An Introduction to *ggbio*

Tengfei Yin, Michael Lawrence, Dianne Cook

February 4, 2012

Contents

1 Introduction

The R package, *ggbio*, provided a toolkit for visualizing biological data, using static R graphics. We mainly focus on visualizing sequencing data, and annotation tracks, but not just limited to it. We also try to implement some most used graphics and at the same time explore more types of new graphics. For example, we have implement *Manhattan plot* for GWAS study, stacked overview for exploring distribution of events cross all chromosomes, splicing summary for summarizing alternative splice events, etc. It's just a start, we hope to provide a static gallery for biologists and analyst, and prototyping for interactive graphics project like *visnab*.

This package extends the *ggplot2* package[1], which provides elegant static graphics, based on the grammar of graphics. The syntax and style of commands follows that of *ggplot2*. For those plots which require a low level manipulation on graphics some plots are enhanced using *grid* or *gridExtra*.

The goal is to provide high quality graphics for both analysis and publication purposes, requiring some rules:

- Be general. Try to focus on most asked questions and provide most used types of graphics at first.
- Be object-oriented, by providing a generic function, eg `qplot`, which automatically can handle and plot different R biological data objects in Bioconductor project.
- To use *ggplot2* to develop graphics much as as possible, but hide the complexity. Most functions return a `ggplot` object, which allow users to further enhance this object, for example, add/change labels, changing color scheme, etc.
- Be easy-to-use and user-friendly. For specific questions or most used graphics, we provide simple functions to that make the necessary graphics.

2 Some Details of *ggplot2*

Extensive details about *ggplot2* are available in the on-line documentation, or the web site `http://had.co.nz/ggplot2/geom_point.html`. The key pieces of describing a plot are these:

data object to be visualized

geoms how elements of the data are mapped into the plot (points, lines, ...)

statistics transformations necessary on the data, eg binning for a histogram

scale scaling data values for plotting, matching colors to values, generating legends

coordinates axes, coordinate transformations, eg polar, and aspect ratios plot

There are very **important** differences between *ggbio* and *ggplot2*:

- The horizontal axis in *ggbio*, for most plots, will be genomic coordinates. You can only specify the x to be *start, end, midpoint* describing interval data with width larger than 1.
- Faceting of the graphics, can only be done by **seqnames** or region on a chromosome. If **seqnames** is in the data structure, it automatically is used for faceting.

3 Generic Visualization Method

In the spirit of being general and object-oriented, following the syntax of *ggplot2*, we redefine the *quick plot* function `qplot` in *ggplot2* package, as a **S4** generic function. Then the `qplot` function could automatically and appropriately plot different R objects. For each object type a new **geom** is defined.

In the following section, we will introduce how to use `qplot` to plot different types of data to generate different displays.

[1] `http://had.co.nz/ggplot2/`

3.1 Plotting a `data.frame`

This is a wrapper around the original `qplot` function. when reading in `data.frame` or `matrix` object, you can use `qplot` as usual without any change, it essentially just call ggplot2::qplot.

```
> library(ggbio)
```

```
> p <- qplot(data = mtcars, mpg, cyl)
> print(p)
```

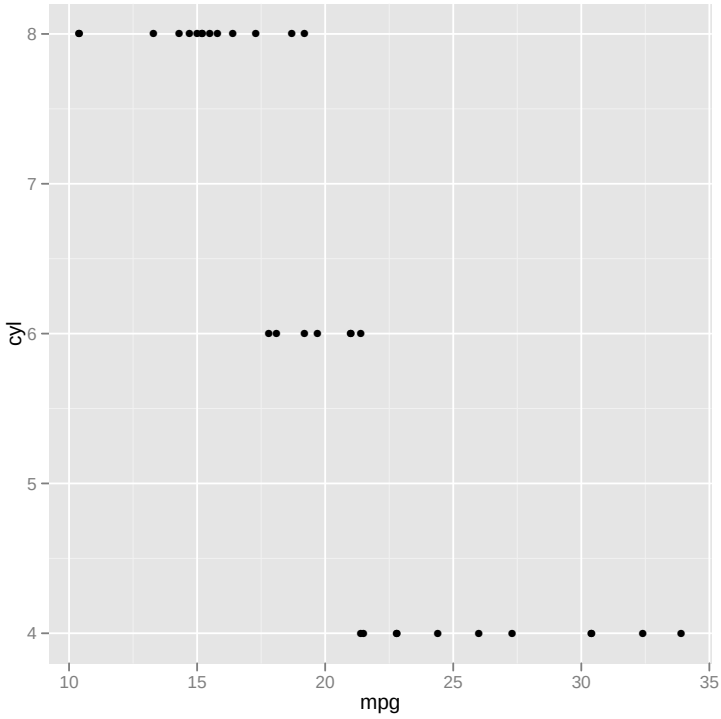

Figure 1: If the data object is simply an R `data.frame`, the `qplot` defaults are used, which is the point geom.

3.2 Plotting `GRanges` object

A `GRanges` object is defined in *GenomicRanges* package. It is one of the most important data structures coding genomic information, organizing genomic data as intervals. Three required fields for this object is `seqnames` for chromosome name, `ranges` for interval position, `strand` for storing direction of strand for particular interval, including +,-,*.Currently R does not have any convenient functions for visualizing interval objects.

3.2.1 Sample `Granges` object

Let's first create a sample `GRanges` object that can be used for following examples. We generate data, mimicking paired RNA-seq data, that contains 1000 rows, 3 chromosomes, and meta data of grouping and pairing information.

```
> set.seed(1)
> N <- 1000
> library(GenomicRanges)
> gr <- GRanges(seqnames =
+               sample(c("chr1", "chr2", "chr3"),
+                      size = N, replace = TRUE),
+               IRanges(
+                       start = sample(1:300, size = N, replace = TRUE),
+                       width = sample(70:75, size = N,replace = TRUE)),
+               strand = sample(c("+", "-", "*"), size = N,
+                 replace = TRUE),
+               value = rnorm(N, 10, 3), score = rnorm(N, 100, 30),
+               group = sample(c("Normal", "Tumor"),
+                 size = N, replace = TRUE),
+               pair = sample(letters, size = N,
+                 replace = TRUE))
```

3.2.2 Supported Geoms

For `GRanges` objects we support following *geoms*:

full Show the full stacked interval, a set of rectangles. Default.

segment Show full stacked interval, as a set of segments.

line Show line, which the user needs to provide.

point Show points, which the user needs to provide.

coverage.line Show coverage by using line.

coverage.polygon Show coverage by using polygon.

This code generates the default plot of a `GRanges` object (Figure 2). It uses the `full geom`, showing the data as stacked intervals, automatically faceted by **seqnames** (chromosome) and colored by the strand.

In this next display (Figure 3), a different theme was used to make the background white.

In this next plot, the `coverage.p geom` is used (Figure 4).

And let's plot all other geoms together by `grid.arrange` from package *gridExtra* (Figure 5).

3.2.3 Layout

Figure 6 shows, when you simply facet by seqnames as default, your can use `nrow` and `ncol` to control the layout of the graphic. It essentially use `facet_wrap` in *ggplot2* automatically.

3.2.4 Faceting

Faceting in *ggbio* is more restricted than in *ggplot2*:

- The faceting variable can only be `seqnames` or regions on the genome. So we limited the formula passing to `facet`, e.g something s̃eqnames, is accepted formula, you can change *something* to variable name in the elementMetadata. But you cannot change the second part.
- Sometimes, we need to view different regions, so we also have a `facet_gr` argument which accept a `GRanges`. If this is provided, it will override the default **seqnames** and use provided region to facet the graphics, which might be useful for different gene centric views.

```
> p <- qplot(gr)
> print(p)
```

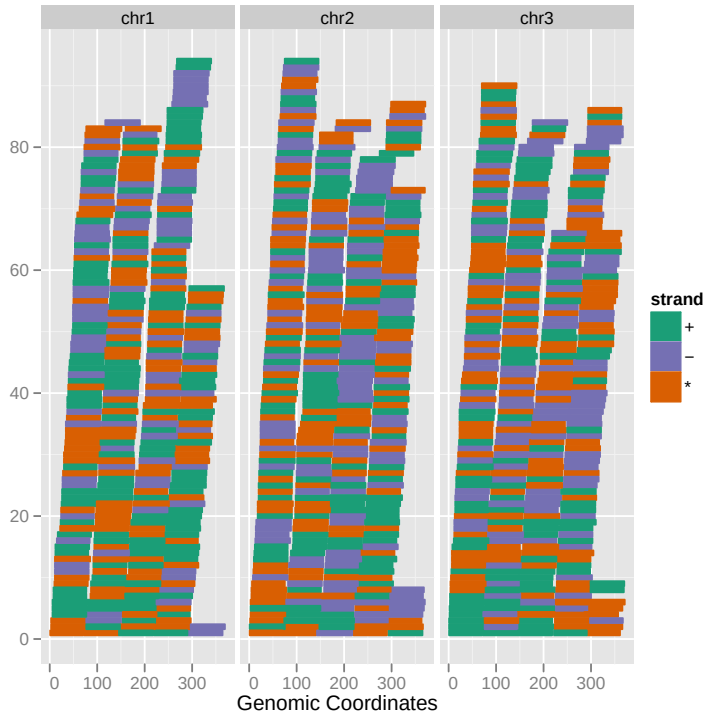

Figure 2: qplot for GRanges as `geom full`. We could see the plot is shown in three columns which facet by seqnames, and colored by strand. The small rectangles are those interval data described in our test `GRanges` object. We can imagine they are short reads on specific region of the genome

```
> p <- p + theme_bw()
> print(p)
```

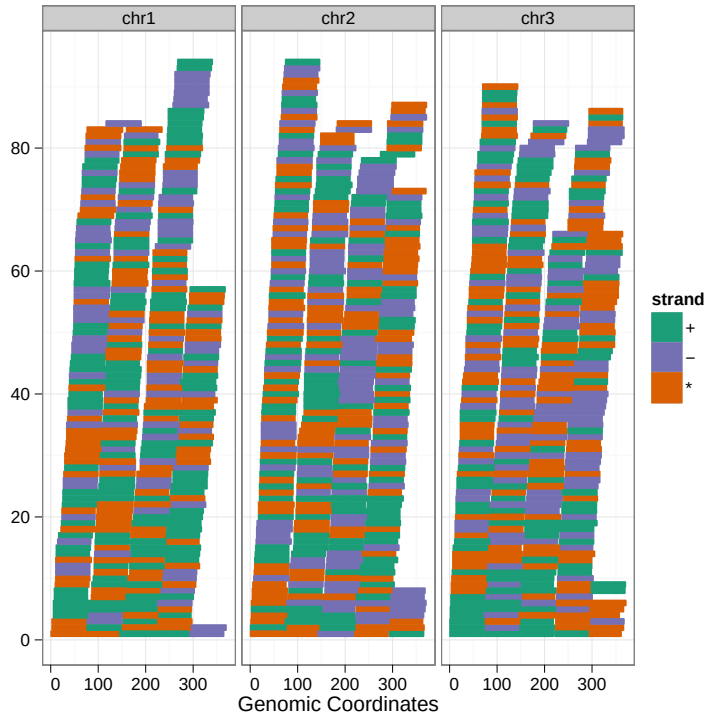

Figure 3: qplot for GRanges as geom full. The theme for this graphic is set to blank

```
> p <- qplot(gr, nrow = 1, geom = "coverage.p")
> p <- p + geom_hline(yintercept = 40, color = "red",
+     size = 1)
> print(p)
```

