Solexa QA report

November 10, 2008

1 Overview

This document provides a quality assessment of Genome Analyzer results. The assessment is meant to complement, rather than replace, quality assessment available from the Genome Analyzer and its documentation. The narrative interpretation is based on experience of the package maintainer. It is applicable to results from the 'Genome Analyzer' hardware single-end module, configured to scan 300 tiles per lane. The 'control' results refered to below are from analysis of φX-174 sequence provided by Illumina.

An R script containing the code used in this document can be created with

```
> fl <- system.file("template", "qa_solexa.Rnw", package = "ShortRead")
> Stangle(fl)
```

2 Run summary

Read counts. Filtered and aligned read counts are reported relative to the total number of reads (clusters). Consult Genome Analyzer documentation for official guidelines. From experience, very good runs of the Genome Analyzer 'control' lane result in 6-8 million reads, with up to 80% passing pre-defined filters.

```
> ppnCount(qa[["readCounts"]])
```

	read	filtered	aligned
s_1_1_export.txt	3668433	0.621	0.521
s_2_1_export.txt	4230424	0.766	0.655
s_3_1_export.txt	4003465	0.772	0.430
s_4_1_export.txt	4521919	0.762	0.569
s_6_1_export.txt	4004807	0.781	0.496
s_7_1_export.txt	3546869	0.771	0.386
s_8_1_export.txt	4232977	0.778	0.562
s_5_1_export.txt	2842633	0.844	0.816

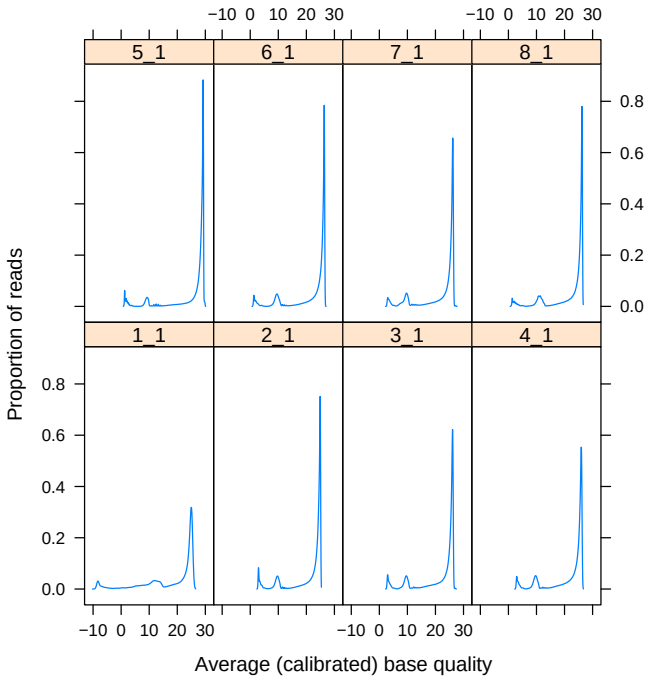
Base call frequency over all reads. Base frequencies should accurately reflect the frequencies of the regions sequenced.

```
> qa[["baseCalls"]]/rowSums(qa[["baseCalls"]])
```

	A	C	G	T	N
s_1_1_export.txt	0.241	0.225	0.207	0.238	0.0893
s_2_1_export.txt	0.245	0.235	0.229	0.232	0.0598
s_3_1_export.txt	0.242	0.244	0.237	0.223	0.0546
s_4_1_export.txt	0.244	0.243	0.236	0.226	0.0509
s_6_1_export.txt	0.237	0.254	0.248	0.213	0.0478
s_7_1_export.txt	0.248	0.249	0.238	0.219	0.0453
s_8_1_export.txt	0.234	0.253	0.245	0.222	0.0452
s_5_1_export.txt	0.284	0.211	0.205	0.251	0.0490

Overall read quality. Lanes with consistently good quality reads have strong peaks at the right of the panel.

```
> df <- qa[["readQualityScore"]]
> print(plotReadQuality(df[df$type == "read", ]))
```



