Glyphs and pixel-oriented glyphs for data visualization in R

By

Jiahua Liu

A research paper presented to the

University of Waterloo

In partial fulfillment of the requirements for the degree of

Master of Mathematics

in

Statistics

Waterloo, Ontario, Canada, *2017*

Abstract

Glyphs are an important way to display multidimensional data and many different glyphs have been proposed in the literature. This paper reviews a number of these and provides functions to create arbitrary glyphs as pngs or pixmaps. Additional methods for constructing pixel-oriented glyphs for very high dimensional data are reviewed. We build an R package called `glyphs` which implements all these. The package's functionality is illustrated on a variety of datasets.

Keywords: glyph, data visualization, pixel-oriented technique, space-filling curve, multidimensional data,

1 Introduction

According to Matthew O. Ward (2008) [1], "A glyph is the visual representation of a piece of data where the attributes of a graphical entity are dictated by one or more attributes of a data record, in the context of data visualization". The glyphs are very useful to visualize multidimensional data since we will run out of axes when the data is beyond three dimensions.

The use of glyphs dates back to at least 1863 and Francis Galton made use of the weather symbol to analyze the weather data in United Kingdom [2]. In 1981, John W. Tukey and P.A. Tukey summarized the types of characters that had been used as glyphs before [3]. Those are weathervane symbols [4], Anderson's glyphs [5], circle-based glyphs with water level , trees or castles [6], schematic cells, inscribed triangles and cartoon faces [7]. (See Figure 1.1)

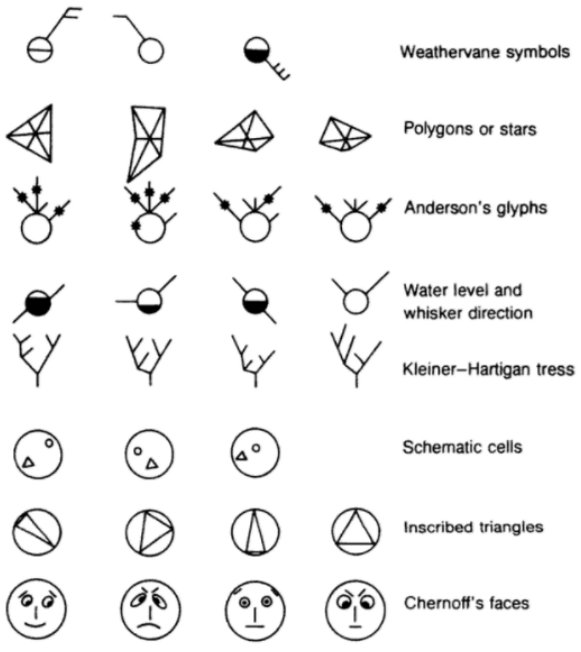

Exhibit 25.3. Several possible individual-value compound character scales.

Figure 1.1: Glyphs summary before 1981

Wickham, Hofmann, Wickham, and Cook treated time series as glyphs to analyze the temperature over Central America in 1995-2000 [8]. They also implemented it in a package called *GGally* in R [9]. (See Figure 1.2)

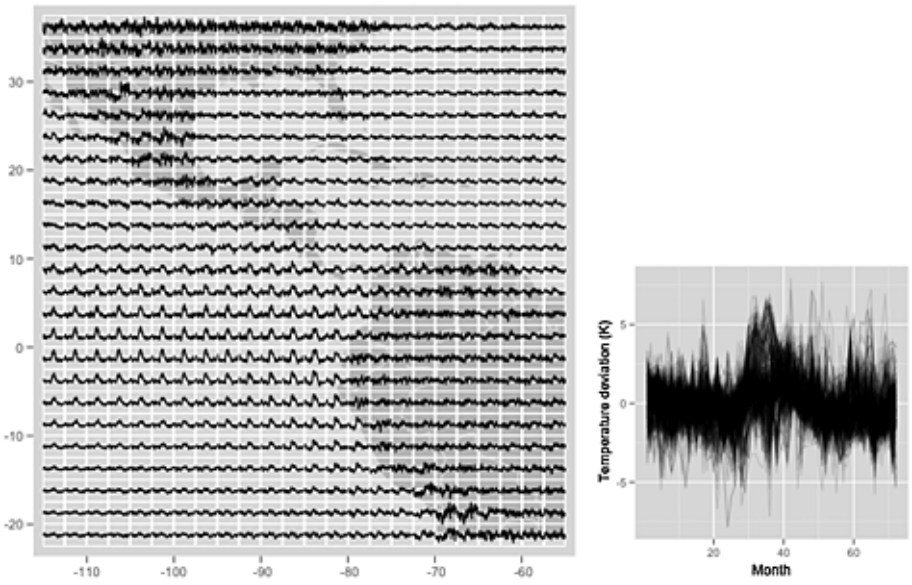

Figure 1.2: Time series glyphs

From above, we can see that glyphs can be very flexible when they are used to visualize multidimensional data. Therefore, given any kinds of drawing methods we can use them to make glyphs and draw the glyphs on a plot.

However, when the number of dimensions in each glyph becomes much higher many previous kinds of glyphs will not work. Therefore, we would like to introduce the pixel-oriented glyphs.

The basic idea of pixel-oriented method is to map each data value to a colored pixel and display them in a plot [10]. It can handle multiple dimensions within each glyph and have many advantages over others. (See Figure 1.3)

An `R` package `glyphs` is made to implement all the ideas above. It offers functionality to make plots given any kind of drawing functions including scatter plot, histogram and so on. It can also make pixel-oriented glyphs with the method reviewed by Keim [10].

In Section 2, we will deal with the layout problem when there are many dimensions in each glyph. We will also talk about the color scale which we may use. Sections 3 gives the descriptions and examples of the `glyphs` package for constructing the plot. In section 4, we explore the use of our functions in various data sets. Lastly, we will give a final remark at the end.

2 High dimensions layout

The original idea of glyphs is to deal with few dimensions for each individual. For example, the Chernoff faces can only deal with around 20 dimensions since the facial features are limited [7]. However, if we

		Clust-ering	multi-variate hot spot	no. of vari-ates	no. of data items	cate-gorical data	visual over-lap	learn-ing curve
Geometric Techniques	Scatterplot Matrices	++	++	+	+	-	o	++
	Landscapes	+	+	-	o	o	+	+
	Prosection Views	++	++	+	+	-	o	+
	Hyperslice	+	+	+	+	-	o	o
	Parallel Coordinates	o	++	++	-	o	--	o
Icon-based Techniques	Stick Figures	o	o	+	-	-	-	o
	Shape Coding	o	-	++	+	-	+	-
	Color Icon	o	-	++	+	-	+	-
Pixel-oriented Techniques	Query-Independent	+	+	++	++	-	++	+
	Query-Dependent	+	+	++	++	-	++	-
Hierarchical Techniques	Dimensional Stacking	+	+	o	o	++	o	o
	Treemap	+	o		o	++		o
	Cone Trees	+	+	o	+	o	+	+
Graph-based Techniques	Basic Graphs	o	o	-	+	o	o	+
	Specific Graphs	++	+	-	+	o	+	+

Fig. 2. An attempt at comparing multidimensional visualization techniques (++: very good, +: good, o: neutral, -: bad, –: very bad).

Figure 1.3: Comparisons between different multidimensional visualization techniques from Keim [10]

treat the idea of glyphs more generally, many previous glyphs will not meet our needs. Considering the data sets of Figure 1.2, for each location in the grid we could not use, say, Chernoff faces or star plots to visualize the data. Pixel-oriented glyphs is an ideal solution for the problem.

The idea of pixel-oriented method is to map data values to colored pixels and present them in a plot. Thus, it is crucial to determine the order of layout in a plot.

2.1 Layout colors

In this section, we will talk about three ways to layout colors in a plot. The first two are motivated by space-filling curves and the last is introduced by Keim [11].

2.1.1 Space-filling curves layout

Space filling curve is a curve whose range contains the whole 2-D unit square (more generally a hyper cube). They provide a good way to display the colors in a plane.

Hilbert curve [12] and Morton curve [13] are typical examples and are used in our `glyphs` package. (See Figure 2.1, 2.2) They are better than the trivial "snake curve" in a sense that they preserve the locality of the points, which means as the order increases, the location change becomes smaller.

