CHOYOA 3

Table of contents


```
#loading libraries
library(ggplot2)
library(tibble)
```

0.1 Exercise 1:

```
mpg |>
  ggplot(aes(displ, hwy, colour = drv)) + #using the mpg dataset to create ggplot
  geom_point(show.legend = FALSE, size = 7) + #adding points
  geom_line(show.legend = FALSE, size = 3, colour = "green") + #converting the dots in the
  coord_polar() + #creating polar coordinates
  theme_void() +
  scale_color_brewer()
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```

0.2 Exercise 2

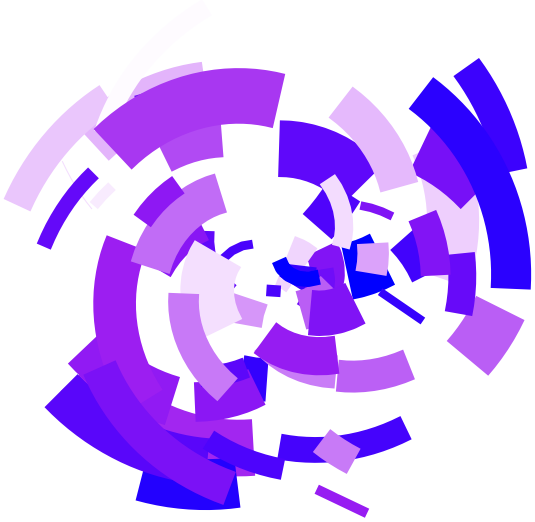
3. We might fall back on using the old-fashioned system of naming files because it helps to remember which file is which, and helps to organize your art.

Getting Started Exercise 2:

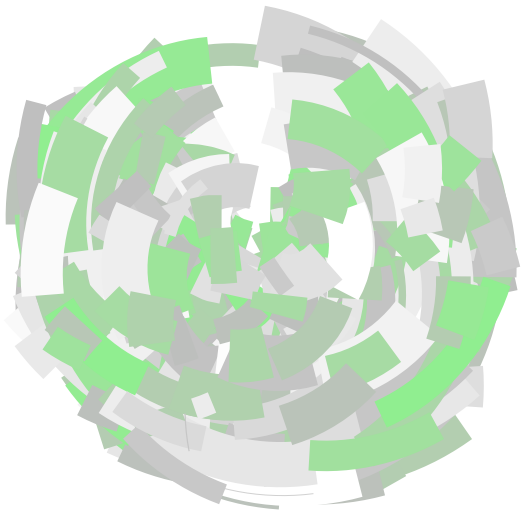
```
library(ggplot2)
library(tibble)

polar_art <- function(seed, n, palette) {

  # set the state of the random number generator
  set.seed(seed)
```

```
  # data frame containing random values for
  # aesthetics we might want to use in the art
  dat <- tibble(
    x0 = runif(n),
    y0 = runif(n),
    x1 = x0 + runif(n, min = -.2, max = .2),
    y1 = y0 + runif(n, min = -.2, max = .2),
    shade = runif(n),
    size = runif(n)
  )

  # plot segments in various colours, using
  # polar coordinates and a gradient palette
  dat |>
    ggplot(aes(
      x = x0,
      y = y0,
      xend = x1,
      yend = y1,
      colour = shade,
      size = size
    )) +
    geom_segment(show.legend = FALSE) +
    coord_polar() +
    scale_y_continuous(expand = c(0, 0)) +
    scale_x_continuous(expand = c(0, 0)) +
    scale_colour_gradientn(colours = palette) +
    scale_size(range = c(0, 10)) +
    theme_void()
}

polar_art(
  seed = 2,
  n = 50,
  palette = c("red", "black", "white")
)
```

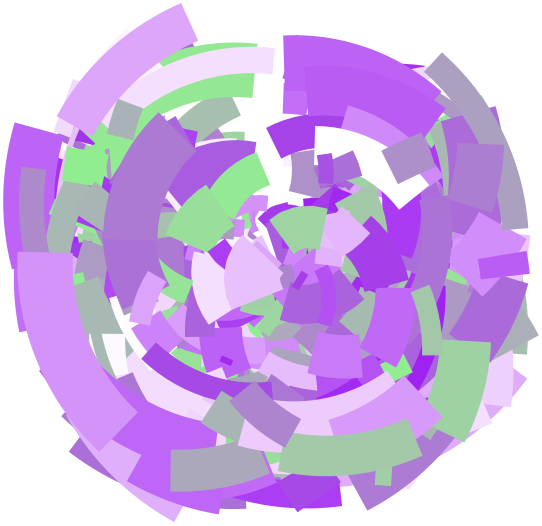
```
polar_art(
  seed = 5, #changing the seed creates a random generation of numbers
  n = 60, #creating 60 numbers
  palette = c("blue", "purple", "white") #changing the palette colors
)
```



```
polar_art(seed = 4, #creating a different random generation
          n = 800, #creating 800 numbers/elements
          palette = c("light green", "grey", "white")) #changing the palette colors
```



```
polar_art(seed = 4, #creating a different random generation
          n = 450, #creating 450 numbers/elements
          palette = c("light green", "purple", "white")) #changing the palette colors
```



Getting Started Exercise 3:

```
library(ggthemes)
library(scales)
library(tidyverse)
```

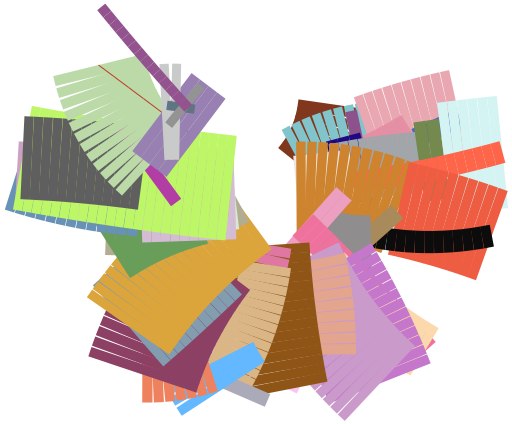
```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x readr::col_factor() masks scales::col_factor()
x purrr::discard()    masks scales::discard()
x dplyr::filter()     masks stats::filter()
x dplyr::lag()        masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

```r
# the original function from the first session
sample_canva <- function(seed = NULL) {
  if(!is.null(seed)) set.seed(seed)
  sample(ggthemes::canva_palettes, 1)[[1]]
}

# the extended function used in later sessions
sample_canva2 <- function(seed = NULL, n = 4) {
  if(!is.null(seed)) set.seed(seed)
  sample(ggthemes::canva_palettes, 1)[[1]] |>
    (\(x) colorRampPalette(x)(n))()
}

show_col(sample_canva())
```


```
#2.

sample_named_colours <- function(n) {
  all <- colors(distinct = TRUE)  #randomizing all the colors
  sample_colours <- sample(all, n, replace = TRUE)  #replacing the n colors
  return(sample_colours) #returning sample colors to randomize the colors
}
show_col(sample_named_colours())
```

```
polar_art <- function(seed, n) {

  # set the state of the random number generator
  set.seed(seed)

  # data frame containing random values for
  # aesthetics we might want to use in the art
  dat <- tibble(
    x0 = runif(n),
    y0 = runif(n),
    x1 = x0 + runif(n, min = -.2, max = .2),
    y1 = y0 + runif(n, min = -.2, max = .2),
    shade = runif(n),
    size = runif(n)
  )

  # plot segments in various colours, using
  # polar coordinates and a gradient palette
  dat |>
    ggplot(aes(
      x = x0,
      y = y0,
      xend = x1,
```

```
      yend = y1,
      colour = shade,
      size = size
    )) +
    geom_line(show.legend = FALSE) + #editing it so that the plot creates lines instead of
    coord_polar() +
    scale_y_continuous(expand = c(0, 0)) +
    scale_x_continuous(expand = c(0, 0)) +
    scale_colour_gradientn(colours = sample_named_colours()) + #changing to sample_named_c
    scale_size(range = c(0, 20)) + #changing the scale size to make the elements bigger
    theme_void()
}

polar_art(
  seed = 2,
  n = 50
)
```

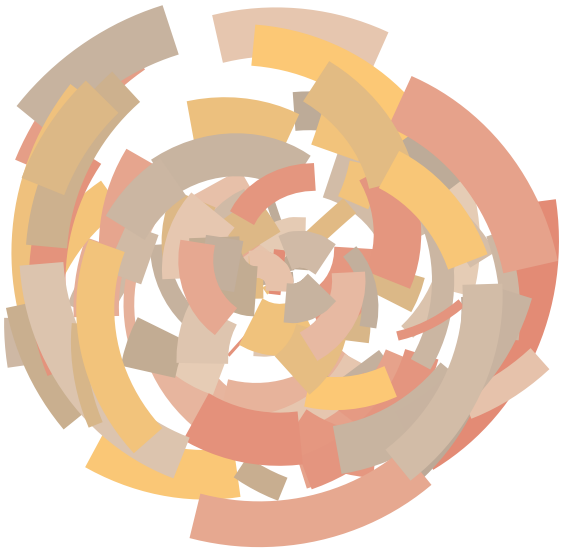


```
#3.

sample_canva <- function(n) {
  all_colours <- unlist(ggthemes::canva_palettes)  #calling vector of 600 colors
  sampled_colours <- sample(all_colours, n) #sampling n colors without replacement
  return(sampled_colours)
}
show_col(sample_canva())
```