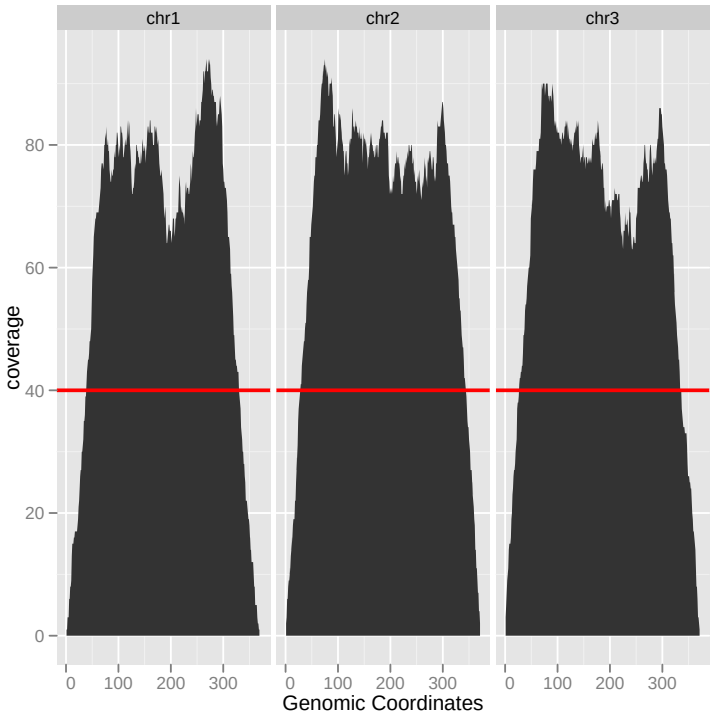

Figure 4: qplot for GRanges as `geom coverage.polygon`, the redline shows the artificial cutoff.

```
> gr.sub <- gr[seqnames(gr) == "chr1"] #or
> p1 <- qplot(gr.sub, geom = "full") + opts(title = "full")
> p2 <- qplot(gr.sub, geom = "point", y = value) + opts(title = "point")
> p3 <- qplot(gr.sub, geom = "line", y = value) + opts(title = "line")
> p4 <- qplot(gr.sub, geom = "coverage.line") + opts(title = "coverage.line")
> p5 <- qplot(gr.sub, geom = "coverage.polygon") + opts(title = "coverage.polygon")
> library(gridExtra)
> grid.arrange(p1, p2, p3, p4, p5,ncol = 2)
```

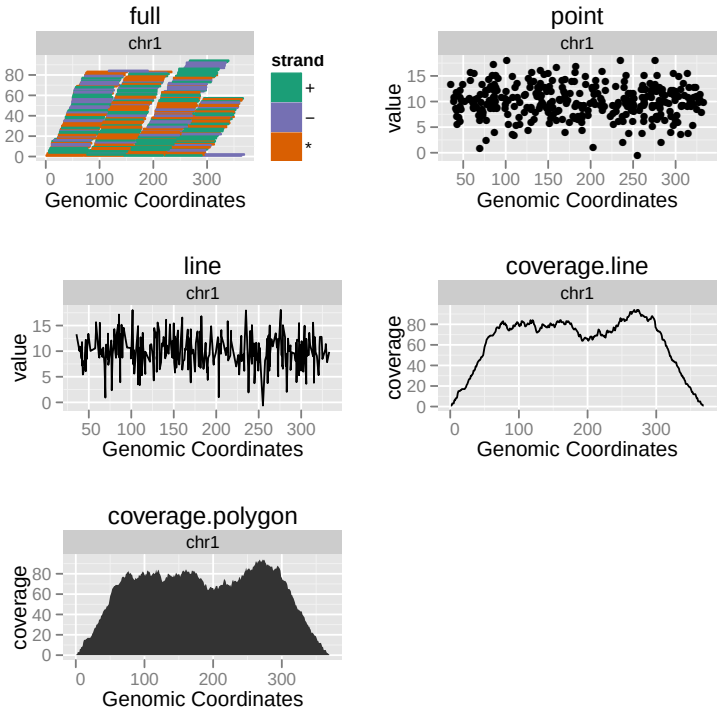

Figure 5: All common geoms(except splice) for GRanges object

@

```
> p <- qplot(gr, ncol = 2)
> print(p)
```

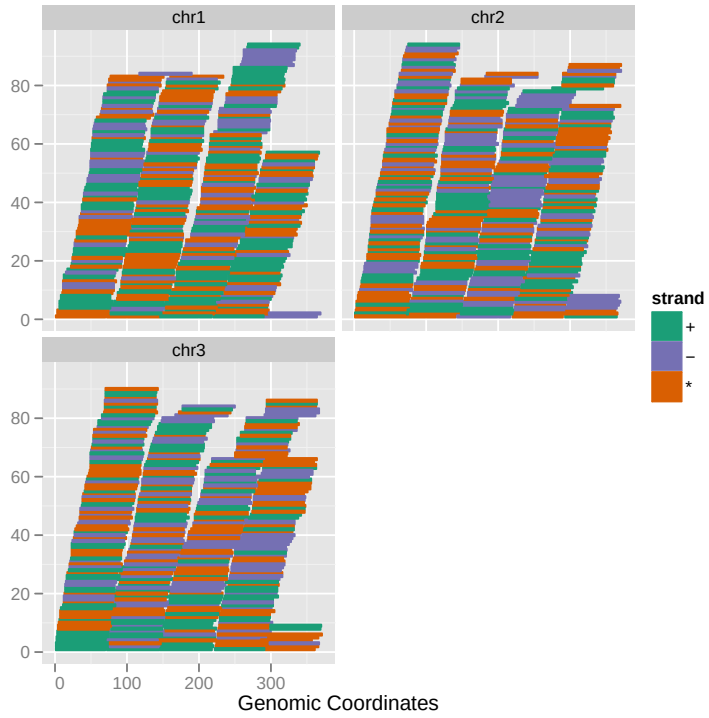

Figure 6: qplot use layout to control the graphics. Here when users set ncol = 2, it automatically wrap to two column when facet by default seqnames.

```
>   p <- qplot(gr, facets = group ~ seqnames)
>   print(p)
```

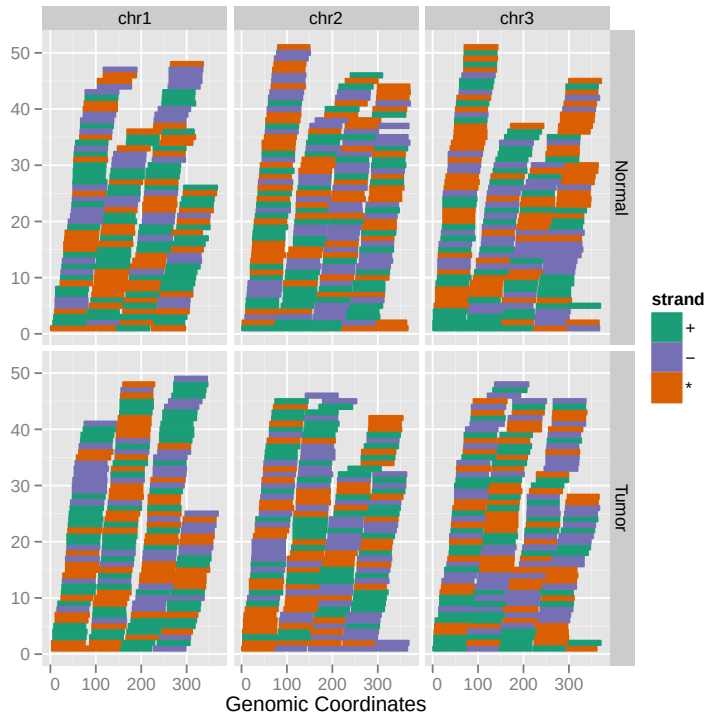

Figure 7: Facet by group

Figure 7 shows we could only specify the facet row to be one the categorical variables in elementMetadata.

NOTICE: User `facets`, NOT `facet`.

Figure 8 shows how to facet by `facet_gr` arguments.

When providing `nrow, ncol`, use facet_wrap, so users need to align y, default is use facet_grid , and free x-scale.

@

```
>  gr.region <- GRanges(c("chr1", "chr2", "chr3"),
+   IRanges(c(100, 200, 250),
+   width = 70))
> ## facet_grid
> p <- qplot(gr, facet_gr = gr.region)
> ## facet_wrap
> p <- qplot(gr, facet_gr = gr.region, nrow = 2) +
+   scale_y_continuous(limits = c(0, 90))
> print(p)
```

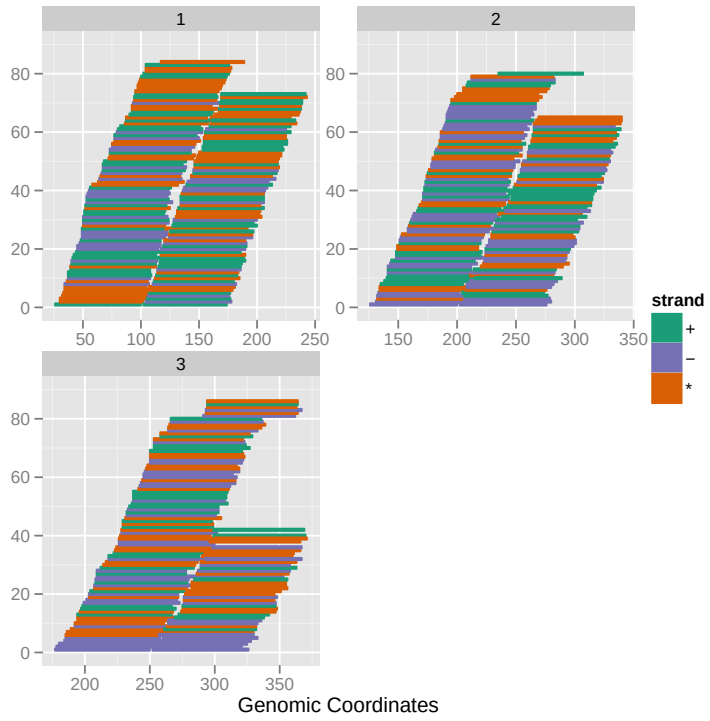

Figure 8: Facet by regions

3.3 For `GRangesList` object

If you want to more control like `GRanges` object, please simply unlist the `GRangesList` object and treat it as `GRanges` object.

But `qplot` for `GRangesList` object is specially designed for gene modeling purpose, since most time a `GRangesList` has a native structure to describe a genomic model or alternative splicing, we make the default automatically adding group information based on the listed data. We assume each list contains a

single isoform or a single model, so they need to be shown on the same level and connected by segments or chevrons.

3.3.1 Chevron Geom

To draw a normal GRanges as Chevron instead of segments or rectangle as shown in previous sections, we need to provide a special geom for this purpose. Chevron is popular in gene viewer or genomoe browser, when they try to show isoforms or gene model. More advanced usage is to use height of chevron or width of chevron to show statistics summary. Here we introduce a new function `geom_chevron`, just like any other `geom_*` function in *ggplot2*, instead, it reads in `GRanges` object, not a data.frame.