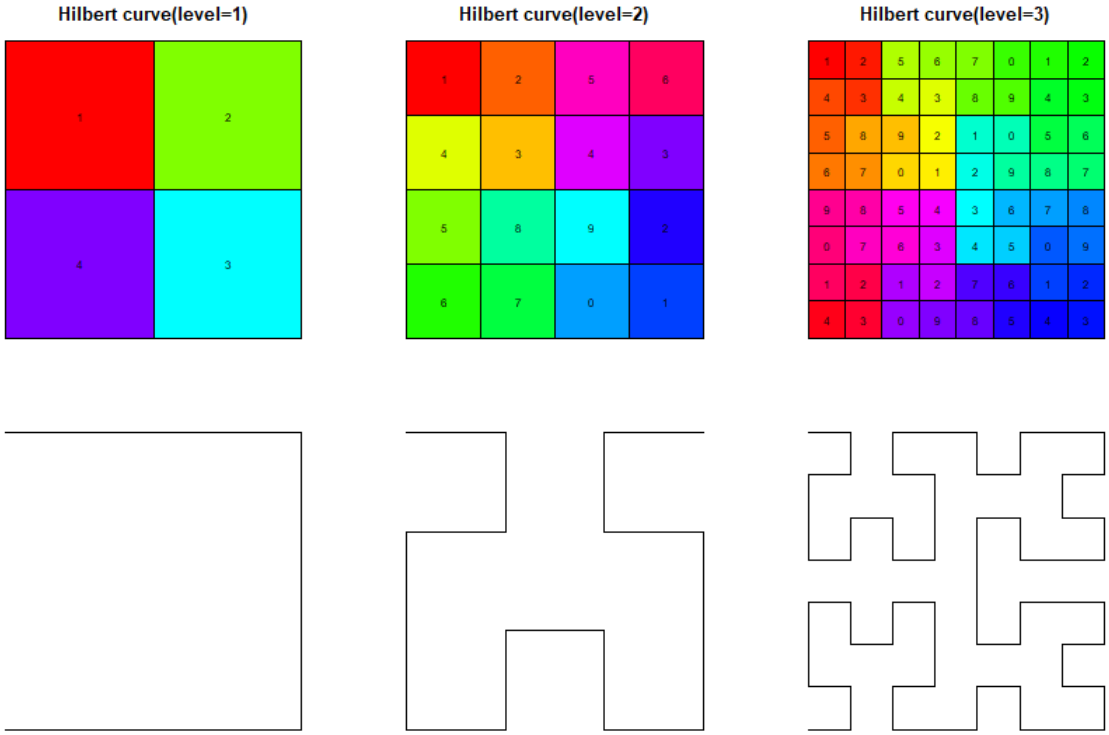

Figure 2.1: Hilbert curve layout

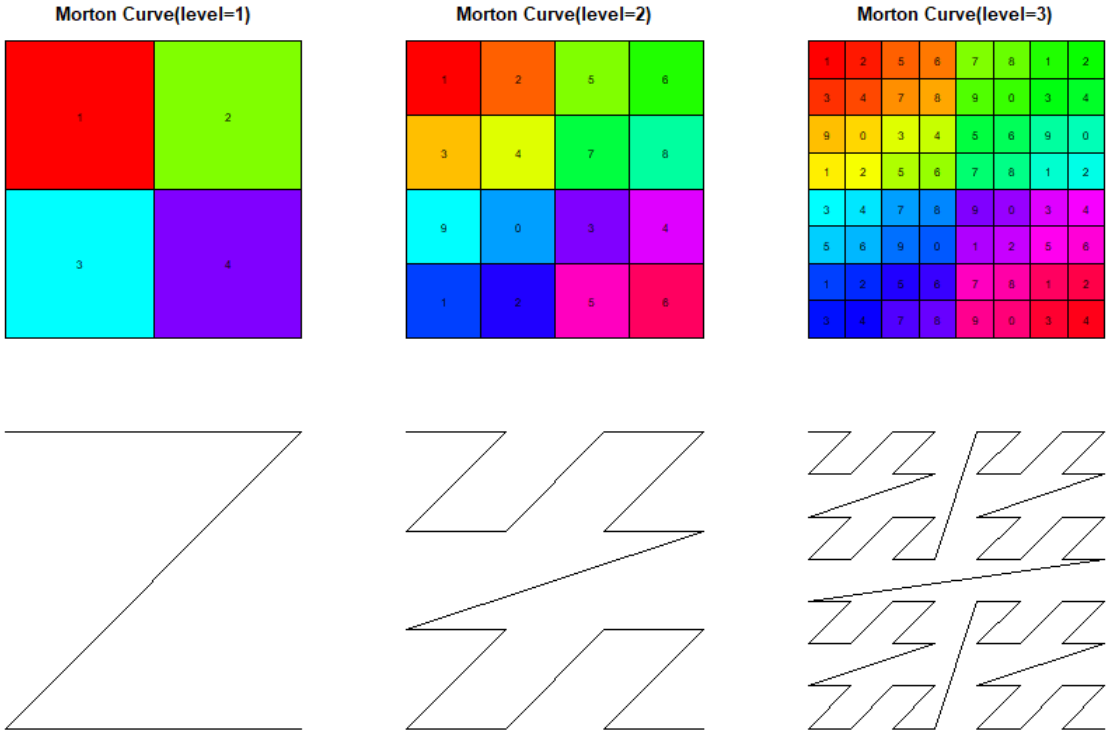

Figure 2.2: Morton curve layout

One can see the order of each layout method from the flow of the color, the number inside each square and the line graph shown in the bottom. The plot starts from the top left corner and the sequence of numbers is module by 10.

2.1.2 Keim's recursive pattern visualization layout

Keim's technique is based on a general recursive scheme using lower-level patterns as a building blocks for higher-level patterns [10]. The method is useful when dealing with data with some inherent structure such as time series which can be reflected by the patterns. (See Figure 2.3)

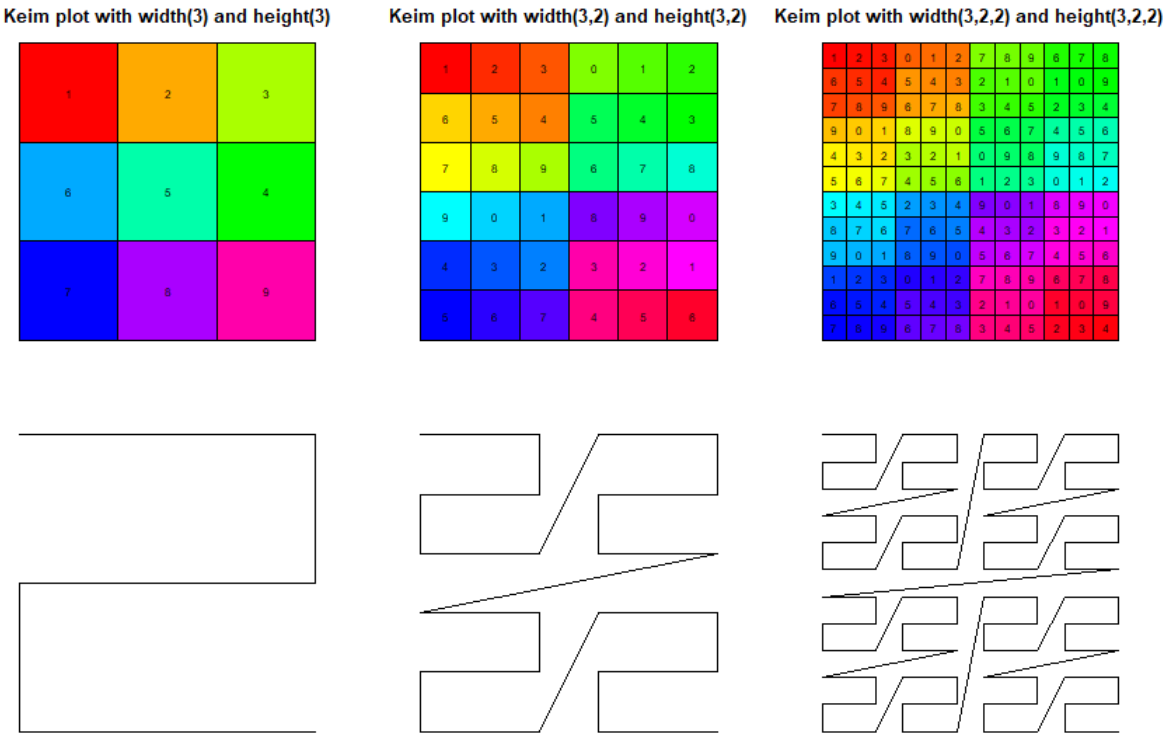

Figure 2.3: Keim's layout

The plot also starts from the top left corner and the sequence of numbers is module by 10. Same as before, one can see the pattern shown on the plot. It is drawn recursively and only the first level is in an "snake" order, and the rest start from the left end.

Since time order is arranged like this way, the method will be preferred to analyze the time series data.

2.2 Color scales

Once the color layout is determined, we need to choose a color scale which the data can map to. The reason why we use the color scale but not grey scale is because color scale has higher Just Noticeable Differences (JNDs) [14].

One good choice is the diverge color palettes which can be produced by `diverge_hcl`, `diverge_hsv` in package `colorspace`. (See Figure 2.4) It can make the comparisons between high and low values easier to see. In addition, it can also give a sense of how far a value is from the center point.

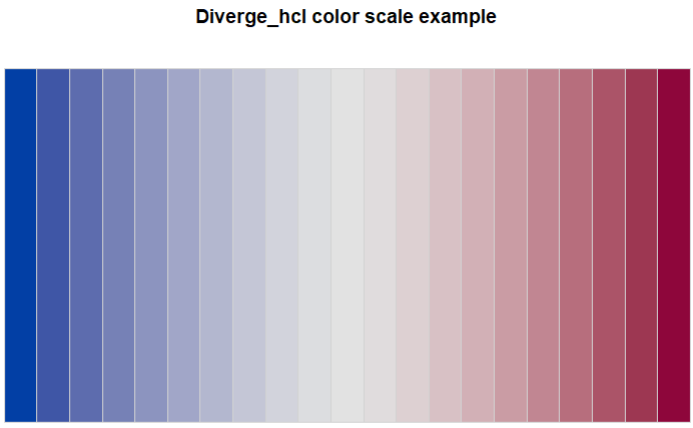

Figure 2.4: Diverge color palettes

Another suggestion by Keim is to use linear interpolation in "hsi" color space [15]. The basic idea is to let saturation, say, (in "hsi" model) remain the same and do a linear interpolation on hue and intensity with the values we set. The "hsi" color space is a variation of the "hsv" color space but it preserves the monotonicity of the brightness better. (See Figure 2.5)

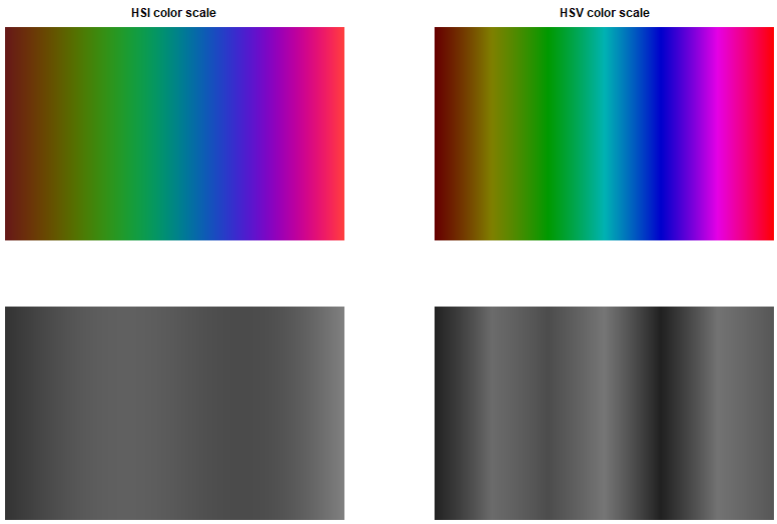

Figure 2.5: Comparisons between hsi and hsv color scale

3 The `glyphs` package

In this section, we will give a brief description for each function in the package and provide an example. A more detailed description is in the Appendix which is the `help()` of each function. One can also see the `vignettes` in the package for more information.

The data used in the following illustration is from the `nasa` data. More detailed information will be in the Data analysis section. In this illustration, we will randomly select 9 of them in the data list by SRSWOR. Each of them is time series data recorded monthly during a span of 6 years. The code is as follows:

```
library(dplyr)
# get the temperature data and organize the data into a list
# each element in the list is corresponding to a longitude and latitude
data_temperature <- list()
for(i in 1:(24*24)){
  col_number <- (i-1) %% 24 + 1
  row_number <- (i-1) %/% 24 + 1
  data_temperature[[i]] <- as.vector(nasa$mets$temperature[col_number,row_number,,])
}
# get 9 of them in the data list
set.seed(1000)
# randomly select by SRSWOR
sample_loc <- sample(length(data_temperature), 9)
temperature_sample <- data_temperature[sample_loc]
```

3.1 The `make_glyphs_draw` function

Given any kinds of drawing function, the `make_glyphs_draw` function can make a corresponding list of glyphs in one of four different classes which are "png", "jpeg", "pixmap" and "tiff".

3.1.1 Histogram glyphs

The following code will generate histogram glyphs.

```
library(glyphs)
glyphs_hist <- make_glyphs_draw(data = temperature_sample,
                                draw_fun = function(data_i){
                                hist(data_i,main = "",
                                 axes = FALSE, col = "steelblue")
                                 },
                                 type = "png", mar = rep(1, 4),
                                 width = 100,height = 100)
str(glyphs_hist[1:3]) #present the first three elements
```

The elements are in the form of matrix:

```
## List of 3
## $ : num [1:100, 1:100, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
## $ : num [1:100, 1:100, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
## $ : num [1:100, 1:100, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
```

3.1.2 Boxplot glyphs

The code to generate boxplot glyphs is given below.

```
glyphs_boxplot <- make_glyphs_draw(data = temperature_sample,
                                   draw_fun = function(data_i){
                                   boxplot(data_i,main = "",
                                    axes = FALSE, col = "steelblue")
                                    },
                                    type = "png", mar = rep(1, 4),
                                    width = 100,height = 100)
```

3.2 The `getGridXY`, `plot_glyphs` functions

The `getGridXY` function will get the x and y coordinates of where we need to plot. The `plot_glyphs` will plot the glyphs according to the coordinates given. More information about the two functions is provided in the Appendix.

Then, we can plot the previous result on a plane. (See Figure 3.1, and Figure 3.2) We can see the different trend between different locations.

