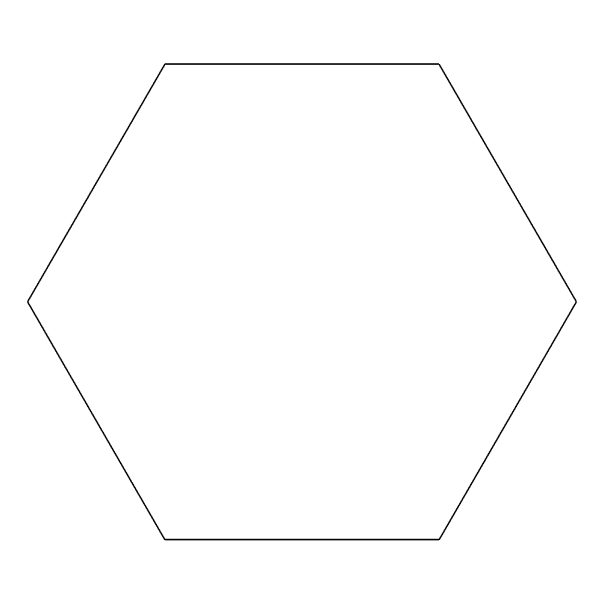
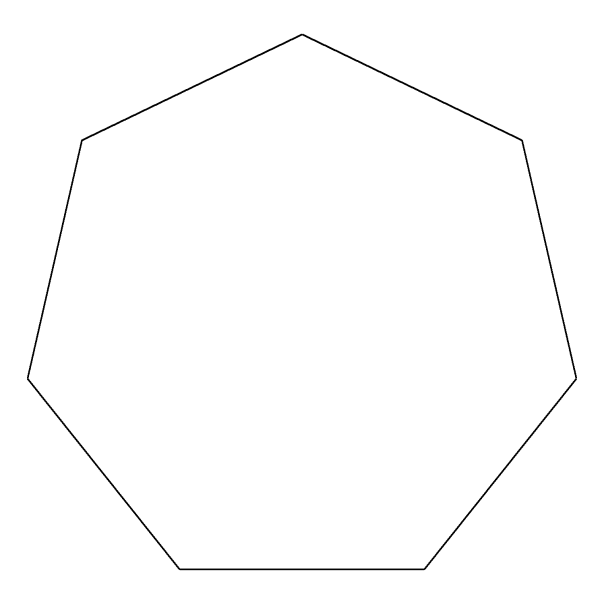
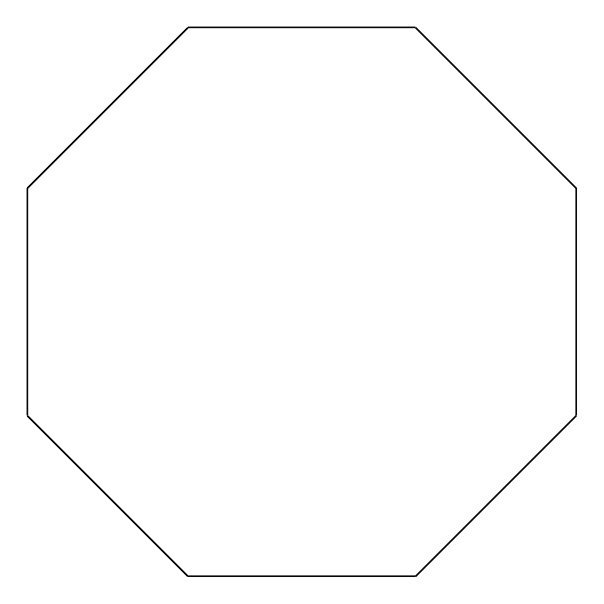
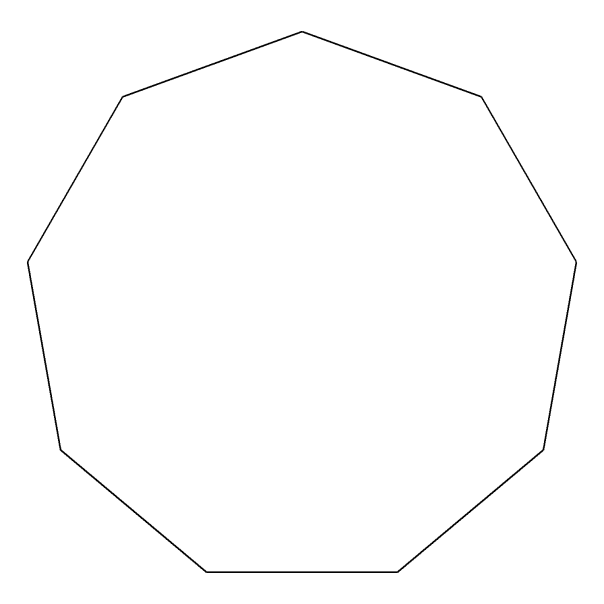
theme\_void()

ggplot(polygon(9))+

geom\_segment(aes(x=x, y=y, xend=xend, yend=yend))+

coord\_equal()+

theme\_void()

* 
* 
* 
* 

Now, let’s place another segment in the middle of each edge, perpendicular to it towards its centre. To do it I *mutate* de data frame to add those segments using *simple* trigonometry: I just have to add pi/2 to the angle wich forms the edge, obtained with atan2 function:

polygon(5) -> df1

df1 %>% mutate(angle = atan2(yend-y, xend-x)+pi/2,

x = 0.5\*x+0.5\*xend,

y = 0.5\*y+0.5\*yend,

xend = x+0.2\*cos(angle),

yend = y+0.2\*sin(angle)) %>%

select(x, y, xend, yend) -> df2

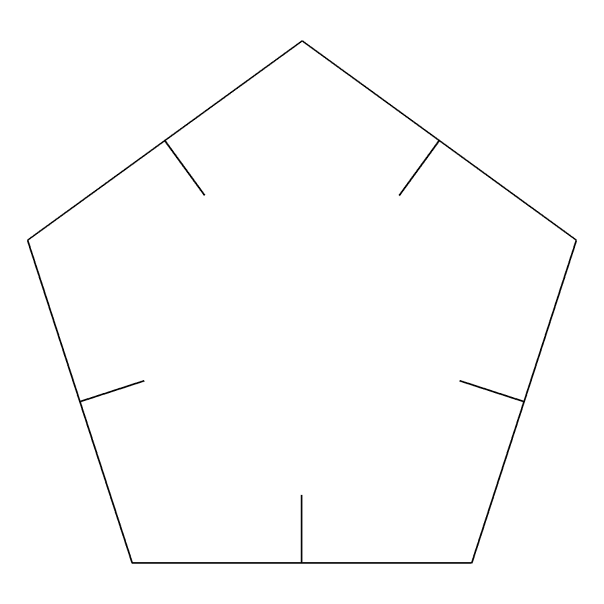
df1 %>% bind\_rows(df2) -> df

ggplot(df)+

geom\_segment(aes(x=x, y=y, xend=xend, yend=yend))+

coord\_equal()+

theme\_void()