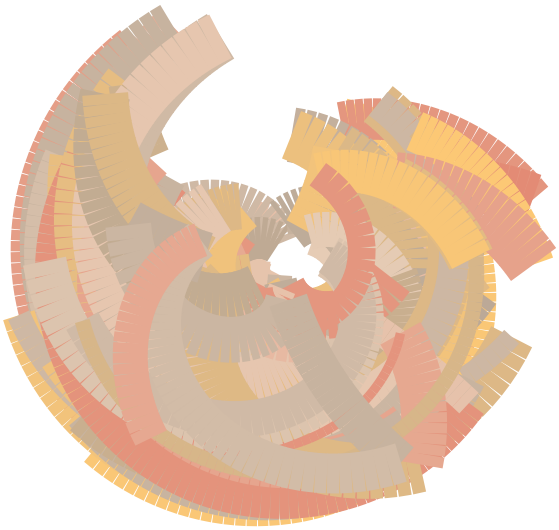
0.3 Getting Started Exercise 4

```
library(ggplot2)
library(tibble)
library(dplyr)

sample_canva <- function(seed = NULL) {
  if(!is.null(seed)) set.seed(seed)
  sample(ggthemes::canva_palettes, 1)[[1]] #adjusting the colors to create colors from can
}

sample_data <- function(seed = NULL, n = 100){ #creating sample data
  if(!is.null(seed)) set.seed(seed)
  dat <- tibble(
    x0 = runif(n),
    y0 = runif(n),
    x1 = x0 + runif(n, min = -.2, max = .2), #adjusting the randomness
    y1 = y0 + runif(n, min = -.2, max = .2),
    shade = runif(n),
    size = runif(n),
    shape = factor(sample(0:22, size = n, replace = TRUE))
  )
}
```

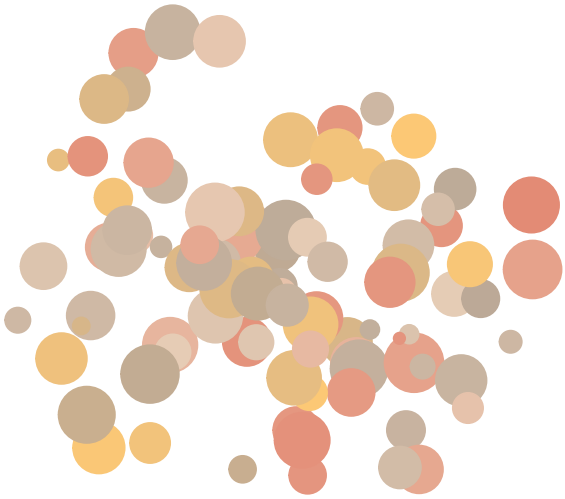
```r
polar_styled_plot <- function(data = NULL, palette) {
  ggplot(
    data = data, #creating ggplots from the data
    mapping = aes(
      x = x0,
      y = y0,
      xend = x1,
      yend = y1,
      colour = shade,
      size = size
    )) +
    coord_polar(clip = "off") +
    scale_y_continuous(
      expand = c(0, 0),
      limits = c(0, 1),
      oob = scales::oob_keep
    ) +
    scale_x_continuous(
      expand = c(0, 0),
      limits = c(0, 1),
      oob = scales::oob_keep
    ) +
    scale_colour_gradientn(colours = palette) + #getting colors from the palette
    scale_size(range = c(0, 10)) +
    theme_void() +
    guides(
      colour = guide_none(),
      size = guide_none(),
      fill = guide_none(),
      shape = guide_none()
    )
}

#1.
dat <- sample_data(n = 100, seed = 1)  #calling data with sample_canva
pal <- sample_canva(seed = 1)

#2.
polar_styled_plot(data = dat, palette = pal) + geom_segment()
```



```
polar_styled_plot(data = dat, palette = pal) + geom_path() #initializing plot with polar_s
```



```
polar_styled_plot(data = dat, palette = pal) + geom_point()
```



```
#3.
library(dplyr) #adding ggplot geoms

dat1 <- sample_data(n = 2000, seed = 123)
dat2 <- sample_data(n = 100, seed = 456) |>
  mutate(y0 = .3 + y0 * .6, y1 = .3)

polar_styled_plot(palette = sample_canva(seed = 7)) +
  geom_segment(
    data = dat1 |> mutate(size = size * 3)
  ) +
  geom_line(
    data = dat2 |> mutate(size = size / 5), #adding ggplot2 geoms- creating lines
    lineend = "round",
    colour = "white"
  ) +
  geom_segment(
    data = dat2 |> mutate(size = size / 40), #adding ggplot 2 geoms- creating segments
    lineend = "square", #creating squares for the end of the lines
    colour = "#222222"
  ) +
  geom_point(
    data = dat2 |> mutate(size = size * 2), #adding ggplot 2 geoms- creating points
    colour = "#222222"
  )
```

```
#2.
my_style_plot <- function(data = NULL, palette) {
  ggplot(
    data = data,
    mapping = aes(
      x = x0,
      y = y0,
      xend = x1,
      yend = y1,
      colour = gradient, #creating a gradient color
      size = size
    )) +
    coord_polar(clip = "o") +
    scale_y_continuous(
      expand = c(0, 0),
      limits = c(0, 1),
      oob = scales::oob_keep
    ) +
    scale_x_continuous(
      expand = c(0, 0),
      limits = c(0, 1),
      oob = scales::oob_keep
    ) +
```

```
    scale_colour_gradientn(colours = palette) +
    scale_size(range = c(0, 10)) +
    theme_void() +
    guides(
      colour = guide_none(),
      size = guide_none(),
      fill = guide_none(),
      shape = guide_none()
    )
}

dat <- sample_data(n = 50, seed = 4) #creating 50 elements, adjusting the randomness
pal <- sample_canva(seed = 6) #adjusting the randomness of the colors


polar_styled_plot(data = dat, palette = pal) + geom_jitter() + geom_line() #creating jitte
```

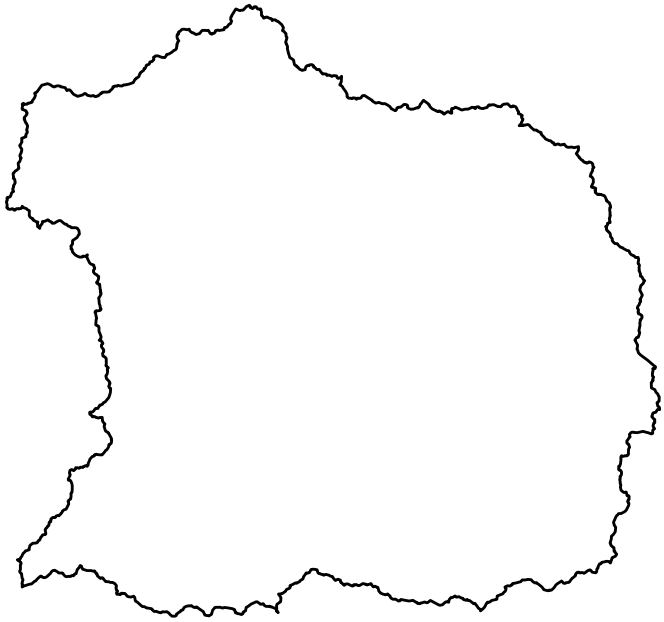
0.4 Polygon Tricks Task 2 (grow.polygon)

```
library(dplyr)
library(purrr)
library(tidyr)
library(tibble)
library(ggplot2)
```

```r
library(ambient)
library(tictoc)
library(ggthemes)
library(gifski)

#Exercise 1
square <- tibble(
  x = c(0, 1, 1, 0, 0),
  y = c(0, 0, 1, 1, 0),
  seg_len = c(1, 1, 1, 1, 0)
)

show_polygon <- function(polygon, show_vertices = TRUE, colour = "black", ...) {

  pic <- ggplot(polygon, aes(x, y)) +
    geom_polygon(fill = NA, colour = colour, show.legend = FALSE, ...) +
    coord_equal() +
    theme_void()

  if(show_vertices == TRUE) {
    pic <- pic + geom_point(size = 2, colour = colour)
  }
  return(pic)
}

sample_edge <- function(polygon) {
  sample(nrow(polygon), 1, prob = polygon$seg_len)
}

edge_length <- function(x1, y1, x2, y2) {
  sqrt((x1 - x2)^2 + (y1 - y2)^2)
}

edge_noise <- function(size) {
  runif(1, min = -size/2, max = size/2)
}

insert_edge <- function(polygon, noise) {

  # sample and edge and remember its length
  ind <- sample_edge(polygon)
```

```
  len <- polygon$seg_len[ind]

  # one endpoint of the old edge
  last_x <- polygon$x[ind]
  last_y <- polygon$y[ind]

  # the other endpoint of the old edge
  next_x <- polygon$x[ind + 1]
  next_y <- polygon$y[ind + 1]

  # location of the new point to be inserted: noise
  # is scaled proportional to the length of the old edge
  new_x <- (last_x + next_x) / 2 + edge_noise(len * noise)
  new_y <- (last_y + next_y) / 2 + edge_noise(len * noise)

  # the new row for insertion into the tibble,
  # containing coords and length of the 'new' edge
  new_row <- tibble(
    x = new_x,
    y = new_y,
    seg_len = edge_length(new_x, new_y, next_x, next_y)
  )

  # update the length of the 'old' edge
  polygon$seg_len[ind] <- edge_length(
    last_x, last_y, new_x, new_y
  )

  # insert a row into the tibble
  bind_rows(
    polygon[1:ind, ],
    new_row,
    polygon[-(1:ind), ]
  )
}

grow_polygon <- function(polygon, iterations, noise, seed = NULL) {
  if(!is.null(seed)) set.seed(seed)
  for(i in 1:iterations) polygon <- insert_edge(polygon, noise)
  return(polygon)
}
```

```r
# modify this code
pic <- square |>
  grow_polygon(iterations = 2000, noise = .5, seed = 2) |> #adjusting iterations, noise, a
  show_polygon(show_vertices = FALSE)

plot(pic)
```