3 Read distribution

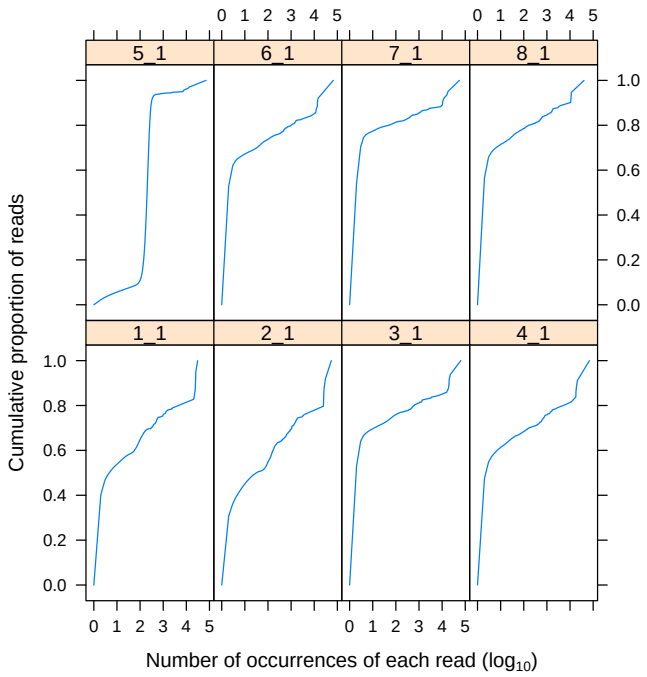
These curves show how coverage is distributed amongst reads. Ideally, the cumulative proportion of reads will transition sharply from low to high.

Portions to the left of the transition might correspond roughly to sequencing or sample processing errors, and correspond to reads that are represented relatively infrequently. 10-15% of reads in a typical Genome Analyzer 'control' lane fall in this category.

Portions to the right of the transition represent reads that are over-represented compared to expectation. These might include inadvertently sequenced primer or adapter sequences, sequencing or base calling artifacts (e.g., poly-A reads), or features of the sample DNA (highly repeated regions) not adequately removed during sample preparation. About 5% of Genome Analyzer 'control' lane reads fall in this category.

Broad transitions from low to high cumulative proportion of reads may reflect sequencing bias or (perhaps intentional) features of sample preparation resulting in non-uniform coverage. the transition is about 5 times as wide as expected from uniform sampling across the Genome Analyzer 'control' lane.

```
> df <- qa[["sequenceDistribution"]]
> print(plotReadOccurrences(df[df$type == "read", ], cex = 0.5))
```



Common duplicate reads might provide clues to the source of over-represented sequences. Some of these reads are filtered by the alignment algorithms; other duplicate reads migth point to sample preparation issues.

```
> freqSequences(qa, "read")
```

```
                                   sequence count             lane
1051 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA 70947 s_5_1_export.txt
451  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA 69116 s_4_1_export.txt
601  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA 66776 s_6_1_export.txt
301  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA 63998 s_3_1_export.txt
751  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA 55729 s_7_1_export.txt
151  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA 54828 s_2_1_export.txt
901  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA 40359 s_8_1_export.txt
1    ANNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 30880 s_1_1_export.txt
152  ANNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 30485 s_2_1_export.txt
153  CNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 26476 s_2_1_export.txt
2    TNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 25600 s_1_1_export.txt
154  GNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 25594 s_2_1_export.txt
3    CNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 25063 s_1_1_export.txt
155  TNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 24965 s_2_1_export.txt
4    GNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 24164 s_1_1_export.txt
302  ANNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 22501 s_3_1_export.txt
5    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA 20996 s_1_1_export.txt
452  TNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 20842 s_4_1_export.txt
303  CNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 20227 s_3_1_export.txt
304  GNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN 19845 s_3_1_export.txt
```