These new segments have longitude equal to 0.2, smaller than the original edges of the pentagon. Now, let’s connect the ending points of these perpendicular segments. It is easy using mutate and first functions. Another smaller pentagon appears:

polygon(5) -> df1

df1 %>% mutate(angle = atan2(yend-y, xend-x)+pi/2,

x = 0.5\*x+0.5\*xend,

y = 0.5\*y+0.5\*yend,

xend = x+0.2\*cos(angle),

yend = y+0.2\*sin(angle)) %>%

select(x, y, xend, yend) -> df2

df2 %>% mutate(

x=xend,

y=yend,

xend=lead(x, default=first(x)),

yend=lead(y, default=first(y))) %>%

select(x, y, xend, yend) -> df3

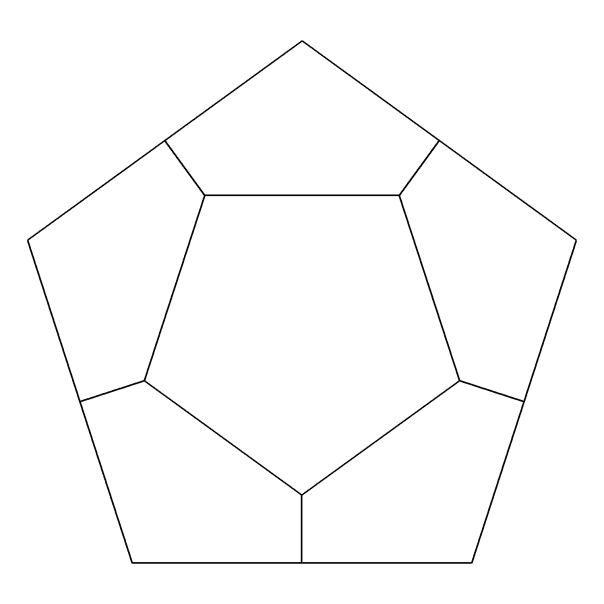
df1 %>% bind\_rows(df2) %>% bind\_rows(df3) -> df

ggplot(df)+

geom\_segment(aes(x=x, y=y, xend=xend, yend=yend))+

coord\_equal()+

theme\_void()



Since we are repeating these steps many times, I will write two functions: one to generate perpendicular segments to the edges called mid\_points and another one to connect its ending points called con\_points. The next code creates both funtions and uses them to add another level to our previous drawing:

mid\_points <- function(d) {

d %>% mutate(

angle=atan2(yend-y, xend-x) + pi/2,

x=0.5\*x+0.5\*xend,

y=0.5\*y+0.5\*yend,

xend=x+0.2\*cos(angle),

yend=y+0.2\*sin(angle)) %>%

select(x, y, xend, yend)

}

con\_points <- function(d) {

d %>% mutate(

x=xend,

y=yend,

xend=lead(x, default=first(x)),

yend=lead(y, default=first(y))) %>%

select(x, y, xend, yend)

}

polygon(5) -> df1

df2 <- mid\_points(df1)

df3 <- con\_points(df2)

df4 <- mid\_points(df3)

df5 <- con\_points(df4)

df1 %>%

bind\_rows(df2) %>%

bind\_rows(df3) %>%

bind\_rows(df4) %>%

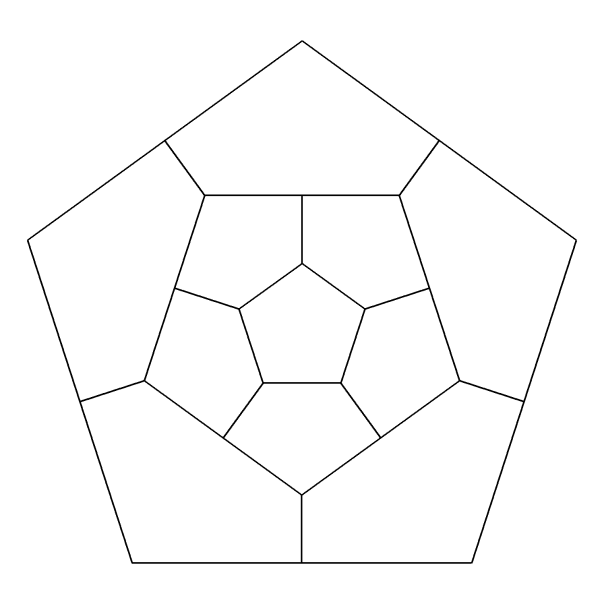
bind\_rows(df5) -> df

ggplot(df)+

geom\_segment(aes(x=x, y=y, xend=xend, yend=yend))+

coord\_equal()+

theme\_void()



This pattern is called Sutcliffe pentagon. In the previous step, I did iterations *manually*. The function accumulate can help us to do it automatically. This code reproduces exactly the previous plot:

edges <- 5

niter <- 4

polygon(edges) -> df1

accumulate(.f = function(old, y) {

if (y%%2!=0) mid\_points(old) else con\_points(old)