3.2.1 Histogram glyphs plot

```
n <- length(temperature_sample)
x <- getGridXY(n) # get the coordinates to plot
# plot it on a plane
plot_glyphs(x, glyphs = glyphs_hist, glyphWidth = 0.8, glyphHeight = 0.6,
            axes = FALSE, xlab = "", ylab = "", main = "Histogram glyphs")
```

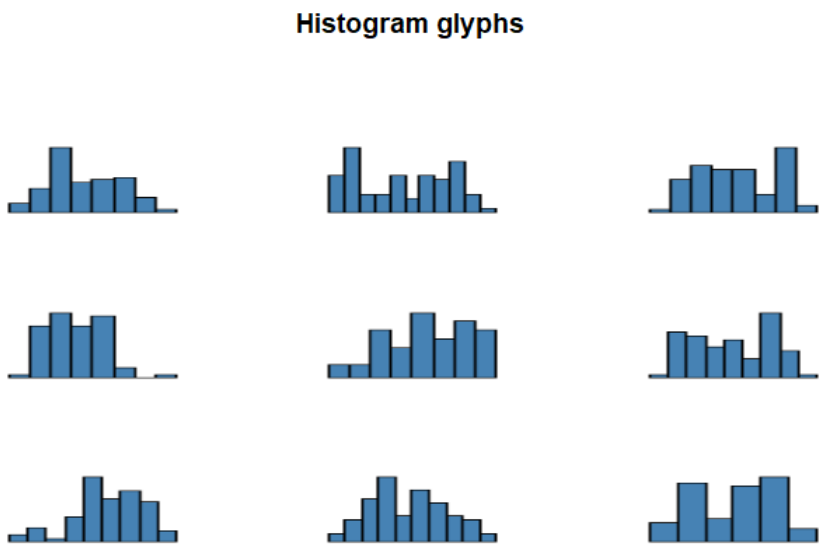

Figure 3.1: Histogram glyphs

3.2.2 Boxplot glyphs plot

```
n <- length(temperature_sample)
x <- getGridXY(n) # get the coordinates to plot
# plot it on a plane
plot_glyphs(x, glyphs = glyphs_boxplot, glyphWidth = 0.8, glyphHeight = 0.6,
            axes = FALSE, xlab = "", ylab = "", main = "Boxplot glyphs")
```

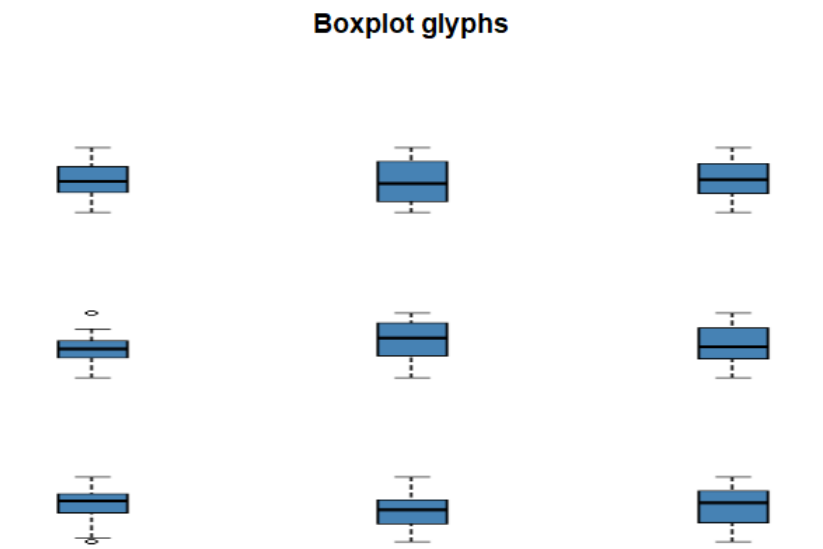

Figure 3.2: Boxplot glyphs

3.3 The `hsi2rgb` function

The `hsi2rgb` transforms colors from HSI space into RGB space. (See Figure 3.3) This is the default color scale that the package use.

```
x=c(0,360)
y=c(0.4,1)
f=approxfun(x,y) # linear introploation function
l <- 100000 # number of colors
H=seq(x[1],x[2],length.out = l)
I=f(H)
S=rep(1,length(H))
cols=rgb(t(hsi2rgb(H,S,I)),maxColorValue = 255)
barplot(rep(1,length(H)),col = cols,border = NA,beside = FALSE,
space = c(0,0), axes = FALSE, main="HSI color scale")
```

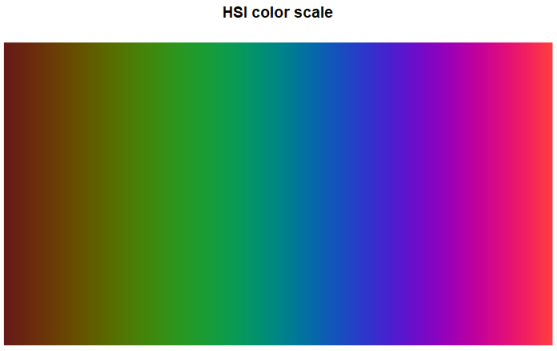

Figure 3.3: HSI color scale

3.4 The `data2col` function

The `data2col` provides mapping from data values to color values by using hsi color space. We first rescale the data to [0,1]. Then, suppose we have n colors in the color scale, we multiply each data by n and take ceiling function of it. Lastly, the number we finally get, say k, will be the kth colors in the color scale.

We can also set the range of data that we are interested in and the origin which is mapping to the center of the color scale as possible parameters. All the data outside the range will be set a unique different color. In this example, smaller data will be light grey and larger will be dark grey. The data within the range will use the mapping mentioned above adjusted by the origin. (See Figure 3.4)

```
pal <- function(col, border = "light gray", ...)
{
    n <- length(col)
    plot(0, 0, type="n", xlim = c(0, 1), ylim = c(0, 1),
         axes = FALSE, xlab = "", ylab = "", ...)
    rect(0:(n-1)/n, 0, 1:n/n, 1, col = col, border = border)
} # a plot function in examples of rainbow_hcl{colorspace}
cols_1 <- data2col(data = 1:10)
cols_2 <- data2col(data = 1:10, xLow = 3, xHigh = 8,
                   outRangeCol = c("grey60", "grey20"))
cols_3 <- data2col(data = 1:10, xLow = 3, xHigh = 8, origin = 6,
                   outRangeCol = c("grey60", "grey20"))
par(mfrow = c(1,3))
pal(cols_1, asp = 1)
title("Default setting", line = -6)
pal(cols_2, asp = 1)
title("Set the range of data in [3, 8]", line = -6)
pal(cols_3, asp = 1)
title("Set the range of data in [3, 8] \n add origin = 6", line = -6)
```

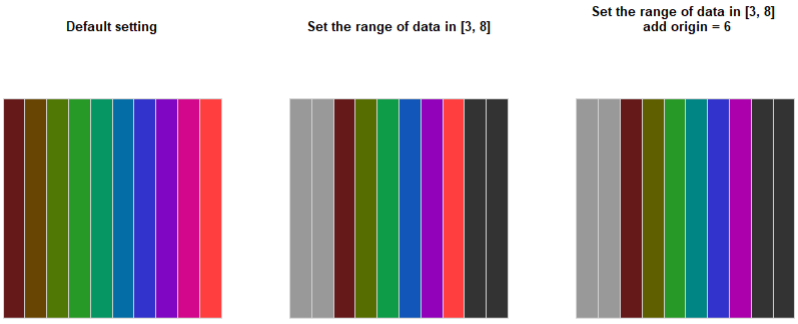

Figure 3.4: The `data2col` examples

3.5 The `HilbertGlyph`, `MortonGlyph`, `KeimGlyph` functions

These three functions generate data matrix of png format or class of "pixmap" which can be used to display color strings in the corresponding layout method. (See Figure 3.5)

```
par(mfrow = c(1,3))
# get colors
cols <- data2col(temperature_sample[[1]])
# Hilbert glyph
myPngmat <- HilbertGlyph(cols) # get a glyph of data matrix in png format
# background plot
plot(0,type='n', xlim=c(0,1), ylim=c(0,1), axes = FALSE,
     xlab = "", ylab = "", asp = 1)
title("HilbertGlyph draw", line = -4)
rasterImage(myPngmat,0,0,1,1) # plot the glyph
# Morton glyph
myPngmat <- MortonGlyph(cols)
plot(0,type='n', xlim=c(0,1), ylim=c(0,1),
     axes = FALSE, xlab = "", ylab = "", asp = 1)
title("MortonGlyph draw", line = -4)
rasterImage(myPngmat,0,0,1,1)
# Keim glyph
myPngmat <- KeimGlyph(cols, width = c(1,12,1), height = c(1,1,6))
plot(0,type='n', xlim=c(0,1), ylim=c(0,1), asp = 1,
     axes = FALSE, xlab = "", ylab = "")
title("KeimGlyph draw", line = -4)
rasterImage(myPngmat,0,0,1,1)
```

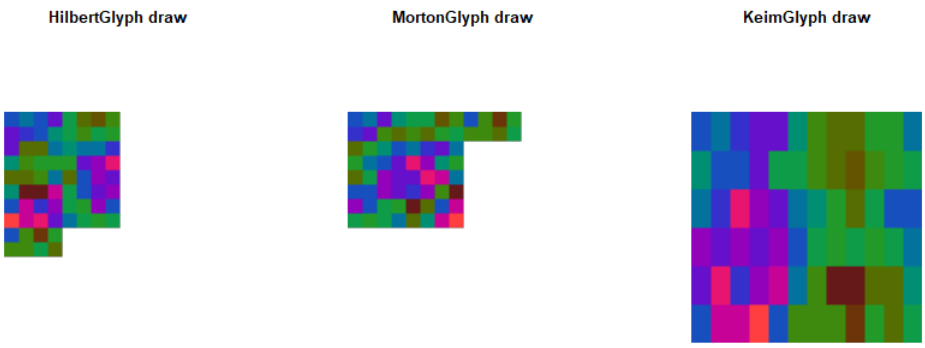

Figure 3.5: Glyph drawing examples

The order is the same as what we illustrated in the section 2.

3.6 The `make_glyphs` function

Given a list of data vectors and a function of data vectors to colors, the `make_glyphs` function can make a corresponding list of glyphs in png format or a class of pixmap.

Similar to `make_glyphs_draw`, we can use `getGridXY` and `plot_glyphs` to plot the glyphs on a plane. All three layout methods are shown in Figure 3.6.