Common duplicate reads after filtering

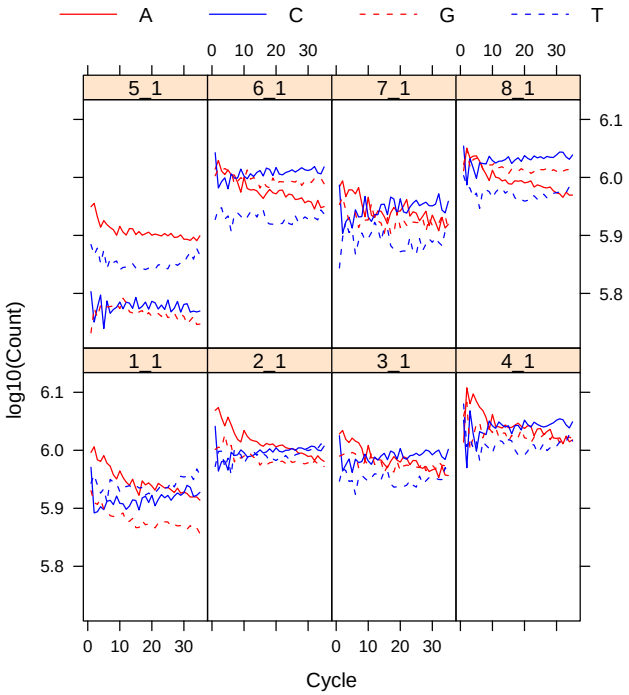
```
> freqSequences(qa, "filtered")
```

```
                                   sequence count             lane
1151 GACGTTTGGTCAGTTCCATCAACATCATAGCCAGA   439 s_5_1_export.txt
1152 CAAATGACGACTTCTACCACATCTATTGACATTAT   438 s_5_1_export.txt
1153 CGAAAGACCAGGTATATGCACAAAATGAGATGCTT   435 s_5_1_export.txt
1154 AAGTAAAGGACGGTTGTCAGCGTCATAAGAGGTTT   433 s_5_1_export.txt
1155 ATTTGGACTGCTCCGCTTCCTCCTGAGACTGAGCT   419 s_5_1_export.txt
1156 AAATGACGACTTCTACCACATCTATTGACATTATG   416 s_5_1_export.txt
1157 GACTTCTACCACATCTATTGACATTATGGGTCTGC   413 s_5_1_export.txt
1158 ATCATAGCCAGATGCCCAGAGATTAGAGCGCATGA   410 s_5_1_export.txt
1159 AACATCATAGCCAGATGCCCAGAGATTAGAGCGCA   409 s_5_1_export.txt
1160 TGATGAACTAAGTCAACCTCAGCACTAACCTTGCG   402 s_5_1_export.txt
1161 TGCTCCGCTTCCTCCTGAGACTGAGCTTTCTCGCC   401 s_5_1_export.txt
1162 AAGCTGCTTATGCTAATTTGCATACTGACCAAGAA   398 s_5_1_export.txt
1163 TGCGTAACCGTCTTCTCGTTCTCTAAAAACCATTT   398 s_5_1_export.txt
1164 AACGACGTTTGGTCAGTTCCATCAACATCATAGCC   397 s_5_1_export.txt
1165 GTTTGGTCAGTTCCATCAACATCATAGCCAGATGC   394 s_5_1_export.txt
1166 AAGGTGATGTGCTTGCTACCGATAACAATACTGTA   391 s_5_1_export.txt
1167 GACCAGAAAACTGGCCTAACGACGTTTGGTCAGTT   385 s_5_1_export.txt
1168 GACGACTTCTACCACATCTATTGACATTATGGGTC   385 s_5_1_export.txt
1169 GCCGTGGATGCCTGACCGTACCGAGGCTAACCCTA   383 s_5_1_export.txt
1170 AAAGACCAGGTATATGCACAAAATGAGATGCTTGC   381 s_5_1_export.txt
```

