Interactive Visualization of Genomic Data
Interfacing Qt and R

Michael Lawrence

November 17, 2010
1 Introduction

2 Qt-based Interactive Graphics Canvas
   Design
   Implementation

3 Looking Forward: Integration
   Bioconductor
   External Tools
• Stacking tracks relates data to annotations along the genome
• Useful picture (if a bit ugly)
• Stacking tracks relates data to annotations along the genome
• Useful picture (if a bit ugly)
• Stacking tracks relates data to annotations along the genome
• Useful picture (if a bit ugly)
R and Genome Browsers: \textit{rtracklayer}

- Original goal of \textit{rtracklayer}: abstraction around genome browsers, internal or external
- First (and still only) implementation was the UCSC Genome Browser

```r
> session <- browserSession("UCSC")
> session\$user <- userTrack
> browserView(session, full = "user")
```
Thinking Outside the Browser

Genome Browser

- Look at the data in multiple, coordinated views
- Jump to the most interesting parts of the data
- Move beyond the genome axis
Thinking Outside the Browser

Genome Browser Data Visualization

- Look at the data in multiple, coordinated views
- Jump to the most interesting parts of the data
- Move beyond the genome axis
Thinking Outside the Browser

Genome Browser Data Visualization

- Look at the data in multiple, coordinated views
- Jump to the most interesting parts of the data
- Move beyond the genome axis
Thinking Outside the Browser

Genome Browser Data Visualization

- Look at the data in multiple, coordinated views
- Jump to the most interesting parts of the data
- Move beyond the genome axis
Thinking Outside the Browser

Genome Browser Data Visualization

- Look at the data in multiple, coordinated views
- Jump to the most interesting parts of the data
- Move beyond the genome axis
Interactive Graphics

- Many types of data
- Many types of plots, interactions
- Many preprocessing approaches, statistics, transformations, ...
Interactive Graphics

- Many types of data
- Many types of plots, interactions
- Many preprocessing approaches, statistics, transformations, ...
Interactive Graphics

- Many types of data
- Many types of plots, interactions
- Many preprocessing approaches, statistics, transformations, ...
Interactive Graphics

- Many types of data
- Many types of plots, interactions
- Many preprocessing approaches, statistics, transformations, ...

Exploring this space requires lots of rapid experimentation
Outline

1. Introduction

2. Qt-based Interactive Graphics Canvas
   Design
   Implementation

3. Looking Forward: Integration
   Bioconductor
   External Tools
Overview

- Support constructing interactive graphics “from scratch” in R
  - R draws every graphical primitive
  - R handles every user event

- Maximize performance
  - Static graphics: Be lazy, little work as possible
  - Dynamic graphics: Only redraw what needs to be redrawn
  - Interactive graphics: Quickly map actions in the plot to actions on the data

- Leave high-level graphics to other packages (e.g., mosaiq)
Optimized Rendering Layer

- Draw each unique glyph only once, blit to buffer
- Use hardware (OpenGL) whenever possible
  - Rarely-used GL_POINT_SPRITE mode draws glyphs (small textures) at roughly same rate as GL_POINTS (fastest primitive)
  - Drawing cached as texture through FBO
Layered Buffering Strategy

- Plot updates tend to be incremental, only a part of the plot changes
- Divide the plot elements into layers, such that elements within the same layer tend to change together
- Cache each layer in a buffer, only redraw when necessary
- Compose plot by compositing/stacking the layers
Spatial Indexing

- Spatial indices (e.g. quad tree) provide fast ($O(\log(n))$) lookup of data elements from coordinates
- Lazily update spatial index whenever element positions have changed and R requests a lookup
- Populate spatial index by passing a special renderer to the R drawing callback, thus indexing is transparent to R user
- Sometimes additional or separate logic is required (e.g. area plots, smooth scatter)
Outline

1 Introduction

2 Qt-based Interactive Graphics Canvas
   Design
   Implementation

3 Looking Forward: Integration
   Bioconductor
   External Tools
Qt

- C++ application library for GUIs, graphics and lots more
- Professionally developed, open-source, maintained by Nokia
- Basis of KDE desktop on Linux, MeeGo
- Official installer for all major platforms
Qt Modules

**Core**  Basic utilities, collections, threads, I/O, ...

**Gui**  Widgets, models, canvas etc for graphical user interfaces

**OpenGL**  Convenience layer (e.g., 2D drawing API) over OpenGL

**Webkit**  Embeddable HTML renderer (shared with Safari, Chrome)

**Others**  DBus, Designer, Help, Multimedia, Network, Xml, Script (Javascript), Sql, Svg, Declarative (UI language)
Qt Modules

Core  Basic utilities, collections, threads, I/O, ...

