# SequenceJuxtaposer: Fluid Navigation For Large-Scale Sequence Comparison In Context

James Slack* Kristian Hildebrand*† Tamara Munzner* Katherine St. John‡

## Abstract

SequenceJuxtaposer is a sequence visualization tool for the exploration and comparison of biomolecular sequences. We use an information visualization technique called "accordion drawing" that guarantees three key properties: context, visibility, and frame rate. We provide context through the navigation metaphor of a rubber sheet that can be smoothly stretched to show more details in the areas of focus, while the surrounding regions of context are correspondingly shrunk. Landmarks, such as user specified motifs or differences between aligned base pairs across multiple sequences, are guaranteed to be visible even if located in the shrunken areas of context. Our graphics infrastructure for progressive rendering provides immediate responsiveness to user interaction by guaranteeing that we redraw the scene at a target frame rate. Our preprocessing algorithms are subquadratic: $O(nk)$ for $k$ sequences of $n$ base pairs each. All runtime rendering algorithms are sublinear in $nk$: they are $O(v)$ where $v$ is the number of items visible onscreen at once, and $v \ll nk$. SequenceJuxtaposer supports interaction at 20 frames per second when browsing collections of several hundred sequences that comprise over 1.7 million total base pairs. We show three example applications with large, publicly available datasets, and we are able to quickly observe many features that had previously required significant analysis to discover.

**Keywords:** sequence analysis, motif and gene finding, data visualization, Focus+Context

## 1 Introduction

Biomolecular sequence comparisons are essential in understanding underlying genomic patterns. Current sequence browsers [14, 17, 34, 37] support examining the data at any level from a global overview down to a detailed view of a small section, where the interaction happens as a discrete jump from one level of magnification to the next. At high levels of magnification only a very small subset of the sequence is visible, so it is easy to lose track of the current location and its position with respect to other areas of interest. The cognitive load of maintaining a mental model of navigation history is very high, and humans have a very limited capacity to do so [38]. Exploration often entails frequent backtracking where people remind themselves of what they have already seen and where they are now in relation to previous viewpoints.

*Dept of Computer Science, U. of British Columbia, Vancouver, BC V6T 1Z4, Canada, {jslack,hilde,tmm}@cs.ubc.ca.

†Dept Media Systems, Bauhaus U., Weimar, Germany, kristian.hildebrand@medien.uni-weimar.de.

‡Dept of Math & Computer Science, Lehman College & the Graduate Center, City U. of New York, Bronx, NY 10468, USA, stjohn@lehman.cuny.edu.