4 Cycle-specific base calls and read quality

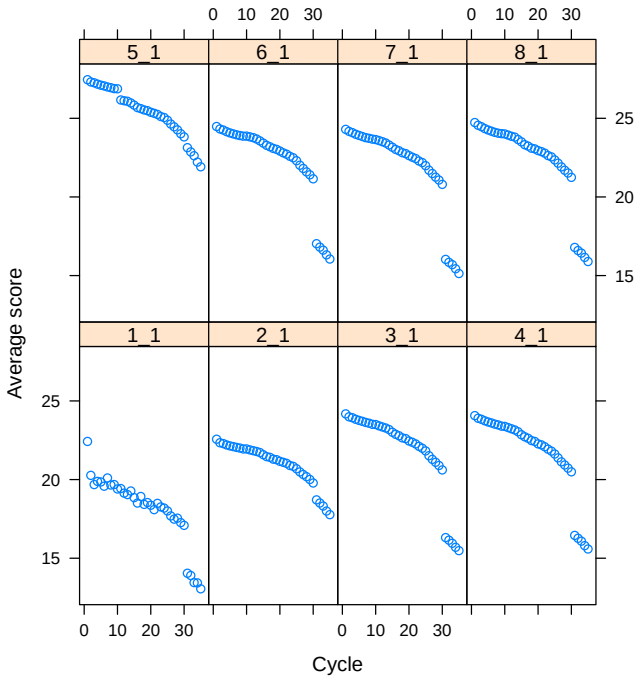
Per-cycle base call should usually be approximately uniform across cycles. Genome Analyzer 'control' lane results often show a deline in A and increase in T as cycles progress. This is likely an artifact of the underlying technology.

```
> perCycle <- qa[["perCycle"]]
> print(plotCycleBaseCall(perCycle$baseCall))
```



Per-cycle quality score. Reported quality scores are 'calibrated', i.e., incorporating phred-like adjustments following sequence alignment. These typically decline with cycle, in an accelerating manner. Abrupt transitions in quality between cycles toward the end of the read might result when only some of the cycles are used for alignment: the cycles included in the alignment are calibrated more effectively than the reads excluded from the alignment.

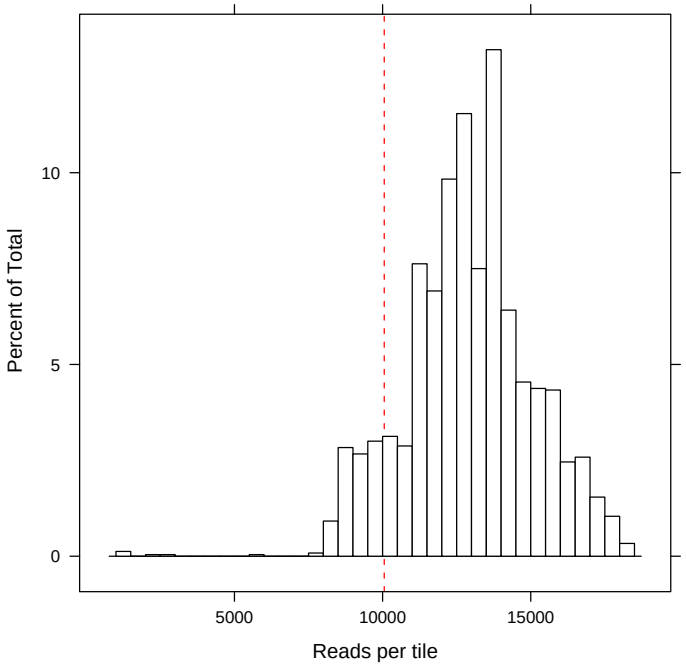
```
> print(plotCycleQuality(perCycle$quality))
```



5 Tile performance

Counts per tile. Dashed red line indicates the 10% of tiles with fewest reads. An approximately uniform distribution suggests consistent read representation in each tile. Distinct separation of 'good' versus poor quality tiles might suggest systematic failure, e.g., of many tiles in a lane, or excessive variability (e.g., due to unintended differences in sample DNA concentration) in read number per lane.

```
> perTile <- qa[["perTile"]]
> readCnt <- perTile[["readCounts"]]
> cnts <- readCnt[readCnt$type == "read", "count"]
> print(histogram(cnts, breaks = 40, xlab = "Reads per tile", panel = function(x,
+     ...) {
+     panel.abline(v = quantile(x, 0.1), col = "red", lty = 2)
+     panel.histogram(x, ...)
+ }, col = "white"))
```

