HilbertVis
April 19, 2009

hilbertCurve  calculate finite approximations of the Hilbert curve

Description
These functions calculate the Hilbert curve in its finite approximations. hilbertCurvePoint
gives the coordinates of one point and hilbertCurve returns an array with the coordinates of
all $4^lv$ points. The functions are not needed for hilbertImage and only provided for demonstration purposes. plotHilbertCurve makes use of them.

Usage
hilbertCurve( lv )
hilbertCurvePoint( t, lv )

Arguments
lv  The iteration level. A hilbert curve of level $lv$ spans a square with side length
    $2^lv$ (coordinates ranging from 0 to $2^lv-1$) and has $4^lv$ points.
t  The point index in the Hilbert curve. Must be an integer in $0:(4^lv-1)$.

Value
hilbertCurvePoint returns a vector of two integer numbers, both in the range $0:(2^lv-1)$,
indicating the coordinates of point $t$. huilbertCurve returns a matrix with $4^lv$ rows and 2
columns, giving all points of the curve at level $lv$.

Author(s)
Simon Anders, EMBL-EBI, ⟨sanders@fs.tum.de⟩

See Also
plotHilbertCurve

Examples
hilbertCurvePoint( 67, 4 )
hilbertCurve( 4 )
hilbertDefaultPalette

Description

Calculates a colour palette of length size. This palette is used as default by hilbertDisplay (in the "HilbertVisGUI" package) and also useful for hilbertImage (see example there).

Usage

hilbertDefaultPalette( size, asArray=TRUE )

Arguments

size

The number of desired colours.

asArray

Whether to return an array of RGB values or a character vector of color specs.

Value

* if asArray=TRUE (default): An array with 3 rows and size columns, containing RGB values. This is the same format as returned by col2rgb.

* if asArray=FALSE: A character vector of color specs, suitable to be passed to the col argument of plot.

Author(s)

Simon Anders, EMBL-EBI, ⟨sanders@fs.tum.de⟩

Examples

# Get a palette
palette <- hilbertDefaultPalette(30)
# Transform from RGB triples to color strings (i.e., do the # reverse of col2rgb)
colors <- apply( palette, 2, function(a) rgb(a[1], a[2], a[3], max=255) )
# Plot the palette
plot.new()
plot.window( xlim=c(.5,30.5), ylim=c(0,1) )
rect( 1:30-.5, 0, 1:30+.5, 1, col=colors )
## hilbertImage

Produce a matrix that visualizes a long data vector along a Hilbert curve

### Description

Calculate a Hilbert curve visualization of a long data vector and return it as a square matrix.

### Usage

hilbertImage(data, level = 9, forEBImage = FALSE)

### Arguments

- **data**: A (potentially very long) vector of numerical data.
- **level**: The level of the Hilbert curve, determining the size of the returned matrix.
- **forEBImage**: If set to TRUE, an Image object (as defined in the EBImage package) will be returned.

**Note to MacOS X Users:** Due to a problem with EBImage's GTK+ bindings you are advised not to use EBImage and HilbertVis/HilbertVisGUI within the same R session as it seems to cause crashes. This shall be rectified soon. [Note added 2008-12-18]

### Details

See the package vignette for an explanation of this visualization technique.

### Value

A matrix of dimension $2^{\text{level}} \times 2^{\text{level}}$. Each matrix element corresponds to a bin of consecutive elements of the data vector, the bins arranged to follow the Hilbert curve of the given level. The value of a matrix element is the maximum value of the data vector elements in the bin.

If forEBImage=TRUE, a true-color Image object is returned instead, where each pixel is colored according to the palette constructed by the function `hilbertDefaultPalette`.

### Note

For an interactive GUI to explore a Hilbert curve visualisation, use the function `hilbertDisplay` in the HilbertVisGUI package.

### Author(s)

Simon Anders, EMBL-EBI, sanders@fs.tum.de

### Examples

```r
# Get a vector with example data
dataVec <- makeRandomTestData()

# Plot it in conventional (linear) fashion
plotLongVector(dataVec)
```
makeRandomTestData

generate a long vector of example data that is suitable to demonstrate the Hilbert curve visualisation

Description

This function generates a long numeric vector and fills it with many narrow Gaussian peaks of varying width and position. Around 30 the distribution of peak width is changed to be substantially larger. This feature is easily visible with the Hilbert curve visualization but much harder to spot with conventional 1D plots.

Usage

makeRandomTestData(len = 1e+07, numPeaks = 500)

Arguments

len Length of the vector
numPeaks Number of peaks to be placed in the vector

Value

A vector, of type 'numeric', with sample data.

Author(s)

Simon Anders, EMBL-EBI, sanders@fs.tum.de

Examples

# See the help page of function 'hilbertImage' for an example.
makeWiggleVector  generate a "wiggle vector" from start/end/value data

Description
Given intervals in the form of a "start" and an "end" vectors and corresponding values, generate a "wiggle vector" of a given length that contains the specified values in the vector elements indicated by the intervals.