For each glyph in the KeimGlyph drawing, each row represents a year while each column represents month, so it can make an accurate comparison between each month and year. Also, we can see the trend of data in each glyph. However, HilbertGlyph and MortonGlyph drawing do not have these good properties. Therefore, if the data is in time order, it could be better to make glyphs by Keim's method.

```
glyphs <- make_glyphs(data = temperature_sample, type = "pixmap", origin = "mean")
x <- getGridXY(length(temperature_sample)) # get the x and y coordinates to plot
# plot glyphs
plot_glyphs(x, glyphs = glyphs, asp = 1,
            glyphWidth = 0.8, glyphHeight = 0.6,
            type = "pixmap", axes = FALSE, xlab = "", ylab = "")
title("HilbertGlyph drawings", line = -5)
glyphs <- make_glyphs(data = temperature_sample, glyph_type = "Morton",
                      type = "pixmap", origin = "mean")
x <- getGridXY(length(temperature_sample)) # get the x and y coordinates to plot
# plot glyphs
plot_glyphs(x, glyphs = glyphs, asp = 1,
            glyphWidth = 0.8, glyphHeight = 0.6,
            type = "pixmap", axes = FALSE, xlab = "", ylab = "")
title("MortonGlyph drawings", line = -5)
glyphs <- make_glyphs(data = temperature_sample,
                      glyph_type = "Keim", width = c(1,12,1), height = c(1,1,6),
                      type = "pixmap", origin = "mean")
x <- getGridXY(length(temperature_sample)) # get the x and y coordinates to plot
# plot glyphs
plot_glyphs(x, glyphs = glyphs, asp = 1,
            glyphWidth = 0.8, glyphHeight = 0.6,
            type = "pixmap", axes = FALSE, xlab = "", ylab = "")
title("KeimGlyph drawings", line = -5)
```

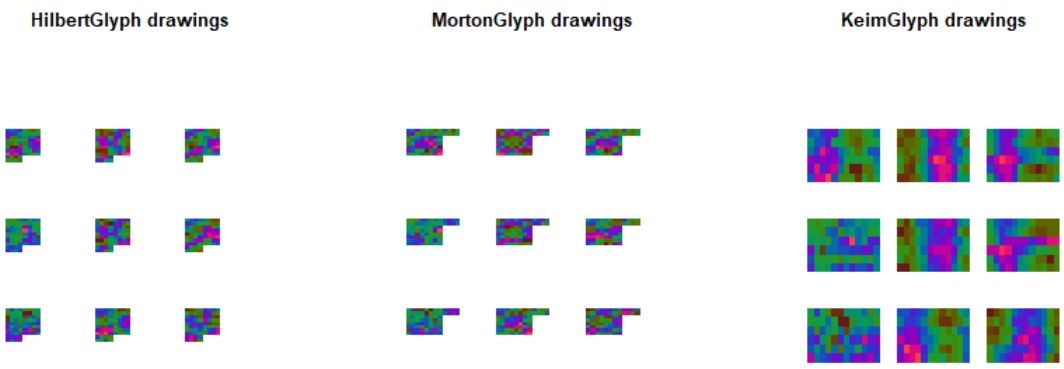

Figure 3.6: The `make_glyphs` examples

4 Data analysis

In this section, we will use the `glyphs` package to analyze three different kinds of data sets. The first is the gene data which does not have inherent order. The other two are time series data. The codes will be provided in the Appendix.

4.1 The `ALL` gene data

The data set is in the `ALL` package in R [16]. The data record microarrays of 128 different individuals with acute lymphoblastic leukemia (ALL). We can use our method to create a plot for an individual's gene. In the plot, each gene expression is scaled over all individuals. (See Figure 4.1)

From the plot, we can get a rough idea of the distribution of one's gene. It can be seen that the distribution varies among the individuals especially the genes colored in blue.

Also, from the information given in the data set, the first 95 individuals have B-cell ALL, and the rest have T-cell ALL. After taking the average of them in each group, we can see their differences. (See Figure 4.2)

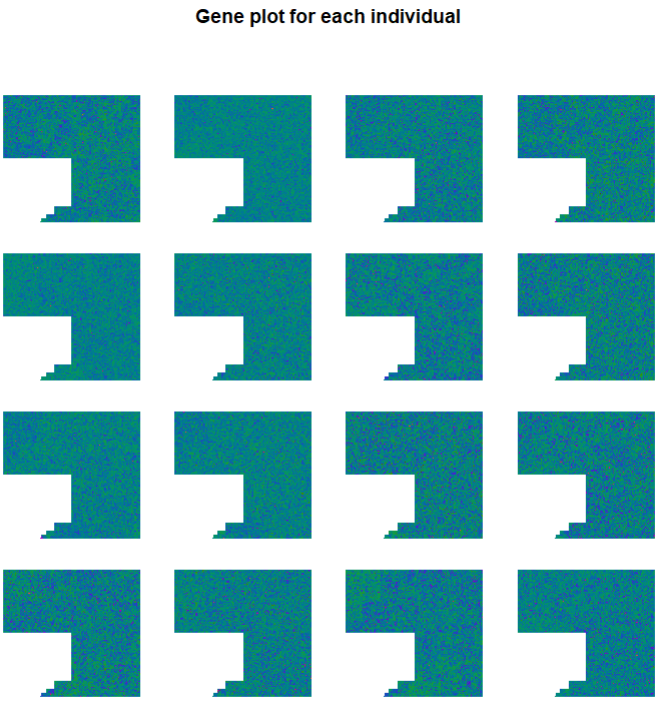

Figure 4.1: The 16 Individuals' gene plot

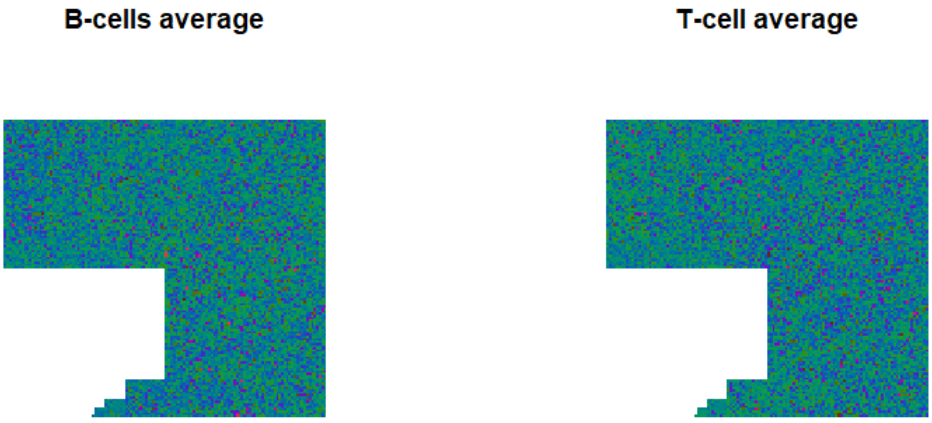

Figure 4.2: B-cell and T-cell ALL comparison

4.2 The `nasa` data

The `nasa` data set is in the `dplyr` package in R [17]. It contains geographic and atmospheric measures on Central America. Temperature, ozone, air pressure, and cloud cover (low, mid, and high) are provided on monthly averages from Jan 1995 to Dec 2000.

It has two variable `mets` and `dims`. The `mets` contains measurements which for each forms a 4-way

matrix. The `dims` provides the corresponding value of each dimension which includes the latitude, longitude, month and year.

Given a specific longitude and latitude, we would like to make a glyph according to the corresponding time series data and then draw them to see if there are some patterns. The analysis will focus on the measurement of temperature.

As we talked about in the description of package, it is better to use Keim's method. For each glyph, the data will be displayed in a 12 by 6 grid, and there will be 6 rows representing 6 years and 12 columns representing 12 months. The color scale we choose is a diverge color scale from blue to red, because blue usually represents low temperature values and red usually represents high.

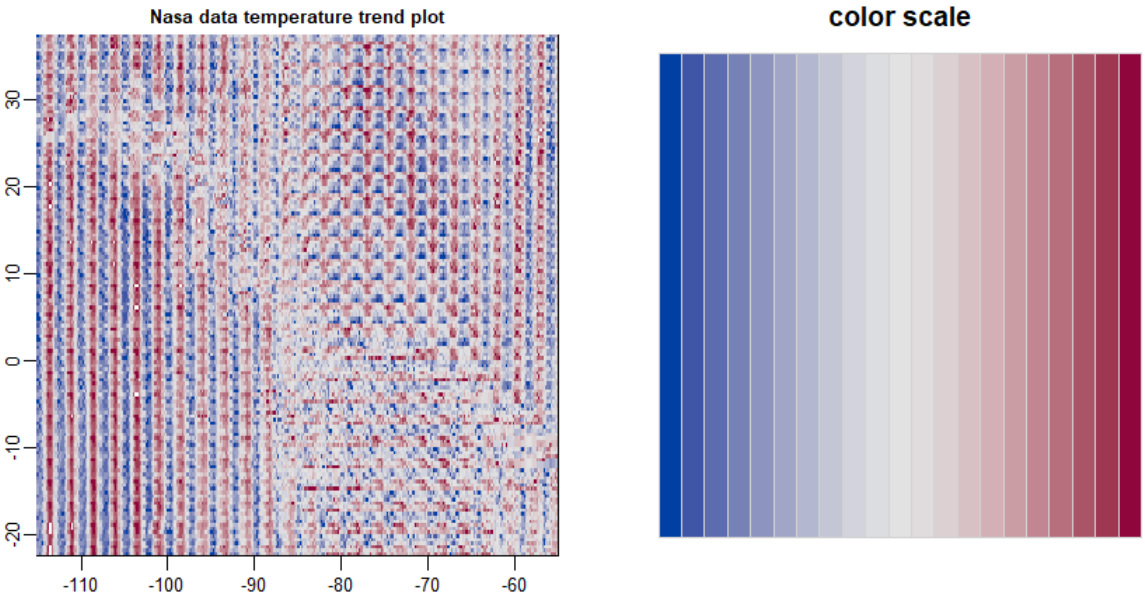

Figure 4.3: The `nasa` trend plot

Figure 4.3 shows the plot. The x-axis and y-axis represent the longitude and latitude.

In the bottom left corner and top right corner, we can see the temperature pattern is very normal and does not change much in these 6 years. However, for the bottom right corner, top left corner and the center, the temperature varies drastically not only between different locations but within each glyph, which suggests that there exists irregular climate changes during these 6 years.

Additionally, we can also draw a map as a background, which can provide more information. (Figure 4.4)

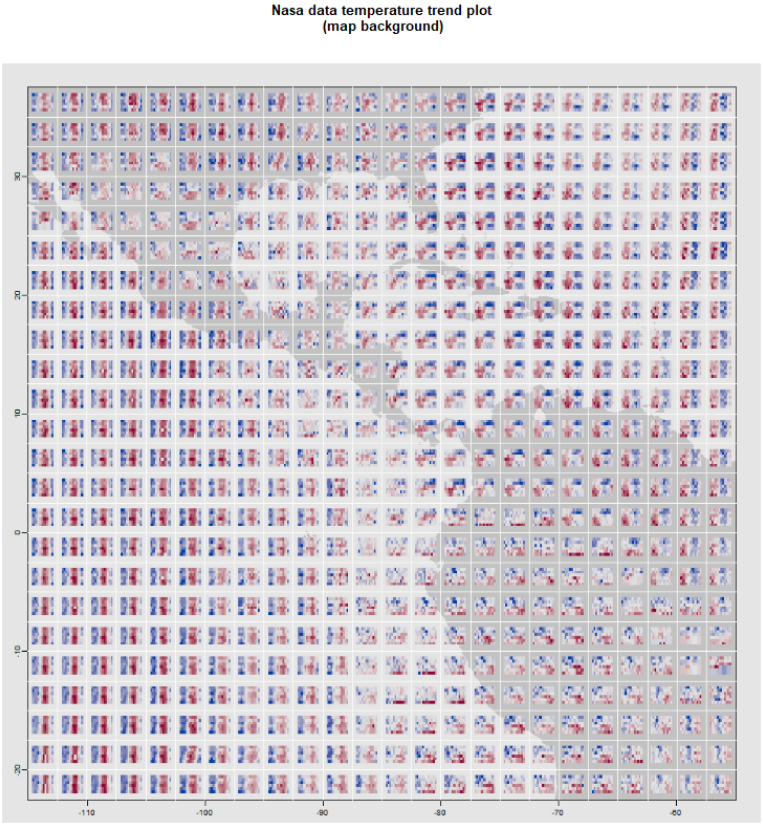

Figure 4.4: The `nasa` trend plot with map background

We can also draw time series glyphs on the map background with common scaling and local scaling. (See Figure 4.5, 4.6)

After adding the map of corresponding area to the plot, it can be seen that most of the regular temperature pattern we detect above is in the sea which is far from the land. The temperature in the land varies much more than in the sea. The most irregular pattern appears at the boundary between land and sea.