Here is a simple example (Figure 9)of how it works, it give the users ability to control the wiggled direction or scale of it when it's point to one of the columns.

```
> gr.chr1 <- GRanges("chr1", IRanges(c(100, 200, 300), width = 50))
> p <- qplot(gr.chr1)
> gr.gaps <- gaps(gr.chr1)[-1]
> values(gr.gaps)$score <- c(1, 100)
> p1 <- p + geom_chevron(gr.gaps)
> p2 <- p + geom_chevron(gr.gaps, aes(size = score), offset = "score",
+                  chevron.height = c(0.1, 0.2))
> p3 <- p + geom_chevron(gr.gaps, offset = -0.1)
> tracks(p1, p2, p3)

NULL
```

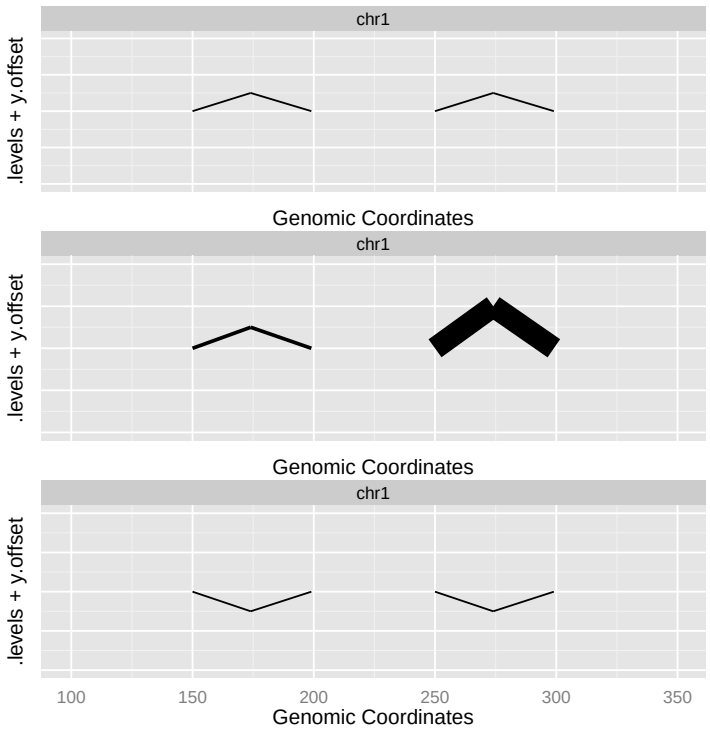

Figure 9: geom chevron. how to use parameters to control it.

3.3.2 Alternative splicing and summary

Here is a case study, how to use `qplot` for `GRangesList` to produce a splicing model and show summary on it. In the following data from a `TranscriptDb` object, and using `freq` argument to pass the summary information to the function.

It's convenient to get model from a `TranscriptDb` object, it's already split by transcript when you specify `by` equals *tx*, and this will return a `GRangesList` object, with name of transcript.

```
> library(TxDb.Hsapiens.UCSC.hg19.knownGene)
> data(genesymbol)
> txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
> exons.tx <- exonsBy(txdb, by = "tx")
> exons.rbm17 <- subsetByOverlaps(exons.tx, genesymbol["RBM17"])
> nms <- names(exons.rbm17)
> freqs <- c(100, 200, 300)
> names(freqs) <- nms
> p.splice1 <- qplot(exons.rbm17)
> ## when turning on frequency
> ## p.splice <- qplot(exons.rbm17, freq = freqs, show.label = TRUE, label.type = "count",
> ##       scale.size = c(1, 5), label.size = 3)
> p.splice2 <- qplot(exons.rbm17, freq = freqs, show.label = TRUE, offset = 0.05,
+                    label.type = "count")
```

Please refer to a special graphic in section 6.1.

3.4 For `IRanges` object

Similar to `qplot` for `GRanges`, supported geoms includes, *full, segment, coverage.line, coverage.polygon.*

No facet support yet, just simply viewing the intervals and coverage.

```
> ir <- ranges(gr[seqnames(gr) == "chr1"])[1:40]
> p1 <- qplot(ir) + opts(title = "full")
> p2 <- qplot(ir, geom = "segment")+ opts(title = "segment")
> p3 <- qplot(ir, geom = "coverage.line")+ opts(title = "coverage.line")
> p4 <- qplot(ir, geom = "coverage.polygon")+ opts(title = "coverage.polygon")
> grid.arrange(p1, p2, p3, p4, ncol = 2)
```

3.5 For `Rle/RleList` object

`Rle` and `RleList` is container for storing an atomic vector(List) that is stored in a run-length encoding format. It is based on the `rle` function from the base package. and please see manual of *IRanges* for more details.

`qplot` for `Rle` and `RleList` provides smoothing or summarizing methods for viewing these types of data. We support three geoms *point, line, segment* and five types of smoothing or summarizing methods,*raw, viewMaxs, viewMins, viewSums, viewMeans* the last four methods are also defined in *IRanges*. When you are using *raw*, we just convert it to a numeric vector and plot it along x. Make sure it's not a huge vector, other wise it's going to be super slow to visualize in *ggplot2*. If use the following four method, you could pass extra arguments, such as `lower` to control how to smooth the data.

For `RleList`, it automatically facet them by list, controlled by parameters `facetByRow`, to facet them side by side all on different tracks. This could be useful for multiple comparison of different samples.