0.5 Polygon Tricks Exercise 2 (grow.multipolygons)

```r
library(dplyr)
library(purrr)
library(tidyr)
library(tibble)
library(ggplot2)
library(ambient)
library(tictoc)
library(ggthemes)
library(gifski)
```

```r
square <- tibble(
  x = c(0, 1, 1, 0, 0),
  y = c(0, 0, 1, 1, 0),
  seg_len = c(1, 1, 1, 1, 0)
)

show_polygon <- function(polygon, show_vertices = TRUE, colour = "white", ...) { #adjustin

  pic <- ggplot(polygon, aes(x, y)) +
    geom_polygon(fill = NA, colour = colour, show.legend = FALSE, ...) +
    coord_equal() +
    theme_void()

  if(show_vertices == TRUE) {
    pic <- pic + geom_point(size = 3, colour = colour) #making the geom points bigger
  }
  return(pic)
}

sample_edge <- function(polygon) {
  sample(nrow(polygon), 1, prob = polygon$seg_len)
}

edge_length <- function(x1, y1, x2, y2) {
  sqrt((x1 - x2)^2 + (y1 - y2)^2)
}

edge_noise <- function(size) {
  runif(1, min = -size/2, max = size/2)
}

insert_edge <- function(polygon, noise) {

  # sample and edge and remember its length
  ind <- sample_edge(polygon)
  len <- polygon$seg_len[ind]

  # one endpoint of the old edge
  last_x <- polygon$x[ind]
  last_y <- polygon$y[ind]
```

```
  # the other endpoint of the old edge
  next_x <- polygon$x[ind + 1]
  next_y <- polygon$y[ind + 1]

  # location of the new point to be inserted: noise
  # is scaled proportional to the length of the old edge
  new_x <- (last_x + next_x) / 2 + edge_noise(len * noise)
  new_y <- (last_y + next_y) / 2 + edge_noise(len * noise)

  # the new row for insertion into the tibble,
  # containing coords and length of the 'new' edge
  new_row <- tibble(
    x = new_x,
    y = new_y,
    seg_len = edge_length(new_x, new_y, next_x, next_y)
  )

  # update the length of the 'old' edge
  polygon$seg_len[ind] <- edge_length(
    last_x, last_y, new_x, new_y
  )

  # insert a row into the tibble
  bind_rows(
    polygon[1:ind, ],
    new_row,
    polygon[-(1:ind), ]
  )
}

grow_polygon <- function(polygon, iterations, noise, seed = NULL) {
  if(!is.null(seed)) set.seed(seed)
  for(i in 1:iterations) polygon <- insert_edge(polygon, noise)
  return(polygon)
}

grow_multipolygon <- function(base_shape, n, seed = NULL, ...) {
  if(!is.null(seed)) set.seed(seed)
  polygons <- list()
  for(i in 1:n) {
    polygons[[i]] <- grow_polygon(base_shape, ...)
```

```
  }
  polygons <- bind_rows(polygons, .id = "id")
  polygons
}

show_multipolygon <- function(polygon, fill, alpha = .05, ...) { #adjusting size of the po
  ggplot(polygon, aes(x, y, group = id)) +
    geom_polygon(colour = NA, alpha = alpha, fill = fill, ...) +
    coord_equal() +
    theme_void()
}

# simplified version of the one in the workshop
tic()
dat <- square |>
  grow_polygon(iterations = 200, noise = .7, seed = 2) |> #adjusting iterations and seed o
  grow_multipolygon(n = 10, iterations = 300, noise = 1, seed = 2)
pic <- show_multipolygon(dat, fill = "blue", alpha = .2) #changing the color of the polygo
plot(pic)
```

```
toc()
```

```
1.383 sec elapsed
```

0.6 Polygon Tricks (splotch.r)

```
library(dplyr)
library(purrr)
library(tidyr)
library(tibble)
library(ggplot2)
library(ambient)
library(tictoc)
library(ggthemes)
library(here)
```

```
here() starts at /Users/celinepark/github/CHOYOA3
```

```
edge_length <- function(x1, y1, x2, y2) {
  sqrt((x1 - x2)^2 + (y1 - y2)^2)
}

edge_noise <- function(size) {
  runif(1, min = -size/2, max = size/2)
}

sample_edge_l <- function(polygon) {
  sample(length(polygon), 1, prob = map_dbl(polygon, ~ .x$seg_len))
}

insert_edge_l <- function(polygon, noise) {

  ind <- sample_edge_l(polygon)
  len <- polygon[[ind]]$seg_len

  last_x <- polygon[[ind]]$x
  last_y <- polygon[[ind]]$y

  next_x <- polygon[[ind + 1]]$x
  next_y <- polygon[[ind + 1]]$y
```

```
  new_x <- (last_x + next_x) / 2 + edge_noise(len * noise)
  new_y <- (last_y + next_y) / 2 + edge_noise(len * noise)

  new_point <- list(
    x = new_x,
    y = new_y,
    seg_len = edge_length(new_x, new_y, next_x, next_y)
  )

  polygon[[ind]]$seg_len <- edge_length(
    last_x, last_y, new_x, new_y
  )

  c(
    polygon[1:ind],
    list(new_point),
    polygon[-(1:ind)]
  )
}

grow_polygon_l <- function(polygon, iterations, noise, seed = NULL) {
  if(!is.null(seed)) set.seed(seed)
  for(i in 1:iterations) polygon <- insert_edge_l(polygon, noise)
  return(polygon)
}

grow_multipolygon_l <- function(base_shape, n, seed = NULL, ...) {
  if(!is.null(seed)) set.seed(seed)
  polygons <- list()
  for(i in 1:n) {
    polygons[[i]] <- grow_polygon_l(base_shape, ...) |>
      transpose() |>
      as_tibble() |>
      mutate(across(.fn = unlist))
  }
  polygons <- bind_rows(polygons, .id = "id")
  polygons
}

show_multipolygon <- function(polygon, fill, alpha = .02, ...) {
  ggplot(polygon, aes(x, y, group = id)) +
```

```
    geom_polygon(colour = NA, alpha = alpha, fill = fill, ...) +
    coord_equal() +
    theme_void()
}

triangle <- transpose(tibble(
  x = c(0, 1, 0.5, 0),
  y = c(0, 0, 1, 0),
  seg_len = c(1, 1, 1, 0)
))

splotch_triangle <- function(seed, layers = 20) {
  set.seed(seed)
  triangle |>  #creating a triangle to replace the polygon
    grow_polygon_l(iterations = 10, noise = .5, seed = seed) |>
    grow_multipolygon_l(n = layers, iterations = 500, noise = .8, seed = seed)
}

tic()
dat_triangle <- splotch_triangle(seed = 1)
```

```
Warning: There was 1 warning in `mutate()`.
i In argument: `across(.fn = unlist)`.
Caused by warning:
! Using `across()` without supplying `.cols` was deprecated in dplyr 1.1.0.
i Please supply `.cols` instead.
```

```
pic_triangle <- dat_triangle |> show_multipolygon(fill = "black", alpha = .2) #adjusting t
ggsave(
  filename = here("output", "splotch_triangle.png"),
  plot = pic_triangle, #plotting the pic of a triangle
  width = 2000,
  height = 2000,
  units = "px",
  dpi = 300,
  bg = "black"
)
toc()
```

```
2.422 sec elapsed
```

```
plot(pic_triangle)
```

```
#Example 1
#Adding more numbers to the layer argument adds more layers of polygons, which thus affect

#Example 3
#The mutate() function helps to switch the position along the x-axis for each polygon
#The arrange() function ensures that the polygons are plotted in a specific order in order
```