In addition, in Figure 4.4 we can see that in the area between land and sea, the temperature varies also between different years. Therefore, in that area we can conclude that an irregular climate change occurred during these years.

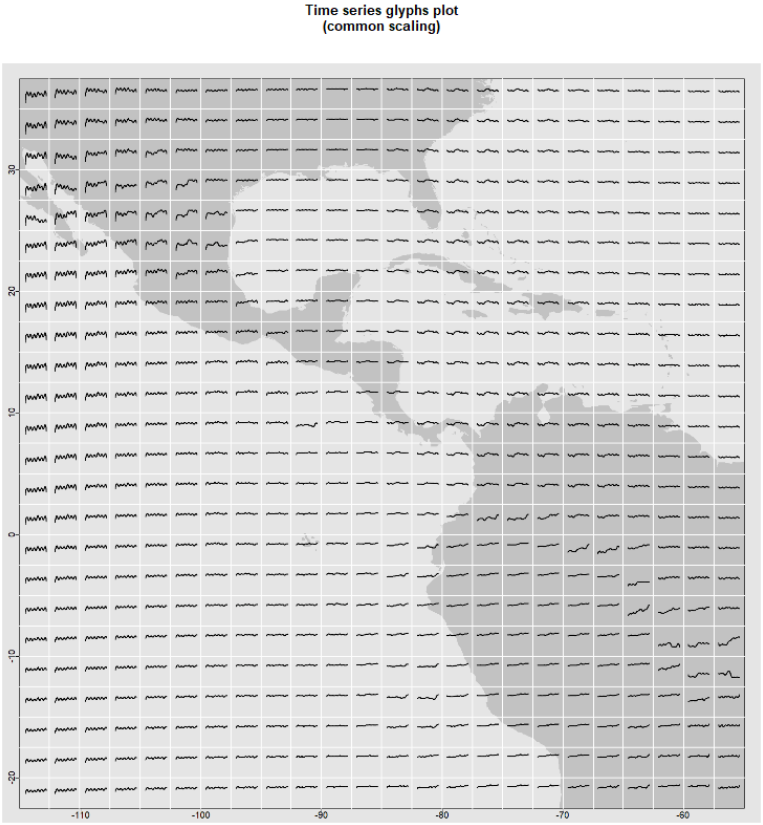

Figure 4.5: Time series glyphs plot with common scaling

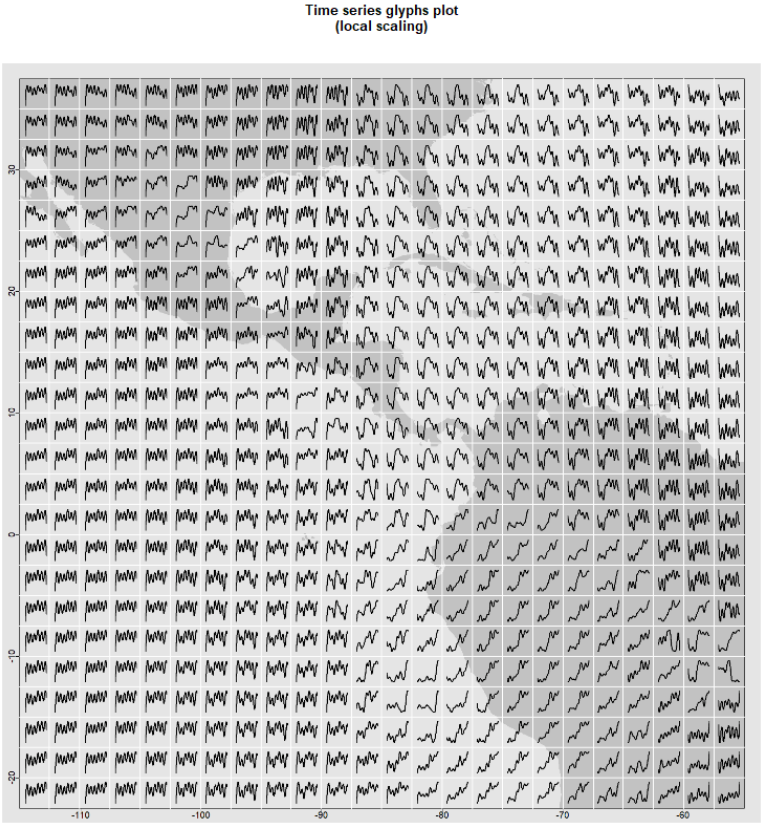

Figure 4.6: Time series glyphs plot with local scaling

4.3 The SP500 data

The SP500 data set comes from qrmdata package in R [18]. It contains S&P 500 constituents stock indices from 1962, when at least one of the constituents is available, to 2015. If the data is not available, it will return missing. The constituents information can be seen in the following website (https://en.wikipedia.org/wiki/List_of_S%26P_500_companies).

The data set also provides the sectors that the stocks belong to. It makes life easier to compare the stocks within common sectors or sub sectors.

In this section, we will focus on the stock indices from 2007 to 2009. Since there is a financial crisis on 2008, we would like to see its impact on the constituents.

Since it is time series data for each stock, Keim's method is preferred. Each day there is only one data value, so the first level of width and height are both 1. On average, 5 days in a week has stock indices, so the second level of width should be 5, and the third level of height should be 1. The cycle continues until we have three years left and the width should be 1 and height 3. Note that because number of weeks varies in each month, the plot will not match the date exactly. However, the approximation is accurate enough.

Because the red represents lower value in stock index, and green represents higher, we want to choose a diverge color palette from red to green. However, many people have red-green blindness, so we change the green color to blue.

4.3.1 General trend

First, we average the stock indices on each day to see the general trend of the stock indices. (See Figure 4.7)

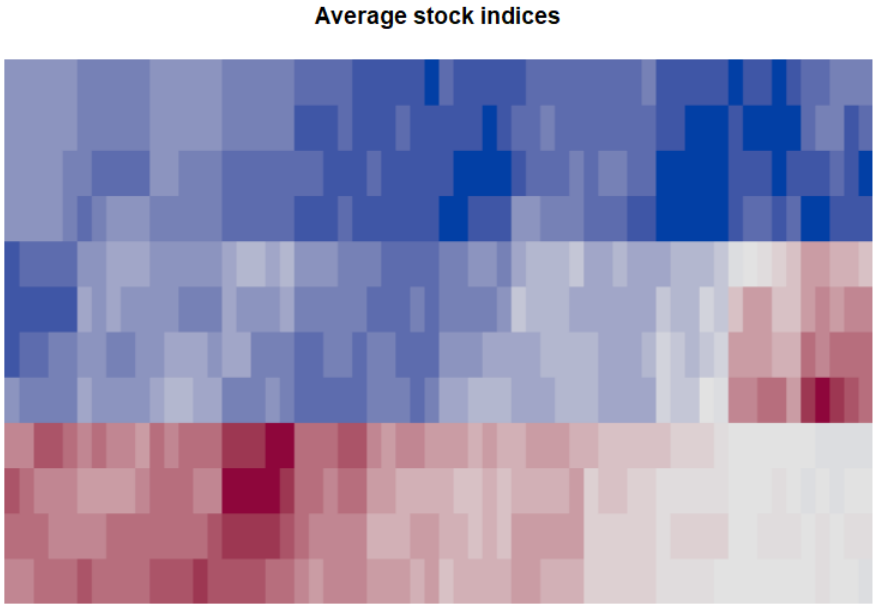

Figure 4.7: SP500 general trend plot

From the plot, the stock indices were relatively high until the last quarter of 2008 and generally had little fluctuation. However, after that the stock indices suddenly went down and lasted almost a year. It recovered a little bit at the end of 2009 but did not return the original status. Therefore, in this plot we can see the dramatic impact of the 2008 financial crisis on the stock indices in the whole stock market.

Then, we want to dig into details about some specific sectors or sub sectors of the stock and see whether they have a similar trend. Some companies may foresee the crisis and do something to prevent the loss or even gain some profit on it. We choose the stocks in `Bank` sector and `Investment banking & Brokerage` sector as an example.

4.3.2 Bank stocks

The stock tickers in Bank sector are listed below.

```
## [1] "BAC"  "BK"   "BBT"  "C"    "CMA"  "FITB" "HBAN" "JPM"  "KEY"  "MTB"  "PNC"
       "STI"  "USB"  "WFC"  "ZION"
```

In total there are 15 stocks. We also calculate the average over all banks and plot it in the last glyph. (See Figure 4.8) The plot is in local scale. When we use common scale, the stock "C" dominates and we will not see patterns for others.

Overall, all the banks are more or less influenced by the financial crisis. There seems to be two groups according to the plot. "BK", "BBT", "JPM", "PNC", "USB", and "WFC" have similar pattern which have roughly no change in the first two years but drop down in the last year. However, the rest have different colors for each year from red to grey and finally to blue showing that their indices generally went down on a yearly basis.

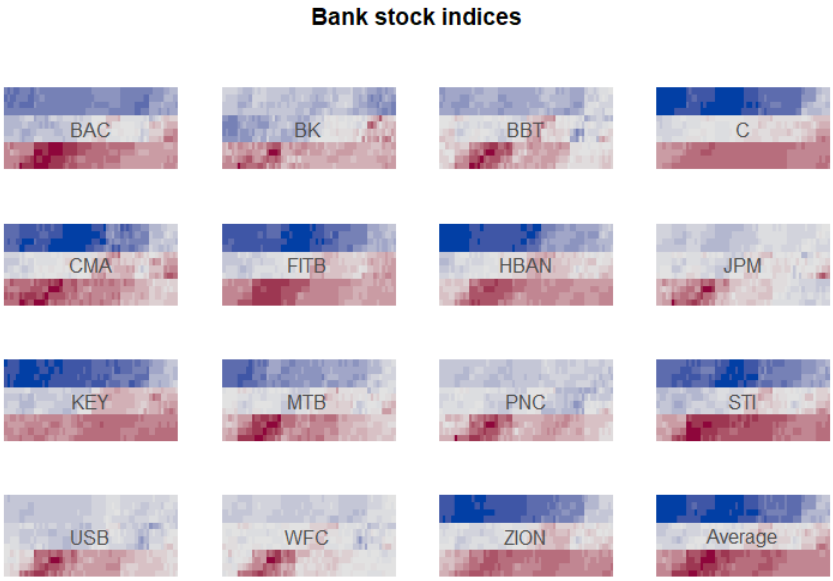

Figure 4.8: Bank stock glyphs plot

4.3.3 Investment Bank stocks

The investment bank tickers are listed below.

```
## [1] "SCHW" "ETFC" "GS"   "MS"
```

There are only 4 stocks. Same as before, the last glyph is the average. (See Figure 4.9) The plot is also in local scale.

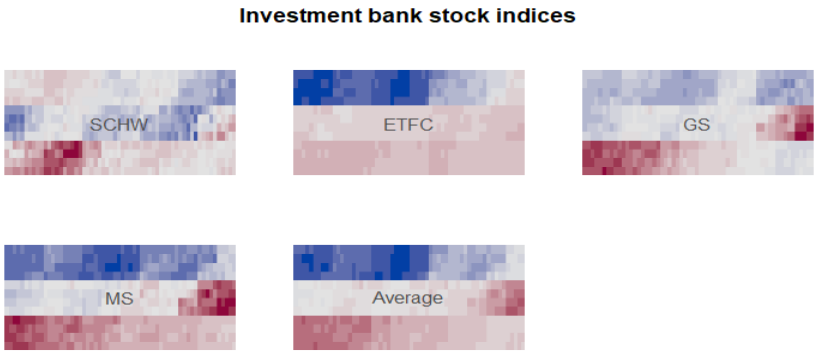

Figure 4.9: Investment bank stock glyphs plot

From the plot, we can see that the situation is similar to above. Most of the investment banks are influenced by the crisis. However, note that the second stock "ETFC" has a different pattern which had a early drop in 2007 but not in the late 2008. Therefore, it seems that the financial crisis did not affect it that much.

