A Visual Comparison of Piano Performance

Kevin Mackenzie
Linus Koepfer
mackek4@rpi.edu
koepfl@rpi.edu
Rensselaer Polytechnic Institute

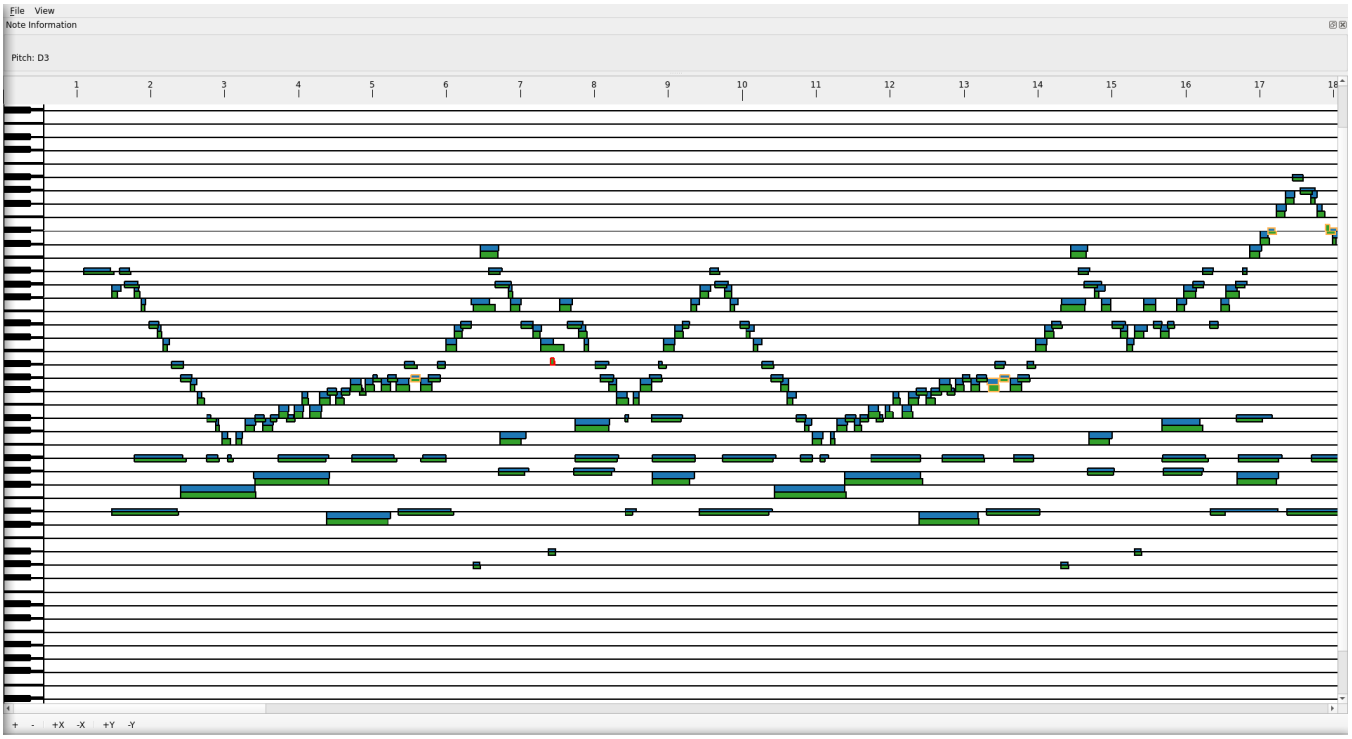

Figure 1: Visualizing the first few phrases of Schubert's Impromptu in E-Flat major (D.899 No.2) with corresponding note alignment enabled; imperfect recordings produced by author

ABSTRACT

When performing a piece of piano music, especially those representative of the Romantic era, there are elements of the music beyond the written pitches and rhythms that establish the emotion of the performance. Professional musicians learn to identify and manipulate these elements in their interpretations, like how articulation, dynamics, and *rubato* affect phrasing, and how voicing draws the user attention to certain sequences of pitches. For non-professionals, it can be difficult to identify and, subsequently, control these higher-order features of the music. We present the groundwork for a method of visualizing two performances of a piece of music comparatively to aid the analysis of such higher-order features.