##Polygon Tricks Exercise 2 (smudged_hexagon)

```
library(dplyr)
library(purrr)
library(tidyr)
library(tibble)
library(ggplot2)
library(tictoc)
library(ggthemes)
library(here)

edge_length <- function(x1, y1, x2, y2) {
  sqrt((x1 - x2)^2 + (y1 - y2)^2)
```

```r
}

edge_noise <- function(size) {
  runif(1, min = -size/2, max = size/2)
}

sample_edge_l <- function(polygon) {
  sample(length(polygon), 1, prob = map_dbl(polygon, ~ .x$seg_len))
}

insert_edge_l <- function(polygon, noise) {

  ind <- sample_edge_l(polygon)
  len <- polygon[[ind]]$seg_len

  last_x <- polygon[[ind]]$x
  last_y <- polygon[[ind]]$y

  next_x <- polygon[[ind + 1]]$x
  next_y <- polygon[[ind + 1]]$y

  new_x <- (last_x + next_x) / 2 + edge_noise(len * noise)
  new_y <- (last_y + next_y) / 2 + edge_noise(len * noise)

  new_point <- list(
    x = new_x,
    y = new_y,
    seg_len = edge_length(new_x, new_y, next_x, next_y)
  )

  polygon[[ind]]$seg_len <- edge_length(
    last_x, last_y, new_x, new_y
  )

  c(
    polygon[1:ind],
    list(new_point),
    polygon[-(1:ind)]
  )
}
```

```r
grow_polygon_l <- function(polygon, iterations, noise, seed = NULL) {
  if(!is.null(seed)) set.seed(seed)
  for(i in 1:iterations) polygon <- insert_edge_l(polygon, noise)
  return(polygon)
}

grow_multipolygon_l <- function(base_shape, n, seed = NULL, ...) {
  if(!is.null(seed)) set.seed(seed)
  polygons <- list()
  for(i in 1:n) {
    polygons[[i]] <- grow_polygon_l(base_shape, ...) |>
      transpose() |>
      as_tibble() |>
      mutate(across(.fn = unlist))
  }
  polygons <- bind_rows(polygons, .id = "id")
  polygons
}

show_multipolygon <- function(polygon, fill, alpha = .02, ...) {
  ggplot(polygon, aes(x, y, group = id)) +
    geom_polygon(colour = NA, alpha = alpha, fill = fill, ...) +
    coord_equal() +
    theme_void()
}

smudged_hexagon <- function(seed, noise1 = 0, noise2 = 2, noise3 = 0.5) {
  set.seed(seed)

  # define hexagonal base shape
  theta <- (0:6) * pi / 3
  hexagon <- tibble(
    x = sin(theta),
    y = cos(theta),
    seg_len = edge_length(x, y, lead(x), lead(y))
  )
  hexagon$seg_len[7] <- 0
  hexagon <- transpose(hexagon)
  base <- hexagon |>
    grow_polygon_l(
      iterations = 60,
```

```
      noise = noise1
    )

  # define intermediate-base-shapes in clusters
  polygons <- list()
  ijk <- 0
  for(i in 1:3) {
    base_i <- base |>
      grow_polygon_l(
        iterations = 50,
        noise = noise2
      )

    for(j in 1:3) {
      base_j <- base_i |>
        grow_polygon_l(
          iterations = 50,
          noise = noise2
        )

      # grow 10 polygons per intermediate-base
      for(k in 1:10) {
        ijk <- ijk + 1
        polygons[[ijk]] <- base_j |>
          grow_polygon_l(
            iterations = 500,
            noise = noise3
          ) |>
          transpose() |>
          as_tibble() |>
          mutate(across(.fn = unlist))
      }
    }
  }

  # return as data frame
  bind_rows(polygons, .id = "id")
}


triangle <- transpose(tibble(
```

```
  x = c(0, 1, 0.5, 0),
  y = c(0, 0, 1, 0),
  seg_len = c(1, 1, 1, 0)
))

splotch_triangle <- function(seed, layers = 20) {
  set.seed(seed)
  triangle |>  #creating a triangle to replace the polygon
    grow_polygon_l(iterations = 10, noise = .5, seed = seed) |>
    grow_multipolygon_l(n = layers, iterations = 500, noise = .8, seed = seed)
}

tic()
dat_triangle <- splotch_triangle(seed = 1)
pic_triangle <- dat_triangle |> show_multipolygon(fill = "black", alpha = .2) #adjusting t
ggsave(
  filename = here("output", "splotch_triangle.png"),
  plot = pic_triangle, #plotting the pic of a triangle
  width = 2000,
  height = 2000,
  units = "px",
  dpi = 300,
  bg = "black"
)
toc()
```

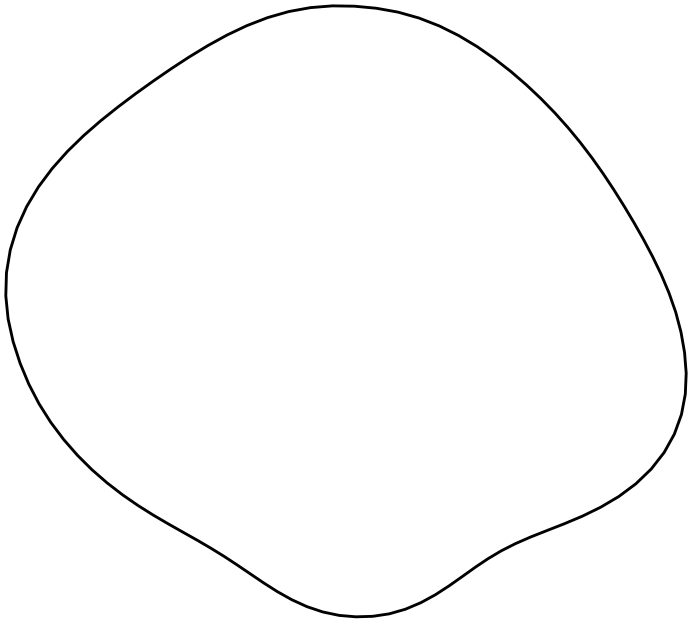
```
2.464 sec elapsed
```

```
plot(pic_triangle)
```

0.7 Polygon Tricks Task 3 (perlin-blob.R)

```
library(dplyr)
library(purrr)
library(tidyr)
library(tibble)
library(ggplot2)
library(ambient)
library(tictoc)
library(ggthemes)
library(gifski)

show_polygon <- function(polygon, show_vertices = TRUE, ...) {

  pic <- ggplot(polygon, aes(x, y)) +
    geom_polygon(colour = "black", fill = NA, show.legend = FALSE, ...) +
    coord_equal() +
    theme_void()

  if(show_vertices == TRUE) {
    pic <- pic + geom_point(colour = "black", size = 2)
  }
```

```
  return(pic)
}

normalise_radius <- function(x, min, max) {
  normalise(x, from = c(-0.5, 0.5), to = c(min, max))
}

perlin_blob <- function(n = 100,
                        freq_init = 0.3,
                        octaves = 2,
                        r_min = 0.5,
                        r_max = 1) {
  tibble(
    angle = seq(0, 2*pi, length.out = n),
    x_base = cos(angle),
    y_base = sin(angle),
    radius = fracture(
      x = x_base,
      y = y_base,
      freq_init = freq_init,
      noise = gen_perlin,
      fractal = fbm,
      octaves = octaves
    ) |>
      normalise_radius(r_min, r_max),
    x = radius * x_base,
    y = radius * y_base
  )
}

set.seed(3);
pic <- perlin_blob(freq_init = .4) |> show_polygon(FALSE)
plot(pic)
```