Here are some simple examples

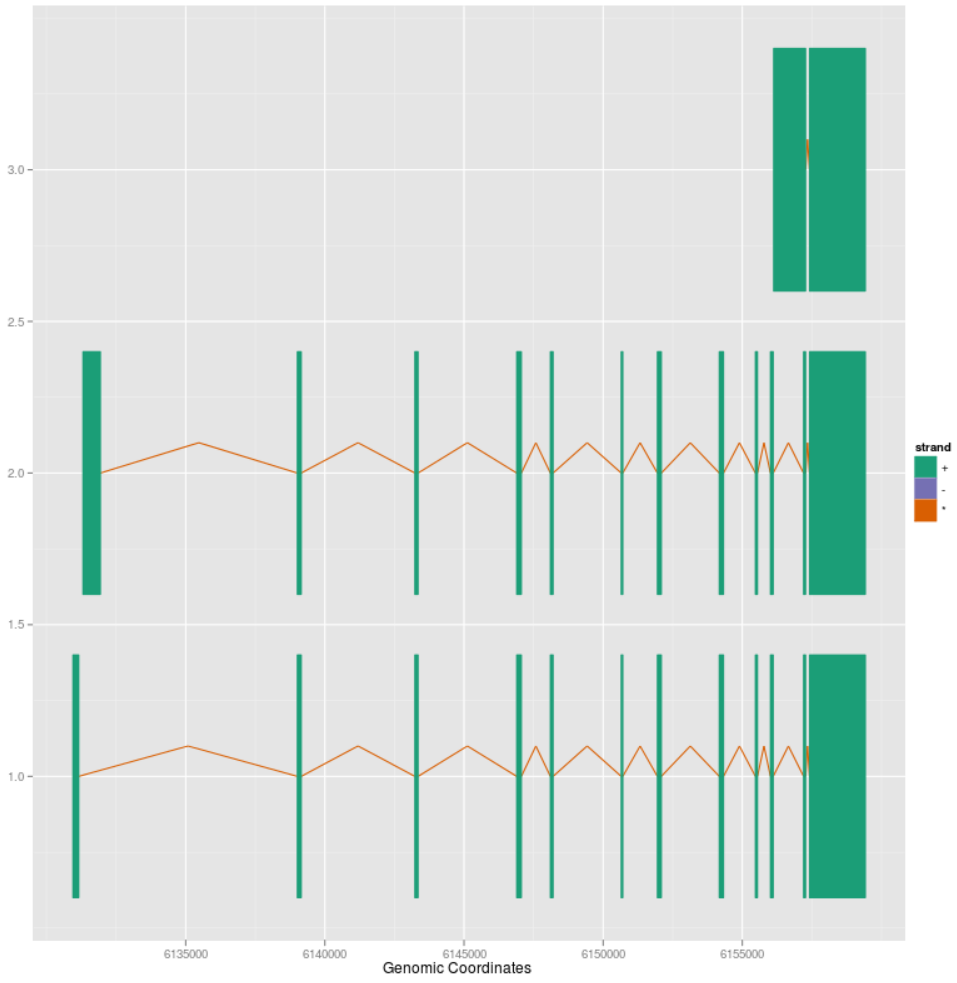

Figure 10: Transcripts model of gene RBM17

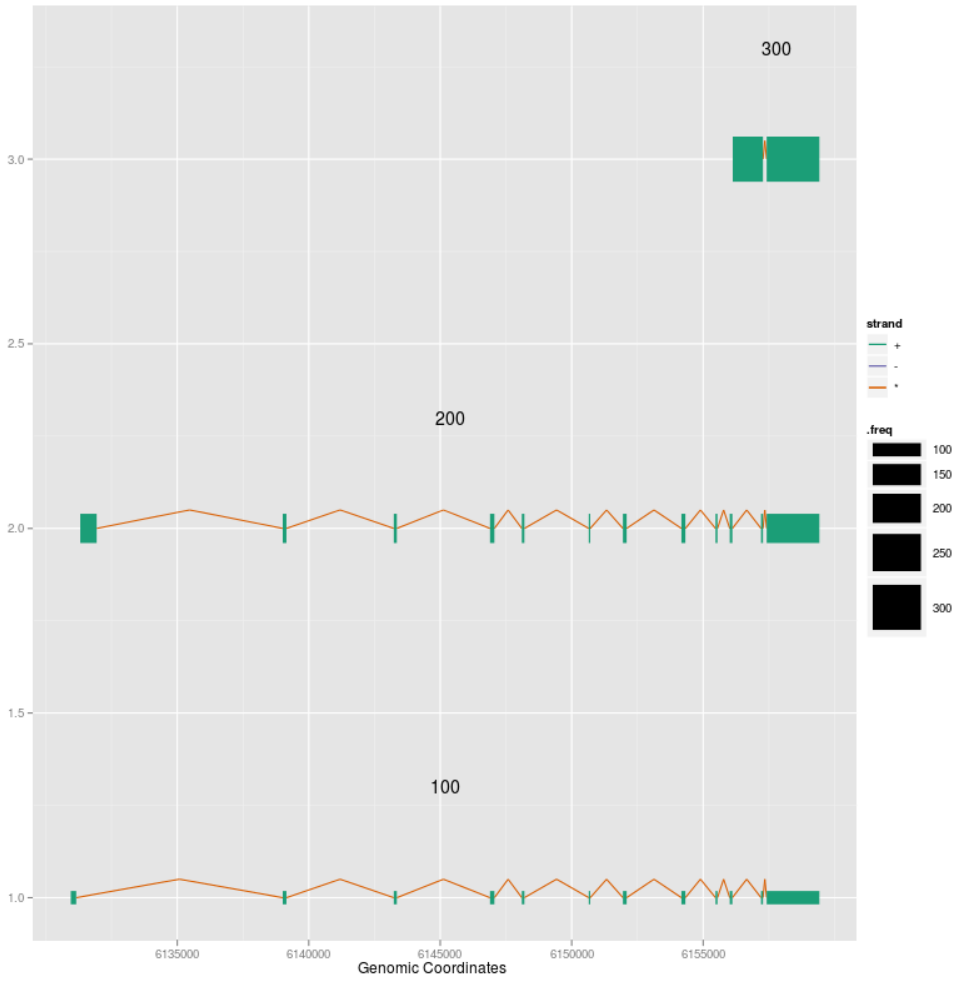

Figure 11: Model with frequency indicated

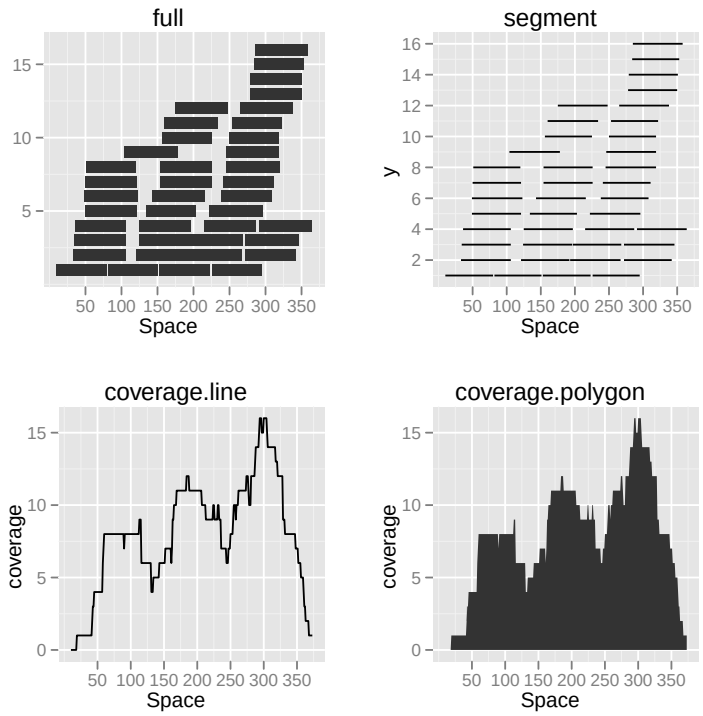

Figure 12: qplot for `IRanges`. Similar to `GRanges`, but we don't have seqnames information about it

```
> set.seed(1)
> lambda <- c(rep(0.001, 4500), seq(0.001, 10, length = 500),
+             seq(10, 0.001, length = 500))
> xVector <- rpois(1e4, lambda)
> xRle <- Rle(xVector)
> xRleList <- RleList(xRle, 2L * xRle)
> p1 <- qplot(xRle) + opts(title = "raw(point)")
> p2 <- qplot(xRle, geom = "line")+ opts(title = "raw(line)")
> p3 <- qplot(xRle, geom = "segment")+ opts(title = "raw(segment)")
> p4 <- qplot(xRle, type = "viewMaxs", geom = "line", lower = 5)+
+    opts(title = "viewMax(line)")
> p5 <- qplot(xRle, type = "viewMins", geom = "line",lower = 5)+
+    opts(title = "viewMins(line)")
> p6 <- qplot(xRle, type = "viewMeans", geom = "line",lower = 5) +
+    opts(title = "viewMeans(line)")
> p7 <- qplot(xRle, type = "viewSums",geom = "line", lower = 5)+
+    opts(title = "viewSums(line)")
>
> ## more control like this
> ## qplot(xRle, type = "viewSums", lower = 5, size = I(10), color = I("red"),
> ##      alpha = y)
```

Figure 13 shows some combination of different geoms and methods.
Figure 14 shows multiple sample comparison.

@

```
> grid.arrange(p1, p2, p3, p4, p5, p6, p7, ncol = 2)
```

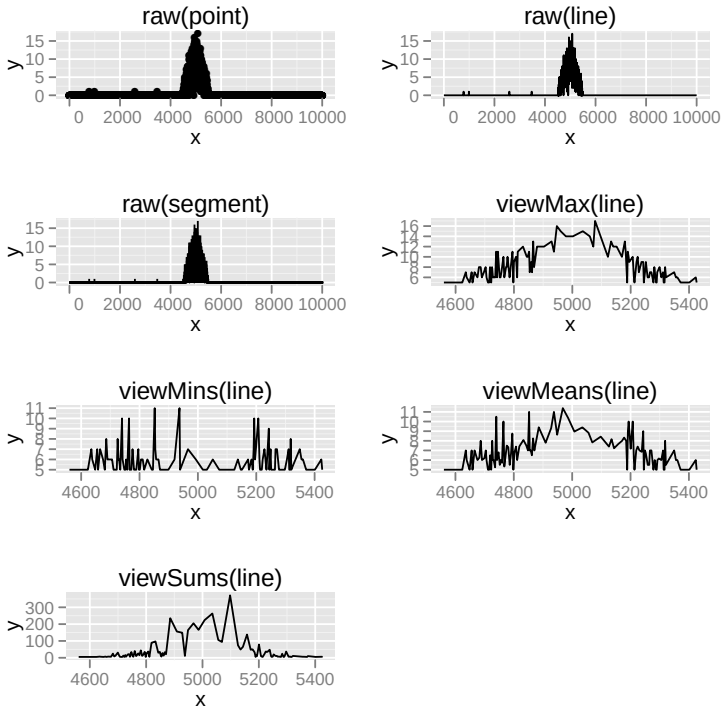

Figure 13: Rle visualization with smoothing method

@

```
> p <- qplot(xRleList, geom = "line", facetByRow = TRUE)
> print(p)
```

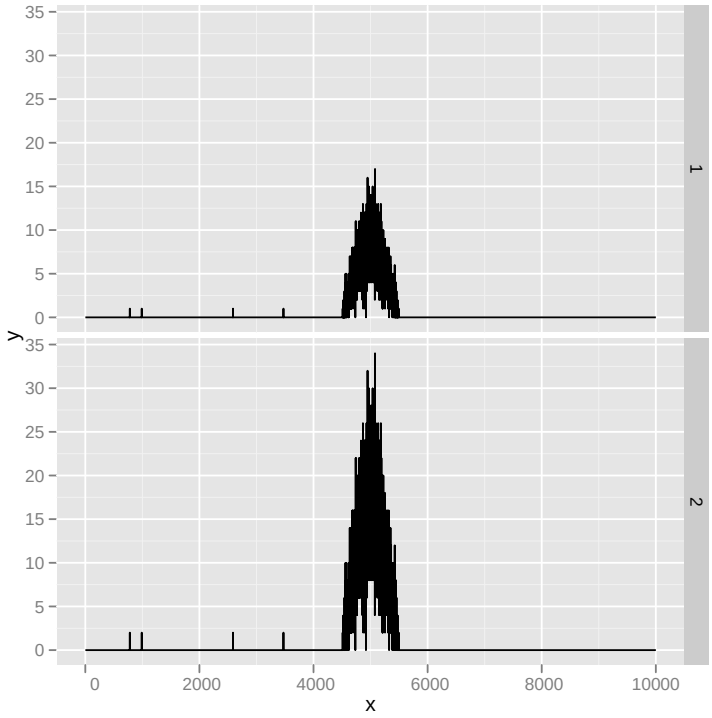

Figure 14: RleList visualization with facets

3.6 For `GappedAlignments` object

Supported geom includes:

gapped.pair Consider junction reads information, and assign stepping levels based on that information, could be used to show junction. The `show.junction` could control show the junction reads or not

full Show as the *full* geom like for `GRanges` object, randomly assign stepping levels.

```
> library(Rsamtools)
> bamfile <- system.file("extdata", "SRR027894subRBM17.bam", package="biovizBase")
> ## need to set use.names = TRUE
> ga <- readBamGappedAlignments(bamfile, use.names = TRUE)
> p1 <- qplot(ga)
> p2 <- qplot(ga, show.junction = TRUE)
> p3 <- qplot(ga, geom = "full")
> grid.arrange(p1, p2, p3, ncol = 1)
```

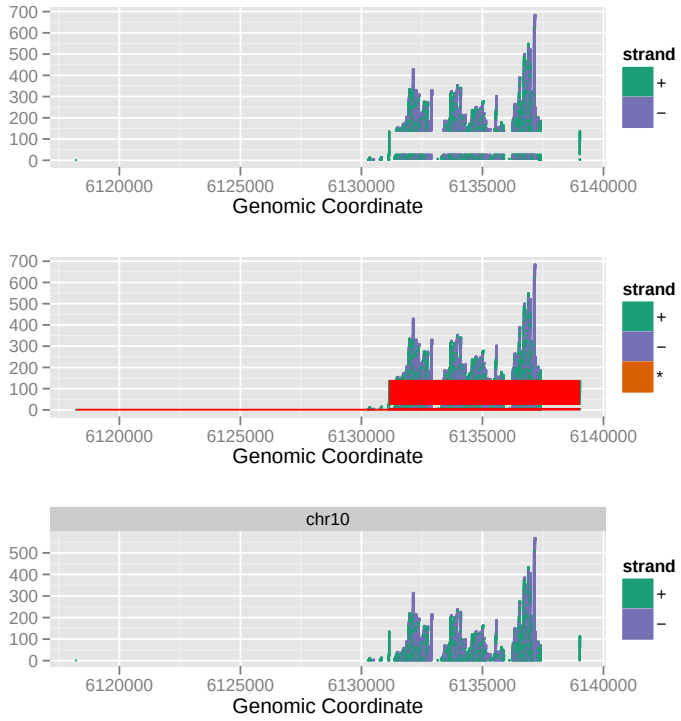

Figure 15: GappedAlignments visualization. The one on the top shows default geom "gapped.pair", which try to put the one belongs to the same read in the same step level, and make sure nothing falls in between, this is useful when show.junction = TRUE. The second shows links, the last one show as geom "full", which just show as formal GRanges.

3.7 For `BamFile` object

Supported geoms includes:

gapped.pair Like for `GappedAlignments` object.

full Like for `GappedAlignments` object.

coverage.line like for `GRanges` object

coverage.polygon like for `GRanges` object

mismatch.summary Mismatch summary, you could refer to the mismatch summary section.

```
> p1 <- qplot(bamfile, which = genesymbol["RBM17"], geom = "gapped.pair")
> p2 <- qplot(bamfile, which = genesymbol["RBM17"], geom = "gapped.pair",
+     show.junction = TRUE)
> p3 <- qplot(bamfile, which = genesymbol["RBM17"], geom = "full")
> p4 <- qplot(bamfile, which = genesymbol["RBM17"], geom = "coverage.line")
> p5 <- qplot(bamfile, which = genesymbol["RBM17"], geom = "coverage.polygon")
> tracks(p1, p2, p3, p4, p5)
```

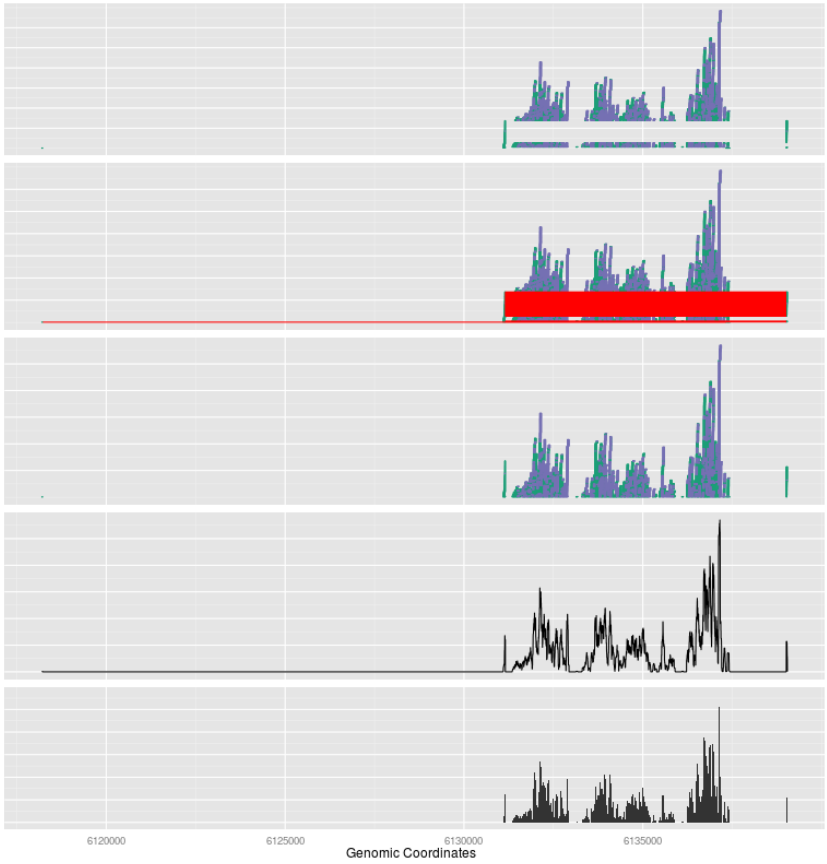

Figure 16: Tracks of different geom of bam file, in the order of gapped.pair, gapped.pair with junction, full, coverage.line, coverage.polygon, mismatch.summary.

3.8 For `TranscriptDb` object

Supported geoms including

full Full model, trying to show all 5'-UTR, 3'-UTR, cds and introns in different transcripts.

single Reduced model.

tx Transcirpts isoforms composed of exons.