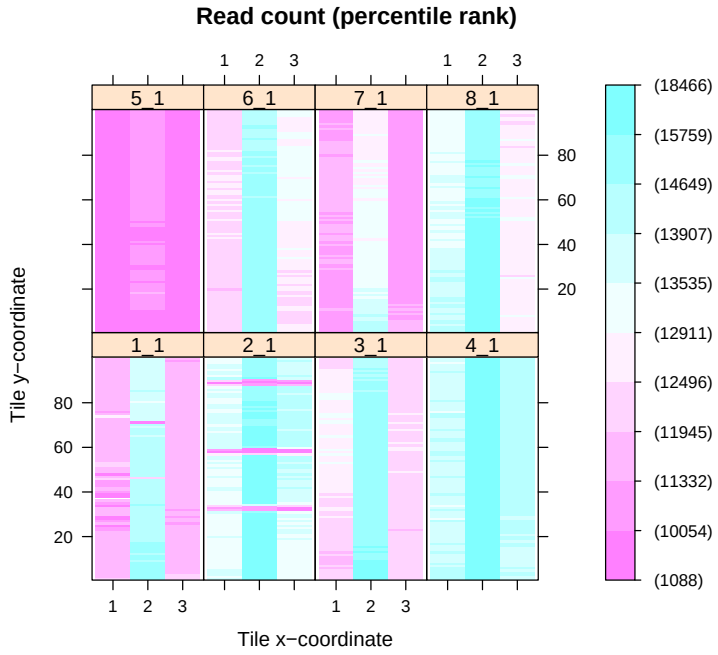


Spatial counts per tile. Divisions on the color scale are quantized, so that the range of counts per tile is divided into 10 equal increments. Parenthetic numbers on the scale represent the break points of the quantized values. Because the scale is quantized, some tiles will necessarily have 'few' reads and other necessarily 'many' reads.

Consistent differences in read number per lane will result in some lanes being primarily one color, other lanes primarily another color. Genome Analyzer data typically have greatest read counts in the center column of each lane. There are usually consistent gradients from 'top' to 'bottom' of each column.

Low count numbers in the same tile across runs of the same flow cell may indicate instrumentation issues.

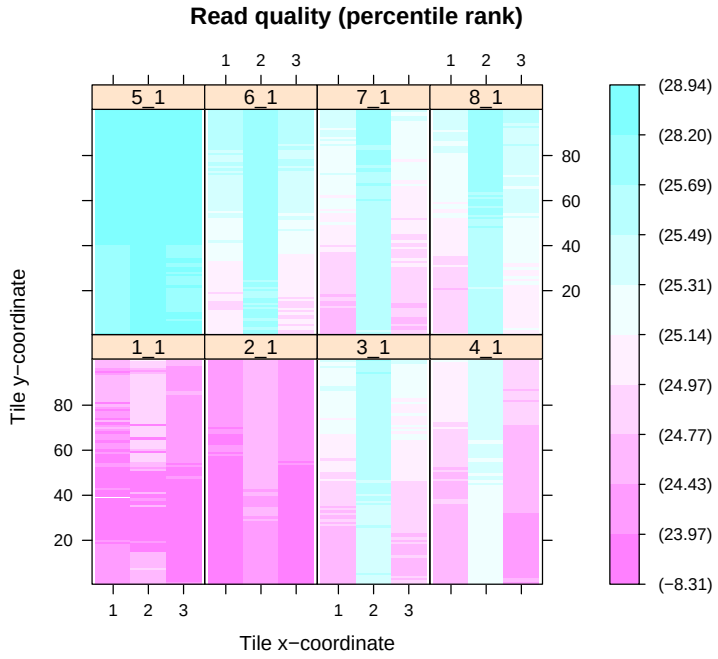
```
> print(plotTileCounts(readCnt[readCnt$type == "read", ]))
```



Median read quality score per tile. Divisions on the color scale are quantized, so that the range of average quality scores per tile is divided into 10 equal increments. Parenthetic numbers on the scale represent the break points of the quantized values.