5 Conclusion

In summary, the `glyphs` package can plot any glyphs in a plane and can also make pixel-oriented glyphs by itself. The package is flexible and can handle various kinds of data sets.

In `ALL` data example, we illustrate how to plot the gene data by using the Hilbert curve layout. The plot can provide a basic exploration of one's gene data.

In `nasa` data example, we can use glyphs to add an additional dimension on another plot such as a map. Not only can we see the trend of data in different locations, but also we can combine the information of the location together with the plot.

In `SP500` data example, we can visualize each individual stock index and compare them in a plot. The Keim's method for time series data makes it easy to see the trend and compare the data between each time period. We can also group the data according to the plot.

One aspect that could be improved is the mapping from data to colors. Our method is to rescale the data into [0,1] but the outliers are lost during the process. Though we can set the range in our method to avoid it, the choice of the range is still a problem. Therefore, one can try to create a mapping that can better deal with the issue.

Acknowledgement

I would like to express my grateful thanks to my supervisor Prof.Wayne Oldford of the Statistics and Actuarial Science at University of Waterloo. His board and deep knowledge has always guided me in the right direction. He patiently taught me how to write rigorous code. He was always willing and enthusiastic to help and solve the problems that I encountered.

Last but not least, I give thanks to God for his generosity to give me wisdom and power to know his creation. I praise his continuous guidance and mercy as I completed the paper.

Appendix

A The `help` of each function in `glyphs` package

A.1 The `make_glyphs_draw` function

make_glyphs_draw {glyphs} R Documentation

Make glyphs in different classes

Description

Given a list of data vectors, make a corresponding list of glyphs in one of four different classes which are "png", "jpeg", "pixmap" and "tiff".

Usage

```
make_glyphs_draw(data, draw_fun, type = c("png", "jpeg", "pixmap", "tiff"),
  width = 50, height = 50, mar = rep(1, 4), ...)
```

Arguments

`data`	a list of data vectors
`draw_fun`	function of drawing in a device given a data vector
`type`	string specifying the format of output. "png" means the output is a list of data matrices in the png format, "jpeg" means the output is in the jpeg format and so on.
`width`	the width of the device
`height`	the height of the device
`mar`	A numerical vector of the form c(bottom, left, top, right) which gives the number of lines of margin to be specified on the four sides of the plot
`...`	Arguments passed to 'png', 'jpeg', 'pixmap', or 'tiff' according to the 'type'

Value

a list of glyphs in different classes corresponding to the list of data

Author(s)

Wayne Oldford, Jiahua Liu

Examples

```
n <- 16
data <- list()
for (i in 1:n){
  data[[i]] <- rnorm(500)
}
glyphs <- make_glyphs_draw(data, hist, type = "jpeg", mar = rep(4, 4), width = 200, height = 200)
x <- getGridXY(n)
plot_glyphs(x, glyphs = glyphs, glyphWidth = 0.8, glyphHeight = 0.6, axes = FALSE, xlab = "", ylab = "")
```

[Package *glyphs* version 0.1.0 Index]

A.2 The `getGridXY` functions

getGridXY {glyphs} R Documentation

Get x and y coordinates in a grid

Description

getGridXY generates a matrix of x and y coordinates in a grid. The justification of each coordinate is in center.

Usage

```
getGridXY(n, nrows, ncols = NULL, byrow = TRUE)
```

Arguments

`n`	number of coordinates to generate
`nrows`	number of rows in the grid
`ncols`	number of columns in the grid
`byrow`	logical value indicating whether the order of coordinates in the grid is byrow or not

Value

a matrix which gives the x and y coordinates of a grid

Author(s)

Jiahua Liu

Examples

```
x <- getGridXY(16)
x
```

[Package *glyphs* version 0.1.0 Index]

A.3 The plot_glyphs function

plot_glyphs {glyphs} R Documentation

Plot glyphs

Description

plot_glyphs plots a list of glyphs by basic plot or the plot in "grid" package

Usage

```
plot_glyphs(x, y = NULL, glyphs, glyphWidth, glyphHeight, just = c("centre",
  "center", "top", "bottom", "left", "right", "bottomleft", "topright",
  "bottomright", "topleft"), type = c("raster", "pixmap", "grid"),
  add = FALSE, ...)
```

Arguments

x	a numeric vector specifying x-location or a matrix whose first column specify x-location and second column specify y-location
y	a numeric vector specifying y-location, if x is a matrix, y is NULL.
glyphs	a list of glyphs which are data matrices in picture format or pixmaps
glyphWidth	width of each glyph
glyphHeight	height of each glyph
just	the justification of the rectangle relative to its (x, y) location
type	string specifying the format used to plot. "raster" means each glyph in the list is a data matrix in the png, jpeg or tiff format, and "pixmap" means it is a class of pixmap. "grid" means it will be ploted by the way in "grid" package. See rasterImage, pixmap, grid.raster
add	logical value, indicate whether the plot is add on to the original plot or not
...	Arguments passed to plot if add=FALSE

Author(s)

Jiahua Liu

Examples

```
library(dplyr)
# ozone
data_ozone <- list()
for(i in 1:(24*24)){
  col_number <- (i-1) %% 24 + 1
  row_number <- (i-1) %/% 24 + 1
  data_ozone[[i]] <- as.vector(nasa$mets$ozone[col_number,row_number,,])
}
glyphs_ozone <- make_glyphs(data = data_ozone, width = c(1,12,1), height = c(1,1,6),
                            glyph_type = "Keim", type = "pixmap")
x <- getGridXY(length(glyphs_ozone))
plot_glyphs(x, glyphs = glyphs_ozone, type = "pixmap")
# rainbow color
glyphs_rainbow <- make_glyphs(data = data_temperature[1:25], width = c(1,12,1), height = c(1,1,6), cols = rainbow(3),
                              glyph_type = "Keim")
x <- getGridXY(length(glyphs_rainbow))
x[,1] <- x[,1]/ceiling(max(x[,1]))
x[,2] <- x[,2]/ceiling(max(x[,2]))
plot_glyphs(x, glyphs = glyphs_rainbow, type = "grid")
```

[Package *glyphs* version 0.1.0 Index]

A.4 The `hsi2rgb` function

hsi2rgb {glyphs} R Documentation

HSI to RGB Conversion

Description

hsi2rgb transforms colors from HSI space (hue/saturation/intensity) into RGB space (red/green/blue)

Usage

```
hsi2rgb(h, s = NULL, i = NULL, maxColorValue = 255)
```

Arguments

h	vector of 'hue' values in [0,360] or 3-row hsi matrix
s	vector of 'saturation' values in [0,1], or NULL when r is a matrix
i	vector of 'intensity' values in [0,1], or NULL when r is a matrix

Details

HSI is a variation of the HSV model, which provides color scales with monotonically increasing or decreasing brightness after linear interpolation

Value

A matrix with a column for each color. The three rows of the matrix indicate "red","green","blue" values and are named "r", "g", and "b" in [0,255] accordingly.

Author(s)

Jiahua Liu

References

Keim, Daniel A, and Hans-Peter Kriegel. 1995. "Issues in Visualizing Large Databases." In *Visual Database Systems* 3, 203-14. Springer.

Examples

```
x=c(0,360)
y=c(1,0.4)
f=approxfun(x,y) # linear introploation function
l <- 100000 # number of colors
H=seq(x[1],x[2],length.out = l)
I=f(H)
S=rep(1,length(H))
R=hsi2rgb(H,S,I)[1,]
G=hsi2rgb(H,S,I)[2,]
B=hsi2rgb(H,S,I)[3,]
col1=rgb(R,G,B,maxColorValue = 255)
gr1=(0.34*R+0.5*G+0.16*B)/255
col2=grey(gr1)
par(mfrow=c(2,1))
barplot(rep(1,length(H)),col = col1,border = NA,beside = FALSE,space = c(0,0),main="HSI color mapping")
barplot(rep(1,length(H)),col = col2,border = NA,beside = FALSE,space = c(0,0))
```

A.5 The `data2col` function

data2col {glyphs} R Documentation

Data to color mapping

Description

data2col provides a mapping from data values to color values by using hsi color space. One can use `hueRange`, `intensityRange`, `alpha`, `alpha`, `numCols`, and `maxColorValue` to adjust the mapping. Also, missingCol specifies the color to the missing data.

Usage

```
data2col(data, cols = NULL, hueRange = c(0, 360), intensityRange = c(0.4,
  1), alpha = 1, numCols = 1e+05, maxColorValue = 255, xLow = min(data,
  na.rm = TRUE), origin = NULL, xHigh = max(data, na.rm = TRUE),
  missingCol = NULL, outRangeCol = NULL)
```

Arguments

`data`	vector of data values
`cols`	a vector of rgb colours or NULL (default). If NULL, cols is constructed from `hueRange`, `intensityRange`, `alpha`, `alpha`, `numCols`, and `maxColorValue`.
`hueRange`	numeric vectors with values in [0, 360], it specify the hue range. It does not need to be in ascending order since it is only used in the linear interpolation
`intensityRange`	numeric vectors with values in [0, 1], it specifies the intersity range. It does not need to be in ascending order since it is only used in the linear interpolation
`alpha`	number in [0,1]. It specifies the `alpha` transparency value.
`numCols`	number. It specifies the number of colors used to do the mapping, and the higher the value is, the more distinguished between different data values.
`maxColorValue`	number in (0, 255]. giving the maximum of the color values range.
`xLow`	lower value of one's interested range
`origin`	middle value in one's interested range in (`xLow`, `xHigh`), which specifies the data value mapping to the center of color scale
`xHigh`	higher value of one's interested range
`missingCol`	color character specifying the color for the missing data. It is in the form of "#rrggbbaa", and the default is yellow.
`outRangeCol`	color characters specifying the color for the data outside our interested range. If the `length` is two, the first will specify the color for the data lower than the range, and the second is for higher. the color should have the same form with `missingCol`, and the default is "blue" and "red".

Value

a vector of color chacacters which can be used to plot

Author(s)

Jiahua Liu, Wayne Oldford

Examples

```
data <- 1:10
col <- data2col(data)
barplot(rep(1:length(data)),col = col)
```

[Package *glyphs* version 0.1.0 Index]

A.6 The HilbertGlyph function

HilbertGlyph {glyphs} R Documentation

Hilbert curve data visualization

Description

HilbertGlyph generates a data matrix of png format or a class of "pixmap" which can be used to display color strings in an order of Morton curve in the space given a vector of colors

Usage

```
HilbertGlyph(col, fill = FALSE, type = c("png", "pixmap"))
```

Arguments

col	vector of colors
fill	logical value indicating whether the color strings repeat from the beginning if number of colors in col is smaller than the total length of Hilbert curve
type	string specifying the format of output. "png" means the output is a data matrix in the png format, and "pixmap" means the output is a class of pixmap.

