BIOINFORMATICS APPLICATIONS NOTE

Vol. 25 no. 4 2009, pages 533–534
doi:10.1093/bioinformatics/btn657

Genome analysis

AnnotationSketch: a genome annotation drawing library

Sascha Steinbiss*,†, Gordon Gremme†, Christin Schärfer, Malte Mader and Stefan Kurtz

Center for Bioinformatics, University of Hamburg, Bundesstraße 43, 20146 Hamburg, Germany

Received on September 25, 2008; revised on December 03, 2008; accepted on December 19, 2008

Advance Access publication December 23, 2008

Associate Editor: John Quackenbush

ABSTRACT

Summary: To analyse the vast amount of genome annotation data available today, a visual representation of genomic features in a given sequence range is required. We developed a C library which provides layout and drawing capabilities for annotation features. It supports several common input and output formats and can easily be integrated into custom C applications. To exemplify the use of *AnnotationSketch* in other languages, we provide bindings to the scripting languages Ruby, Python and Lua.

Availability: The software is available under an open-source license as part of *GenomeTools* (http://genometools.org/annotationsketch.html).

Contact: steinbiss@zbh.uni-hamburg.de

1 INTRODUCTION

Genome annotations are often provided in the GFF3 format (Stein, 2007) using the vocabulary of the Sequence Ontology (Eilbeck *et al.*, 2005). This format defines a directed acyclic graph (*annotation graph*) with individual annotated entities (*features*) as nodes and edges representing *part_of* relationships between them. It is not uncommon for annotations to contain tens of thousands of features. This makes it difficult to obtain an overview of the structure and hierarchy of the features in a particular genomic location by looking at tabular data. For this reason, annotation browsers like the *UCSC Genome Browser* (Kent *et al.*, 2002) or *GBrowse* (Stein *et al.*, 2002) as well as curation tools like *Apollo* (Lewis *et al.*, 2002) provide an intuitive graphical representation of annotated features, allowing, for example, to jump to a specific feature. However, such drawing components are often tied to the particular tool's data model and programming language, limiting their reusability in other contexts. While the BioPerl toolkit (Stajich *et al.*, 2002) includes the *Bio::Graphics* module as an established reusable and extensible solution for genome annotation drawing, it has the disadvantage to be conveniently usable in Perl applications only. Furthermore, its output is limited to files, which is inefficient in desktop GUI applications because temporary files must be created. Another disadvantage is the need for a database backend and the explicit definition of *aggregators* to visualize feature relationships.

*To whom correspondence should be addressed.

†The authors wish it to be known that, in their opinion, the first two authors should be regarded as joint First Authors.

2 DESIGN AND IMPLEMENTATION

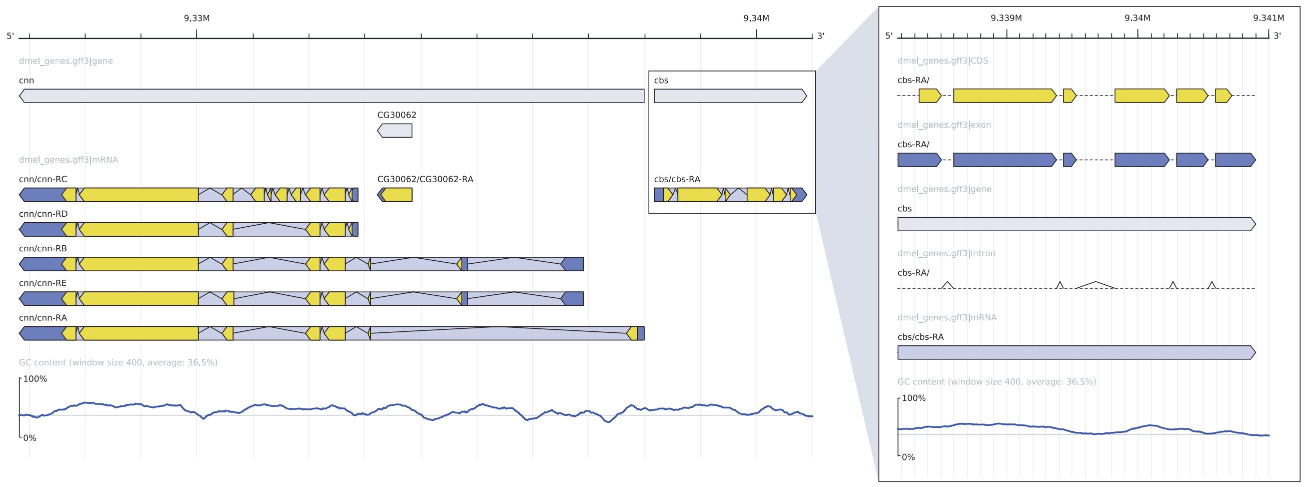
AnnotationSketch is designed to be a small and efficient drawing library for genome annotations with a focus on simplicity, allowing to draw any given annotation in a wide variety of application fields while automatically considering feature relationships. *AnnotationSketch* directly uses annotation graphs as its underlying data model. They can be created and manipulated dynamically by user code (e.g. in custom gene prediction software) via library functions available in *GenomeTools*. Alternatively, they can be imported from GFF3, GTF or BED files by using the respective *GenomeTools* parser. The actual drawing process is divided into following three separate phases.

(1) Feature selection phase. Obtain a collection of features, either by retrieving, from an efficiently searchable *feature index*, all features overlapping the range of sequence positions to draw, or by supplying an array of features. Based on user preferences and feature relationships, group single features into *blocks*, the smallest units which can be laid out.
(2) Layout phase. Distribute the blocks into a hierarchical structure representing vertical *tracks* (containing all blocks with a common feature, e.g. type) and *lines* (each containing non-overlapping blocks) such that the obtained packing in the 2D representation is most compact.
(3) Rendering phase. Use the track and line structure as a blueprint for drawing to a specific output format.