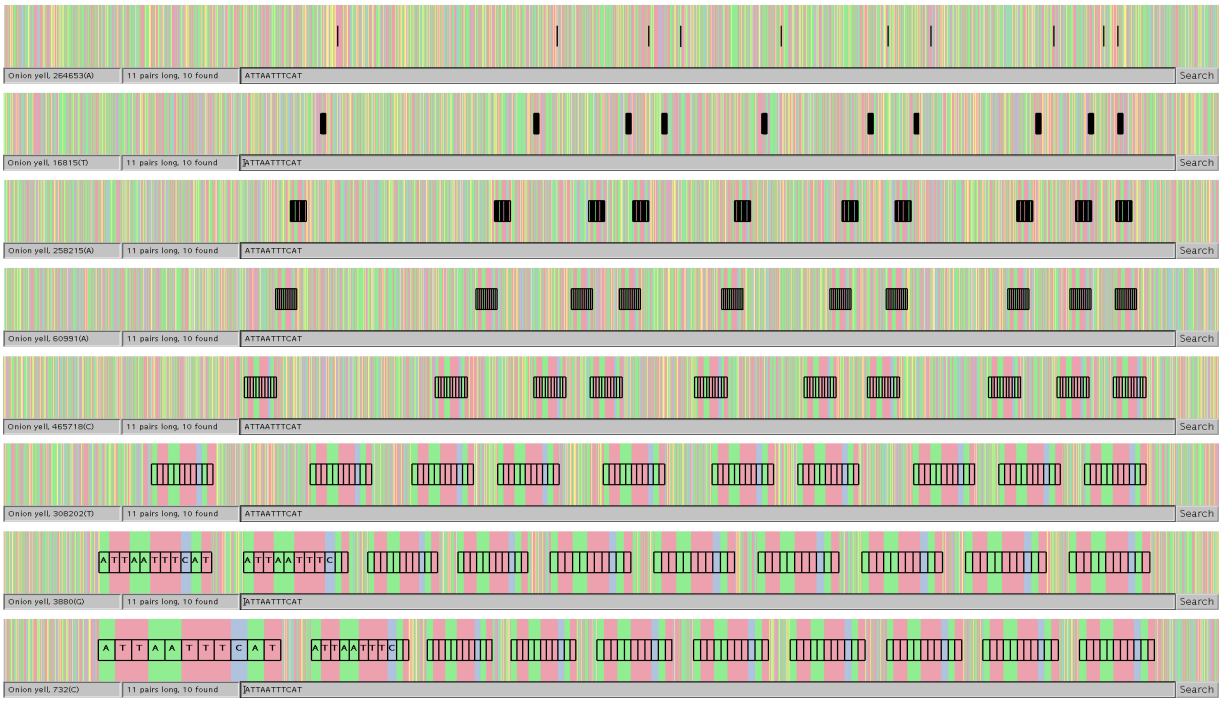


Figure 1: The entire *onion yellows phytoplasma* (OY) genome of 860 Kbp. The eight successive screenshots show the exploration of a motif in the genome using the accordion drawer navigation technique, where we interact with sequence data as if it were drawn on a stretchable rubber sheet with the borders tacked down. Top: after searching for the motif ATTAATTTCAT, the found bases are marked as landmarks that are guaranteed to be visible, even if they would ordinarily be smaller than a single pixel in this overview. Second-Sixth: steps along the animated transition that automatically expands the entire found group. Seventh-Eighth: expansion by selecting an onscreen region and dragging the mouse to resize it.

SequenceJuxtaposer allows people to interact with sequence data as if it were drawn on a rubber sheet with the borders tacked down [29], as shown in Figure 1. Stretching certain areas so more detail can be seen causes the rest of the sheet to shrink accordingly, but landmarks remain visible in the periphery. We call this approach **accordion drawing** because the effect is similar to the stretching and shrinking of an accordion bellows. It is an example of a class of techniques known as "Focus+Context" in the information visualization literature [5, 11, 15, 19], where overview and detail are integrated into a single combined view.

Accordion drawing is a powerful new information visualization technique that we recently introduced with the TreeJuxtaposer system for visually comparing large phylogenetic trees [23]. In this work, we introduce accordion drawing for biomolecular sequences. Accordion drawing provides fluid exploration of large datasets through guarantees of three key properties: context, visibility and frame rate. We always provide visible context around the expanded areas of interest. Ensuring that marked areas stay visible, so that even in the shrunken regions of the rubber sheet they act as landmarks, requires significant graphics infrastruc-

ture. We accomplish this in $O(v)$ rather than $O(t)$ time, where $v$ is the number of visible nucleotides, $t$ is the total number of nucleotides, and $v \ll t$. Without that guarantee, marks would simply disappear if the group of nucleotides was shrunk to an area smaller than a single pixel, which would lead to user disorientation. Maintaining a requested frame rate for real-time drawing, for example 20 frames per second, provides smooth transitions during stretching or shrinking. Transitions allow people to easily track visual landmarks, which is faster than reacquiring visual targets after an abrupt jump cut from one view to the next [27].

# 2 Prior Work

Comparing and analyzing sequences is a fundamental part of bioinformatics. Many text-based alignment tools were developed to address this problem, including [10, 12, 20]. The text-based alignment tools work well for aligning and analyzing only a few small sequences. However, viewing the overall sequence structure becomes difficult when the length of a sequence exceeds the window size (often 80-100 nucleotides).

Alternative approaches to sequence viewing include embedding sequences or properties of the sequences in three dimensional space [1, 6, 8, 35], using fractals [2], using "dekapentagonal" summary figures [39], representing alignments by proportional sized letters at each position [30] and formatting sequences for publication [3]. Further advances in visualization of pairwise aligned sequences include Artemis [28], PipMaker [31], and BARD [33].

SequenceJuxtaposer complements recent web-based viewers that allow the search and display of genomic sequences integrated with annotation databases [14, 17, 34, 37]. Other viewers, such as VISTA [21] or phyloVISTA [32], are intended to run on a local client. Although these approaches have strengths, and many of them allow navigation of a great deal of information at different magnification levels, none of them allow users to compare multiple sequences with context preserved and fluid navigation between different magnification scales.

# 3 Applications

SequenceJuxtaposer allows fluid navigation of whole genomes, where the overall context of the genome is visible even when regions of interest are being examined in-depth. Our application currently handles DNA and RNA sequence data (or strings over the alphabet that include the nucleotide bases $\{$A,C,G,T,U$\}$, a symbol N for undetermined bases, and a symbol '-' for gaps).

**Onion Yellows Genome:** As a case in point, we look at the genome for Onion yellows phytoplasma (OY), a recently completed whole genome (December 2003) [25] pathogen transmitted by insects. The difficulty in culturing it *in vitro* has hampered the study of its biology. With the sequencing of the whole genome, bioinformatics techniques can be used to shed light on both this and other important phytoplasmas. Finding new ways to target its spread is of economic importance due to the damage it causes to commercial plants, because the only known effective treatment is by the antibiotic tetracycline [9].