1 INTRODUCTION

It is of interest to identify the differences between two performances of a piece of music to gain better control over the material as a player. This is true both for comparing one's own recording to a textbook or reference recording for "correctness" and for comparing two equally correct recordings for difference in style. While a skilled musician can identify these differences by ear, an automated tool for identifying differences could be useful for learning at a lower level, especially if consistent professional instruction is not accessible to the student.

We thus set out to find a way for learning musicians to identify subtle differences between musical performances that they may be unable to perceive aurally. We hypothesize that a visualization of two performances of a piano piece where notes are presented on the same keyboard, with auxiliary information on tempo and velocity will allow untrained musicians to perceive more subtle differences in performance than they could by ear.

Using Musical Instrument Digital Interface (MIDI) recordings of piano pieces, we propose a tool to create a visual representation of the difference between two artists' performances to help novice to intermediate musicians recognize differences in both correctness and style that they would not have recognized aurally. In addition

to simple user interaction such as moving and scaling the visualization, we describe a technique to analyze performance differences through a modified edit-distance algorithm and visualize the resulting discrepancies. Our current tool focuses on a note-by-note pitch comparison of two performances, but we also discuss comparisons of tempo and dynamics of a piece.

2 RELATED WORK

2.1 Music Visualization

Previous forms of music visualization have grouped features of a musical performance into first-order data, such as dynamics, note onset and offset, and pedal pressure, from which one can derive second-order data such as note duration, beat, tempo, and articulation[4]. First-order data may be captured directly from MIDI recordings or through algorithmic analysis of audio recordings[5]. Second order data can then be calculated from these first-order events as described in the following section.

Many methods exist for visualizing both first- and second-order data of a single musical performance. The classic score using musical notation is legible for trained musicians, but discretizes note timing to rational intervals. A piano-roll with time on the horizontal axis and pitch on the vertical is commonly found in Digital Audio Workstations (DAWs). Many such programs allow for the quantization of notes, realigning them to fractional subdivisions of a musical section, but piano roll visualizations also allow for notes to be displayed at specific points in time not correlated with musical subdivisions. While auxiliary markings on a musical score can be used to indicate tempo and dynamics of a performance, standard piano rolls lack this capability. This leads to more exotic visualizations such as the 3D Piano Roll [4], which displays dynamics, represented by MIDI note velocity, on the third axis. Other less common visualizations of musical performances include line graphs of tempo and velocity over time, and even the *performance worm*[3], which tracks tempo on the horizontal axis, and loudness on the vertical. By animating these performance characteristics over time, with past points either fading in saturation and/or moving further into the background, this creates an interesting visualization of the way a piece rises and falls in both speed and volume.

2.2 Analysis of Second-Order Performance Data

While it is relatively trivial to calculate second-order musical data after the completion of a performance, it is desirable to do so in real-time, to allow the performer live feedback on their interpretation of the piece. One of the more complex tasks is identifying the location of primary beats in a piece of music without the information provided by a musical score. An initial attempt at this task was the use of a single-notion model, which uses a weighted average of previous perceived tempos and a confidence mechanism to compute the current perceived tempo. The limitations of this single required state led also to a beam search model that simultaneously considers multiple interpretations of the performance[1] and prunes the search tree as new information is perceived. Neither of these methods have been extended to compare the analysis with a prerecorded performance of the piece.

2.3 Comparison of Musical Performances

Trivial comparison of two versions of a musical performance involve simply displaying visualizations of the two pieces side-by-side, or perhaps plotting relevant line graphs or performance worms on the same axes. Such visualizations generally do not display specific note information, due to the vast area required to display a score or piano roll. Our tool thus aims to provide note-by-note information so that a user may analyze not only the overall differences in interpretation of a piece, but also the correctness of individual performances and the liberties taken in improvisations. We do so by taking advantage of the interactivity provided by a digital environment to allow us to display the full set of notes for nontrivial pieces without overwhelming the user.