Gui   Widgets, models, canvas etc for graphical user interfaces

OpenGL Convenience layer (e.g., 2D drawing API) over OpenGL

Webkit Embeddable HTML renderer (shared with Safari, Chrome)

Others DBus, Designer, Help, Multimedia, Network, Xml, Script (Javascript), Sql, Svg, Declarative (UI language)
The *qtpaint* Package
The `qtpaint` Package

- `qtpaint` implements our approach using QGraphicsView
- Each layer of the plot corresponds to a canvas item, which is cached and composited by Qt
- In a callback, R draws canvas items through optimized renderer based on Q Painter
- All user events handled in R, using Qt’s spatial index for fast mapping of event coordinates to data elements
Impressionistic Example: Interactive Scatterplot

circle <- qglyphCircle()
scatterplot <- function(item, painter) {
    qdrawGlyph(painter, circle, df[,1], df[,2], 
        fill = fill)
}
scene <- qscene()
root <- qlayer(scene)
points <- qlayer(root, scatterplot, 
    hoverMove = pointIdentifier)
labels <- qlayer(root, labeler, cache = FALSE)
view <- qplotView(scene = scene, opengl = TRUE)
print(view)
Implementation

**VisNAB: VisNAB is Not A Browser**

- Toolbox for interactive visualization of genomic data in R
- Joint work with Teng Fei Yin (intern), Nicholas Lewin-Koh
- Leverages Bioconductor infrastructure
  - Preprocessing *ShortRead, Biostrings*
  - Data Manipulation *IRanges*
- Relies on *plumbr* package from GGobi foundation for its reference-based data model
- Demonstration on ChIP-seq data
Outline

1. Introduction

2. Qt-based Interactive Graphics Canvas
   Design
   Implementation

3. Looking Forward: Integration
   Bioconductor
   External Tools
Every interactive graphics software needs to compute on the data

During drawing, and in response to user input

Data must be mutable, not copy-on-write, so that graphic implicitly updates upon user changes

plumbr package provides a mutable data frame and list
Interval Data into Pipeline

- Need mutable data structures specific for interval data
- `MutableRanges` extends data structures in `IRanges` and `GenomicRanges` using “R5” reference classes
- Defines `Signal` class for registering R functions as callbacks that are invoked upon data changes

```r
connect(mutableRanges, "rangesChanged",
        function(i) { ... })
```
# Outline

1. **Introduction**

2. **Qt-based Interactive Graphics Canvas**
   - Design
   - Implementation

3. **Looking Forward: Integration**
   - Bioconductor
   - External Tools
Examples of Existing Browsers

- **Web-based**
  - *UCSC* (restrictive license)
  - *Ensembl* (open)
- **Desktop (Java, open-source)**
  - *IGB* (Integrated Genome Browser, Affy)
  - *IGV* (Integrative Genomics Viewer, Broad)
IGB: Integrated Genome Browser
**Key IGB Features**

**Data**  All standard track formats, including SAM/BAM, also DAS and QuickLoad

**Visualization**  Many plot types, combining plots, smooth zoom, some limited linked views (slice, external)

**Computation**  Coverage, thresholding, track arithmetic
Future IGB Features

- **Adaptability**: Plugins, more configurable GUI
- **Visualization**: Multiple views
- **Computation**: More operations on tracks
IGB + R Integration: Requirements

Data  Set genome, load URL, synchronize through shared data models (mutable ranges)

Visualization  Set view range, select features, control tier visibility, respond to changes in range and selection, customize

GUI  Add R-driven actions, customize
IGB + R Integration: Design

- Direct embedding through R and Java bridge (SJava).
- Embedding should be bi-directional.
- Implement rtracklayer API for basic operations.
Big Picture Design

- Computational Biologists
- Bench Biologists
- R
- IGB
- R tracklayer
- SJava
- R Visualizations
- Other Vis Software
Availability

qtinterfaces  http://github.com/ggobi
visnab  http://github.com/tengfei/visnab
External Tools

Acknowledgements

Deepayan Sarkar
Hadley Wickham

Teng Fei Yin
Nicholas Lewin-Koh
Robert Gentleman

GGobi Foundation