The OY genome has a relatively small size of 860,631 bp which means it can be loaded into SequenceJuxtaposer in its entirety, as seen in Figure 1. Using the search feature, we focus in on thymidylate kinase (TMK), an anti-viral target for phytoplasmosis. Several bacterial homologues with catalytic activity were recently found by Miyata *et al.* [22], and we examine on one of them, *tmk-a* (GenBank accession number

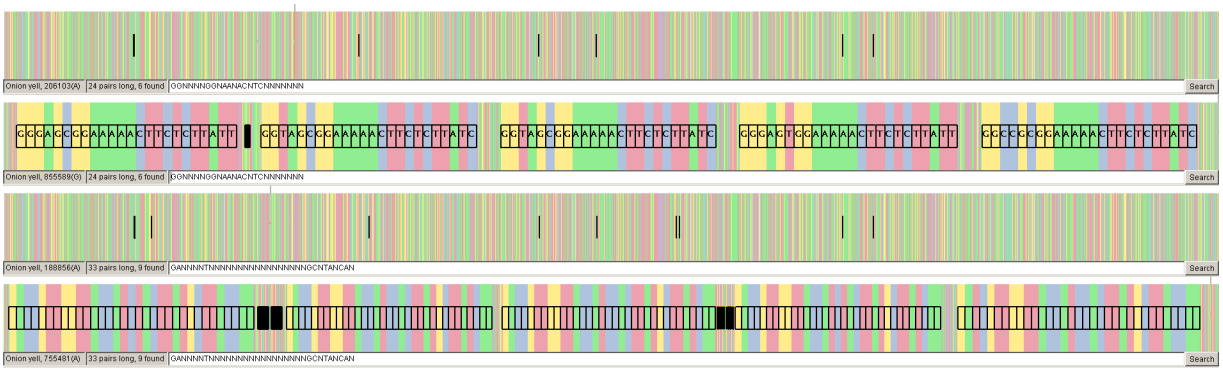

Figure 2: Searching for motifs found in TMK genes in the *onion yellows phytoplasma* (OY) genome. The first and second views show the locations of P-loop motifs in the context of the genome that were found with search string GGNNNNGGNAANACNTCNNNNNNN in unstretched and magnified views, respectively. The third and fourth views show the locations of TMP binding motifs found with search string GANNNNTNNNNNNNNNNNNNNNNGCNTANCAN, again with an unstretched view followed by a magnified one.

AB010446). We can quickly find several potential copies of the gene by typing a segment of it, ATTAATTTCAT, into the search field to find the beginning of the coding region. The first line of Figure 1 shows their relative position in the whole genome. The following five lines in Figure 1 show the animated transition of growing the search results as a group. The bottom two lines of Figure 1 show two frames during a final zoom on the left side, where the mouse was used to select a single copy of the gene in an on-screen rectangle that was interactively resized. The ability to focus in on duplicate genes, in the context of the whole genome, allows the quick visualization of related areas such as regulatory and functional regions upstream of genes themselves.

Sequence Juxtaposer also allows searches by regular expressions to find more sophisticated motifs. For example, we can locate important regions on the TMK gene such as the P-loop and TMP binding site identified by Miyata *et al.* [22]. Figure 2 shows both a P-loop motif search string which has 6 matches across the whole genome, and a TMP-binding site motif search with 9 matches. The simple search strings we used also match a few false positives, and after distinguishing the possible TMK matches through manual exploration we show those regions in expanded form. The regular expression searches supported by Sequence-Juxtaposer allow easy exploration for duplicate genes, interesting structures, and regulatory elements.

**Mammalian Mitochondrial Dataset:** The Murphy *et al.* [24] data set consists of molecular data for 22 genes and 44 mammals. The sequences are of length 16,397 bp and include 19 nuclear and 3 mitochondrial gene sequences for 42 placental and 2 marsupial outgroups. The placental mammals fall into four superordinal groups and the analysis of this dataset focused on resolving the interrelationships among these groups. Figure 3 top shows all 44 sequences for the single gene CNR1, and when we find the appropriate difference threshold with the slider we can immediately see that whales, dolphins, and hippos are distinct from the other sequences. These three mammals form a clade, and further

adjusting of the slider shows other clades in Figure 3 second through fourth. The ability to see the phylogenetic signal of clade membership by simply manipulating the difference slider shows SequenceJuxtaposer's power.

**RbcL Gene from Seed-Plants:** The rbcL gene found in plants encodes the large subunit of ribulose bisphosphate carboxylase, a crucial enzyme in photosynthesis [7]. Given its important functional role, the protein sequence is highly conserved across species. The 714 Kbp Chase *et al.* `treezilla` dataset contains 500 seed-plant rbcL sequences of 1428 base pairs each [7, 26]. Kellogg and Juliano found that in this dataset 105 sequences (22%) are absolutely conserved across all, while another 110 had only one change [16]. While the function (and amino acids coded for) are highly conserved, the actual codons vary from sequence to sequence.