},

1:niter,

.init=df1) %>%

bind\_rows() -> df

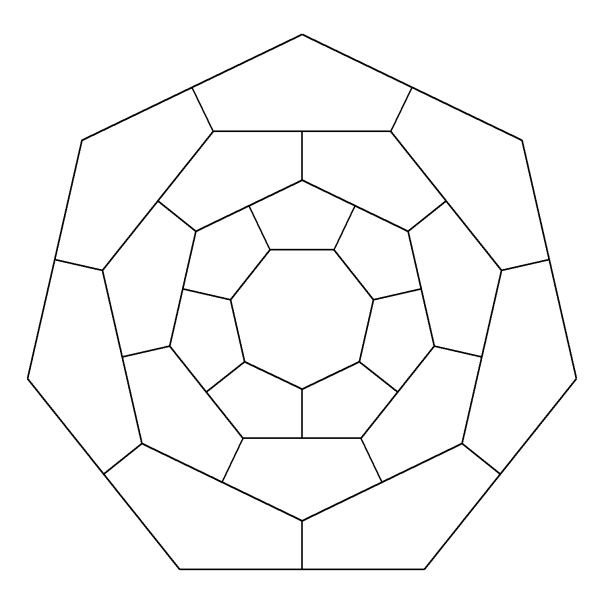
ggplot(df)+

geom\_segment(aes(x=x, y=y, xend=xend, yend=yend))+

coord\_equal()+

theme\_void()

Substituting edges by 7 and niter by 6 as well in the first two rows of the previous code, generates a different pattern, in this case heptagonal:



Let’s start to play with the parameters to change the appearance of the drawings. What if we do not start the perpendicular segments from the midpoints of the edges? It’s easy: we just need to add a parameter that will name p to the function mid\_points (p=0.5 means starting from the middle). This is our heptagon pattern when p is equal to 0.3:

mid\_points <- function(d, p) {

d %>% mutate(

angle=atan2(yend-y, xend-x) + pi/2,

x=p\*x+(1-p)\*xend,

y=p\*y+(1-p)\*yend,

xend=x+0.2\*cos(angle),

yend=y+0.2\*sin(angle)) %>%

select(x, y, xend, yend)

}

edges <- 7

niter <- 6

polygon(edges) -> df1

accumulate(.f = function(old, y) {

if (y%%2==0) mid\_points(old, 0.3) else con\_points(old)

},

1:niter,

.init=df1) %>%

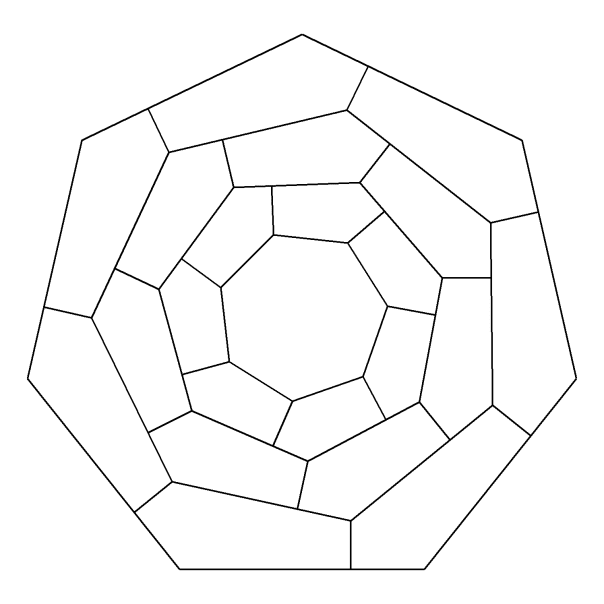
bind\_rows() -> df

ggplot(df)+

geom\_segment(aes(x=x, y=y, xend=xend, yend=yend))+

coord\_equal()+

theme\_void()



Another simple modification is to allow any angle between edges and next iteration segments (perpendicular until now ) so let’s add another parameter, called a, to themid\_points function:

mid\_points <- function(d, p, a) {

d %>% mutate(

angle=atan2(yend-y, xend-x) + a,

x=p\*x+(1-p)\*xend,

y=p\*y+(1-p)\*yend,

xend=x+0.2\*cos(angle),

yend=y+0.2\*sin(angle)) %>%

select(x, y, xend, yend)

}

edges <- 7

niter <- 18

polygon(edges) -> df1

accumulate(.f = function(old, y) {

if (y%%2!=0) mid\_points(old, 0.3, pi/5) else con\_points(old)

},

1:niter,

.init=df1) %>%

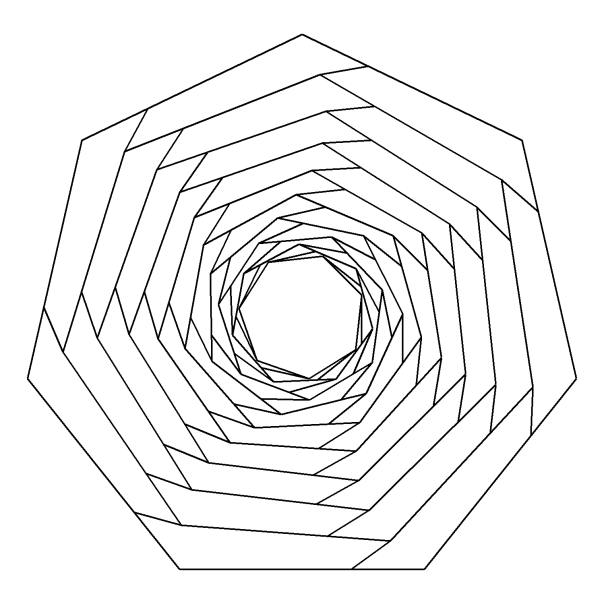
bind\_rows() -> df

ggplot(df)+

geom\_segment(aes(x=x, y=y, xend=xend, yend=yend))+

coord\_equal()+

theme\_void()



That’s nice! It looks like a shutter. Now it’s time to change the longitude of the segments starting from the edges (those perpendicular in our first drawings). Now all them measure 0.2. I will take advantage of the parameter y of accumulate and apply a user defined function to modify that longitude each iteration. This example uses the identity function (FUN = function(x) x) to increase longitude step by step:

mid\_points <- function(d, p, a, i, FUN = function(x) x) {

d %>% mutate(

angle=atan2(yend-y, xend-x) + a,

radius=FUN(i),

x=p\*x+(1-p)\*xend,

y=p\*y+(1-p)\*yend,

xend=x+radius\*cos(angle),

yend=y+radius\*sin(angle)) %>%

select(x, y, xend, yend)