Value

a data matrix of png format or a class of "pixmap" that is used to plot. See pixmap, writePNG, rasterImage

Author(s)

Jiahua Liu, Wayne Oldford

Examples

```
myPngmat <- HilbertGlyph()
myPngmat <- HilbertGlyph(col = colors())
myPngmat <- HilbertGlyph(col = colors(), fill = TRUE)
writePNG(myPngmat, target = "myplot.png")
plot(0,type='n', xlim=c(0,1), ylim=c(0,1), axes = FALSE,xlab = "", ylab = "")
rasterImage(myPngmat, 0, 0, 1, 1)
mypixmap <- HilbertGlyph(type = "pixmap")
mypixmap <- HilbertGlyph(col = colors(), fill = TRUE, type = "pixmap")
plot(mypixmap)
```

[Package *glyphs* version 0.1.0 Index]

A.7 The MortonGlyph function

MortonGlyph {glyphs} R Documentation

Morton curve data visualization

Description

MortonGlyph generates a data matrix of png format or a class of "pixmap" which can be used to display color strings in an order of Morton curve in the space given a vector of colors

Usage

```
MortonGlyph(col, fill = FALSE, type = c("png", "pixmap"))
```

Arguments

col	vector of colors
fill	logical value indicating whether the color strings repeat from the beginning if number of colors is smaller than the total length of Morton curve
type	string specifying the format of output. "png" means the output is a data matrix in the png format, and "pixmap" means the output is a class of pixmap.

Value

a data matrix of png format or a class of "pixmap" which can be used to plot. See pixmap, writePNG, rasterImage

Author(s)

Jiahua Liu, Wayne Oldford

Examples

```
myPngmat <- MortonGlyph()
myPngmat <- MortonGlyph(col = colors())
myPngmat <- MortonGlyph(col = colors(), fill = TRUE)
writePNG(myPngmat, target = "myplot.png")
plot(0,type='n', xlim=c(0,1), ylim=c(0,1), axes = FALSE, xlab = "", ylab = "")
rasterImage(myPngmat,0,0,1,1)
mypixmap <- MortonGlyph(type = "pixmap")
mypixmap <- MortonGlyph(col = colors(), fill = TRUE, type = "pixmap")
plot(mypixmap)
```

[Package *glyphs* version 0.1.0 Index]

A.8 The KeimGlyph function

KeimGlyph {glyphs} R Documentation

Keim's recursive pattern algorithm data visualization

Description

KeimGlyph generates a data matrix of png format or a class of "pixmap" to display color strings in an order of recursive pattern provided by Keim in the space given a vector of colors

Usage

```
KeimGlyph(col, width, height, fill = FALSE, type = c("png", "pixmap"))
```

Arguments

col	vector of colors
width	vector of width in each level of algorithm
height	vector of height in each level of algorithm
fill	logical value indicating whether the color strings repeat from the beginning if number of colors in col is smaller than the total number of colors to display
type	string specifying the format of output. "png" means the output is a data matrix in the png format, and "pixmap" means the output is a class of pixmap.

Value

a data matrix of png format or a class of "pixmap" that is used to plot. See pixmap, writePNG, rasterImage

Author(s)

Jiahua Liu, Wayne Oldford

References

Keim, Daniel A. 1996. "Pixel-Oriented Visualization Techniques for Exploring Very Large Data Bases." *Journal of Computational and Graphical Statistics* 5 (1). Taylor & Francis: 58-77.

Examples

```
width=c(1,6,1,12,1)
height=c(1,1,4,1,3)
myPngmat <- KeimGlyph(width=width, height=height)
myPngmat <- KeimGlyph(width=width, height=height, col = colors(), fill = TRUE)
writePNG(myPngmat, target = "myplot.png")
plot(0,type='n', xlim=c(0,1), ylim=c(0,1), axes = FALSE,xlab = "", ylab = "")
rasterImage(myPngmat,0,0,1,1)
myPixmap <- KeimGlyph(width=width, height=height, col = colors(), type = "pixmap")
plot(myPixmap)
```

A.9 The make_glyphs function

make_glyphs {glyphs} R Documentation

Make glyphs

Description

Given a list of data vectors and a function of data vectors to colors, make a corresponding list of glyphs in png format or a class of pixmap. The colors in each glyph is placed in the order of Hilbert curve, Morton curve or a method provided by Keim which is better used when the data has some inherent structure such as time series. By default, the data2col function is given, and one can adjust it by the parameters given (see `data2col`), or one can provide his own function.

Usage

```
make_glyphs(data, glyph_type = c("Hilbert", "Morton", "Keim"), width, height,
  cols = NULL, hueRange = c(0, 360), intensityRange = c(0.4, 1),
  alpha = 1, numCols = 1e+05, maxColorValue = 255, xLow = NULL,
  origin = NULL, xHigh = NULL, missingCol = NULL, outRangeCol = NULL,
  data2colfn, type = c("png", "pixmap"), ...)
```

Arguments

`data`	a list of data vectors
`glyph_type`	if "Hilbert"(Default) colors in each glyph is placed in the order of Hilbert curve, if "Morton", it is in Morton curve, and if "Keim" it is the method provided by Keim.
`width`	vector of width in each level of algorithm provided by Keim, must be given if glyph_type is "Keim"
`height`	vector of height in each level of algorithm provided by Keim if glyph_typbe is "Keim"
`cols`	a vector of rgb colours or NULL (default). If NULL, cols is constructed from hueRange, intensityRange, alpha, numCols, and maxColorValue.
`hueRange`	numeric vectors with values in [0, 360], it specifies the hue range. It does not need to be in ascending order since it is only used in the linear interpolation
`intensityRange`	numeric vectors with values in [0, 1], it specifies the intersity range. It does not need to be in ascending order since it is used in the linear interpolation
`alpha`	number in [0,1]. It specifies the `alpha` transparency value.
`numCols`	number. It specifies the number of colors used to do the mapping, and the higher the value is, the more distinguished between different data values.
`maxColorValue`	number in (0, 255] giving the maximum of the color values range.
`xLow`	lower value of one's interested range
`origin`	middle value in one's interested range in (`xLow`, `xHigh`), which specifies the data value mapping to the center of color scale, or "mean", "median" can be specified to set the origin as the mean or median value for each glyph
`xHigh`	higher value of one's interested range
`missingCol`	color character specifying the color for the missing data. It is in the form of "#rrggbbaa", and the default is yellow.
`outRangeCol`	color characters specifying the color for the data outside our interested range. If the `length` is two, the first will specify the color for the data lower than the range, and the second is for higher. the color should have the same form with `missingCol`, and the default is "blue" and "red".
`data2colfn`	function mapping from data values to color values, if missing, the function will be `data2col`
`type`	string specifying the format of output. "png" means the output is a list of data matrices in the png format, and "pixmap" means the output is a list of class of pixmaps.
`...`	Arguments passed to `data2colfn`

Value

a list of data matrices of png format or a list of pixmaps

Author(s)

Jiahua Liu, Wayne Oldford

Examples

```
library(qrmdata)
data("SP500_const") # load the constituents data from qrmdata
time <- c("2007-01-03", "2009-12-31") # specify time period
x <- SP500_const[paste0(time, collapse = "/"),] # grab out data
data_c <- list() # complete data
for (i in 1:ncol(x)){
  data_c[[i]] <- x[,i]
}
x <- t(na.omit(t(x)))
data <- split(x,col(x)) # omit the missing data

Glyphs <- make_glyphs(data[1:24]) # list of glyphs in the png format
Glyphs <- make_glyphs(data_c[1:24], glyph_type = "Morton")
width=c(1,6,1,12,1)
height=c(1,1,4,1,3)
Glyphs <- make_glyphs(data_c[1:24], width = width, height = height, glyph_type = "Keim", alpha = 0.7)
Glyphs_pixmap <- make_glyphs(data_c[1:24], glyph_type = "Morton", type = "pixmap", intensityRange = c(1,0.4))

# list of png format plot
sideLength <- ceiling(sqrt(length(Glyphs)))
plot(0,type='n', xlim=c(0, sideLength), ylim=c(0,sideLength),axes = FALSE,xlab = "", ylab = "")
glyph_x <- 0.7
glyph_y <- 0.5
plot <- Map(function(Glyphs_i,i){
  rasterImage(Glyphs_i,(i-1)%%sideLength, sideLength-1-(i-1)%/%sideLength, (i-1)%%sideLength+glyph_x, sideLength-(1-glyph_y)-(i-1)%/%sideLength)
}, Glyphs,1:length(Glyphs))

# list of class of pixmap plot
sideLength <- ceiling(sqrt(length(Glyphs)))
glyph_x <- 0.7
glyph_y <- 0.5
plot(0,type='n', xlim=c(0, sideLength), ylim=c(0,sideLength),axes = FALSE,xlab = "", ylab = "")
plot <- Map(function(Glyphs_i,i){
  addlogo(Glyphs_i, px=c((i-1)%%sideLength, (i-1)%%sideLength+glyph_x), py=c(sideLength-1-(i-1)%/%sideLength, sideLength-(1-glyph_y)-(i-1)%/%sideLength))
},Glyphs_pixmap, 1:length(Glyphs_pixmap))
```

[Package *glyphs* version 0.1.0 Index]

B Data analysis code

B.1 The 16 Individuals' gene plot

```
# gene data
library(ALL)
data(ALL)
expr <- exprs(ALL)
# Start with scaled data so that every expression is on the same scale
expr_scale <- t(scale(t(expr)))
data_gene <- split(expr_scale, col(expr_scale))
set.seed(1000)
# get a sample of 16 individuals
loc <- sample(length(data_gene), 16)
gene_sample <- data_gene[loc]
# get glyphs
glyphs_gene <- make_glyphs(data = gene_sample, type = "pixmap", origin = "median")
# plot them
x <- getGridXY(length(glyphs_gene))
plot_glyphs(x, glyphs = glyphs_gene, type = "pixmap",
            glyphWidth = 0.8, glyphHeight = 0.8,
            axes = FALSE, xlab = "", ylab = "",
            main = "Gene plot for each individual")
```

B.2 B-cell and T-cell ALL comparison

```
data_Bcell <- data_gene[1:95]
data_Tcell <- data_gene[96:128]
# get the average
data_average_Bcell <- list(Reduce("+", data_Bcell) / length(data_Bcell))
data_average_Tcell <- list(Reduce("+", data_Tcell) / length(data_Tcell))
par(mfrow = c(1,2))
# B-cells average glyph
glyphs_gene_Bcell <- make_glyphs(data = data_average_Bcell,
                                 type = "pixmap", origin = "median")
x <- getGridXY(length(glyphs_gene_Bcell))
plot_glyphs(x, glyphs = glyphs_gene_Bcell, type = "pixmap",
            glyphWidth = 0.8, glyphHeight = 0.6,
            axes = FALSE, xlab = "", ylab = "")
title("B-cells average", line = -1)
# T-cells average glyph
glyphs_gene_Tcell <- make_glyphs(data_average_Tcell,
                                 type = "pixmap", origin = "median")
x <- getGridXY(length(glyphs_gene_Tcell))
plot_glyphs(x, glyphs = glyphs_gene_Tcell, type = "pixmap",
            glyphWidth = 0.8, glyphHeight = 0.6,
            axes = FALSE, xlab = "", ylab = "")
title("T-cell average", line = -1)
```

B.3 The nasa trend plot (Figure 4.3)

```
library(dplyr)
data_temperature <- list()
temperature <- nasa$mets$temperature
for(i in 1:(24*24)){
    row_number <- (i-1) %% 24 + 1
    col_number <- (i-1) %/% 24 + 1
    data_temperature[[i]] <- as.vector(temperature[row_number,col_number,,])
}
cols <- diverge_hcl(21) # get colors
glyphs_temperature <- make_glyphs(data = data_temperature,
                                  width = c(1,12,1), height = c(1,1,6),
                                  cols = cols, origin = "mean",
                                  glyph_type = "Keim", type = "pixmap")
# get the latitude and longitude as coordinates
x <- expand.grid(nasa$dims$long, nasa$dims$lat)
par(mfrow = c(1, 2), mar = rep(1.5, 4), pty = "s")
plot_glyphs(x, glyphs = glyphs_temperature, type = "pixmap", xlab = "", ylab = "",
            xaxs = "i", yaxs = "i", cex.axis = 0.8, mgp = c(3, 0.5, 0))
title("Nasa data temperature trend plot", cex.main = 0.8)
pal(cols, main = "color scale")
```