Figure 17 shows two tracks for a `TranscriptDb` object.

```
> library(ggbio)
> data(genesymbol)
> pdf("intro-txdb-plot")
> p.full <- qplot(txdb, geom = "full", which = genesymbol["RBM17"])
> p.single <- qplot(txdb, geom = "single", which = genesymbol["RBM17"])
> p.tx <- qplot(txdb, geom = "tx", which = genesymbol["RBM17"])
> tracks(p.full, p.single, p.tx, heights = c(400, 100, 400))
> dev.off()
```

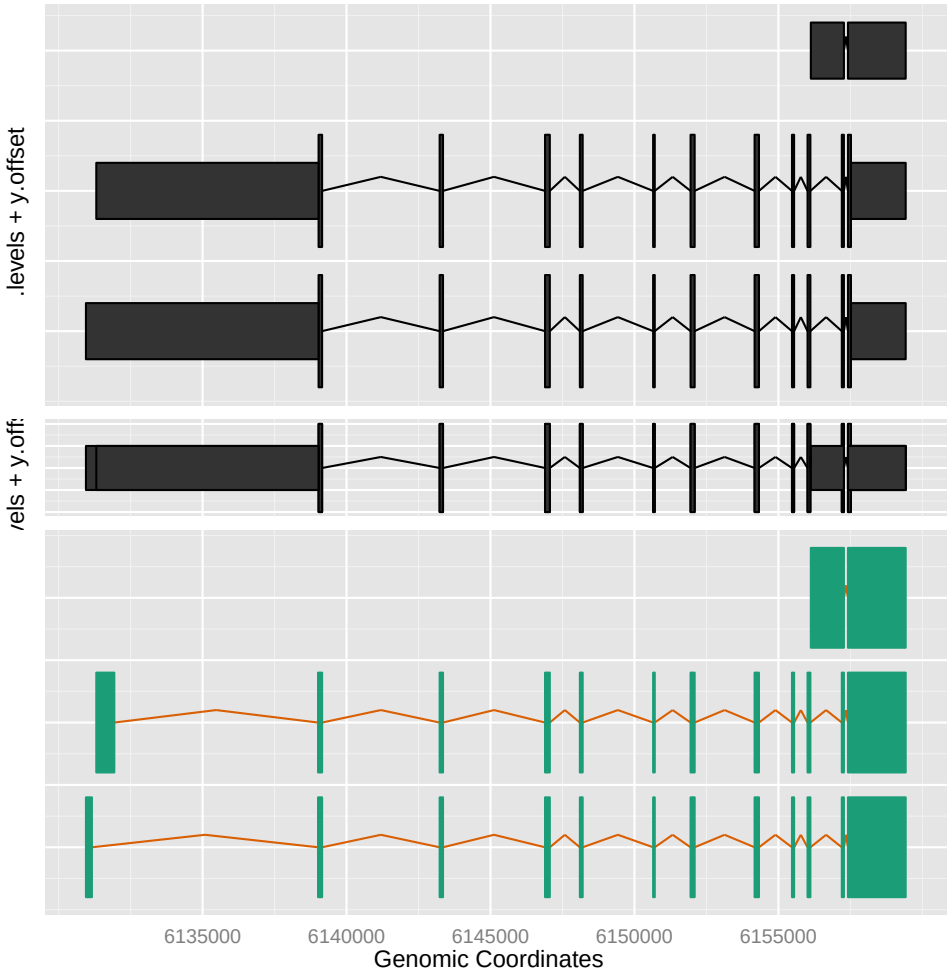

Figure 17: TranscriptDb visualization. First track showing full model, second track showing reduced one. and last one show exons in isoforms.

3.9 For BSgenome object

Package *BSgenome* provides infrastructure for Biostrings-based genome data packages. And users could download many full genome as provided by UCSC which stored in Biostrings objects from metadata database in Bioconductor project website.

So `qplot` function for this object tries to:

text Showing nucleotides as text.

segment Showing nucleotides as color-coded segment.

point Showing nucleotides as color-coded point.

rectangle Showing nucleotides as color-coded rectangle.

Figure 18 shows four tracks(with four geom) for a `BSgenome` object.

4 Overview Plot

Bird Eye Overview is useful to see the overall distribution of certain events. For static graphic, we currently only support stacked overview as ideogram, or for single chromosome.

4.1 Stacked Overview

Stacked overview is useful to visualize the annotation across the genome, you can use `plotOverview` function to directly plot the result from `getIdeogram` for certain species. And you could control whether to plot the cytoband or not.

Figure 19 shows how to plot stacked overview with cytoband. We change the name to make the label more clear. `renameSeqlevels` function from *GenomicRanges* is a good choice.

Clearly, it's not good for visualizing the annotation at the same time, so we could plot it without cytoband. This accept a full ideogram which will be reduced automatically. You could also just use **hg19Ideogram** dataset.

Then we could simply use `geom_hotregion` function to read in a `GRanges` object as other geoms(except they read in `data.frame`). And use + to simply add a annotation track on top with overview, they will automatically plot on the same chromosome and on the same x scale.

Figure 21 shows an example of subset of RNA editing set.

NOTE: user need to make sure the `seqnames` are consistency in two data, otherwise you are going to make weird graphics, because they can not be mapped in the right way.

We can also use `color` argument to use color to indicate a column in the elementMetadata.

Let's try to use color to indicate which exon region the editing site is, *3* means 3' UTR, *5* means 5' UTR, *C* means it's *CDS*, and **NA** indicate missing.

4.2 Single Chromosome Overview

Only support cytoband, used for those user who want to embed this as one of many tracks. One could draw red rectangle on it which indicate the zoomed region.

```
> p <- plotSingleChrom(hg19IdeogramCyto, subchr = "chr1")
> p.zoom <- plotSingleChrom(hg19IdeogramCyto, subchr = "chr1",
+                 zoom.region = c(1e8, 1.5e8))
>
> ## you could also run the following code to make nice adjusted
> ## overview
> ## vp1 <- viewport(width = 1, height = 0.14)
```

```
> library(BSgenome.Hsapiens.UCSC.hg19)
> gr <- GRanges("chr1", IRanges(5e7, 5e7+50))
> p1 <- qplot(Hsapiens, name = gr, geom = "text") +
+   theme_bw()
> p2 <- qplot(Hsapiens, name = gr, geom = "point") +
+     theme_bw()
> p3 <- qplot(Hsapiens, name = gr, geom = "segment") +
+     theme_bw()
> p4 <- qplot(Hsapiens, name = gr, geom = "rectangle") +
+     theme_bw()
> tracks(p1, p2, p3, p4)

NULL
```

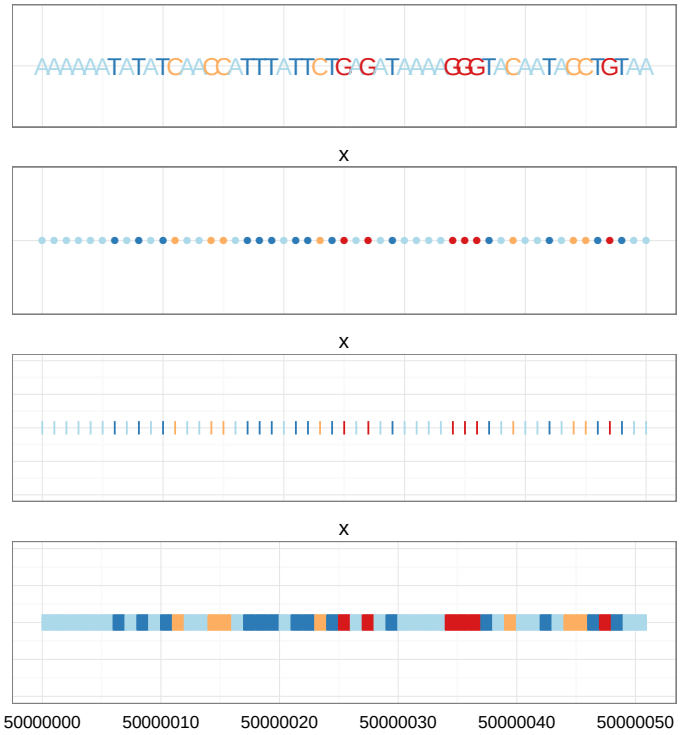

Figure 18: BSgenome with four geoms, first one is text, second is point, third one is segment, and fourth one is rectangle

```
> data(hg19IdeogramCyto)
> ## make shorter and clean labels
> old.chrs <- seqnames(seqinfo(hg19IdeogramCyto))
> new.chrs <- gsub("chr", "", old.chrs)
> ## lst <- as.list(new.chrs)
> names(new.chrs) <- old.chrs
> new.ideo <- renameSeqlevels(hg19IdeogramCyto, new.chrs)
> p <- plotOverview(new.ideo, cytoband = TRUE)
> print(p)
```

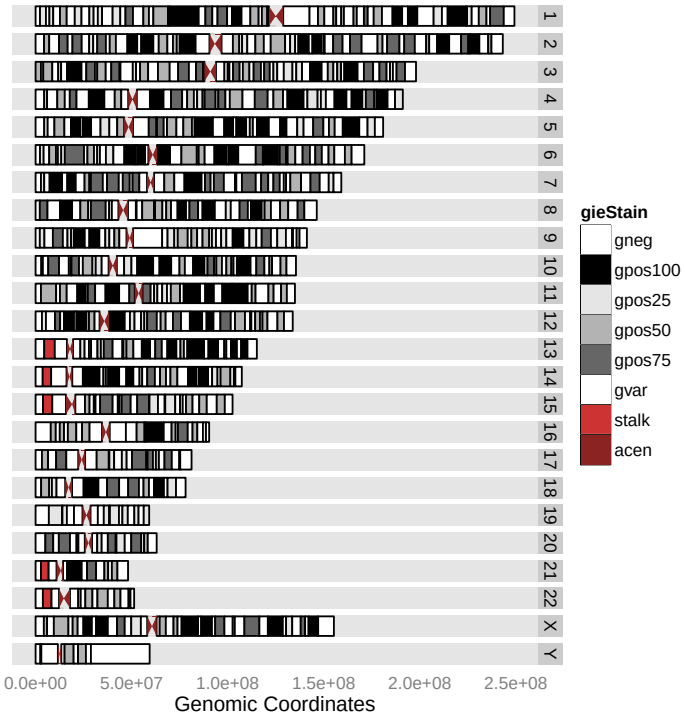

Figure 19: Stacked overview with cytoband

```
> p <- plotOverview(new.ideo, cytoband = FALSE)
> print(p)
```

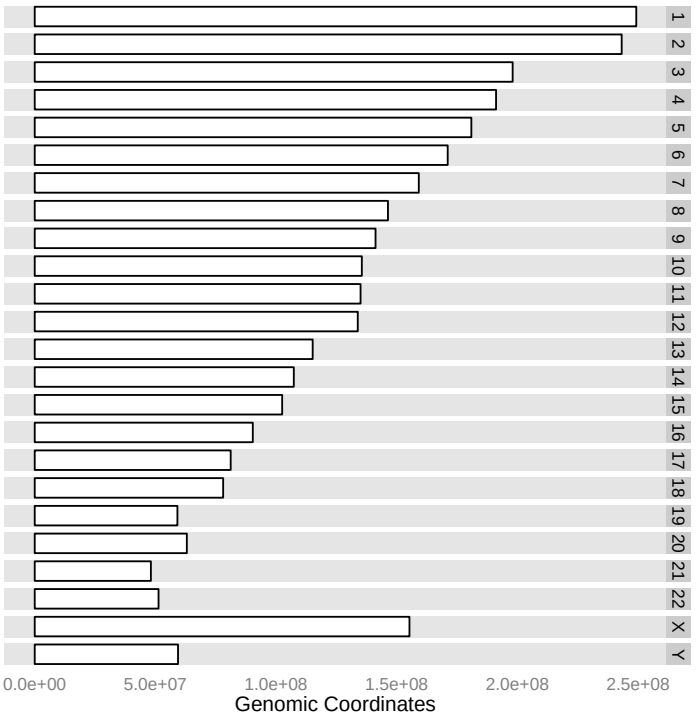

Figure 20: Stacked overview without cytoband

```
> data(darned_hg19_subset500)
> ## rename
> old.chrs <- seqnames(seqinfo(darned_hg19_subset500))
> new.chrs <- gsub("chr", "", old.chrs)
> names(new.chrs) <- old.chrs
> new.darned <- renameSeqlevels(darned_hg19_subset500, new.chrs)
> p <- p + geom_hotregion(new.darned)
> print(p)
```