}

edges <- 7

niter <- 18

polygon(edges) -> df1

accumulate(.f = function(old, y) {

if (y%%2!=0) mid\_points(old, 0.3, pi/5, y) else con\_points(old)

},

1:niter,

.init=df1) %>%

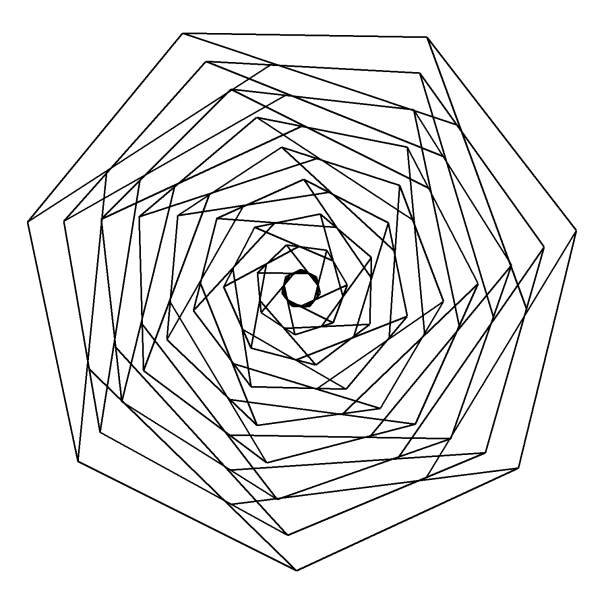
bind\_rows() -> df

ggplot(df)+

geom\_segment(aes(x=x, y=y, xend=xend, yend=yend))+

coord\_equal()+

theme\_void()



What if we increase niter from 18 to 250?

edges <- 7

niter <- 250

step <- 2

polygon(edges) -> df1

accumulate(.f = function(old, y) {

if (y%%step!=0) mid\_points(old, 0.3, pi/5, y) else con\_points(old)

},

1:niter,

.init=df1) %>%

bind\_rows() -> df

ggplot(df)+

geom\_curve(aes(x=x, y=y, xend=xend, yend=yend),

curvature = 0,

color="black",

alpha=0.1)+

coord\_equal()+

theme(legend.position = "none",

panel.background = element\_rect(fill="white"),

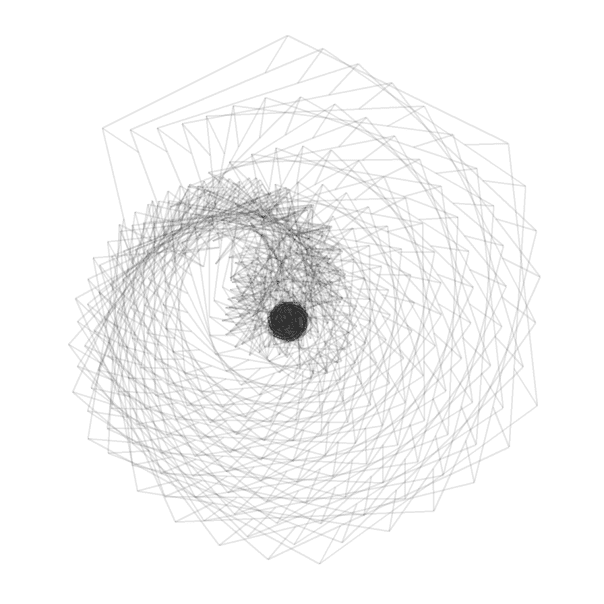
plot.background = element\_rect(fill="white"),

axis.ticks = element\_blank(),

panel.grid = element\_blank(),

axis.title = element\_blank(),

axis.text = element\_blank())



Not bad, but we can do it better. First of all, note that appart of adding transparency with the parameter alpha inside the ggplot function, I changed the geometry of the plot from geom\_segment to geom\_curve. Setting curvature = 0 as I did generates straight lines so the result is the same as geom\_segment but it will give us an additional *degree of freedom* to do our plots. I also changed the theme\_void by an explicit customization some of the elements of the plot. Concretely, I want to be able to change the background color. This is the definitive code explained:

library(tidyverse)

# This function creates the segments of the original polygon

polygon <- function(n) {

tibble(

x = accumulate(1:(n-1), ~.x+cos(.y\*2\*pi/n), .init = 0),

y = accumulate(1:(n-1), ~.x+sin(.y\*2\*pi/n), .init = 0),

xend = accumulate(2:n, ~.x+cos(.y\*2\*pi/n), .init = cos(2\*pi/n)),

yend = accumulate(2:n, ~.x+sin(.y\*2\*pi/n), .init = sin(2\*pi/n)))

}

# This function creates segments from some mid-point of the edges

mid\_points <- function(d, p, a, i, FUN = ratio\_f) {

d %>% mutate(

angle=atan2(yend-y, xend-x) + a,

radius=FUN(i),

x=p\*x+(1-p)\*xend,

y=p\*y+(1-p)\*yend,

xend=x+radius\*cos(angle),

yend=y+radius\*sin(angle)) %>%

select(x, y, xend, yend)

}

# This function connect the ending points of mid-segments

con\_points <- function(d) {

d %>% mutate(

x=xend,

y=yend,

xend=lead(x, default=first(x)),

yend=lead(y, default=first(y))) %>%

select(x, y, xend, yend)

}

edges <- 3 # Number of edges of the original polygon

niter <- 250 # Number of iterations

pond <- 0.24 # Weight to calculate the point on the middle of each edge

step <- 13 # No of times to draw mid-segments before connect ending points

alph <- 0.25 # transparency of curves in geom\_curve

angle <- 0.6 # angle of mid-segment with the edge

curv <- 0.1 # Curvature of curves

line\_color <- "black" # Color of curves in geom\_curve

back\_color <- "white" # Background of the ggplot

ratio\_f <- function(x) {sin(x)} # To calculate the longitude of mid-segments

# Generation on the fly of the dataset

accumulate(.f = function(old, y) {

if (y%%step!=0) mid\_points(old, pond, angle, y) else con\_points(old)