Wall and Herbeck did a sophisticated probabilistic analysis, using maximum likelihood techniques and factoring in the G+C content of each species [36]. A goal was to determine whether codon bias changes over time and if evolutionary trends can be found to explain codon choice and differing amounts of bias. They found that certain residues were significantly correlated with codon bias. These were the twofold amino acids (with exactly two possible codons): glutamic acid, lysine, cystine, phenylalanine, glutamine, and tyrosine, plus leucine as the only sixfold one. Proline, a fourfold amino acid, was the only exception to the easily characterized behavior of the other fourfold amino acids, because its codon bias was unusually divergent across clades.

Figure 4 shows that we can quickly observe this phenomenon in the `treezilla` rbcL dataset using the visualization capabilities built into SequenceJuxtaposer. A few minutes of interactive exploration with the slider locates the interesting difference threshold of 62%, where we just begin to see red marks in the image. Figure 4 top shows the areas around these differences after expansion, where the red mark falls at the end of the codon triplet. For example, we see that in the second column of highlighted differences, the codons are mostly CTN (where N matches any character), because the vertical strip corresponding to the first nucleotide site is mostly blue. A few pink areas show that TTN also occurs. Six codon possibilities code for leucine and the other two code for phenylalanine. Both of those residues are correlated strongly with codon bias. Similarly, we can inspect the third area of differences to see that the dominant codon is the yellow-green of GAN. In this case, the identity of the third site in the codon affects which residue it encodes, but the red markings hide this information. Figure 4 bottom shows the same view with the red difference marking turned off, so that we can then easily see the colors that represent nucleotide type in those vertical strips. Glutamic acid codons end with A (green) or G (yellow), which occur a significant fraction of the time. The fourth difference column of CCN encodes proline.

Although SequenceJuxtaposer does not visually communicate all of the fine-grained details of the codon bias captured in Wall and Herbeck's maximum likelihood analysis technique, our tool does render many of their results visually apparent after a minimal amount of visual exploration. The set of techniques we describe above are: focusing on regions of difference, zooming in on neighboring codons, toggling difference higlighting, and viewing other species with similar patterns. Examining the regions of differences provides intuition about which amino acids are coded by several different codons and shows their relative position in the sequence. The new visualization capabilities of accordion drawing will provide similar

biological insight to guide future experimental and computer analysis of sequences.

# 4 Algorithms

SequenceJuxtaposer uses advanced algorithms for drawing, searching and interaction. Our dataset contains a total number $t = nk$ of nucleotides, where $n$ is the number of nucleotides per sequence and $k$ is the number of sequences. In the common case of redrawing frames, which happens many times per second, our runtime algorithms are all sublinear in $t$. The user-initiated actions of changing difference thresholds or searching for motifs are linear in $t$. Our preprocessing algorithms are all subquadratic in $t$. Gaps that occur in all loaded sequences are automatically elided during preprocessing.

**Differences:** Computing the differences between aligned nucleotides in sequences has a processing cost of $O(nk)$, where $n$ is the number of nucleotide positions and $k$ is the number of sequences. We make a first pass through the sequences at a nucleotide position to find the "majority" nucleotide, that occurs most often, and what percentage of nucleotides in that column differ. If the percentage is beyond the threshold, which is interactively controllable by the user through a slider, then we make a second pass through that column to mark the nucleotides that do not match the majority. Gaps and undetermined base pairs are ignored. This $O(t)$ processing happens at startup time and whenever the user moves the slider.

**Searching:** Users initiate a search either by entering characters in a text box or by simply dragging the mouse in the window across an expanse of nucleotides while the shift key is held down. Our algorithm is based on the linear-time Knuth-Morris-Pratt string-matching algorithm [18]. Our variant supports gap-aware search, skipping gaps when checking if a sequence matches a search string. We also support a limited form of regular expressions by optionally allowing search strings to contain the undetermined base character N that matches any nucleotide. The search cost is also $O(t)$.

**Interaction:** The look and feel of the fluid interaction is difficult to communicate through words and still images, so we also provide an accompanying video showing the system in action*.

Rectangular areas can be expanded or contracted by selecting either an area of the window to freely resize with the mouse, or by picking a group of nucleotides to grow or shrink. Although the absolute position of an item changes when we resize regions, the relative ordering between items stays constant in both the horizontal and vertical directions. With interactive resizing, the user drags the mouse in the window to define the active area and then resizes that area by moving the visible rubberband from one of its corners. Users can also trigger an animated transition to grow or shrink all nucleotides belonging to a particular group with a button press. The three main groups are the nucleotides marked as current search results, as alignment gaps, and as different according to the current threshold value.