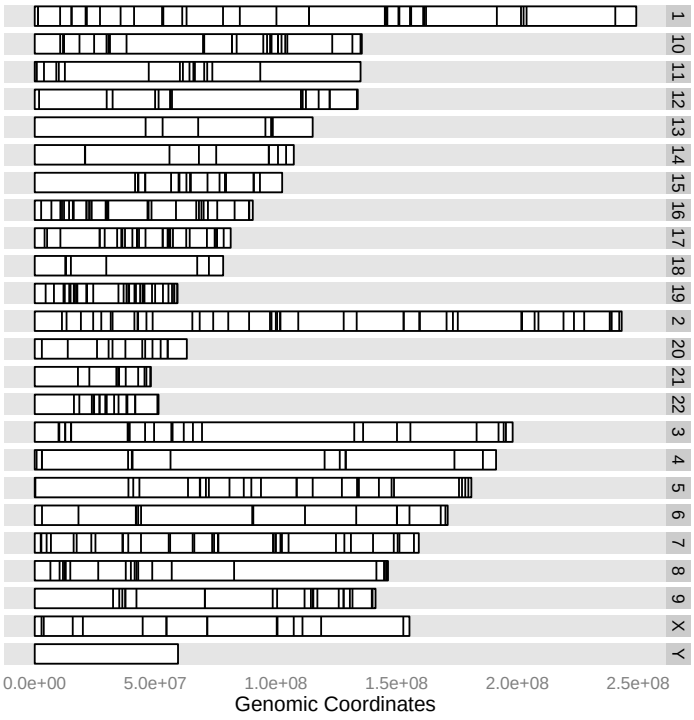

Figure 21: Stacked overview without cytoband and with subseted DARNED data on it

```
> p <- plotOverview(new.ideo, cytoband = FALSE)
> p <- p + geom_hotregion(new.darned, aes(color = exReg))
> print(p)
```

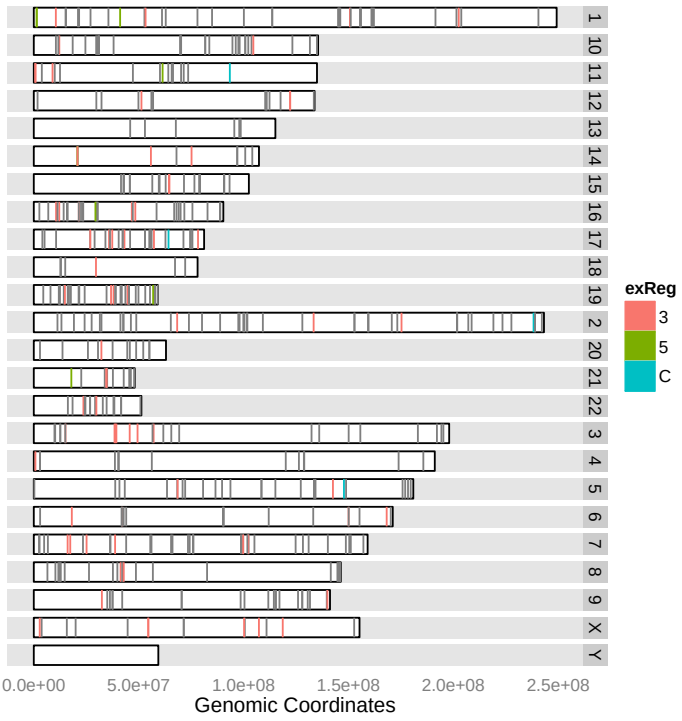

Figure 22: Stacked overview without cytoband and with subseted DARNED data on it

```
> ## p <- plotSingleChrom(hg19IdeogramCyto, subchr = "chr1")
> ## print(p, vp = vp1)
```

Figure 23: Single Chromosome as Ideogram. The top plot shows simple single chromosome, and the bottom one shows the one with fixed zoom window.

4.3 Grand Linear View and Manhattan Plots

Grand linear view is one of the useful overviews for the whole genome, we put chromosomes side by side along x-axis, and make it compact. Simply view the statistics across the genome and align multiple samples as different tracks.

A Manhattan plot is special case for grand linear view, one kind of scatter plot used to visualize data with a large number of data points, with a distribute of some higher-magnitude values. For example, in the GWAS(genome-wide association studies). Here we mainly focus on GWAS Manhattan plots. X-axis is genomic coordinates and Y-axis is negative logarithm of the associated P-value for each single nucleotide polymorphism. So higher the value, more stronger the association they are.

But we support both *point* and *line* geom.

Here we simulate some SNP data, as Figure 24shows, you can plot it in different ways, and use different tricks.

Default is assigning color for all chromosomes, and use a free space. Users can have more delicate control such as adding title, using customized two-color scheme, and adding cut off, removing legend.. Please refer to the manual.

One of the most important features is multiple sample comparison, suppose you have extra columns contains categorical data, you can facet easily by certain factor as shown if Figure25. More delicate control like change color, size, alpha blending, please refer to the manuals.

4.4 Circular Overview

Circular view is inspired by the Circos project [2] which is essentially written Perl[3].Circos visualize data in a circular layout, originally starting from visualize the genomic data, then extends to many other fields, turn out to be an elegant and useful way to visualize some other information.

The static version of circular view is not implemented in this package yet, but it's definitely in the TODO. For users who are really interested in using a circular view in R, we have a highly experimental circular view in another package *visnab*, which is interactive visualization toolkit for genomic data.

5 Building Tracks for Linear View

In most genome browsers, they all have such a view that including many tracks, could be any annotation data along genomic coordinate. So we try to provide a convenient constructor for building tracks, which here in this package is simply vertically binding of several plots. It's essentially a `grid.arrange`. So if users want to have more delicate control over their tracks, they need manipulate the graphics in *ggplot2* level or grid levels.

`tracks` function has some extra features and limitations compare to `grid.arrange`.

- Always sitting on genomic or protein space.

[2] `http://circos.ca/`
[3] `http://www.perl.org/`

```
> data(hg19Ideogram)
> chrs <- as.character(levels(seqnames(hg19IdeogramCyto)))
> seqlths <- seqlengths(hg19Ideogram)[chrs]
> set.seed(1)
> nchr <- length(chrs)
> nsnps <- 500
> gr.snp <- GRanges(rep(chrs,each=nsnps),
+               IRanges(start = do.call(c, lapply(chrs, function(chr){
+                 N <- seqlths[chr]
+                 runif(nsnps,1,N)
+               })), width = 1),
+               SNP=sapply(1:(nchr*nsnps), function(x) paste("rs",x,sep='')),
+               pvalue =  -log10(runif(nchr*nsnps)),
+                    group = sample(c("Normal", "Tumor"), size = nchr*nsnps,
+                      replace = TRUE)
+               )
> ## processing the name, to make it shorter
> nms <- seqnames(seqinfo(gr.snp))
> nms.new <- gsub("chr", "", nms)
> names(nms.new) <- nms
> gr.snp <- renameSeqlevels(gr.snp, nms.new)
> p1 <- plotGrandLinear(gr.snp, y = pvalue, geom = "point")
> p2 <- plotGrandLinear(gr.snp, y = pvalue, geom = "point", color.type = "identity",
+                 color = I("black"), size = I(1))
> ## remove ticks and laebels and set cutoff
> p3 <- plotGrandLinear(gr.snp, y = pvalue, geom = "point", cutoff = 4) +
+             opts(axis.text.x = theme_blank(),
+                  axis.ticks=theme_blank())
> grid.arrange(p1, p2, p3, ncol = 1)
```

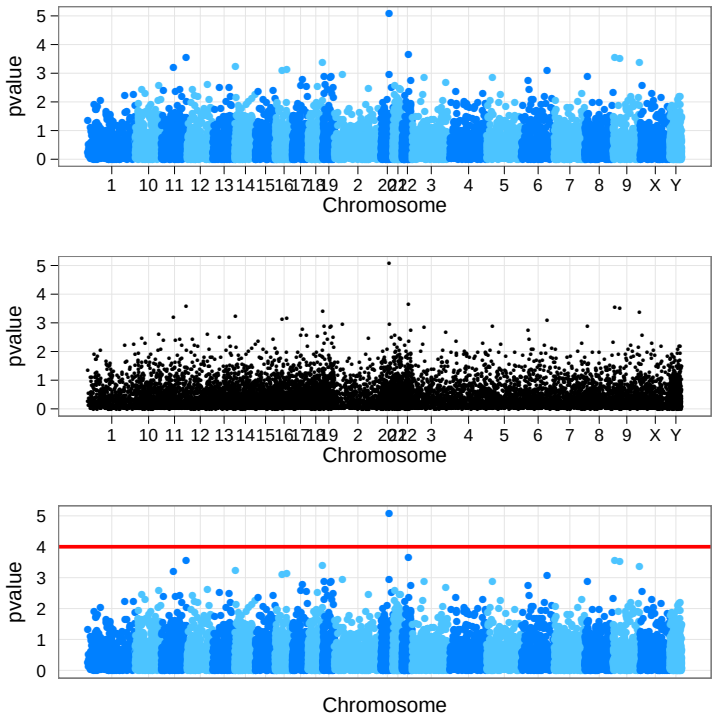

Figure 24: Grand linear view for simulated SNP data.

```
> p <- plotGrandLinear(gr.snp, y = pvalue, geom = "point",
+                      facet = group ~ .,  color.type = "twocolor")
> print(p)
```

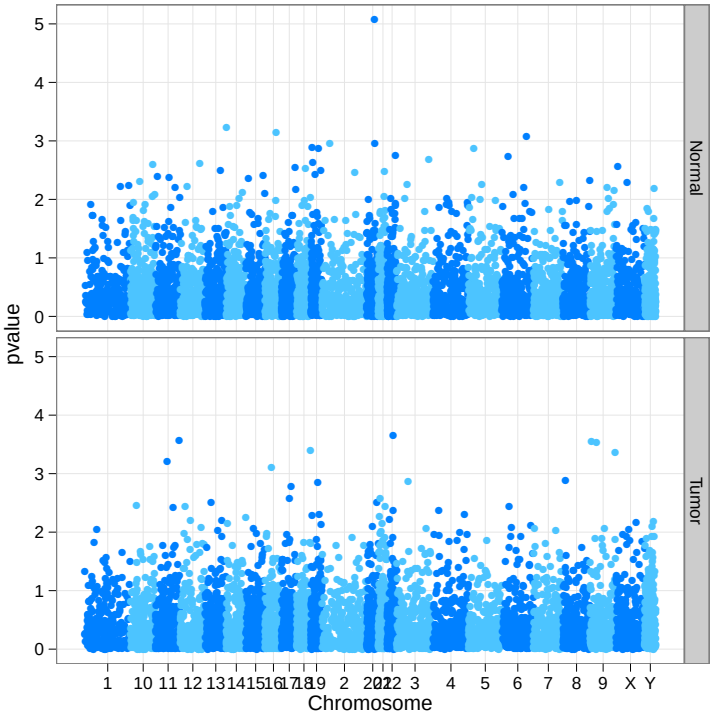

Figure 25: Grand linear view for simulated SNP data.