Treviño and Sapp[6] provide a music-staff based visualization method for multiple historical performances, but it is limited in that it assumes all recordings contain exactly the same musical events as the reference score. Additionally, a score based visualization requires knowledge of musical notation, creates visual distance between two performances making it harder to compare note onsets, and discretizes note durations to standard musical durations rather than exact times. They recognize that their use of similar notation elements to standard sheet music in non-standard ways can also be confusing. Our aim is thus to create a tool for comparison of performance styles that takes advantage of an alternate visualization to enable more clear and direct comparison.

The work done by Dannenberg [2] for real-time accompaniment is based on a similar dynamic programming as the one we chose and considers time of note onset in addition to pitch. For this application, they consider various heuristics based on timing information and the assumption of coherence in the matching between the performance and the score. It is designed for monophonic performances, so they assume that the events are in a set order and make no attempt to reorder events in the matching. Because of this, their method is not suitable for matching piano performances specifically.

3 VISUALIZATION TECHNIQUE

3.1 Performance Visualization

As discussed above, the quantization to rational subdivisions of musical measures provided by a musical score does not allow us to effectively visualize the slight differences in note timing between two musicians' *rubato*s, *ritardando*s, or *accelerando*s. We therefore use a piano-roll based visualization, with pitch displayed aligned with a piano keyboard on one axis, and time in seconds on the other. Following the layout of most DAWs and time-based visualizations, we place time on the x-axis and pitch on the y. We propose small changes to this common visualization to indicate additional metrics, such as the use of glyphs or different note shapes to indicate articulation. We additionally suggest displaying current tempo information as a line graph on the same time axis, and indicating pedal data either with another overlaid graph, or with a special pitch value at the top or bottom of the keyboard. Due to the narrow time frame for this project, we chose to eschew the development of these additional features to focus on the primary comparative visualization.

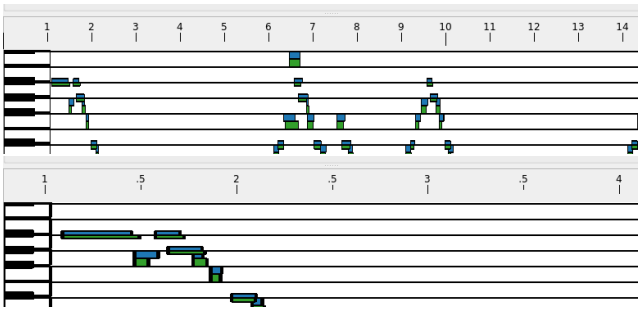

Figure 2: Viewing detail and overview length time scales

3.2 Interactivity

3.2.1 Navigation. To allow the display of note data for a full performance without overwhelming the user, we allow for standard pan and zoom interactions with the piano roll visualization. Users may use the toolbar at the bottom of the visualization to zoom either the entire visualization or one axis at a time. Zooming on the pitch axis rescales the displayed keyboard along the pitch axis, but keeps the length of the keys consistent to avoid the keyboard disappearing at overview levels and taking up too much of the screen at detail levels. Zooming on the time axis adds or removes axis ticks at additional time subdivisions, ensuring that the axis is not cluttered at overview levels, and that enough marks exist at detail levels to allow for analysis, as seen in Figure 2.

Users may use the scroll bars at the bottom and left of the visualization and the corresponding scroll wheel(s) on their mouse to move around the piano roll when zoomed in. The scroll bar for an axis disappears when the visualization is zoomed out to a level where the full range of that axis is in view. For additional interaction, we propose a feature like the "hand" tool in many image manipulation programs, that allows the user to select the tool in order to click and drag to move the canvas, and then deselect the tool to return to standard interaction. Once again, there was insufficient time to implement this feature.