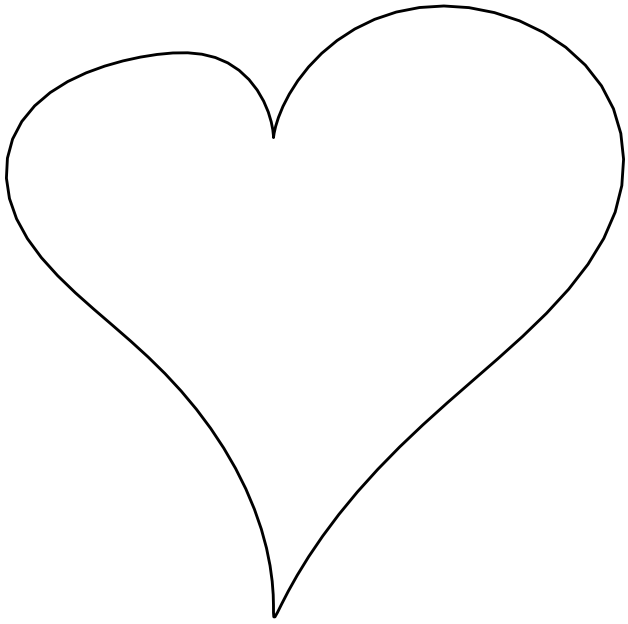


0.8 Polygon Tricks Exercise 3 (perlin-heart.R)

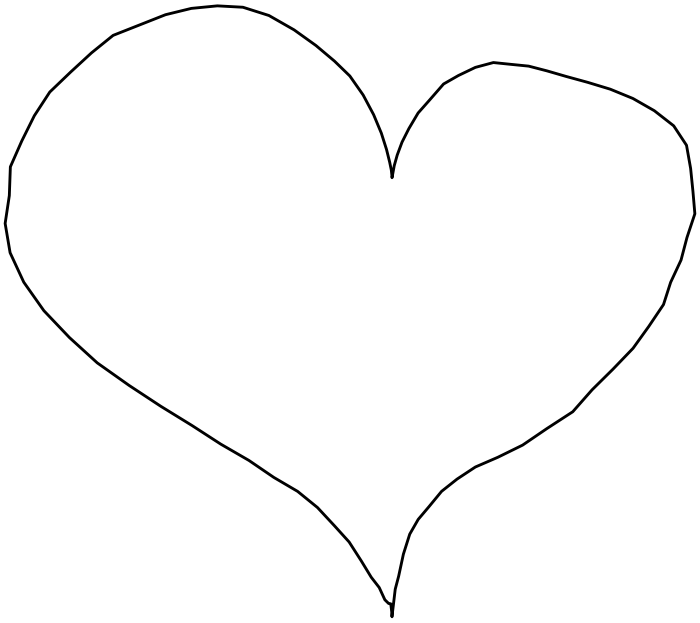
```
library(dplyr)
library(purrr)
library(tidyr)
library(tibble)
library(ggplot2)
library(ambient)
library(tictoc)
library(ggthemes)
library(gifski)

show_polygon <- function(polygon, show_vertices = TRUE, ...) {

  pic <- ggplot(polygon, aes(x, y)) +
    geom_polygon(colour = "black", fill = NA, show.legend = FALSE, ...) +
    coord_equal() +
    theme_void()

  if(show_vertices == TRUE) {
    pic <- pic + geom_point(colour = "black", size = 2)
  }
```

```
  return(pic)
}

heart_x <- function(angle) {
  x <- (16 * sin(angle) ^ 3) / 17
  return(x - mean(x))
}

heart_y <- function(angle) {
  y <- (13 * cos(angle) - 5 * cos(2 * angle) - 2 * cos(3 * angle) -
          cos(4 * angle)) / 17
  return(y - mean(y))
}

normalise_radius <- function(x, min, max) {
  normalise(x, from = c(-0.5, 0.5), to = c(min, max))
}

perlin_heart <- function(n = 100,
                         freq_init = 0.3,
                         octaves = 2,
                         r_min = 0.5,
                         r_max = 1,
                         x_shift = 0,
                         y_shift = 0,
                         id = NA,
                         seed = NULL) {
  if(!is.null(seed)) set.seed(seed)
  tibble(
    angle = seq(0, 2*pi, length.out = n),
    x_base = cos(angle),
    y_base = sin(angle),
    radius = fracture(
      x = x_base,
      y = y_base,
      freq_init = freq_init,
      noise = gen_perlin,
      fractal = fbm,
      octaves = octaves
    ) |>
      normalise_radius(r_min, r_max),
```

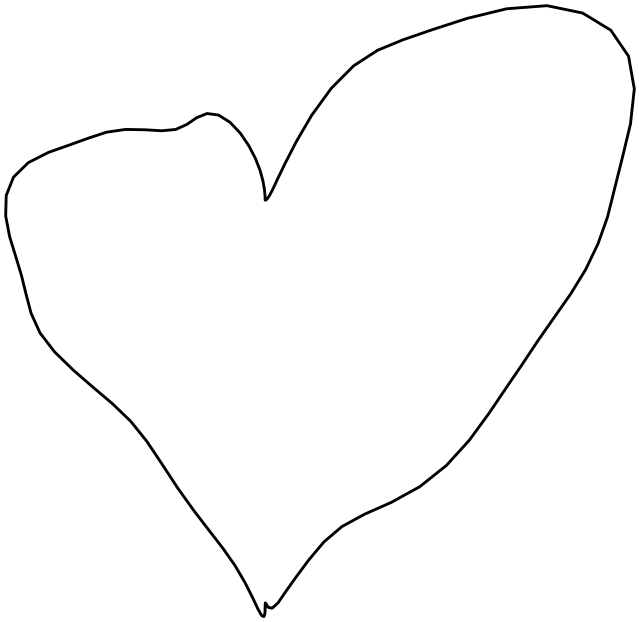
```
    x = radius * heart_x(angle) + x_shift,
    y = radius * heart_y(angle) + y_shift,
    id = id
  )
}

set.seed(3)
pic1 <- perlin_heart(freq_init = 0.4) |> show_polygon(FALSE)
pic2 <- perlin_heart(freq_init = 0.2, octaves = 20) |> show_polygon(FALSE) #adjusting the
pic3 <- perlin_heart(freq_init = 0.6, octaves = 3) |> show_polygon(FALSE)

# Plotting the results
plot(pic1)
```



```
plot(pic2)
```



```
plot(pic3)
```

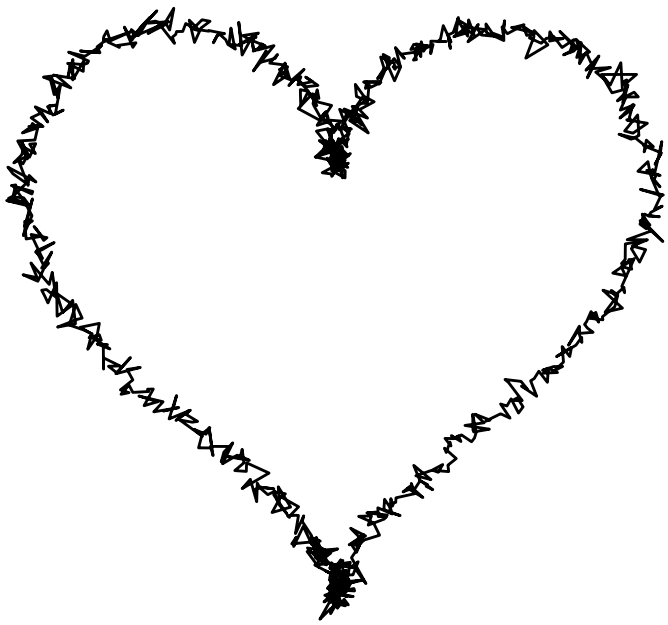


0.9 Polygon Tricks Task 4 (perlin. grid. R)

```
library(dplyr)
library(purrr)
library(tidyr)
library(tibble)
library(ggplot2)
library(ambient)
library(tictoc)
library(ggthemes)
library(gifski)

sample_canva <- function(seed = NULL) {
  if(!is.null(seed)) set.seed(seed)
  sample(ggthemes::canva_palettes, 1)[[1]]
}

show_polygon <- function(polygon, show_vertices = TRUE, ...) {

  pic <- ggplot(polygon, aes(x, y)) +
    geom_polygon(colour = "black", fill = NA, show.legend = FALSE, ...) +
    coord_equal() +
    theme_void()

  if(show_vertices == TRUE) {
    pic <- pic + geom_point(colour = "black", size = 2)
  }
  return(pic)
}

heart_x <- function(angle) {
  x <- (16 * sin(angle) ^ 3) / 17
  return(x - mean(x))
}

heart_y <- function(angle) {
  y <- (13 * cos(angle) - 5 * cos(2 * angle) - 2 * cos(3 * angle) -
          cos(4 * angle)) / 17
  return(y - mean(y))
}
```

```r
normalise_radius <- function(x, min, max) {
  normalise(x, from = c(-0.5, 0.5), to = c(min, max))
}

perlin_heart <- function(n = 100,
                         freq_init = 0.3,
                         octaves = 2,
                         r_min = 0.5,
                         r_max = 1,
                         x_shift = 0,
                         y_shift = 0,
                         id = NA,
                         seed = NULL) {
  if (!is.null(seed)) set.seed(seed)
  tibble(
    angle = seq(0, 2*pi, length.out = n),
    x_base = cos(angle),
    y_base = sin(angle),
    radius = fracture(
      x = x_base,
      y = y_base,
      freq_init = freq_init,
      noise = gen_perlin,
      fractal = fbm,
      octaves = octaves
    ) %>%
      normalise_radius(r_min, r_max),
    x = radius * heart_x(angle) + x_shift + rnorm(n, mean = 0, sd = 0.02),  #using rnorm t
    y = radius * heart_y(angle) + y_shift + rnorm(n, mean = 0, sd = 0.02),  #using rnorm t
    id = id #mean controls the mean value of the noise, and sd measures the standard devia
  )
}


perlin_heart_grid <- function(nx = 10, ny = 3, seed = NULL) { #only showing half of the he
  if(!is.null(seed)) set.seed(seed)

  heart_settings <- expand_grid(
    r_min = .3,
    r_max = .4,
    x_shift = 1:nx,
```

```
    y_shift = 1:ny
  ) |>
    mutate(id = row_number())

  heart_data <-  pmap_dfr(heart_settings, perlin_heart)

  heart_data |>
    ggplot(aes(x, y, group = id, fill = sample(id))) +
    geom_polygon(size = 0, show.legend = FALSE) +
    theme_void() +
    scale_fill_gradientn(colours = sample_canva(seed)) +
    coord_equal(xlim = c(0, nx + 1), ylim = c(0, ny + 1))
}

set.seed(1);
perlin_heart(
  n = 1000,
  freq_init = 10, #using freq_init to adjust the overall noise level to create rough edges
  r_min = .95,
  r_max = 1
) |>
  show_polygon(FALSE)
```



```
pic <- perlin_heart_grid(seed = 451)
plot(pic)
```