**Drawing:** Our drawing algorithms are designed for scalable performance even when the total number of nucleotides $t$ in the dataset is much larger than the number of visible nucleotides $v$ that can be seen in any single view. Using algorithms that are $O(t)$ would lead to unacceptably slow interactive performance,

*http://www.cs.ubc.ca/~tmm/papers/sj

whereas $O(v)$ or $O(\log t)$ yields acceptable runtime results. The number of visible nucleotides depends on the number of pixels $p$ available for display. We must always keep the entire dataset in memory, because with rubber sheet navigation the set of visible items could change drastically in just a few frames.

One obvious way to ensure $O(v)$ rendering is to cull the geometric elements in highly compressed areas, where they would be invisible because they are smaller than a single onscreen pixel. However, our guarantee of landmark visibility requires checking if any of the nucleotides corresponding to a highly compressed onscreen area should be marked. The naive approach to this would be linear in $t$, the total number of nucleotides. Instead, at preprocessing time we build a quadtree in $O(t \log t)$ time that maps the colored boxes that are the geometric representation of a nucleotide into a hierarchical decomposition of space into quadtree cells. In our previous description of accordion drawing for trees, we describe a variant of quadtrees that efficiently supports navigation through growing and shrinking areas [23]. Before culling, we check a quadtree cell against each of the two marked groups (differences or search results). The groups are kept as $r$ contiguous ranges, where $r$ can be at most $t/2$, since adjacent ranges are merged when items are added or removed from groups. We store the $r$ ranges in a sorted tree for an $O(\log r)$ lookup, so the total cost of the guaranteed visibility check $O(v \log r)$, which is indeed sublinear in the total node count $t$.

We want immediate realtime response when resizing regions, even if the dataset is very large. If we simply send every geometric item in the scene through the conventional rendering pipeline supported in graphics hardware, we have no such guarantee: the time it takes to render depends on the size of the dataset. We instead want to draw for a fixed amount of time and then check back for user interaction before continuing. This approach is called *progressive rendering* [4], and although popular with high-end graphics applications [13] it is unfortunately uncommon in information visualization systems. We use a priority queue to draw items in order of current onscreen size so that any partial frame along the way has the best possible picture, where the magnified areas are drawn before compressed ones.

The guarantees on frame rate and visibility incur overhead compared to simply using the standard rendering pipeline. However, we argue that the cost is outweighed by the benefits of immediate interactive response and persistent landmarks. Our approach provides very high information density as well as extremely lightweight and fluid navigation.

**Performance:** Web-based sequence browsers impose minimal requirements on the local client machine, because all of the major computation is done with a large remote server farm [14, 17, 34, 37]. In contrast, SequenceJuxtaposer requires significant memory resources to load very large datasets in order to support fluid navigation. We require immediate access in local memory not only to the entire raw dataset, but also to our supporting data structures such as the quadtree-based accordion drawers.

Our prototype is written in Java using the GL4Java bindings to the OpenGL graphics library. The benchmarks below were run on a 3.0GHz Pentium 4 with 2GB of main memory running the Linux 2.4.20-24.9smp kernel, Java 1.4.1_02-b06 (HotSpot) with a 1.5GB heap, and an nVidia GeForceFX 5900 Ultra graphics card.

Figure 5 shows that we have achieved linear memory use, and can support exploration for a maximum of 1.7 million base pairs. Our system's linear memory footprint allows it to be

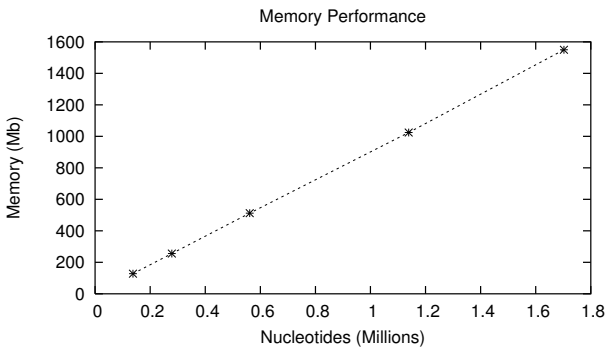


Figure 5: Our memory footprint is linear in the number of base pairs, and our system handles real-time exploration in context for datasets of up to 1.7 million base pairs.

used for interactive exploration of both huge datasets on high-end desktop machines, and medium-sized datasets on low-end laptops such as iBooks.

Our second benchmark tests the time required to preprocess and draw frames for the entire Murphy *et al.* [24] dataset of 44 sequences of 16397 bp each, 721 Kbp total. Preprocessing to compute differences and build our accordion drawer data structures took 25 seconds of wall clock time. The time to draw the entire scene in full detail ranged from 5 seconds for the best case with no marked regions to check for guaranteed visibility, to 7 seconds for the worst case where every column that has differences is marked. Recall that we always provide interactive response in 100 milliseconds, so users can interact without waiting for the system to progressively finish filling in the full scene.

Finally, our approach adapts well to very high resolution displays such as the 9Mpixel IBM T221 flatpanel display. In our accompanying video, we show that our progressive rendering approach provides realtime responsiveness.