3.2.2 Note Information. In order to allow the user to further analyze the performance, we provide for the highlighting of notes that are moused over. When the user hovers the mouse over a note, the note changes color as seen in Figure 3, and note information is displayed in a toolbar window. This window can be opened and closed from the view menu, and moved to different areas of the screen. The window currently displays only the pitch of the note, but could easily be modified to display note velocity, onset and offset time, and duration.

3.2.3 The Scrubber. We also implemented a scrubber, as found in many DAWs, which is a secondary cursor movable along the time axis. The scrubber spans the width of the pitch axis, and highlights all notes currently beneath it, as seen in Figure 3. It can be dragged to a different location by the user, and jumps to the location of the mouse when the mouse is clicked on the visualization. Other than these interactions, it remains in a fixed position and can be scrolled or zoomed off the visible screen. The scrubber is a useful interaction for allowing the highlighting of multiple notes played at the same time, allowing the analysis of musical chords (groups

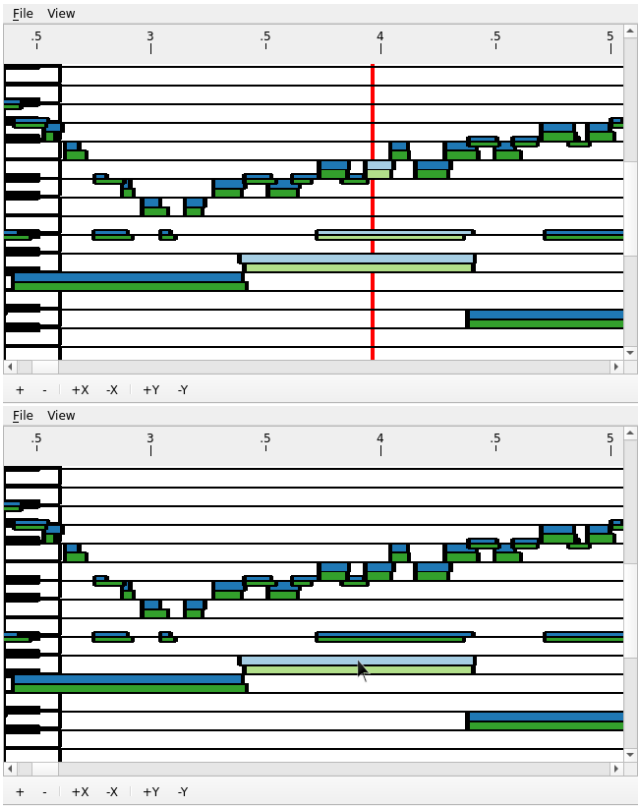

Figure 3: Note relationships can be highlighted with the scrubber or with the mouse

of simultaneous notes with a harmonic relationship). Additionally, while we focused on the visual component of the tool, if audio playback were implemented, the scrubber would be used to indicate the corresponding location in the visualization of the current audio when playing and to set the playback point of the audio when paused.

3.2.4 Menu Interactions. We also implemented a small form of the standard menu found in most applications. The file menu allows the user to load their own MIDI performances into the visualization, making it a tool rather than a single visualization. The view menu allows toggling the visibility of the note information pane (discussed above), and the alignment of notes for comparison (discussed below).

3.3 Performance Comparison

To enable the comparison of two performances of a piece, we display the MIDI data of both performances on the same piano roll. Each performance's note takes up half the space on the pitch axis for that note, with notes of one performance always above those of the other. This would visualize two identical performances as a standard single-performance piano rolls with split-color notes, and shows discrepancies in note timing or pitch through non-aligned note boxes. We also differentiate the performances through note