Usage
makeWiggleVector(start, end, value, chrlength)

Arguments
start The start coordinates of the intervals. As usual in R, these are 1-based.
end The end coordinates of the intervals. As usual, the end points are included.
value The values to be put in the wiggle vector. Where intervals overlap, the values are added.
chrlength The desired length of the returned vector.

Value
A vector as described above.

Author(s)
Simon Anders, EMBL-EBI, sandersfs.tum.de

See Also
For a value vector containing only ones, this function acts similar as the pileup function in the ShortRead package.

Examples
intervalStarts <- c(3,10,17,22)
intervalEnds <- c(7,13,20,26)
values <- c(2, 1.5, .3, 4)
chrlength <- 30
wig <- makeWiggleVector( intervalStarts, intervalEnds, values, chrlength )
# The same effect can be achieved with the following R code, which, however # is much slower:
wig2 <- numeric(chrlength)
for( i in 1:length(values) )
  wig2[ intervalStarts[i]:intervalEnds[i] ] <-
    wig2[ intervalStarts[i]:intervalEnds[i] ] + values[i]
# Let's check that we got the same:
all( wig == wig2 )
**plotHilbertCurve**  
*Plotting the Hilbert curve (for demonstration purposes).*

**Description**

This function plots the Hilbert curve fractal at a chosen iteration level in order to give you an impression how it looks like.

**Usage**

```r
plotHilbertCurve(lv, new.page = TRUE)
```

**Arguments**

- `lv`  
  The iteration level. A Hilbert curve of level `lv` spans a square with side length $2^{lv}$ (coordinates ranging from 0 to $2^{lv}-1$) and has $4^{lv}$ points. Values $lv > 7$ will take very long and yield a cluttered mash of undistinguishable lines.

- `new.page`  
  Boolean indicating whether to start a new graphics page (default: yes).

**Value**

An invisible NULL is returned. Furthermore, a plot is created.

**Author(s)**

Simon Anders, EMBL-EBI, ⟨sanders@fs.tum.de⟩

**See Also**

- `hilbertCurve`

**Examples**

```r
plotHilbertCurve(3)
```

---

**plotLongVector**  
*A simple function to plot a very long vector.*

**Description**

This function does basically the same as just calling `plot(vec)` but is much faster in case of a very long vector. This is because it first calls `shrinkVector`.

**Usage**

```r
plotLongVector(vec, offset = 1, shrinkLength = 4000, xlab = "", ylab = "", type
```
Arguments

- **vec**: The numerical vector to be plotted.
- **offset**: The x axis is labelled with numbers from \( \text{offset} \) to \( \text{offset} + \text{length}(\text{vec}) - 1 \).
- **shrinkLength**: To which length to shrink the vector before plotting it. Should be at least the width of your plot in pixels.
- **xlab**: The label of the x axis, to be passed to `plot`.
- **ylab**: The label of the y axis, to be passed to `plot`.
- **type**: The plot type, to be passed to `plot`. By default, type 'h', i.e., needles, are used.
- **...**: Further arguments to be passed to `plot`.

Value

Invisible Null and a plot.

Author(s)

Simon Anders, EMBL-EBI, sanders@fs.tum.de

Examples

```r
plotLongVector( rep( 1:100000, 20 ) )
```

---

**shrinkVector**

*shrink a vector by partitioning it into bins and taking the maxima in the bins*

Description

Given a (potentially very long) vector, the vector is partitioned into a given number of (up to rounding errors) equally long parts, and a vector of the maxima within each of the parts it returned.

Usage

```r
shrinkVector(vec, newLength)
```

Arguments

- **vec**: The vector to be shrunk. May be numeric or integer.
- **newLength**: The desired size of the return vector, i.e., the number of partitions

Details

This function is useful when plotting a very long vector such as those returned by `ShortRead::pileup`. For a sufficiently long vector \( x \), the two commands `plot( x, type="h" )` and `plot( shrinkVector( x, 10000 ), type="h" )` produce essentially the same plot (up to a rescaling of the x axis) but the former takes much longer to execute because R plots many needles on top of each other. The function `plotLongVector` is a simple wrapper around this function.
shrinkVector

Value

A vector of length `newLength` with the maxima of each of the partitions of `vector`.

Note

Note that, as the maximum is returned, the result may not be what you want if negative numbers are present. Note further that NAs may not be handled correctly.

Author(s)

Simon Anders, EMBL-EBI (sandersfs.tum.de)

See Also

`plotLongVector`, `ShortRead::pileup`, `HilbertVisGui::simpleLinPlot`

Examples

```
shrinkVector( 100000 + 1:1000, 17 )
```
Index

col2rgb, 2
hilbertCurve, 1, 6
hilbertCurvePoint(hilbertCurve), 1
hilbertDefaultPalette, 2, 3
hilbertDisplay, 2, 3
hilbertImage, 1, 2, 3
HilbertVisGui::simpleLinPlot, 8
makeRandomTestData, 4
makeWiggleVector, 5
plot, 2, 7
plotHilbertCurve, 1, 6
plotLongVector, 6, 7, 8
ShortRead::pileup, 8
shrinkVector, 6, 7