}, 1:niter,

.init=polygon(edges)) %>% bind\_rows() -> df

# Plot

ggplot(df)+

geom\_curve(aes(x=x, y=y, xend=xend, yend=yend),

curvature = curv,

color=line\_color,

alpha=alph)+

coord\_equal()+

theme(legend.position = "none",

panel.background = element\_rect(fill=back\_color),

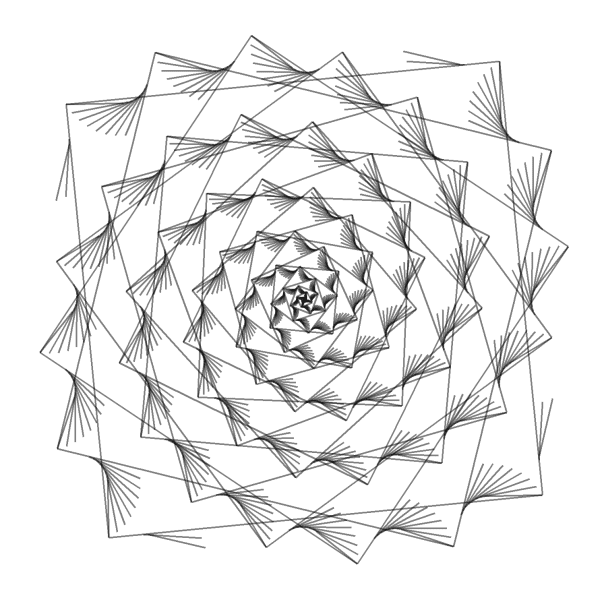
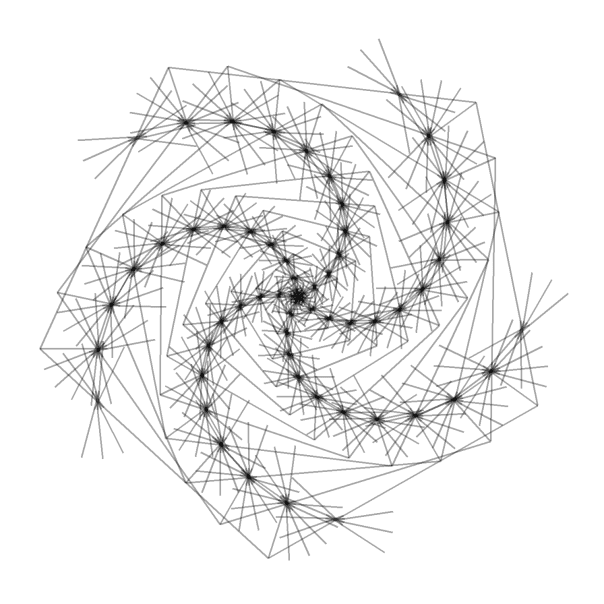