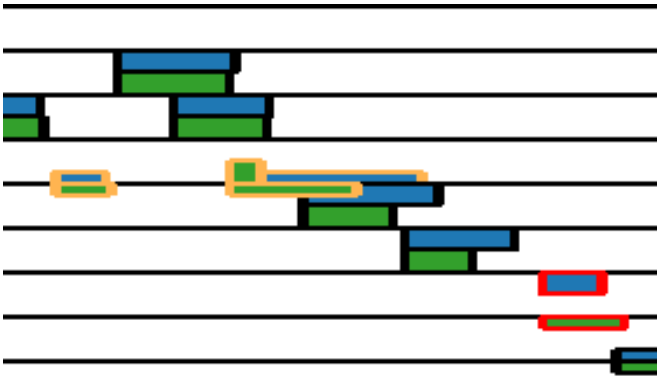

Figure 4: The stroke color of the note rectangles indicate player "mistakes" (red) and failures to make one-to-one matchings (orange)

colors, using a paired qualitative brewer color scheme. Each performance has its own qualitative pair, with the lighter color in the pair indicating currently highlighted notes. Since the black keys on a piano are narrower than the white keys the MIDI data on black keys appears narrower. Some software, such as Ableton Live 10, use uniformly sized keys instead. We speculate that the different key size would make more sense to piano players, but we have not collected data to verify this.

3.3.1 Mistakes and Imperfect Matching. Keeping with the theme of supporting imperfect data, we must visualize when we detect that notes are incorrect and indicate when perfect matchings cannot be found. One otherwise unused visual dimension is the stroke color of the notes. We define mistakes as a one-to-one match of notes where the pitches of the two notes differ. We use red to indicate this since red is commonly used for indicating error in software.

If a connected component of a given note has more than two notes in it, which means it is not a one-to-one relationship, we consider this a matching failure. Matching failures could be caused by a error by the performer or the algorithm. We want to indicate this as an error that is less severe than a mistake, but also want to maintain good contrast with the two other colors we chose. The light-orange we use strikes a reasonable balance of contrast with both the two note colors and the white background, but is not optimal. Other colors with better contrast may be less intuitive, so perhaps other visual dimensions could be employed with better results.

3.3.2 Note Alignment. In addition to highlighting related notes on mouse and scrubber hover, we allow the user to align matching notes on the time axis. This option can be toggled via the *View* menu. In its current state, it can be used to see how time is stretched throughout a piece through the variation in onsets of corresponding events. One method we did not implement that would help the user explore specific parts of a piece would be to let them choose a custom focus note that is always aligned instead of aligning at time zero. This would allow them to see the expansion of time around any point.

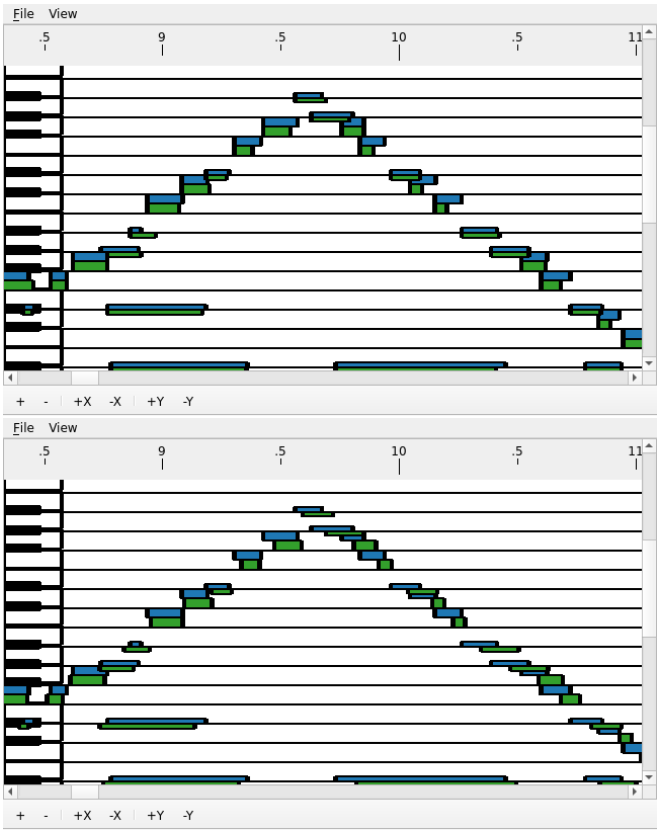

Figure 5: The aligned on and aligned off views can show the variance of onset times and slight changes in tempo