Often, quality and count show an inverse relation.

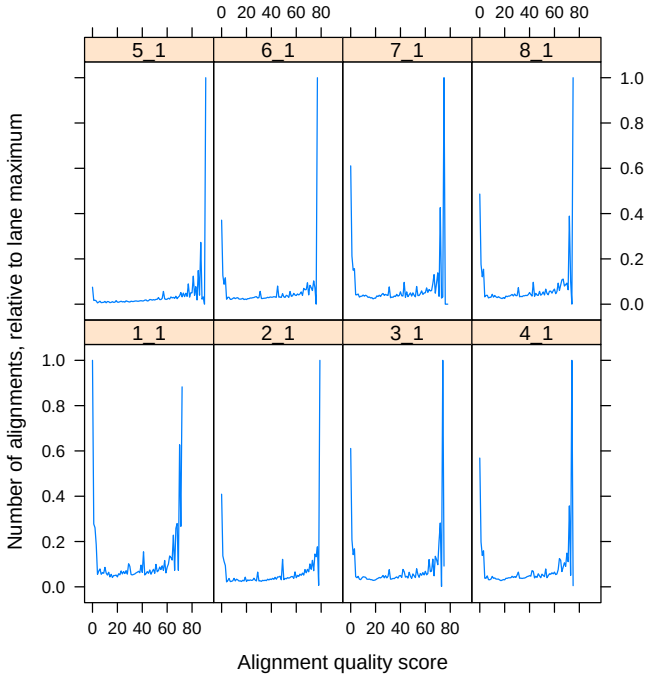
```
> qscore <- perTile[["medianReadQualityScore"]]
> print(plotTileQualityScore(qscore[qscore$type == "read", ]))
```



6 Alignment

Mapped alignment score. Counts measured relative to counts in score category with maximum representation. Successful alignments will be reflected in a strong peak to the right of each panel.

```
> print(plotAlignQuality(qa[["alignQuality"]]))
```