# 5 Future Work

SequenceJuxtaposer is open source, and a beta version is currently available at `http://www.cs.ubc.ca/~tmm/projects/SequenceJuxtaposer/`.

We envision a wide range of possible future work for SequenceJuxtaposer. We would like to add support for proteins, and allow selection of amino acid coloring based on standard classifications of proteins such as polarity or hydrophylic vs. hydrophobic status. Displaying gene annotations in SequenceJuxtaposer would enable to navigation through a rich and user-controllable set of landmarks. Current genome browsers display such information, and our challenge would be to do so efficiently enough to maintain our real-time fluid navigation capabilities. Ultimately, we could use annotations to create a hierarchical series of landmarks that provide layers of abstraction, allowing navigation at multiple levels of detail, ranging from base pairs to genes to chromosomes to entire genomes.

In the short term, we plan to join TreeJuxtaposer with SequenceJuxtaposer for automatic linked navigation between phylogenetic trees and multiple sequences. We would like to use phylogenetic trees directly as the source of hierarchical guides to allow the viewing of thousands of sequences in a meaningful way. We propose using trees not only as a navigational tool, but also as a way to locate possible regulatory elements in aligned whole genomes, and to classify higher level differences between very large datasets.

# 6 Conclusion

SequenceJuxtaposer incorporates the powerful information visualization technique of accordion drawing where details are always shown

within their global context, landmarks are guaranteed to be visible, and the frame rate for redrawing the scene is guaranteed to provide realtime response. We achieve a runtime complexity of $O(v)$, where $v$ is the visible number of nucleotides bounded by the number of pixels $p$ in the display, rather than the $O(t)$ cost of the total number of base pairs. Our start-up and preprocessing time is bounded by $O(t \log t)$ time, and our overall memory footprint is linear in the number of aligned sequences. We can load hundreds of sequences with a total of over 1.7 million base pairs, maintaining the ability to fluidly resize areas of interest. We allow exploration of differences between the sequences by interactively changing the threshold for marking differences at each nucleotide site. We also support searching for motifs with immediate visual feedback. SequenceJuxtaposer's innovative and powerful new capabilities for visualizing and exploring biomolecular sequence data can quickly lead users to the kinds of biological insights that previously required extensive analysis.

# 7 Acknowledgements

We thank Wayne Maddison for originally suggesting we attack this problem, David Hillis, David Haussler, Nina Amenta for helpful conversations, and Ciarán Llachlan Leavitt for comments on paper drafts. We also thank the Hillis lab at UT-Austin for the test dataset of first benchmark. Funding was provided by NSF/DEB-0121651/0121682, NSERC/RGPIN 262047-03, and Hildebrand was supported by the German Academic Exchange Service.

# References

[1] R. Mark Adams, Blaze Stancampiano, Michael McKenna, and David Small. Case study: A virtual environment for genomic data visualization. In *IEEE Visualization 2002*, pages 513–516, 2002.

[2] Dan Ashlock and James W. Goldin III. *Evolutionary Computation and Fractal Visualization of Sequence Data*, chapter 11. Morgan Kauffman, 2001.

[3] E. Beitz. TeXshade: Shading and labeling of multiple sequence alignments using LaTeX2e. *Bioinformatics*, 16:135–139, 2000.

[4] Larry Bergman, Henry Fuchs, Eric Grant, and Susan Spach. Image rendering by adaptive refinement. In *SIGGRAPH*, pages 29–37, 1986.

[5] M. Sheelagh T. Carpendale, David J. Cowperthwaite, and F. David Fracchia. Three-Dimensional Pliable Surfaces: For effective presentation of visual information. In *Proc. UIST*, pages 217–226, 1995.

[6] Hsuan T. Chang, Neng-Wen Lo, Wei C. Lu, and Chung J. Kuo. Visualization and comparison of DNA sequences by use of three-dimensional trajectories. In *Proc. First Asia-Pacific Bioinf. Conf*, pages 81–85. Australian Computer Society, Inc., 2003.

[7] M.W. Chase et al. Phylogenetics of seed plants: An analysis of nucleotide sequences from the plastid gene rbcl. *Ann. Missouri Bot. Gard.*, 80:528–580, 1993.

[8] Ed Huaihsin Chi et al. Visualization of biological sequence similarity search results. In *Proc. of IEEE Visualization '95*, page 44, 1995.

[9] D. L. Davies and M. F. Clark. Maintenance of mycoplasma-like organisms occurring in pyrus species by micropropagation and their elimination by tetracycline therapy. *Plant Pathol*, 43:819–823, 1994.

[10] L. Duret, E. Gasteiger, and G. Perrire. LalnView: A graphical viewer for pairwise sequence alignments. *Comput. Applic. Biosci.*, 12:507–51, 1996.

[11] George W. Furnas. Generalized fisheye views. In *Proc. SIGCHI*, pages 18–23, 1986.