3.4 Real-Time Comparison

Given more time for the project, our aim was to implement a real-time component, in which the user could play the piece on a MIDI-enabled keyboard with played notes animating onto the visualization below a pre-loaded reference performance. This would require dynamic recomputation of the analysis algorithm (discussed below), and eventually real-time analysis of second-order characteristics to display volume and tempo information. Ultimately, we determined that such analysis is computationally complex, and that the only required real-time components would be MIDI playback and recording. Analysis could be performed either with a delay (as for note alignment), or in a single pass at the conclusion of the recording (as for tempo and volume information).

Allowing the user to record themselves in-program is a valuable convenience since they may want to iteratively record and analyze themselves against a reference to emulate others' techniques, verify evenness, or evaluate any other metric they choose. Allowing them to play-back both recordings separately in-program could help reinforce the aural recognition of such elements.

4 IMPLEMENTATION

4.1 Note-by-Note Performance Comparison

As the back-end of the comparative elements of the visualization is an algorithm for matching MIDI events between two streams of note-on/note-off events. We do a short pre-processing step that reduces the event stream into only note-on events and encode the duration of the note in each note-on event. Each of these consolidate events is treated as a character in a string and the pair of strings are run through a variant of edit-distance we devised.

4.1.1 Algorithm Formulation. The classic edit-distance problem between string $A = A_1A_2..A_i$ and string $B = B_1B_2..B_j$ is defined as

$$weight(i, 0) = i * w_{insert}$$
$$weight(0, j) = j * w_{delete}$$
$$weight(i, j) = min \begin{cases} weight(i-1, j) + w_{delete} \\ weight(i, j-1) + w_{insert} \\ weight(i-1, j-1) + w_{transform}(A_i, B_j) \end{cases}$$

If one considers each *note* to be a character in the string, then one can apply the edit-distance algorithm literally. However, one key feature of the piano, and of non-monophonic music in general, is that multiple pitches may be played at nearly the same time (i.e. chords), so the order they occur cannot be predicted. Since our method is designed to work on imperfect reference files, we cannot assume the order of corresponding notes.

To achieve reordering of pitches, we accumulate a bipartite graph at each tile. For each the $(i, j-1)$ (insertion), $(i-1, j)$ (deletion), and $(i-1, j-1)$ (match/transform) case, we add corresponding edges to the graphs. For the $(i, j-1)$ case, instead of attaching an edge from A_i to B_{j-1}, we instead search backwards within a small time window of B_{j-1} to find the pitch that most closely matches the pitch of A_i. For closely matching inputs, this will find the most recent note of the same pitch in the other string. The $(i-1, j)$ case is analogous to the $(i, j-1)$ case and the $(i-1, j-1)$ case performs this edge insertion method both from A_i to B_j and from B_j to A_i. Note that these graphs are not multigraphs and duplicate edges are ignored. While this edge insertion logic does help to ensure an optimal solution is found at every step, the meaning inherited from edit distance of matching or transforming characters is lost.

To assign a weight to a given graph to determine which case is optimal, we considered several different metrics. The most obvious metric is pitch, but additional metrics like note velocity, and timing-related features were also of interest. Since the variance of dynamics is more closely tied to the use of second-order methods, we found it not suitable for matching. For timing, we did consider a metric that would use the onset frequency in the neighborhood of the two notes to project the position of the two notes in each other's time-space, but this was largely unreliable, and, similar to dynamics, highly dependent on the use of second-order methods that require higher-level analysis. Relative timing does not necessarily predict matching well.

Due to the unreliable nature of these other metrics, we decided to use pitch as the sole metric for weight on the edges of the graph. However, another metric for the quality of a matching is the number of orphan nodes. Since our edge insertion method allows reordering within a time window, any nodes of degree zero within the reorder window are not considered orphans, but all notes before are. We found a good total weight to be the sum of the squared pitch-difference of edges and the number orphan nodes times a constant. Further tuning will be required to handle cases with non-trivial missing or added sections in a way that makes sense musically, however.

