Exploring weavable patterns using matrices

David O'Sullivan

2021-10-29

```
library(pracma)
library(tidyr)
library(dplyr)
library(sf)
library(tmap)
library(raster)
library(RColorBrewer)
```

This function makes a random pattern (as a matrix of values) from two provided list of (distinct) integers. Default values will make a 4 x 4 repeating unit.

The `repeat_warp` and `repeat_weft` parameters will specifying how many times across and down respectively to repeat the pattern.

```
make_random_pattern <- function(warp = seq(1, 24, 6), weft = seq(27, 50, 6),
                                repeat_warp = 4, repeat_weft = 4) {
  size <- length(warp) * length(weft)
  mat <- matrix(sample(0:1, size, replace = TRUE), 4, 4)
  treadling <- diag(length(weft)) %>%
    # matrix(sample(0:1, size, replace = TRUE),
    #                   nrow = length(weft), ncol = length(warp)) %>%
    repmat(m = repeat_weft, n = 1)
  threading <- diag(length(warp)) %>%
    # matrix(sample(0:1, size, replace = TRUE),
    #                   nrow = length(weft), ncol = length(warp)) %>%
    repmat(m = 1, n = repeat_warp)
  ww <- threading %*% mat %*% treadling > 0
  fabric <- matrix(0, nrow = nrow(ww), ncol = ncol(ww))
  for (i in seq_along(weft)) {
    rows <- seq(i, nrow(ww), length(weft))
    idxs <- which(ww[rows, ])
    fabric[rows, ][idxs] <- weft[i]
  }
  for (i in seq_along(warp)) {
    cols <- seq(i, ncol(ww), length(warp))
    idxs <- which(!ww[, cols])
    fabric[, cols][idxs] <- warp[i]
  }
  return(fabric)
}
```

This function makes an `sf` polygon from a list of coordinate supplied as an (x1, y1, x2, y2, ...) list.

```
# convenience function to make a sfc POLYGON from a vector
# of points as c(x0,y0,x1,y1,...xn,yn) - note not closed
# this function will close it
get_polygon <- function(pts) {
  mpts <- matrix(c(pts, pts[1:2]), ncol = 2, byrow = TRUE)
  return(st_polygon(list(mpts)))
}
```

A function to translate a polygon by the supplied (x, y). The ordering of parameters facilitates use in an `lapply`.

```
shift_poly <- function(pt, poly) {
  return(poly + pt)
}
```

Function to make an `sf` dataset from the supplied matrix. `w` specifies the size of the grid elements in the units of the provided CRS.

```
make_polygons_from_matrix <- function(width, mat, crs) {
  dxdy <- expand_grid(x = 0:(ncol(mat) - 1),
                      y = 0:(nrow(mat) - 1)) %>%
    mutate(x = width * x, y = width * y) %>%
    # next steps convert to a list of coordinate pairs
    t() %>%
    as.data.frame() %>%
    as.list()
  # base polygon centred on (0, 0)
  p <- get_polygon(width / 2 * c(-1, -1, 1, -1, 1, 1, -1, 1))
  # the ids from the supplied matrix (these should be integers)
  ids <- c(mat)
  return(
    # apply the dxdy offsets to the base polygon
    lapply(dxdy, shift_poly, poly = p) %>%
      # and convert to an sf dataset
      st_as_sfc() %>%
      st_sf() %>%
      st_set_crs(crs) %>%
      # add in the id attribute and dissolve
      mutate(id = ids) %>%
      group_by(id) %>%
      dplyr::summarise()
  )
}
```

In use

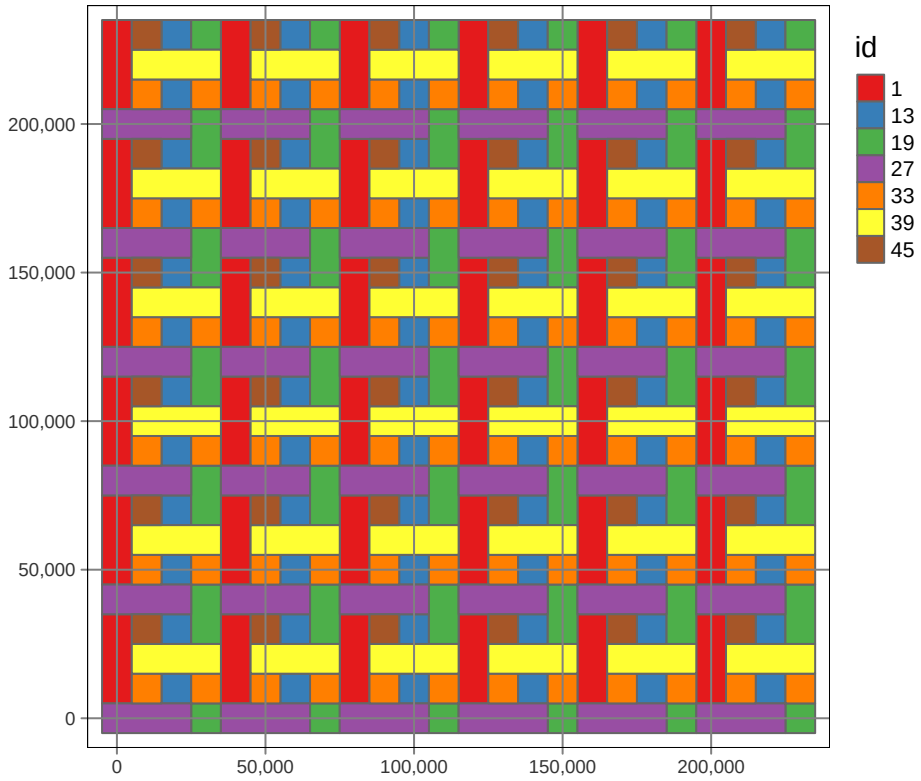
As a vector map

```
tmap_mode("plot")
```

```
## tmap mode set to plotting
```

```
make_polygons_from_matrix(
  10000, # big enough to see on a web map!
  make_random_pattern(repeat_warp = 6, repeat_weft = 6), 3857) %>%
    tm_shape() +
```