0.10 Polygon Tricks Exercise 5 (perlin-heart-grid-2.R)

```
library(dplyr)
library(purrr)
library(tidyr)
library(tibble)
library(ggplot2)
library(ambient)
library(tictoc)
library(ggthemes)
library(gifski)

sample_canva <- function(seed = NULL) {
  if(!is.null(seed)) set.seed(seed)
  sample(ggthemes::canva_palettes, 1)[[1]]
}

show_polygon <- function(polygon, show_vertices = TRUE, ...) {

  pic <- ggplot(polygon, aes(x, y)) +
    geom_polygon(colour = "black", fill = NA, show.legend = FALSE, ...) +
    coord_equal() +
    theme_void()

  if(show_vertices == TRUE) {
    pic <- pic + geom_point(colour = "black", size = 2)
  }
  return(pic)
}
```

```
heart_x <- function(angle) {
  x <- (16 * sin(angle) ^ 3) / 17
  return(x - mean(x))
}

heart_y <- function(angle) {
  y <- (13 * cos(angle) - 5 * cos(2 * angle) - 2 * cos(3 * angle) -
          cos(4 * angle)) / 17
  return(y - mean(y))
}

normalise_radius <- function(x, min, max) {
  normalise(x, from = c(-0.5, 0.5), to = c(min, max))
}

perlin_heart2 <- function(n = 100,
                          freq_init = 0.3,
                          octaves = 2,
                          r_min = 0.5,
                          r_max = 1,
                          w_min = 0,
                          w_max = 4,
                          rot = 0,
                          x_shift = 0,
                          y_shift = 0,
                          id = NA,
                          seed = NULL) {
  if(!is.null(seed)) set.seed(seed)
  tibble(
    angle = seq(0, 2*pi, length.out = n),

    radius = fracture(
      x = cos(angle),
      y = sin(angle),
      freq_init = freq_init,
      noise = gen_perlin,
      fractal = fbm,
      octaves = octaves
    ) |>
      normalise_radius(r_min, r_max),
```

```
    x = radius * heart_x(angle) + x_shift,
    y = radius * heart_y(angle) + y_shift,

    width = fracture(
      x = cos(angle + rot),
      y = sin(angle + rot),
      freq_init = freq_init,
      noise = gen_perlin,
      fractal = fbm,
      octaves = octaves
    ) |>
      normalise(to = c(w_min, w_max)),

    id = id
  )
}

perlin_heart_grid2 <- function(nx = 4, ny = 2, seed = NULL) {
  if(!is.null(seed)) set.seed(seed)

  heart_settings <- expand_grid(
    r_min = .3,
    r_max = .4,
    w_min = .01,
    w_max = 6,
    x_shift = 1:nx,
    y_shift = 1:ny
  ) |>
    mutate(
      n = 200,
      x_shift = x_shift + runif(n(), -.1, .1),
      y_shift = y_shift + runif(n(), -.1, .1),
      rot = runif(n(), -.1, .1),
      id = row_number()
    )

  heart_data <-  pmap_dfr(heart_settings, perlin_heart2)

  heart_data |>
    ggplot(aes(x, y, group = id, colour = sample(id), size = width)) +
    geom_path(show.legend = FALSE) +
```

```
    theme_void() +
    scale_size_identity() +
    scale_colour_gradientn(colours = sample_canva(seed)) +
    scale_x_continuous(expand = c(0, 0)) +
    scale_y_continuous(expand = c(0, 0)) +
    coord_fixed(xlim = c(0, nx + 1), ylim = c(0, ny + 1))
}

pic <- perlin_heart_grid2(seed = 666)
plot(pic)
```

0.11 Polygon Tricks Task 2 (perlin.heart-animated)

```
library(dplyr)
library(purrr)
library(tidyr)
library(tibble)
library(ggplot2)
library(ambient)
library(tictoc)
library(ggthemes)
library(gifski)
library(here)

sample_canva <- function(seed = NULL) {
  if(!is.null(seed)) set.seed(seed)
  sample(ggthemes::canva_palettes, 1)[[1]]
}
```

```r
show_polygon <- function(polygon, show_vertices = TRUE, ...) {

  pic <- ggplot(polygon, aes(x, y)) +
    geom_polygon(colour = "black", fill = NA, show.legend = FALSE, ...) +
    coord_equal() +
    theme_void()

  if(show_vertices == TRUE) {
    pic <- pic + geom_point(colour = "black", size = 2)
  }
  return(pic)
}

heart_x <- function(angle) {
  x <- (16 * sin(angle) ^ 3) / 17
  return(x - mean(x))
}

heart_y <- function(angle) {
  y <- (13 * cos(angle) - 5 * cos(2 * angle) - 2 * cos(3 * angle) -
          cos(4 * angle)) / 17
  return(y - mean(y))
}

normalise_radius <- function(x, min, max) {
  normalise(x, from = c(-0.5, 0.5), to = c(min, max))
}

perlin_heart2 <- function(n = 100,
                          freq_init = 0.3,
                          octaves = 2,
                          r_min = 0.5,
                          r_max = 1,
                          w_min = 0,
                          w_max = 4,
                          rot = 0,
                          x_shift = 0,
                          y_shift = 0,
                          id = NA,
                          seed = NULL) {
  if(!is.null(seed)) set.seed(seed)
```

```
  tibble(
    angle = seq(0, 2*pi, length.out = n),

    radius = fracture(
      x = cos(angle),
      y = sin(angle),
      freq_init = freq_init,
      noise = gen_perlin,
      fractal = fbm,
      octaves = octaves
    ) |>
      normalise_radius(r_min, r_max),

    x = radius * heart_x(angle) + x_shift,
    y = radius * heart_y(angle) + y_shift,

    width = fracture(
      x = cos(angle + rot),
      y = sin(angle + rot),
      freq_init = freq_init,
      noise = gen_perlin,
      fractal = fbm,
      octaves = octaves
    ) |>
      normalise(to = c(w_min, w_max)),

    id = id
  )
}

perlin_heart_data <- function(nhearts = 10, scatter = .05, seed = NULL) {

  if(!is.null(seed)) set.seed(seed)

  palette <- sample_canva(seed) |>
    (\(x) colorRampPalette(x)(nhearts))()

  heart_settings <- tibble(
    id = 1:nhearts,
    n = 500,
    r_min = .35,
```

```
    r_max = .4,
    w_min = -10,
    w_max = 10,
    x_shift = runif(nhearts, -scatter/2, scatter/2),
    y_shift = runif(nhearts, -scatter/2, scatter/2),
    rot = runif(nhearts, -pi, pi)
  )

  heart_settings |>
    pmap_dfr(perlin_heart2) |>
    group_by(id) |>
    mutate(
      shade = sample(palette, 1),
      width = abs(width)
    )
}

generate_one_frame <- function(dat) {

  pic <- dat |>
    ggplot(aes(x, y, group = id, size = width, colour = shade)) +
    geom_path(show.legend = FALSE) +
    theme_void() +
    scale_x_continuous(expand = c(0, 0)) +
    scale_y_continuous(expand = c(0, 0)) +
    scale_colour_identity() +
    scale_size_identity() +
    coord_fixed(xlim = c(-.6, .6), ylim = c(-.6, .6))

  print(pic)
}

rotate_vector <- function(x, percent) {

  len <- length(x)
  ind <- ceiling(len * percent)
  if(ind == 0) return(x)
  if(ind == len) return(x)
  c(x[(ind+1):len], x[1:ind])
}
```

```
generate_all_frames <- function(dat, nframes = 100) {

  for(frame in 1:nframes) {
    dat |>
      group_by(id) |>
      mutate(width = width |> rotate_vector(frame / nframes)) |>
      generate_one_frame()
  }
}

animated_perlin_heart <- function(seed, ...) {

  gif_file <- paste0("animated-perlin-heart-", seed, ".gif")
  save_gif(
    expr = perlin_heart_data(seed = seed, ...) |> generate_all_frames(),
    gif_file = here("output", gif_file),
    height = 1000,
    width = 1000,
    delay = .1,
    progress = TRUE,
    bg = "#222222"
  )
  invisible(NULL)
}

tic()
animated_perlin_heart(seed = 99)
toc()
```