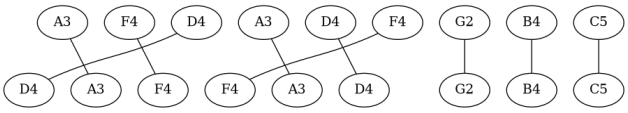

Figure 6: Graphviz render from Chopin B.150 matching

One artifact of this method, since we don't penalize edge count, is that it produces connected components of notes of the same pitch when the distance between any connected pair is less than the reorder window. For the case that the connected component has the same number of notes on each side, we can simply remove edges that aren't between notes of the same index in time-order. For the case that the connected components do not have the same number of notes on each side, we leave it be. It should be noted that in the latter case, cutting some of the edges would make sense improve the results, but we did not investigate such improvements.

To validate results for small inputs prior to a sufficient visualization, we use Graphviz. Fig 6 shows a small excerpt from Chopin B.150, which was a piece we chose for testing since its pitch content is relatively simple and can be played very expressively. The edge crossings on this snippet shows the same second-inversion D chord played twice in sequence and, in both recordings, the three notes appear in different order the second time.

4.1.2 Memory Optimizations. There are two primary optimizations to make this algorithm run with satisfactory memory usage.

First, edit-distance only relies on two diagonal lines at a time while writing to a third. This means that we only need to keep track of three diagonal lines worth of memory at a time (which is the length of the smaller string at most). This insight brings the number of tiles in memory from $O(n^2)$ to $O(n)$.

Second, if one assumes an upper-bound on the connectivity of the graph, since single notes should rarely be connected to very many other notes, the size of the graph can be reduced from $O(V + E) \approx O(n^2)$ to just $O(V) = O(n)$. Additionally, this optimization lets us define the graph edges as a pair of contiguous arrays (left-to-right and right-to-left), which yields constant-time lookup, insertion, and deletion and fast copying. This reduces per-tile memory usage from $O(n^2)$ to $O(n)$.

With these two optimizations, we reduce memory usage from $O(n^4)$ to $O(n^2)$, which, in practical terms, reduces memory usage from petabytes to single gigabytes for 2-minute long pieces of high complexity ($n \approx 2500$).

4.1.3 Parallelism. We use the diagonal-line method for parallelizing the edit-distance algorithm. This method takes advantage of the fact that tiles along diagonal lines running from the bottom-left to the top-right are independent of each other. Each of these can be run in parallel and synchronize when the diagonal line is complete.

A GPU-based method we did not have time to implement would be to design a kernel program that processes all tiles along a diagonal line running from top-left to bottom-right and synchronizes with all of the other running kernels after each step. Although there is some overhead with synchronization, we suspect this would be faster than re-launching the kernels from the CPU for each diagonal line. Large numbers of tiles would be idle in the beginning and towards the end as the length of the diagonal lines are small, but most clusters of threads would either all be running, or all be idle. Only two clusters of threads should suffer from a shared instruction pointer, where some of the threads are idle and some are doing work, which means overall utilization scales well with the string size. Since each tile only accesses tiles in its neighborhood, we can get good memory locality with this method.

4.2 Tools and Technology

Due to the computational complexity of the analysis algorithm, we chose to implement our tool using the C++ programming language due to its overall low overhead, low-level memory operations, and first-class support for parallel libraries like OpenMP. We chose the Qt User Interface framework for our user interface due to its extensive documentation and cross platform support. By choosing a C++ user interface framework, we attempt to avoid the difficulties with language interop. We used Qt's QWidgets to assemble the primary interface, including file loading, window management, and toolbars. We used Qt's QGraphics framework as the canvas element for our piano roll visualization. While this canvas is lower level and less convenient than web-based user interface, it offers an acceptably high level of control without too much complexity. As the authors were using different operating systems, we used CMake to ease cross-platform development, but did not add linking declarations required for Windows platforms.