[12] N. Galtier, M. Gouy, and C. Gautier. SeaView and Phylo_win, two graphic tools for sequence alignment and molecular phylogeny. *Comput. Applic. Biosci.*, 12:543–548, 1996.

[13] Hugues Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH*, pages 189–198, 1997.

[14] T Hubbard et al. The Ensembl genome database project. *Nucleic Acids Research*, 30(1):38–41, 2002. www.ensembl.org.

[15] T. Alan Keahey and Edward L. Robertson. Nonlinear magnification fields. In *Proc. IEEE Symposium on Information Visualization*, pages 51–58, 1997.

[16] E.A. Kellogg and N.D. Juliano. The structure and function of rubisco and their implications for systematic studies. *Amer. J. Botany*, 84(413), 1997.

[17] W.J. Kent et al. The human genome browser at UCSC. *Genome Res.*, 12:996–1006, 2002. genome.ucsc.edu.

[18] Donald E. Knuth, James H. Morris, and Vaughn R. Pratt. Fast pattern matching in strings. *SIAM J. Computing*, 6(1):323–350, 1977.

[19] John Lamping, Ramana Rao, and Peter Pirolli. A Focus+Content technique based on hyperbolic geometry for viewing large hierarchies. In *Proc. SIGCHI*, pages 401–408, 1995.

[20] Wayne P. Maddison and David R. Maddison. *MacClade: Analysis of Phylogeny and Character Evolution. (User's manual)*. Sinauer Associates, Sunderland, MA, 1992.

[21] C. Mayor et al. VISTA: Visualizing global DNA sequence alignments of arbitrary length. *Bioinformatics*, 16:1046–1047, 2000. Application Note.

[22] Shinichi Miyata et al. Two different thymidylate kinase gene homologues, including one that has catalytic activity, are encoded in the onion yellows phytoplasma genome. *Microbiology*, 149:2243–2250, 2003.

[23] Tamara Munzner et al. TreeJuxtaposer: Scalable tree comparison using Focus+Context with guaranteed visibility. *SIGGRAPH*, pages 453–462, 2003.

[24] W. J. Murphy et al. Resolution of the early placental mammal radiation using Bayesian phylogenetics. *Science*, 294(5550):2348–51, 2001.

[25] K. Oshima et al. Reductive evolution suggested from the complete genome sequence of a plant-pathogenic phytoplasma. *Nature Genetics*, 2004. In press.

[26] K. Rice, M. Donoghue, and R. Olmstead. Analyzing large data sets: rbcL 500 revisited. *Syst. Biol.*, 46:554–562, 1997.

[27] George Robertson, Stuart Card, and Jock Mackinlay. The cognitive coprocessor architecture for interactive user interfaces. In *Proc. UIST*, pages 10–18, 1989.

[28] K. Rutherford et al. Artemis: Sequence visualization and annotation. *Bioinformatics*, 16(10):944–5, 2000. Application Note.

[29] Manojit Sarkar, Scott S. Snibbe, Oren J. Tversky, and Steven P. Reiss. Stretching the rubber sheet: A metaphor for viewing large layouts on small screens. In *Proc. UIST*, pages 81–91, 1993.

[30] T. D. Schneider, G. D. Stormo, M. A. Yarus, and L. Gold. Delila system tools. *Nucl. Acids Res.*, 12:129–140, 1984.

[31] Scott Schwartz et al. PipMaker: A web server for aligning two genomic DNA sequences. *Genome Research*, 10(4), 2000.

[32] Nameeta Shah et al. Phylo-VISTA: an interactive visualization tool for multiple dna sequence alignments. *Bioinformatics*, 19, 2003. To appear, Application Note.

[33] Rhazes Spell, Rachael Brady, and Fred Dietrich. BARD: A visualization tool for biological sequence analysis. In *Proc. IEEE Symposium on Information Visualization*, pages 219–226, 2003.

[34] L.D. Stein et al. The generic genome browser: a building block for a model organism system database. *Genome Res.*, 12(10):1599–610, October 2002. www.gmod.org.

[35] Praveen Thiagarajan and Guang Gao. Visualizing biosequences using texture mapping. In *Proc. IEEE Symposium on Information Visualization*, 2002.

[36] Dennis P. Wall and Joshua T. Herbeck. Evolutionary patterns of codon usage in the chloroplast gene rbcl. *J. Mol. Evol.*, 56:673–688, 2003.

[37] D.L. Wheeler et al. Database resources of the National Center for Biotechnology Information: 2002 update.

[38] Jiajie Zhang. The interaction of internal and external representations in a problem solving task. In *Proc. 13th Annual Conf. of Cog. Sci. Society*. Lawrence Erlbaum Assoc., 1991.

[39] Olga Zhaxybayeva, Lutz Hamel, Jason Raymond, and J Peter Gogarten. Visualization of the phylogenetic content of five genomes using dekapentagonal maps. *Genome Biology*, 5(20), 16 February 2004.