B.4 The nasa trend plot with map background (Figure 4.4)

```
par(mar = rep(2,4))
library(maps)
long <- nasa$dims$long
lat <- nasa$dims$lat
interval_x <- (max(long)-min(long))/(length(unique(long))-1)
interval_y <- (max(lat)-min(lat))/(length(unique(lat))-1)
# draw map and make the map able to change the size
map_glyph <- make_glyphs_draw(data = list(glyphs_temperature),
                              draw_fun = function(glyph_i){
    # draw map background
    map("world", xlim = c(min(long)-interval_x/2, max(long)+interval_x/2),
        ylim = c(min(lat)-interval_y/2, max(lat)+interval_y/2), border = FALSE,
        col=adjustcolor("grey70", alpha.f = 0.7), fill=TRUE,
                        bg = "grey90", myborder = 0.001)
    map.axes(cex.axis = 0.8, mgp = c(3, 0.5, 0)) # add axes on the map
    grid(nx = length(nasa$dims$long), col = "white", lty = 1) # add grid
    # plot the glyphs
    plot_glyphs(x, glyphs = glyph_i, type = "pixmap",
                glyphWidth = 1.8, glyphHeight = 1.5, add = TRUE)
}, mar = rep(0,4), width = 960, height = 960)
plot(0,type='n', xlim=c(0, 1), ylim=c(0, 1), axes = FALSE, xlab = "", ylab = "")
title("Nasa data temperature trend plot\n (map background)",
      line = 0.1, cex.main = 0.8)
rasterImage(map_glyph[[1]], 0, 0, 1, 1)
```

B.5 Time series glyphs plot with common scaling (Figure 4.5)

```
xnew <- seq(1, 72, length.out = 100)
# get the latitude and longitude as coordinates
x <- expand.grid(nasa$dims$long, nasa$dims$lat)
ylim <- c(min(unlist(data_temperature)), max(unlist(data_temperature)))
timeseries_glyph_commonscale <- make_glyphs_draw(data = data_temperature,
draw_fun = function(data_i){
  sm_i <- smooth.spline(data_i, df = 12) # smooth.spline fitting with df = 12
  ypred_i <- predict(sm_i, x = xnew)$y
  plot(xnew, ypred_i, type = "l", lwd = 4, axes = FALSE, xlab = "", ylab = "",
       xaxs = "i", yaxs = "i", ylim = ylim)
}, type = "png", width = 100, height = 100, bg = NA)
# draw map and make the map able to change the size
map_glyph_timeseries <- make_glyphs_draw(data = list(timeseries_glyph_commonscale),
draw_fun = function(glyph_i){
  # draw map background
  map("world", xlim = c(min(long)-interval_x/2, max(long)+interval_x/2),
      ylim = c(min(lat)-interval_y/2, max(lat)+interval_y/2), border = FALSE,
      col=adjustcolor("grey70", alpha.f = 0.7), fill=TRUE,
      bg = "grey90", myborder = 0.001)
  map.axes(cex.axis = 1.5, mgp = c(3, 0.5, 0)) # add map axis
  grid(nx = length(nasa$dims$long), col = "white", lty = 1, lwd = 2) # add grids
  plot_glyphs(x, glyphs = glyph_i, add = TRUE) # plot glyphs
}, mar = rep(0,4), width = 1500, height = 1500)
plot(0,type='n', xlim=c(0, 1), ylim=c(0, 1), axes = FALSE, xlab = "", ylab = "")
title("Time series glyphs plot\n(common scaling)",
      line = 0.1, cex.main = 0.8)
rasterImage(map_glyph_timeseries[[1]], 0, 0, 1, 1)
```

B.6 Time series glyphs plot with local scaling (Figure 4.6)

```
# get time series glyphs
timeseries_glyph <- make_glyphs_draw(data = data_temperature,
                                     draw_fun = function(data_i){
  # smooth.spline fitting with df = 12
  sm_i <- smooth.spline(data_i, df = 12)
  ypred_i <- predict(sm_i, x = xnew)$y
  plot(xnew, ypred_i, type = "l", lwd = 4, axes = FALSE, xlab = "", ylab = "",
       xaxs = "i", yaxs = "i")
}, type = "png", width = 100, height = 100, bg = NA)
# draw map and make the map able to change the size
map_glyph_timeseries <- make_glyphs_draw(data = list(timeseries_glyph),
                                         draw_fun = function(glyph_i){
  # draw map background
  map("world", xlim = c(min(long)-interval_x/2, max(long)+interval_x/2),
      ylim = c(min(lat)-interval_y/2, max(lat)+interval_y/2), border = FALSE,
      col=adjustcolor("grey70", alpha.f = 0.7), fill=TRUE,
      bg = "grey90", myborder = 0.001)
  map.axes(cex.axis = 1.5, mgp = c(3, 0.5, 0)) # add map axis
  grid(nx = length(nasa$dims$long), col = "white", lty = 1, lwd = 2) # add grids
  plot_glyphs(x, glyphs = glyph_i, add = TRUE) # plot glyphs
}, mar = rep(0,4), width = 1500, height = 1500)
plot(0,type='n', xlim=c(0, 1), ylim=c(0, 1), axes = FALSE, xlab = "", ylab = "")
title("Time series glyphs plot\n(local scaling)",
      line = 0.1, cex.main = 0.8)
rasterImage(map_glyph_timeseries[[1]], 0, 0, 1, 1)
```

B.7 The SP500 general trend plot (Figure 4.7)

```
library(qrmdata)
data("SP500_const") # load the constituents data from qrmdata
time <- c("2007-01-03", "2009-12-31") # specify time period
data_sp500 <- SP500_const[paste0(time, collapse = "/"),] # grab out data
data_complete <- list() # complete data
for (i in 1:ncol(data_sp500)){
 data_complete[[i]] <- as.vector(data_sp500[,i])
}
x <- t(na.omit(t(data_sp500))) # omit the missing data
data_omitNA <- split(x,col(x)) # split the data into list
data_average <- list(Reduce("+", data_omitNA) / length(data_omitNA))
width=c(1,5,1,12,1) # set the width
height=c(1,1,4,1,3) # set the height
cols <- diverge_hcl(21, h = c(0, 260)) # diverge color from red to blue
# make glyph
average_glyph <- make_glyphs(data = data_average, glyph_type = "Keim", cols = cols,
                             width = width, height = height)
x <- getGridXY(length(average_glyph)) # get x and y coordinates to plot
# plot it
plot_glyphs(x, glyphs = average_glyph, axes = FALSE, xlab = "", ylab = "",
            main = "Average stock indices")
```

B.8 Bank stock glyphs plot (Figure 4.8)

```
bank_loc <- which(SP500_const_info$Subsector == "Banks") # get the stock location
data_bank <- data_complete[bank_loc] # get the bank stock data
bank_average <- Reduce("+", data_bank)/length(bank_loc) # calculate the average
data_bank[[length(bank_loc)+1]] <- bank_average # add the average to the data list
# make glyphs
glyphs_bank <- make_glyphs(data_bank, width = width, height = height,
                           glyph_type = "Keim", origin = "mean", cols = cols)
x <- getGridXY(length(glyphs_bank)) # get grid
plot_glyphs(x, glyphs = glyphs_bank, glyphWidth = 0.8, glyphHeight = 0.6,
            axes = FALSE, xlab = "", ylab = "",
            main = "Bank stock indices") # plot the glyphs
text(x, labels = c(as.vector(SP500_const_info[bank_loc,]$Ticker), "Average"),
     col = "grey30")
```

B.9 Investment bank stock glyphs plot (Figure 4.9)

```
investment_loc <- which(SP500_const_info$Subsector ==
                        "Investment Banking & Brokerage")
data_investment <- data_complete[investment_loc] # get the bank stock data
# calculate the average
bank_average <- Reduce("+", data_investment)/length(investment_loc)
# add the average to the data list
data_investment[[length(investment_loc)+1]] <- bank_average
# make glyphs
glyphs_investment <- make_glyphs(data_investment, width = width, height = height,
                                 glyph_type = "Keim",origin = "mean", cols = cols)
x <- getGridXY(length(glyphs_investment)) # get grid
plot_glyphs(x, glyphs = glyphs_investment, glyphWidth = 0.8, glyphHeight = 0.6,
            axes = FALSE, xlab = "", ylab = "") # plot the glyphs
title("Investment bank stock indices", line = 0)
text(x, labels = c(as.vector(SP500_const_info[investment_loc,]$Ticker), "Average"),
     col = "grey30")
```

References

[1] Matthew O Ward. Multivariate data glyphs: Principles and practice. In *Handbook of data visualization*, pages 179–198. Springer, 2008.

[2] Francis Galton. *Meteorographica, or methods of mapping the weather*. Macmillan, 1863.

[3] William S Cleveland. *The Collected Works of John W. Tukey: Graphics 1965-1985*, volume 5. CRC Press, 1988.

[4] J Friedman, E Farrell, R Goldwyn, M Miller, and J Sigel. A graphic way of describing changing multivariate patterns. In *Proceedings Sixth Interface Symposium on Computer Science and Statistics*, pages 56–59, 1972.

[5] Edgar Anderson. A semigraphical method for the analysis of complex problems. *Proceedings of the National Academy of Sciences*, 43(10):923–927, 1957.

[6] Beat Kleiner and John A Hartigan. Representing points in many dimensions by trees and castles. *Journal of the American Statistical Association*, 76(374):260–269, 1981.

[7] Herman Chernoff. The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association*, 68(342):361–368, 1973.

[8] Hadley Wickham, Heike Hofmann, Charlotte Wickham, and Dianne Cook. Glyph-maps for visually exploring temporal patterns in climate data and models. *Environmetrics*, 23(5):382–393, 2012.

[9] Barret Schloerke, Jason Crowley, Di Cook, Francois Briatte, Moritz Marbach, Edwin Thoen, Amos Elberg, and Joseph Larmarange. *GGally: Extension to 'ggplot2'*, 2017. R package version 1.3.2.

[10] Daniel A Keim. Designing pixel-oriented visualization techniques: Theory and applications. *IEEE Transactions on visualization and computer graphics*, 6(1):59–78, 2000.

[11] Daniel A Keim. Pixel-oriented visualization techniques for exploring very large data bases. *Journal of Computational and Graphical Statistics*, 5(1):58–77, 1996.

[12] David Hilbert. Ueber die stetige abbildung einer line auf ein flächenstück. *Mathematische Annalen*, 38(3):459–460, 1891.

[13] Guy M Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Company New York, 1966.

[14] Haim Levkowitz and Gabor T Herman. The design and evaluation of color scales for image data. *IEEE Computer Graphics and Applications*, 12(1):72–80, 1992.

[15] Daniel A Keim and Hans-Peter Kriegel. Issues in visualizing large databases. In *Visual Database Systems 3*, pages 203–214. Springer, 1995.

[16] Xiaochun Li. *ALL: A data package*, 2009. R package version 1.18.0.

[17] Hadley Wickham, Romain Francois, Lionel Henry, and Kirill Müller. *dplyr: A Grammar of Data Manipulation*, 2017. R package version 0.7.2.

[18] Marius Hofert and Kurt Hornik. *qrmdata: Data Sets for Quantitative Risk Management Practice*, 2016. R package version 2016-01-03-1.