4.3 Division of Labor

The first author (Mackenzie) focused mainly on the note matching algorithm and the visualization of the calculated note relationships (or lack thereof). The second author (Koepfer) focused primarily on interactivity, including navigation, the axes and the scrubber, and note highlighting. Both authors contributed evenly to overall development and refactoring of the code, to the creation of the project proposal, presentation, and this paper, and to the presentation of the project.

5 EVALUATION

During the development and presentation of this work, we received feedback about the visualization in the form of questions and critiques from students and the professor, but did not reach a point in completeness to work directly with out target audience for feedback.

One question was about the placement of the piano axis on the vertical axis versus the horizontal. While it is unnatural to see a piano vertically as we have it, it is common to use the horizontal axis as the time-axis in audio and video editing software. The horizontal piano orientation may be more natural for players who are unfamiliar with software, especially if recording in-program, so we would like to support both layouts.

Another question was the use of outlining on notes instead of shading to indicate mistakes or mismatches. While it is possible to encode several more note states in the shaded color of the two parts, we decided that the we wanted to preserve the information notated by the two states (normal and highlighted) without needing to consider combinations (e.g. highlighted + mistake). However, we recognize that other visual variables may be more appropriate for indicating such relationships.

One critique is the inconsistent, clumsy, and primitive appearance of the notes on the canvas. Since the canvas is not anti-aliased and since we do not maintain a constant stroke-width when scaling, the notes appear to be very pixelated when zoomed in. Either by scaling the stroke width when zooming in, or using a different method to facilitate zoom we could mitigate this issue. Maintaining a consistent stroke would also improve the appearance of mistake and mismatch notes' outlines as described above.

We received several questions about non-piano instruments. While this method is not confined to the piano, it becomes more complex since other instruments have a wider range of controllable timbre. For non-western music and continuous-pitch instruments, we would need to develop a more general concept of musical events and analyses specific to those musics and instruments.

6 CONCLUSION

The method we developed in this work seems to be a good proof-of-concept for a visualization of two piano performances with note relationship information based on the feedback we received from our peers. We believe this representation can be refined to be more visually appealing and can be extended to show more data in the same visual space for faster and more precise visual analysis of the differences between two performances.

6.1 Future Work

Due to time constraints, there were several comparison features we did not get to implement. In the future, we would like to make visualizations for higher-order data, such as relative time contraction and expansion, voicing, and phrasing. For first order data, we would like visualize dynamics to increase visual salience of louder notes and adjust note durations to transform between time-spaces of the input when viewing aligned data.

In addition to the previously discussed additional navigation interactions and note information, it may be beneficial to implement an optional pane similar to the note information pane that displays comparisons of overall performance metrics, such as bar graphs comparing average tempo and average volume.

REFERENCES

[1] Paul E Allen and Roger B Dannenberg. [n.d.]. Tracking Musical Beats in Real Time. https://www.cs.cmu.edu/~rbd/papers/beattrack.pdf
[2] Roger B Dannenberg. 1984. An on-line algorithm for real-time accompaniment. In *ICMC*, Vol. 84. 193–198.
[3] Simon Dixon, Werner Goebl, and Gerhard Widmer. 2002. The Performance Worm: Real Time Visualisation of Expression based on Langner's Tempo-Loudness Animation. (04 2002).
[4] Martin Gasser. 2005. *Interactive visualization of expressive piano performance.* na. http://www.ofai.at/~martin.gasser/archive/thesis.pdf
[5] Lucas Maia and Luiz Biscainho. 2014. On the extraction of parameters from expressive musical performances. http://www02.smt.ufrj.br/~lucas.maia/papers/MaiaLS_AESBR2014.pdf

[6] Jeffrey Treviño and Craig Sapp. 2014. Automated Notation of Piano Recordings for Historic Performance Practice Study. *J. Comput. Cult. Herit.* 7, 3, Article Article 17 (Aug. 2014), 7 pages. https://doi.org/10.1145/2597179