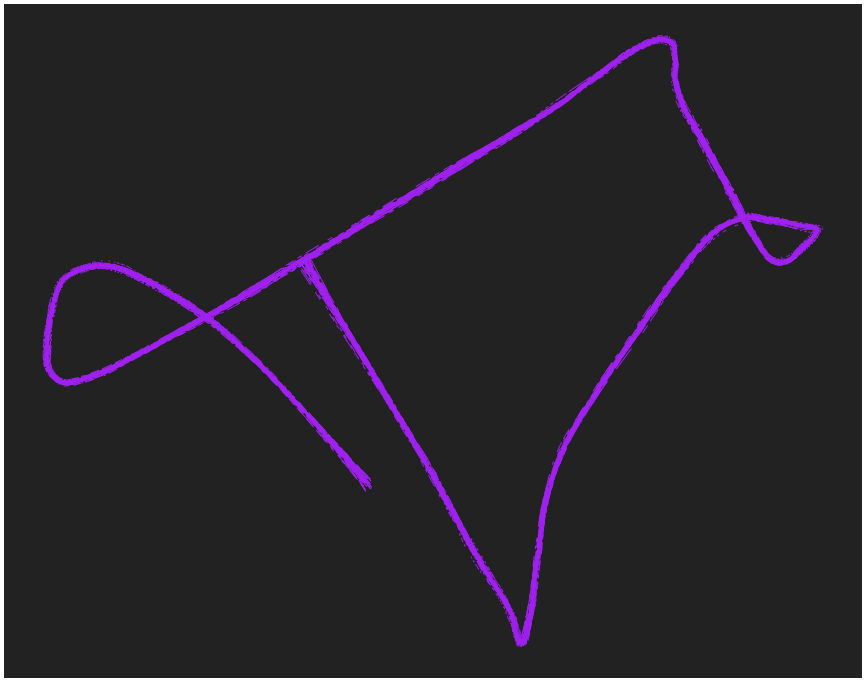
```
10.761 sec elapsed
```

0.12 Exercise 8 textured-lines

```
library(dplyr)
library(tibble)
library(ggplot2)
library(here)
library(e1071)

smooth_loess <- function(x, span) {
```

```
  n <- length(x)
  dat <- tibble(time = 1:n, walk = x)
  mod <- loess(walk ~ time, dat, span = span)
  predict(mod, tibble(time = 1:n))
}

smooth_path <- function(n = 1000, smoothing = .4, seed = NULL) {
  if(!is.null(seed)) set.seed(seed)
  tibble(
    x = smooth_loess(rbridge(1, n), span = smoothing),
    y = smooth_loess(rbridge(1, n), span = smoothing),
    stroke = 1
  )
}

perturb <- function(path, noise = .01, span = .1) {
  path |>
    group_by(stroke) |>
    mutate(
      x = x + rnorm(n(), 0, noise),
      y = y + rnorm(n(), 0, noise),
      x = smooth_loess(x, span),
      y = smooth_loess(y, span),
      alpha = runif(n()) > .5,
      size = runif(n(), 0, .2)
    )
}

brush <- function(path, bristles = 50, seed = 1, ...) { #adjusting the bristles to make it
  set.seed(seed)
  dat <- list()
  for(i in 1:bristles) {
    dat[[i]] <- perturb(path, ...)
  }
  return(bind_rows(dat, .id = "id"))
}

stroke <- function(dat, geom = geom_path, colour = "purple", ...) {
  dat |>
    ggplot(aes(
      x = x,
```

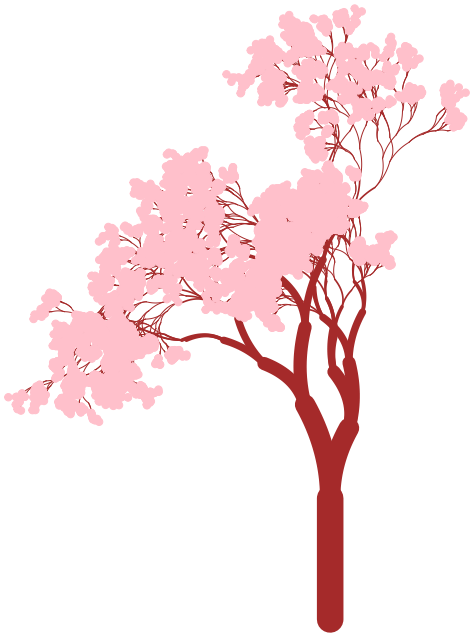
```
      y = y,
      alpha = alpha,
      size = size,
      group = paste0(stroke, id)
    )) +
    geom(
      colour = colour,
      show.legend = FALSE,
      ...
    ) +
    coord_equal() +
    scale_alpha_identity() +
    scale_size_identity() +
    theme_void() +
    theme(plot.background = element_rect(
      fill = "#222222",
      colour = "#222222"
    ))
}

path <- smooth_path(seed = 1) #adjusting the ssed to make the stroke more random
pic <- path |> brush() |> stroke()
plot(pic)
```



0.13 Pixel Filters Task 3 (flame tree)

```
library(dplyr)
library(tibble)
library(ggplot2)
library(ggforce)
library(flametree)

tree <- flametree_grow(
  seed = 6, #adjusting the seed to change the shape of the tree
  time = 12, #adjusting the time to make the tree bigger
  angle = c(-15, 15, 30)
)

leaf <- tree |> filter(id_leaf == TRUE)

base <- ggplot() +
  scale_size_identity() +
  theme_void() +
  coord_equal()
```

```r
leaves <- geom_point(
  mapping = aes(coord_x, coord_y),
  data = leaf,
  size = 1.3,
  stroke = 0,
  colour = "pink" #adjusting the color of the leaves
)

trunk <- geom_bezier(
  mapping = aes(coord_x, coord_y, group = id_pathtree, size = seg_wid),
  data = tree,
  lineend = "round",
  colour = "brown", #changing the color of the trunk to brown
  show.legend = FALSE
)

plot(base + trunk + leaves)
```



0.14 Pixel filters (glow)

```
library(dplyr)
library(tibble)
library(ggplot2)
library(ggforce)
library(flametree)
library(ggfx)

tree <- flametree_grow(
  seed = 10, #adjusting the shape of the tree
  time = 12, #adjusting the size of the tree
  angle = c(-15, 15, 30)
)

leaf <- tree |> filter(id_leaf == TRUE)

base <- ggplot() +
  scale_size_identity() +
  theme_void() +
  coord_equal()

leaves <- geom_point(
  mapping = aes(coord_x, coord_y),
  data = leaf,
  size = 1.3,
  stroke = 0,
  colour = "pink" #adjusting the leaves color
)

trunk <- geom_bezier(
  mapping = aes(coord_x, coord_y, group = id_pathtree, size = seg_wid),
  data = tree,
  lineend = "round",
  colour = "brown", #adjusting the color of the base
  show.legend = FALSE
)

plot(
  base +
    trunk +
```

```
    with_inner_glow(leaves, colour = "#555555", sigma = 5, expand = 5) #creating inner glo
) #expanding each point by 5 pixels
```

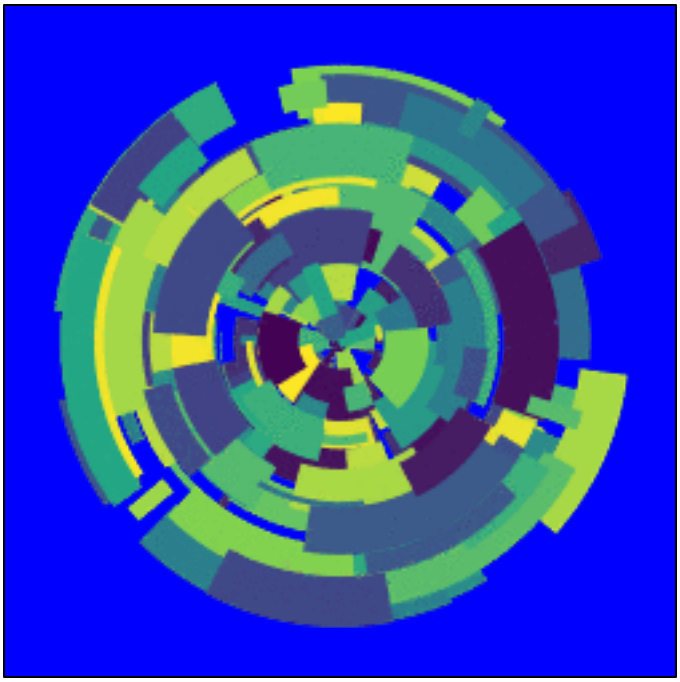
0.15 Pixel Filters (task 3)

```
library(dplyr)
library(tibble)
library(ggplot2)
library(ggfx)

set.seed(5) #adjusting the randomness of the plots
polar <- tibble(
  arc_start = runif(200),
  arc_end = arc_start + runif(200, min = -.2, max = .2),
  radius = runif(200),
  shade = runif(200),
  size = runif(200)
)

base <- ggplot(
```

```
    data = polar,
    mapping = aes(
      x = arc_start,
      y = radius,
      xend = arc_end,
      yend = radius,
      colour = shade,
      size = size
    )
  ) +
    coord_polar(clip = "off") +
    scale_y_continuous(limits = c(0, 1), oob = scales::oob_keep) +
    scale_x_continuous(limits = c(0, 1), oob = scales::oob_keep) +
    scale_colour_viridis_c(option = "pink") + #adjusting the scale color
    guides(colour = guide_none(), size = guide_none()) +
    scale_size(range = c(0, 10)) +
    theme_void() +
    theme(panel.background = element_rect(fill = "blue")) #adjusting the background
```

```
Warning in viridisLite::viridis(n, alpha, begin, end, direction, option):
Option 'pink' does not exist. Defaulting to 'viridis'.
```

```
plot(base + with_dither(geom_segment())) #using Floyd-Steinberg dithering
```