While some concepts (such as the use of tracks) are shared with *Bio::Graphics*, tracks need not be explicitly created by the programmer but are determined from the feature types encountered in the input data. This minimizes programming overhead. Nevertheless, user-defined tracks can be created according to arbitrary block properties. Each feature can also optionally be drawn transparently on top of its parent feature (e.g. all *exon* and *intron* features are placed into their parent *mRNA* or *gene* track). Relationships are implicitly given by the annotation graph. This approach, called *collapsing*, can significantly improve visual clarity in renderings of annotations with many levels of hierarchy.

The *AnnotationSketch* library is implemented using ANSI C in an object-oriented style. This approach makes it straightforward to create bindings to other object-oriented languages. Bindings for the Ruby, Python and Lua scripting languages are included with the software. We also provide an *AnnotationSketch*-based command line tool, allowing to draw GFF3, GTF and BED annotation data.

An image is represented by one class instance per structural (*Element*, *Block*, *Line*, *Track* and *Diagram*) or processing result



(a) collapsed view

(b) uncollapsed view

Fig. 1. Two images drawn by *AnnotationSketch*, showing the *cnn* and *cbs* gene region from the Ensembl *Drosophila melanogaster* annotation [release 51, (**a**) 2R:9326816–9341000, (**b**) 2R:9338169–9341000]. At the bottom, the GC content of the respective sequence (calculated on the fly) is drawn via an example custom track attached to the diagram. Image (a) shows the *exon* (dark blue), *CDS* (yellow) and *intron* type features collapsed into their *mRNA* type parent features (medium blue) for a more concise view. Image (b) shows an uncollapsed rendering of the *cbs* gene.

(*Layout*). Additionally, *custom track* objects – instances of special classes implementing a common interface—can be added to a *Diagram* and allow development of user-defined drawing functionality, e.g. to display arbitrary plots along the annotation (see Fig. 1 for example output). On user request, the *Layout*'s *sketch* method invokes rendering methods for each component in a drawing surface abstraction called *Canvas*, which in turn calls primitive shape drawing methods of a *Graphics* object wrapping a graphics back-end. The currently used *Graphics* implementation uses the *Cairo* 2D graphics library (Worth *et al*., 2003), currently allowing output to PNG bitmaps as well as PDF, SVG and PostScript vector formats. *Cairo* also facilitates integration into GUI-based applications by providing native rendering surfaces for windowing systems like the X Window System or Mac OS X *Quartz*. Attaching a rendered image to a user interface is possible by mapping 2D image coordinates to the respective feature. This enables *AnnotationSketch* to be used, for example, in a genome annotation browser or editor.

User preferences are stored in instances of the *Style* class. Configuration options (colours, borders, collapsing flags, etc.) can be set and retrieved both globally and for specific feature types. Additionally, it is possible to supply callback functions to make colours or captions dependent on individual feature properties.

To evaluate the performance, 100 random regions of 500 kb length from the *Drosophila melanogaster* Ensembl release 50 GTF gene annotation were drawn to a 800 pixels wide PNG image from a *FeatureIndex* held in memory, resulting in an average rendering time of 0.61 s per image. It has to be noted that the time-consuming part appeared to be the bitmap rastering process. Using SVG or PDF for output reduced the time to 0.05 s (SVG) and 0.04 s (PDF) per image. In contrast, creating a comparable output with a Perl script using *Bio::Graphics* took 3.98 s on average per PNG image and 4.58 s per SVG file. This makes *AnnotationSketch* favourable in SVG-based web applications to reduce server load. Memory usage of the *AnnotationSketch*-based program peaked at 34.1 MB for a single run, of which 33.3 MB were occupied by the feature index for the 15.6 MB GFF3 file. The Perl script's average peak memory usage for a single run was 15.45 MB. However, the memory usage of the Perl script did not include the MySQL database storing the features.

3 CONCLUSION

AnnotationSketch provides a fast and easy to use library for drawing annotations to be used in any application in which a light-weight visualization of annotation data compatible with an annotation graph format is desired. By implementing all functionality in C and using foreign function interfaces to add high-level bindings to a variety of other languages afterwards, applications can benefit from both portability and interface consistency across all bindings.

Conflict of Interest: none declared.

REFERENCES

Eilbeck,K. *et al*. (2005) The sequence ontology: a tool for the unification of genome annotations. *Genome Biol*., **6**, R44.

Kent,W.J. *et al*. (2002) The human genome browser at UCSC. *Genome Res*., **12**, 996–1006.

Lewis,S.E. *et al*. (2002) Apollo: a sequence annotation editor. *Genome Biol*., **3**, RESEARCH0082.

Stajich,J.E. *et al*. (2002) The bioperl toolkit: perl modules for the life sciences. *Genome Res*., **12**, 1611–1618.

Stein,L.D. (2007) Generic feature format version 3. Available at http://www.sequenceontology.org/gff3.shtml (Last accessed date August 25, 2008).

Stein,L.D. *et al*. (2002) The generic genome browser: a building block for a model organism system database. *Genome Res*., **12**, 1599–1610.

Worth,C.D. *et al*. (2003) Cairo: cross-device rendering for vector graphics. *Proceedings of the 2003 Linux Symposium*. Available at http://cworth.org/~cworth/papers/xr_ols2003/ (last accessed date August 25, 2008).