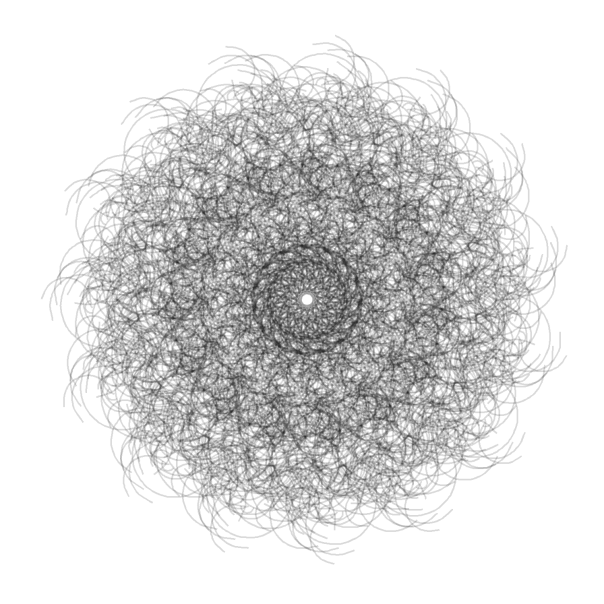
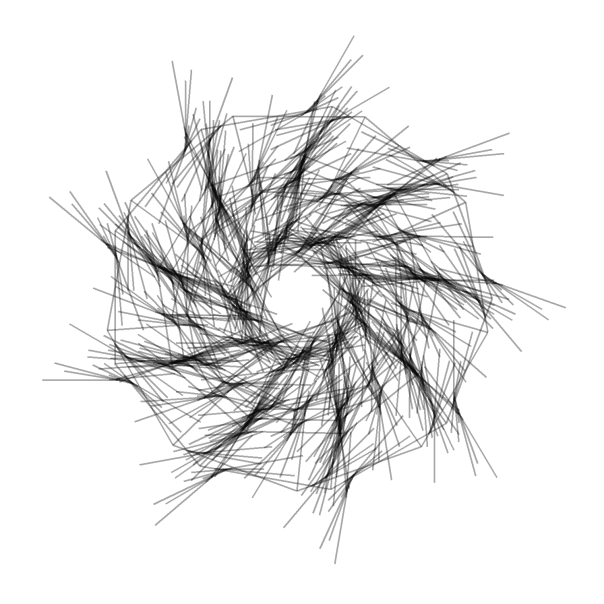
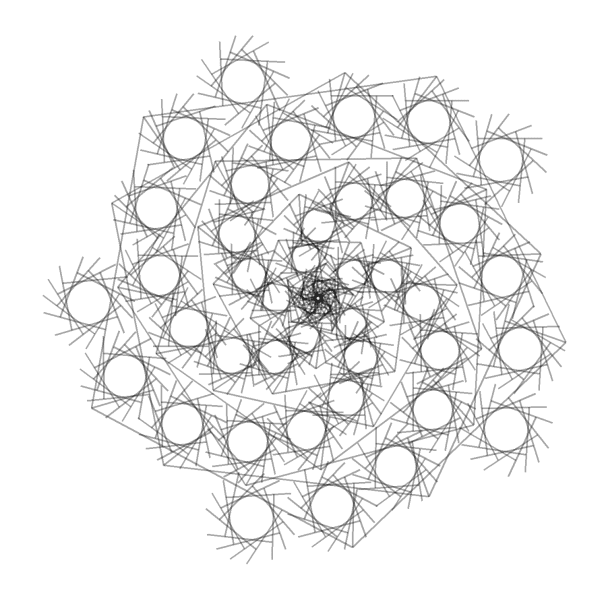
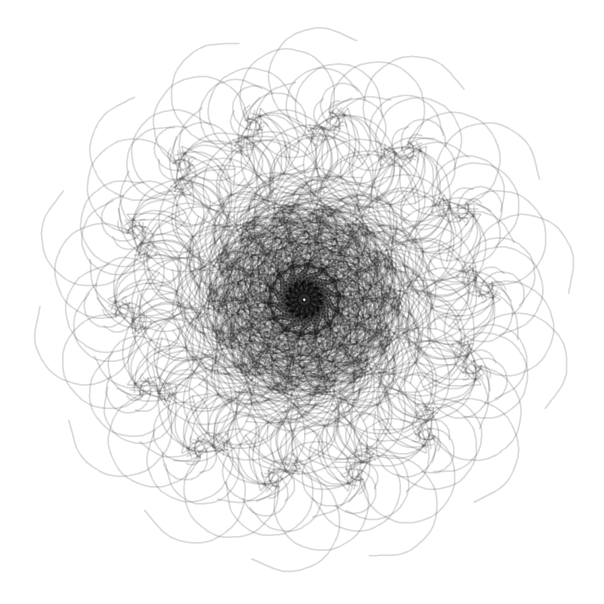
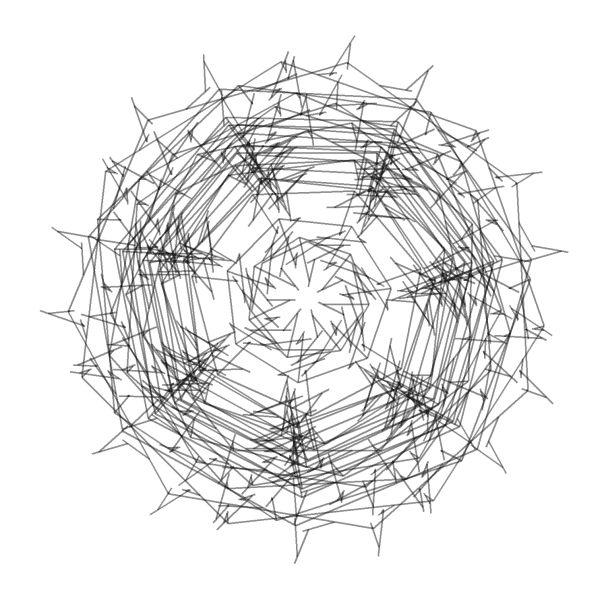
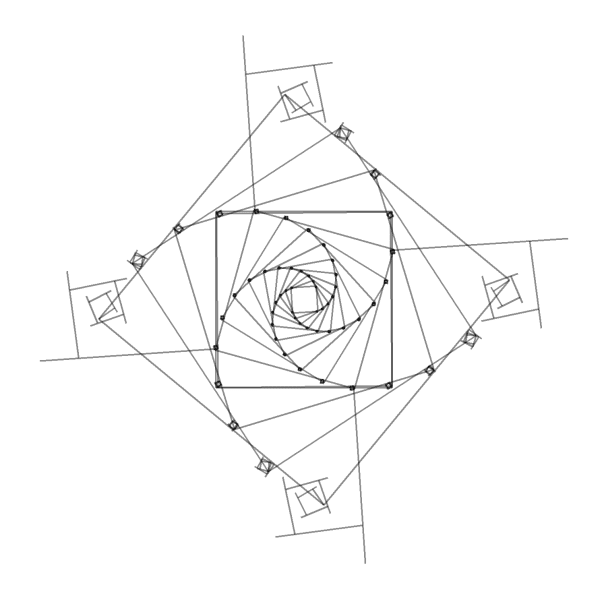
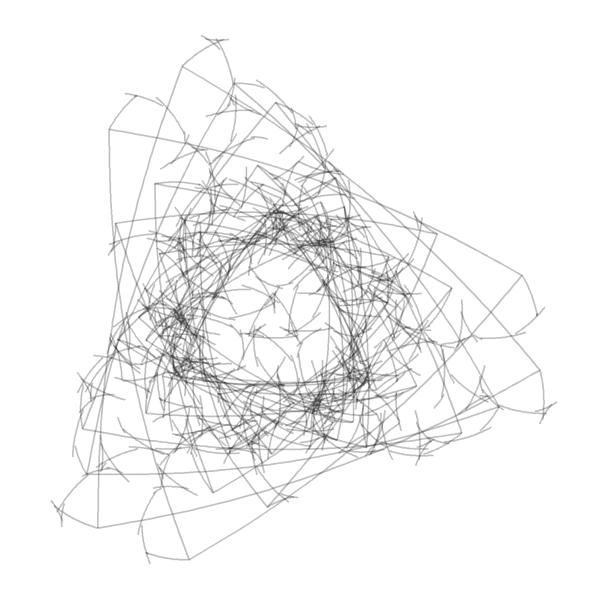
plot.background = element\_rect(fill=back\_color),