- Always using ncol = 1 as default arguments.
- For now, since the unbalanced legend and labels in *ggplot2* has been solved (maybe just I haven't found such features). We simply remove legend and y axis labels to make sure all tracks are aligned exactly in the same way.
- Remove the x-axis for most track except the last one.
- Does the adjustment of margins for you automatically.
- Doesn't like `qplot`, tracks doesn't return `ggplot` object. so processing your plot before you pass them to `tracks`.

NOTICE:

- Tracks cannot guarantee all tracks are aligned well in all the cases simply because control of grobs sometimes are tricky and based on what users passed. Like theme change, or uneven labels or legend could affact the alignments. Even though I tried hard to align them in most cases.
- When y axis text length are not equal across tracks, it's hard to align. So you will see small wiggles. Even missing y label would affect the accurate alignments too.
- So by default, we remove y axis text and legend to make sure they are aligned exactly at the same position. Clearly this is not a good practice.

We will try our best to fix this restriction in the future release. So advanced users could use *gridExtra* package to modify the graphics before passing to `grid.arrange`.

6 Question-oriented Specific Graphics

In package *ggbio*, We also make individual graphic for special case instead of using a generic function `qplot`, when it meets following conditions:

- This function doesn't return a ggplot object. Usually it's just composed of multiple tracks, has some special control over the layout.
- The API is too different from `qplot`.
- The object is not quite clear, or we don't want it to dispatch on any other objects.
- In some cases, we may include one special geom in the qplot function for certain graphic.

These kind of graphics may be used widely, so we hope to provides a set of utilities to generate those graphics.

6.1 Splicing Summary

As we introduced before, `qplot` could plot the `GRangesList` as slicing models, but here we make another function which could read RNA-seq bam file directly and summarize the junction read, then plot the result directly as graphic.

Internally we use biovizBase:::spliceSummary for simple counting by overlapped junction reads which cover consecutive two exons of each model in weighted or unweighted way, but we encourage users to use their own robust way to make splicing summary and store it as GRangesList, then plot the summary by `qplot` function. Please check the previous section.

```
> p1 <- plotSpliceSum(bamfile, exons.rbm17)
> ## all more control
> p2 <- plotSpliceSum(bamfile, exons.rbm17, weighted = FALSE, offset = 0.01)
> p3 <- plotSpliceSum(bamfile, txdb, which = genesymbol["RBM17"],
+                     show.label = TRUE,
+                     label.type = "count")
> print(p1)
```

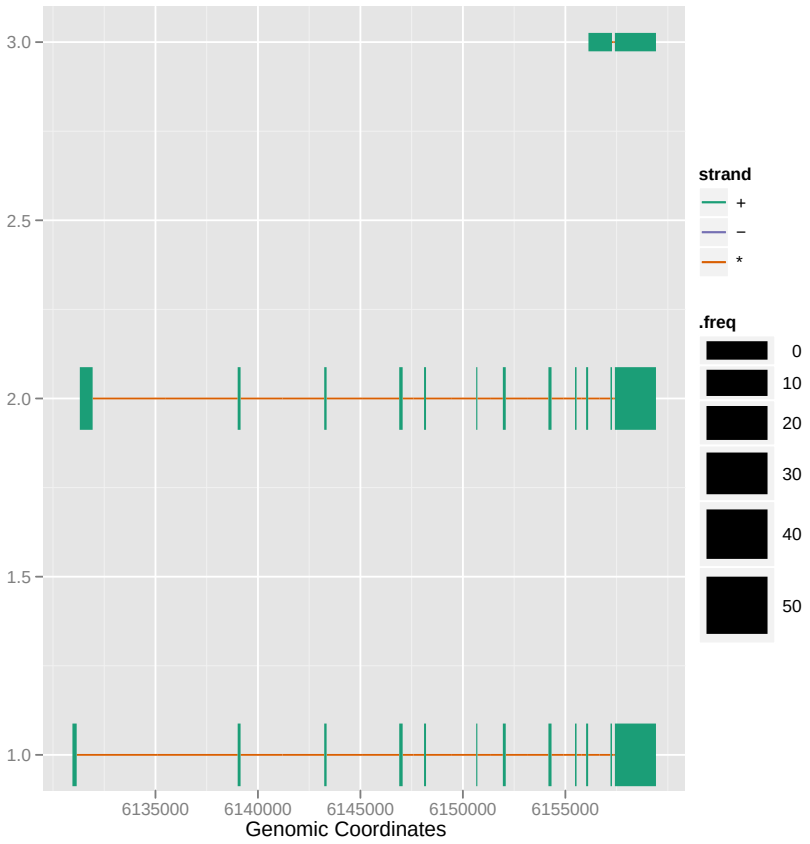

Figure 26: Plot for splicing summary, intervally converted to GRangesList. the height of the rectangle shows the supported evidence.

6.2 Fragment Length for paired-end RNA-seq

Here is just a simple example to plot estimated fragment length, the algorithm could be various, you could use your own robust way to make you own fragment length plot in a similar way.

The idea of graphics is originally from Melanie Huntley and her colleges, I can not say for sure it's implemented in exactly the same way as she described, because this package is now well tested yet. But here is the idea, we collect all paired reads and model, reduce model(exons) first, then find common gaps between exons and reads, remove common gaps between paired-end reads, and compute the new estimated fragment length.

The graphic is implemented in function `plotFragLength`, which read in a paired-end RNA-seq bam file and give the graphics as shown in Figure27 (with annotations) and Figure28 (without annotation) and Figure29 (after shrinking) which is a compact view.

NOTE: the "bamfile" used in the example is from different source, which contains pair-end information. So plese make sure your bamfile contains paired-end reads.

```
> model <- exonsBy(txdb, by = "tx")
> model.new <- subsetByOverlaps(model, genesymbol["RBM17"])
> exons.rbm17 <- subsetByOverlaps(exons(txdb), genesymbol["RBM17"])
> exons.new <- reduce(exons.rbm17)
> ## the graphic shows here is using different bamfile from the one
> ## in ext, it contains more pair-end reads.
> plotFragLength(bamfile, exons.new, geom = c("point","segment"))
> plotFragLength(bamfile, exons.new, geom = c("point","segment"),
+                       annotation = FALSE)
> plotFragLength(bamfile, exons.new, geom = c("point","segment"),
+                       type = "cut", gap.ratio = 0.001)
```

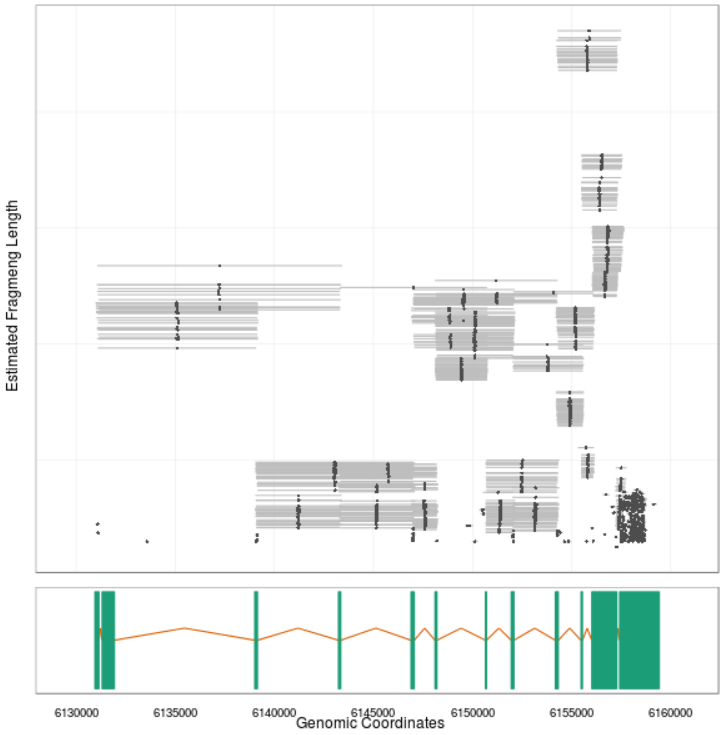

Figure 27: Fragment length with annotation. y value of "point" and "line" indicate the estimated fragment length. and if geom is set to "segment", the segment is from the left most position to paired right most position, should be equal to "isize".

6.3 Mismatch Summary

For NGS(Next generation sequencing) data, sometimes instead of showing all the alignments, we hope to show just summary information for particular region. And mismatch summary is one of those interesting

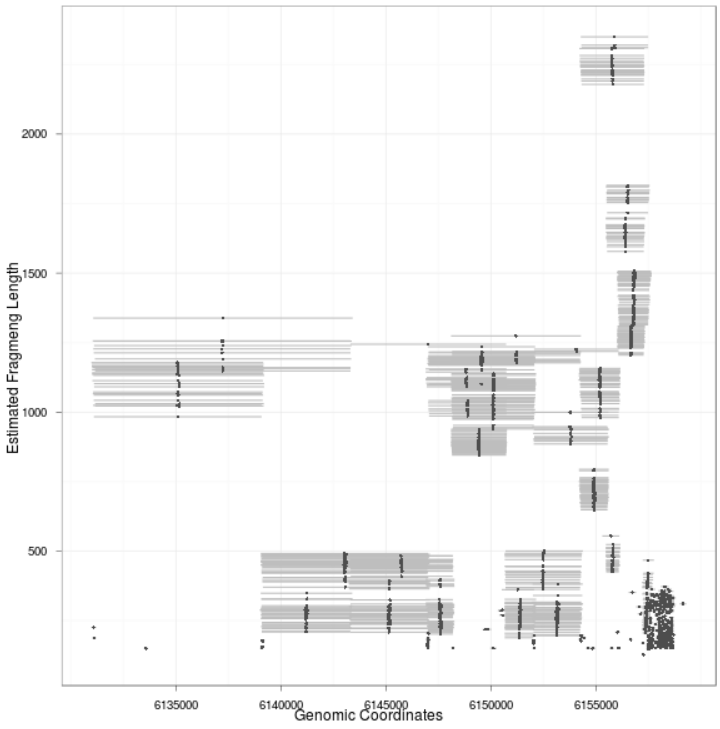

Figure 28: Fragment length without annotation, but showing y scales.

summaries.