A Report-specific functions

```
> ppnCount <- function(m) {
+     ## scale subsequent columns to be proportions of
+     ## first column
+     m[,-1] <- m[,-1] / m[,1]
+     m
+ }
> .laneLbl <- function(lane) sub("s_(.*)_.*", "\\1", lane)
> plotReadQuality <- function(df) {
+     df$lane <- .laneLbl(df$lane)
+     xyplot(density~quality|lane, df,
+            type="l",
+            xlab="Average (calibrated) base quality",
+            ylab="Proportion of reads",
+            aspect=2)
+ }
> plotReadOccurrences <- function(df, ...) {
+     df$lane <- .laneLbl(df$lane)
+     df <- with(df, {
+         nOccur <- tapply(nOccurrences, lane, c)
```

```
+         cumulative <- tapply(nOccurrences*nReads, lane, function(elt) {
+             cs <- cumsum(elt)
+             (cs-cs[1] + 1) / diff(range(cs))
+         })
+         lane <- tapply(lane, lane, c)
+         data.frame(nOccurrences=unlist(nOccur),
+                    cumulative=unlist(cumulative),
+                    lane=unlist(lane))
+     })
+     xyplot(cumulative~log10(nOccurrences)|factor(lane), df,
+            xlab=expression(paste(
+                "Number of occurrences of each read (",
+                log[10], ")", sep="")),
+            ylab="Cumulative proportion of reads",
+            aspect=2, type="l", ...)
+ }
> freqSequences <- function(qa, read, n=20)
+ {
+     cnt <- qa[["readCounts"]]
+     df <- qa[["frequentSequences"]]
+     df1 <- df[df$type==read,]
+     df1[["ppn"]] <- df1[["count"]] / cnt[df1[["lane"]], read]
+     head(df1[order(df1$count, decreasing=TRUE),
+              c("sequence", "count", "lane")], n)
+ }
> plotAlignQuality <- function(df) {
+     df$lane <- .laneLbl(df$lane)
+     xyplot(count~score|lane, df,
+            type="l",
+            prepanel=function(x, y, ...) {
+                list(ylim=c(0, 1))
+            },
+            panel=function(x, y, ...) {
+                panel.xyplot(x, y/max(y), ...)
+            },
+            xlab="Alignment quality score",
+            ylab="Number of alignments, relative to lane maximum",
+            aspect=2)
+ }
> .plotTileLocalCoords <- function(tile, nrow) {
+     row <- 1 + (tile - 1) %% nrow
+     col <- 1 + floor((tile -1) / nrow)
+     row[col%%2==0] <- nrow + 1 - row[col%%2==0]
+     list(row=as.integer(row), col=as.factor(col))
+ }
> .atQuantile <- function(x, breaks)
```

```
+ {
+     at <- unique(quantile(x, breaks))
+     if (length(at)==1)
+         at <- at * c(.9, 1.1)
+     at
+ }
> .colorkeyNames <- function(at, fmt) {
+     paste(names(at), " (", sprintf(fmt, at), ")", sep="")
+ }
> plotTileCounts <- function(df, nrow=100) {
+     df <- df[!is.na(df$count),]
+     xy <- .plotTileLocalCoords(df$tile, nrow)
+     df[,names(xy)] <- xy
+     at <- .atQuantile(df$count, seq(0, 1, .1))
+     df$lane <- .laneLbl(df$lane)
+     levelplot(cut(count, at)~col*row|lane, df,
+               main="Read count (percentile rank)",
+               xlab="Tile x-coordinate",
+               ylab="Tile y-coordinate",
+               cuts=length(at)-2,
+               colorkey=list(
+                 labels=.colorkeyNames(at, "%d")),
+               aspect=2)
+ }
> plotTileQualityScore <- function(df, nrow=100) {
+     df <- df[!is.na(df$score),]
+     xy <- .plotTileLocalCoords(df$tile, nrow)
+     df[,names(xy)] <- xy
+     at <- .atQuantile(df$score, seq(0, 1, .1))
+     df$lane <- .laneLbl(df$lane)
+     levelplot(cut(score, at)~col*row|lane, df,
+               main="Read quality (percentile rank)",
+               xlab="Tile x-coordinate",
+               ylab="Tile y-coordinate",
+               cuts=length(at)-2,
+               colorkey=list(
+                 labels=.colorkeyNames(at, "%.2f")),
+               aspect=2)
+ }
> plotCycleBaseCall <- function(df) {
+     col <- rep(c("red", "blue"), 2)
+     lty <- rep(1:2, each=2)
+     df <- df[df$Base != "N",]
+     df$lane <- .laneLbl(df$lane)
+     xyplot(log10(Count)~as.integer(Cycle)|lane,
+            group=factor(Base), df,
```

```
+            type="l", col=col, lty=lty,
+            key=list(space="top",
+              lines=list(col=col, lty=lty),
+              text=list(lab=as.character(unique(df$Base))),
+              columns=length(unique(df$Base))),
+            xlab="Cycle",
+            aspect=2)
+ }
> plotCycleQuality <- function(df)
+ {
+     qnum <- as(SFastqQuality(as.character(df$Quality)), "numeric")
+     df$qtot <- qnum * df$Count
+
+     aveScore <- with(df,
+                      tapply(qtot, list(lane, Cycle), sum) /
+                      tapply(Count, list(lane, Cycle), sum))
+     score <- data.frame(AverageScore=as.vector(aveScore),
+                         Cycle=as.vector(col(aveScore)),
+                         Lane=.laneLbl(rownames(aveScore)))
+     xyplot(AverageScore~Cycle | Lane, score,
+            ylab="Average score",
+            aspect=2)
+ }
```