axis.ticks = element\_blank(),

panel.grid = element\_blank(),

axis.title = element\_blank(),

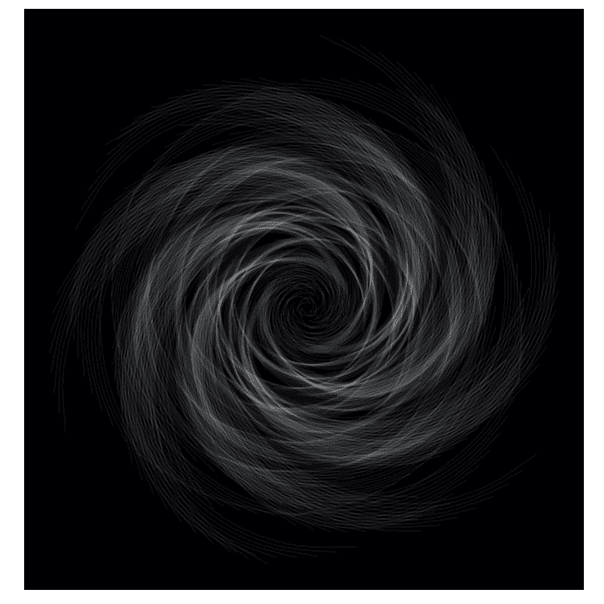
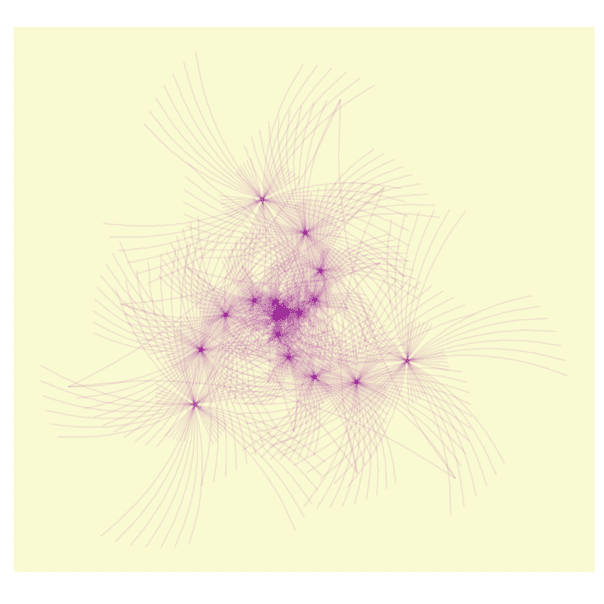
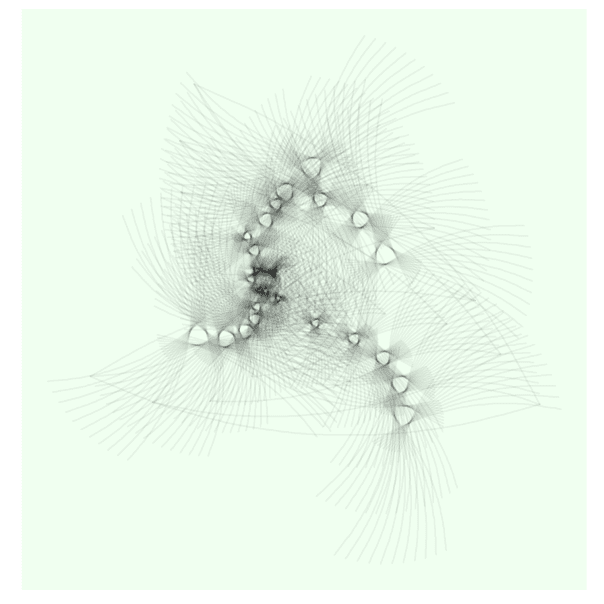
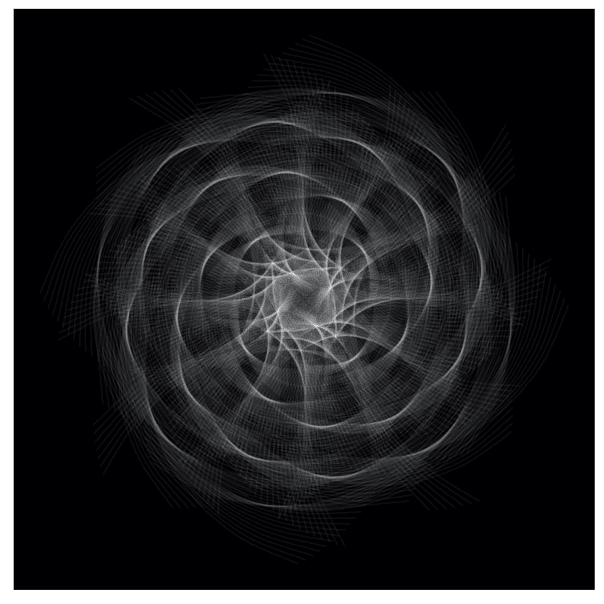
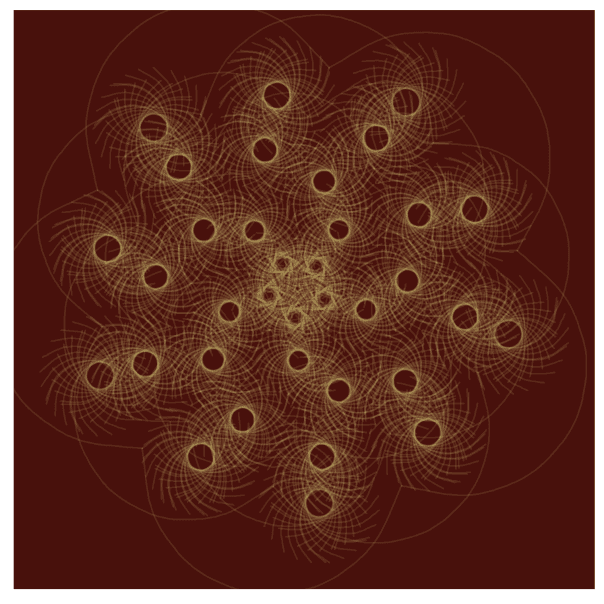
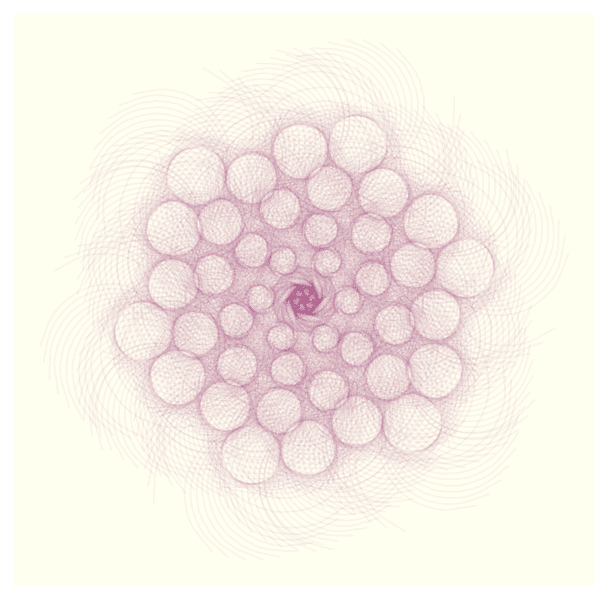
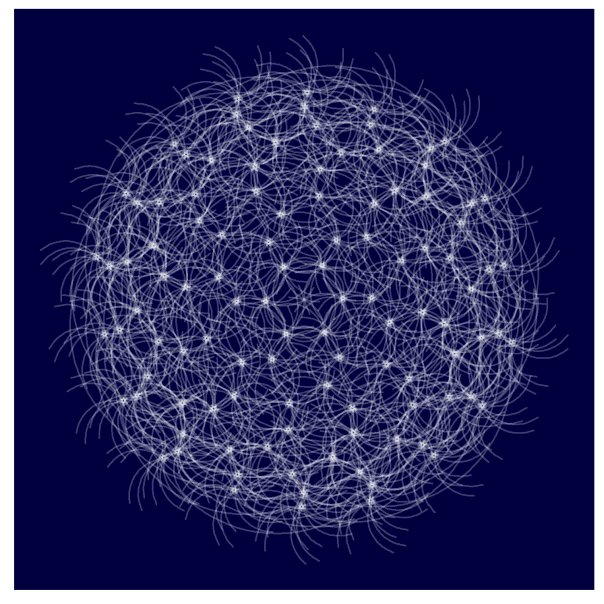
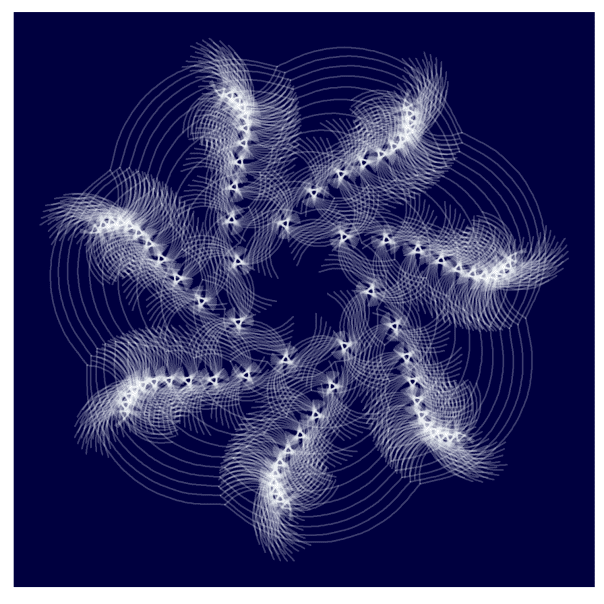
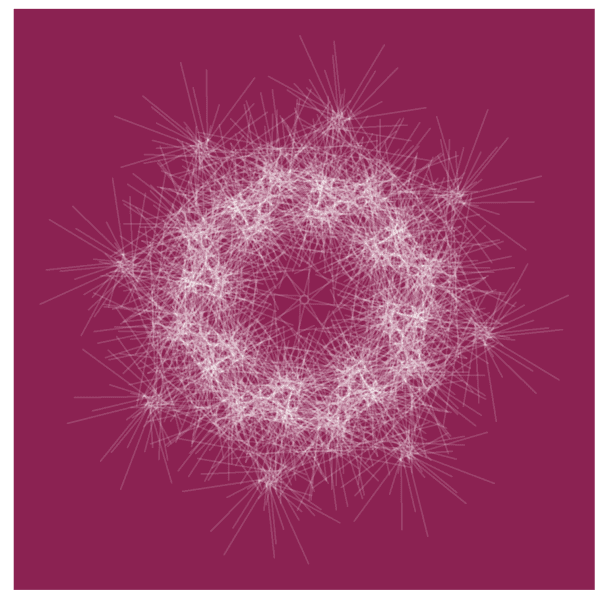
axis.text = element\_blank())