`plotMismatchSum` shows only mismatched read counts as color coded stacked bar chart or with coverage as background. We also implement this as one geom for certain object, such `BamFile` and `GRanges`, so you could use `qplot` for simply generating some mismatch summary. When the object is `GRanges`. we check inside the function to see if the `GRanges` is from `pileupGRangesAsVariantTable` or not, or it need to contains certain column required for plotting mismatch summary.

we can also use generic `qplot` function.

```
> p <- qplot(test.match, geom = "mismatch.summary")
> library(Rsamtools)
> ## for character
> p <- qplot(bamfile, which = gr, bsgenome = Hsapiens,
+       geom = "mismatch.summary", show.coverage = TRUE)
> ## for BamFile
> p <- qplot(BamFile(bamfile), which = gr, bsgenome = Hsapiens,
+       geom = "mismatch.summary", show.coverage = TRUE)
```

6.4 Plot Ranges Linked with Data

Inspired by some graphics produced in some other packages, for example in package *DEXseq*, the author provides graphics with gene models and linked to an even spaced statistics summary. This is useful because we always plot everything along the genomic coordinates, but genomic features like exons are not evenly distributed, so we could actually treat the statistics associated with exons like categorical data, and show them as *Parallel Coordinates Plots*. This is one special layout which represent the data in a nice manner and

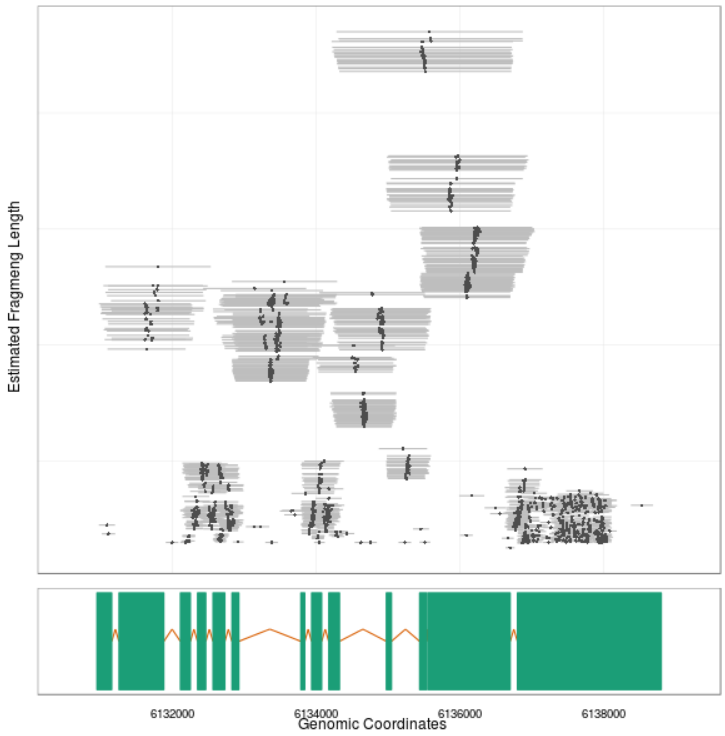

Figure 29: Fragment length with annotation after shrinking

also keep the genomic structure information. With ability of `tracks`, it's possible to generate such type of a graphic along with other annotations.

The data we want is a normal `GRanges` object, and make sure the intervals are not overlapped with each other(currently), and you may have multiple columns which store the statistics for multiple samples. then we produce the graphic we introduced above and users could pass other annotation track in the function which will be shown below it.

The reason you need to pass annotation into the function instead of binding them by `tracks` later is because binding manually with annotation tracks is tricky and this function doesn't return a ggplot object.

Here are some examples for function `plotRangesLinkedToData`.

```
> exons <- exons(txdb)
> exon17 <- subsetByOverlaps(exons, genesymbol["RBM17"])
> ## reduce to make sure there is no overlap
> ## just for example
> exon.new <- reduce(exon17)
> p <- qplot(model.new)
> ## simulated data
> values(exon.new)$sample1 <- rnorm(length(exon.new), 10, 3)
> values(exon.new)$sample2 <- rnorm(length(exon.new), 10, 10)
> values(exon.new)$score <- rnorm(length(exon.new))
> plotRangesLinkedToData(exon.new, stat.col = c("sample1", "sample2"))
> ## show as figure, no annotation
> ## the same as
> ## plotRangesLinkedToData(exon.new, stat.col = 1:2)
```

```
> gr <- GRanges("chr10", IRanges(6134000, 6135000))
> test <- pileupAsGRanges(bamfile, region = gr)
> test.match <- pileupGRangesAsVariantTable(test, Hsapiens)
> ## use plotMismatchSum directly
> pmis1 <- plotMismatchSum(test.match, show.coverage = FALSE)
> pmis2 <- plotMismatchSum(test.match, show.coverage = TRUE)
> grid.arrange(pmis1, pmis2, ncol = 1)
```

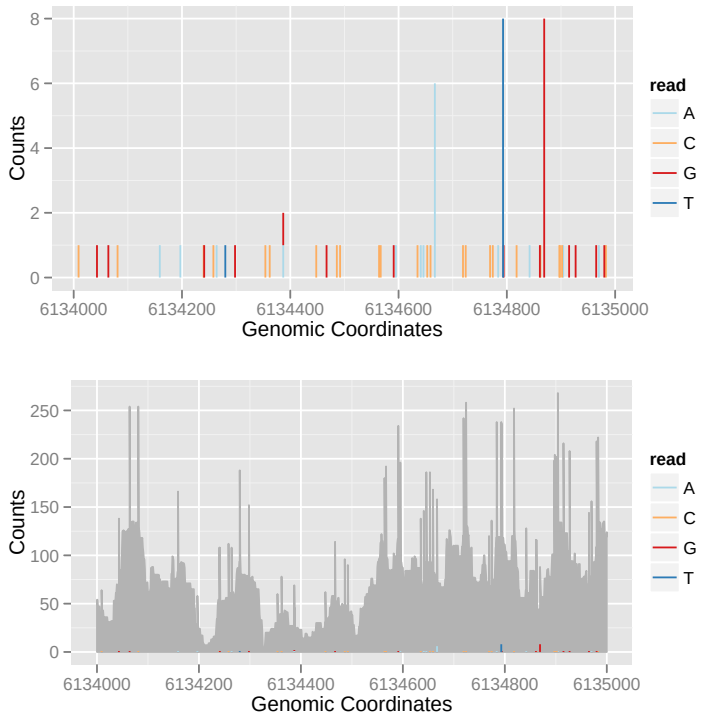

Figure 30: Mismatch summary: The top one shows only mismatched counts without showing coverage, and the bottom one shows coverage in gray color as background. For each position, reads that as same as reference will be shown as gray. And we only color the reads that is mismatch from the reference and color-coded by default.

```
> ## show as figure, adding annotation
```

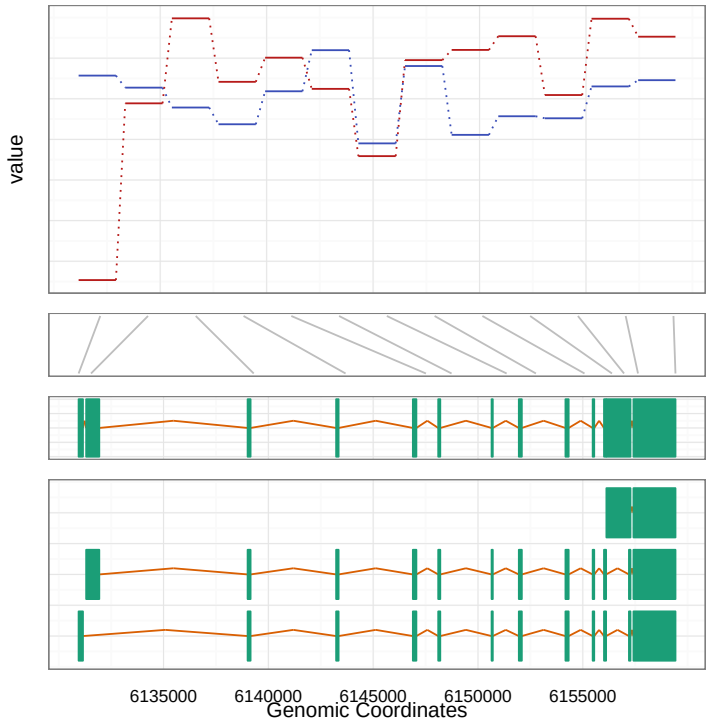

Figure 31: Single Granges object for multiple samples

```
> plotRangesLinkedToData(exon.new, stat.col = 1:2, annotation = list(p))
```

Figure 31 shows, A GRanges with two samples as column. and Figure 32 show the same graphic instead of containing other annotation tracks.

TODO

- Due to limitation of tracks, if showing legend and y axis text, can not align them well currently, will find appropriate solution later.

7 Bugs Report and Features Request

Latest code are available on github `https://github.com/tengfei/ggbio`

Please file bug/request on issue page, this is preferred way. or email me at yintengfei <at> gmail dot com.

It's a new package and under active development.

Thanks in advance for any feedback.

8 Acknowledgement

I wish to thank all those who helped me. Without them, I could not have started this project.

Genentech Sponsorship and valuable feed back and help for this project and my other project.

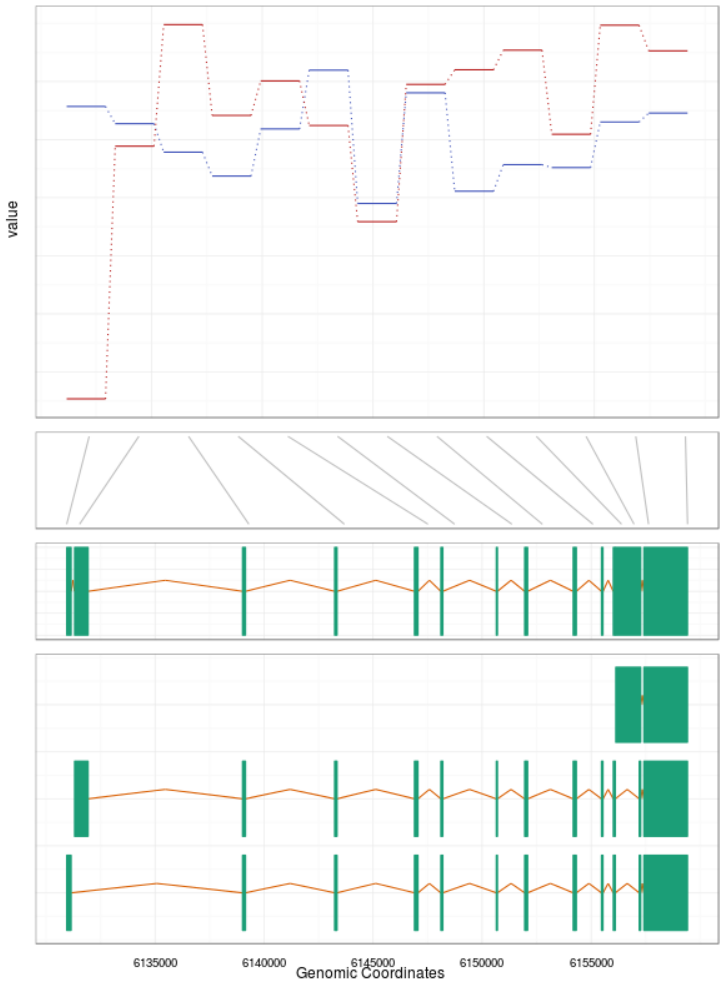

Figure 32: Single Granges object for multiple samples with annotation track.

9 Session Info

```
> sessionInfo()

R version 2.14.1 (2011-12-22)
Platform: i386-apple-darwin9.8.0/i386 (32-bit)

locale:
[1] C/en_US.US-ASCII/en_US.US-ASCII/C/en_US.US-ASCII/en_US.US-ASCII

attached base packages:
[1] grid      stats     graphics  grDevices utils     datasets
[7] methods   base

other attached packages:
 [1] BSgenome.Hsapiens.UCSC.hg19_1.3.17
 [2] BSgenome_1.22.0
 [3] Rsamtools_1.6.3
 [4] Biostrings_2.22.0
 [5] gridExtra_0.9
 [6] GenomicRanges_1.6.6
 [7] IRanges_1.12.5
 [8] ggbio_1.0.4
 [9] ggplot2_0.9.0
[10] biovizBase_1.0.4

loaded via a namespace (and not attached):
 [1] AnnotationDbi_1.16.11 Biobase_2.14.0        DBI_0.2-5
 [4] GenomicFeatures_1.6.7 MASS_7.3-16           RColorBrewer_1.0-5
 [7] RCurl_1.9-5           RSQLite_0.11.1        XML_3.9-4
[10] biomaRt_2.10.0        bitops_1.0-4.1        colorspace_1.1-1
[13] dichromat_1.2-4       digest_0.5.1          memoise_0.1
[16] munsell_0.3           plyr_1.7.1            proto_0.3-9.2
[19] reshape2_1.2.1        rtracklayer_1.14.4    scales_0.1.0
[22] stringr_0.6           tools_2.14.1          zlibbioc_1.0.0
```