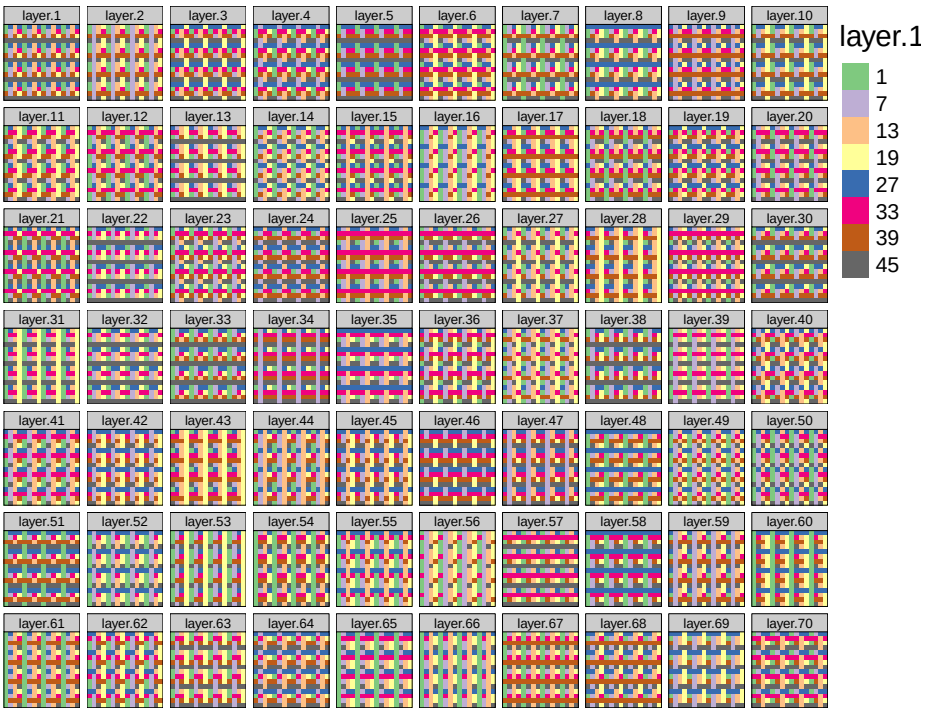
```
    tm_polygons(col = "id", style = "cat", palette = "Set1") +
    tm_grid() +
    tm_layout(legend.outside = TRUE)
```



As a raster stack

It's nice to see a bunch all at once. It's easy enough to use `image()` to plot the patterns, but it's a pain to arrange them in a grid. Handled as a raster stack `tmap` does the job nicely!

```
layers <- list()
for (i in 1:70) {
  layers[[i]] <- raster(make_random_pattern(), crs = st_crs(3857)$wkt)
}
r <- stack(layers)
tm_shape(r) +
  tm_raster(palette = "Accent", style = "cat")
```


layer.1
layer.2
layer.3
layer.4
layer.5
layer.6
layer.7
layer.8
layer.9
layer.10
layer.11
layer.12
layer.13
layer.14
layer.15
layer.16
layer.17
layer.18
layer.19
layer.20
layer.21
layer.22
layer.23
layer.24
layer.25
layer.26
layer.27
layer.28
layer.29
layer.30
layer.31
layer.32
layer.33
layer.34
layer.35
layer.36
layer.37
layer.38
layer.39
layer.40
layer.41
layer.42
layer.43
layer.44
layer.45
layer.46
layer.47
layer.48
layer.49
layer.50
layer.51
layer.52
layer.53
layer.54
layer.55
layer.56
layer.57
layer.58
layer.59
layer.60
layer.61
layer.62
layer.63
layer.64
layer.65
layer.66
layer.67
layer.68
layer.69
layer.70
layer.1
1
7
13
19
27
33
39
45