* 
* 
* 
* 
* 
* 
* 
* 
* 

The next table shows the parameters of each of the previous drawings (from left to right and top to bottom):

|  | **edges** | **niter** | **pond** | **step** | **alph** | **angle** | **curv** | **line\_color** | **back\_color** | **ratio\_f** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 4 | 200 | 0.92 | 9 | 0.50 | 6.12 | 0.0 | black | white | function (x) { x } |
| 2 | 5 | 150 | 0.72 | 13 | 0.35 | 2.96 | 0.0 | black | white | function (x) { sqrt(x) } |
| 3 | 15 | 250 | 0.67 | 9 | 0.15 | 1.27 | 1.0 | black | white | function (x) { sin(x) } |
| 4 | 9 | 150 | 0.89 | 14 | 0.35 | 3.23 | 0.0 | black | white | function (x) { sin(x) } |
| 5 | 5 | 150 | 0.27 | 17 | 0.35 | 4.62 | 0.0 | black | white | function (x) { log(x + 1) } |
| 6 | 14 | 100 | 0.87 | 14 | 0.15 | 0.57 | -2.0 | black | white | function (x) { 1 – cos(x)^2 } |
| 7 | 7 | 150 | 0.19 | 6 | 0.45 | 3.59 | 0.0 | black | white | function (x) { 1 – cos(x)^2 } |
| 8 | 4 | 150 | 0.22 | 10 | 0.45 | 4.78 | 0.0 | black | white | function (x) { 1/x } |
| 9 | 3 | 250 | 0.24 | 13 | 0.25 | 0.60 | 0.1 | black | white | function (x) { sin(x) } |

You can also play with colors. By the way: [this document](http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf) will help you to choose them by their name. Some examples:

* 
* 
* 
* 
* 
* 
* 
* 
* 

I will not *unveil* the parameters of the previous drawings. Maybe it can encourage you to try by yourself and find your own patterns. If you do, I will love to see them. I hope you enjoy this reading. The code is also available [here](https://github.com/aschinchon/drrrawing-with-purrr).