Figure 3: We show the 1001 bp CNR1 gene, a subset of the Murphy *et al.* 44-sequence dataset [24], after exploration revealed four meaningful thresholds of differences in nucleotide site variability: 67%, 60%, 50%, and 25% from top to bottom . The columns marked at the 60% level were expanded, with a middle region between two of them stretched out.

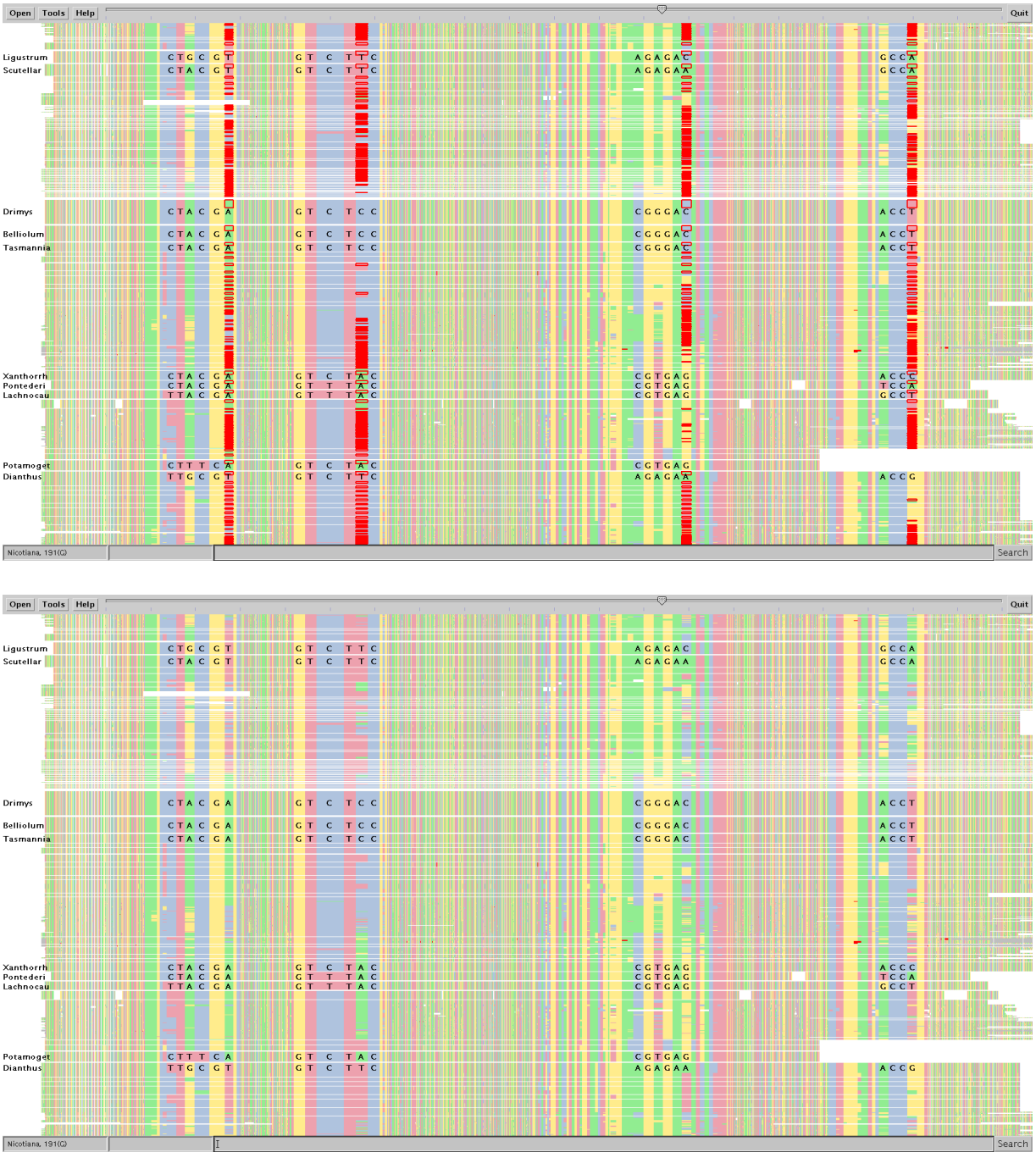


Figure 4: The complete 714 Kbp `treezilla` dataset [7, 26] contains 500 seed-plant rbcL sequences of 1428 bp each. The highlighted marks in the top view are at the difference threshold of 62%, which we found after brief interactive exploration. The regions around highlighted areas are expanded, and the bottom view shows the same scene without the red highlight marks so that the distribution of colors that represent nucleotide type can be also seen in those vertical strips. The visible patterns of color show many of the codon bias results described by Wall and Herbeck [36], as we describe in the main text.