0.16 Pixel Filters (mask)

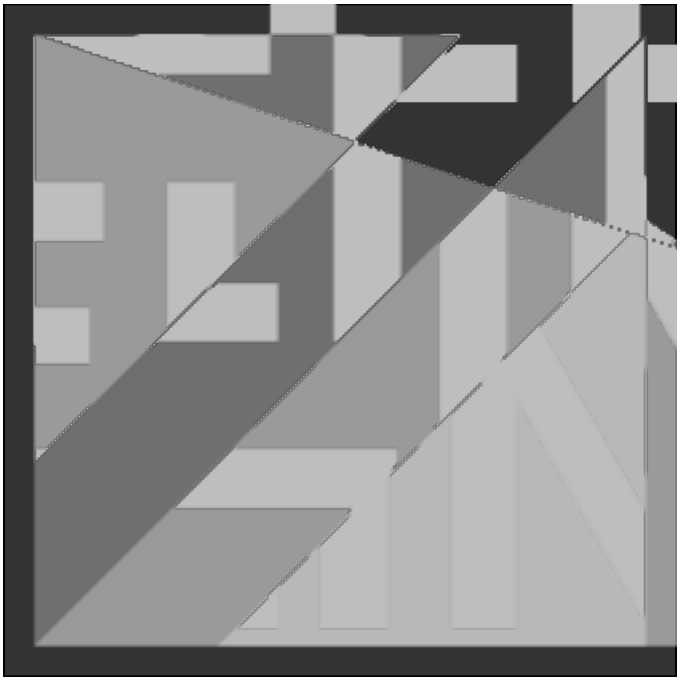
```
library(dplyr)
library(tibble)
library(ggplot2)
library(ggfx)
library(ambient)

texture <- geom_raster(
  mapping = aes(x, y, fill = paint),
  data = long_grid(
    x = seq(from = -1, to = 1, length.out = 1000),
    y = seq(from = -1, to = 1, length.out = 1000)
  ) |>
    mutate(
      lf_noise = gen_simplex(x, y, frequency = 2, seed = 1234),
      mf_noise = gen_simplex(x, y, frequency = 20, seed = 1234),
      hf_noise = gen_simplex(x, y, frequency = 99, seed = 1234),
      paint = lf_noise + mf_noise + hf_noise
    )
)
```

```
hex <- tibble(x = sin((0:6)/6 * 2 * pi), y = cos((0:6)/6 * 2 * pi))
mask <- geom_polygon(aes(x, y), hex, fill = "white")

base <- ggplot() +
  theme_void() +
  coord_equal() +
  scale_x_continuous(expand = c(0, 0)) +
  scale_y_continuous(expand = c(0, 0)) +
  scale_fill_gradientn(
    colours = c("#222222","#e83e8c"),
    guide = guide_none()
  )

border <- geom_path(aes(x, y), hex, colour = "white", size = 15)

text <- geom_text(
  mapping = aes(x, y, label = text),
  dat = tibble(x = 0, y = 0, text = "ART"),
  size = 36,
  colour = "white",
  fontface = "bold"
)

plot(
  base +
    as_group(texture, text, border, id = "content") +
    as_reference(mask, id = "mask") +
    with_mask("content", "mask")
)
```

0.17 Pixel Filters (displace)

```
library(ggplot2)
library(ggfx)

polygon_layer <- function(x, y, fill = "white", alpha = .3) { #adjusting the transparency
  geom_polygon(aes(x, y), fill = fill, alpha = alpha)
}
poly1 <- polygon_layer(x = c(3, 0, 0), y = c(0, 0, 1)) #creating more layers of the polygo
poly2 <- polygon_layer(x = c(0, 1, 1), y = c(0, 0, 1))
poly3 <- polygon_layer(x = c(.3, 2, 1), y = c(0, 0, .7))
poly4 <- polygon_layer(x = c(0, 0, .7), y = c(.3, 1, 1))

base <- ggplot() +
  coord_equal(xlim = c(0, 1), ylim = c(0, 1)) +
  theme_void() +
  theme(panel.background = element_rect(fill = "#333333"))

text <- geom_text(
  mapping = aes(0.5, 0.5, label = "CELINE"), #making my name be displaced
  size = 60,
```

```
  colour = "gray", #adjusting the color of the text
  fontface = "bold"
)

plot(
  base +
    as_group(poly1, poly2, poly3, poly4, id = "polygons", include = TRUE) +
    as_reference("polygons", id = "displacement_map") +
    with_displacement(
      text,
      x_map = ch_alpha("displacement_map"),
      y_map = ch_alpha("displacement_map"),
      x_scale = 150,
      y_scale = -150
    )
)
```



0.18 Pixel filters (blend)

```
library(dplyr)
library(tibble)
library(ggplot2)
library(ggforce)
library(flametree)
library(ggfx)

tree <- flametree_grow(
  seed = 1,
  time = 9,
  angle = c(-15, 20, 30) #adjusting the angle of the tree
)

leaf <- tree |> filter(id_leaf == TRUE)

leaves <- geom_point(
  data = leaf,
  mapping = aes(coord_x, coord_y, colour = seg_col),
  colour = "white",
  size = 5,  #adjusting the size of the trees
  stroke = 0
)

trunk <- geom_bezier(
  data = tree,
  mapping = aes(
    x = coord_x,
    y = coord_y,
    size = seg_wid,
    group = id_pathtree
  ),
  colour = "white",
  lineend = "round"
)

polygon_layer <- function(x, y, fill = "white", alpha = .5) {
  geom_polygon(aes(x, y), fill = fill, alpha = alpha)
}
```

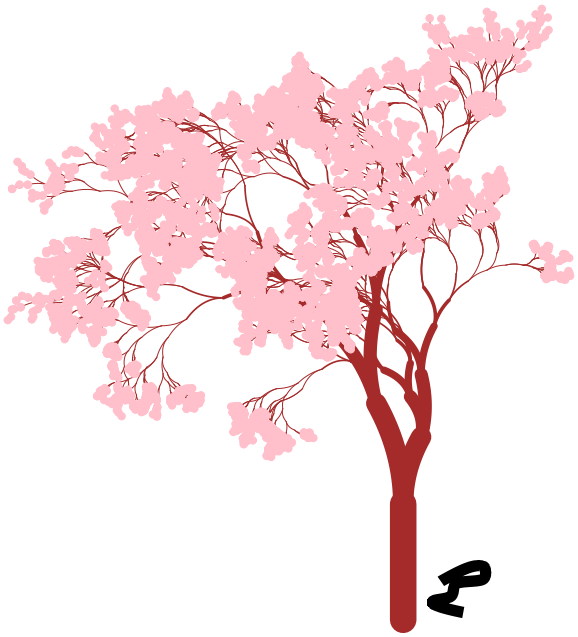
```r
triangle <- polygon_layer(
  x = c(-4, 2, 2),
  y = c(0, 0, 6),
  fill = "white",
  alpha = 3 #adjusting the transparency
)

base <- ggplot() +
  theme_void() +
  theme(panel.background = element_rect(
    fill = "black", colour = "black"
  )) +
  coord_equal(xlim = c(-3, 1), ylim = c(1, 5)) +
  scale_x_continuous(expand = c(0, 0)) +
  scale_y_continuous(expand = c(0, 0)) +
  scale_size_identity(guide = guide_none())

plot(
  base +
    as_group(trunk, leaves, id = "tree") +
    with_blend(triangle, "tree", blend_type = "xor")
)
```

Combining Elements

```
#loading libraries
library(dplyr)
library(tibble)
library(ggplot2)
library(ggforce)
library(flametree)
library(dplyr)
library(purrr)
library(tidyr)
library(tibble)
library(ggplot2)
library(ambient)
library(tictoc)
library(ggthemes)
library(gifski)
```

```
library(ggplot2) #installing libraryes
library(ggforce)
library(e1071)

tree <- flametree_grow(
  seed = 286, #adjusting the randomness of the angle of the tree
  time = 12,
  angle = c(-15, 15, 30), #adjusting the angle of the tree
  split = 2, #adjusting the split of the tree
  trees = 1
)

leaf <- tree |> filter(id_leaf == TRUE) #creating the leaves

base <- ggplot() +  #creaing the base of the tree
  scale_size_identity() +
  theme_void() + #getting rid of the base
  coord_equal()

leaves <- geom_point(
  mapping = aes(coord_x, coord_y),
  data = leaf,
  size = 1.3,  #adjusting the size
```

```
  stroke = 0,
  colour = "pink" #adjusting the color of the leaves to pink
)

trunk <- geom_bezier(
  mapping = aes(coord_x, coord_y, group = id_pathtree, size = seg_wid),
  data = tree,
  lineend = "round", #adjusting the shape of the trunk
  show.legend = FALSE,
  colour = "brown" #adjusting the color of the trunk
)

smooth_loess <- function(x, span) { #creating a smooth curve
  n <- length(x)
  dat <- tibble(time = 1:n, walk = x)
  mod <- loess(walk ~ time, dat, span = span)
  predict(mod, tibble(time = 1:n))
}

smooth_path <- function(n = 1000, smoothing = .7, seed = NULL) {
  if(!is.null(seed)) set.seed(seed)
  tibble(
    x = smooth_loess(rbridge(1, n), span = smoothing), #creating smooth lines
    y = smooth_loess(rbridge(1, n), span = smoothing),
    stroke = 1 #adjusting the stroke of the line
  )
}

path <- smooth_path(seed = 50) #adjusting the randomness of the path

combined_plot <- base + trunk + leaves + geom_path(data = path, aes(x, y), colour = "black

plot(combined_plot)
```



0.19 Reflection

This whole process was relatively new to me. I thought that it was interesting how you could create all different types of artwork from R. I also thought what was interesting was that you could use data points and distort them into pieces of artwork. I also didn't know that R had built-in functions such as flame-tree that help to create artwork. What was familiar to me was when I would adjust the plots. Using functions such as scale_y_continuous was familiar to me as I had used these functions when graphing plots as well. Thus, I learned a lot from this CHOYOA. I learned of many new functions I could use to help create artwork in Quarto Markdown. I also learned of how to manipulate data in order to create different types of visualizations.