MusicScroller: A Solution for Automating

the Traversal of a Digital Composition

Bryan Hutto

University of North Georgia

Department of Computer Science and Information Systems

82 College Cir, Dahlonega, GA 30597

**ABSTRACT**

For the musicians of the information age, access to sheet music is better and easier than it has ever been, but this advantage comes at a cost. Digital sheet music is not conducive to sight reading, particularly in the case of long pieces with many page turns. In this paper, I describe MusicScroller, a tool for practicalizing the use of digital sheet music by seamlessly scrolling through the selection at its intended pace. This paper will detail its methods of action and demonstrate its efficacy.

**1. INTRODUCTION**

In an increasingly post-print world, much information that was traditionally stored on paper has been almost entirely transferred to digital media. The areas where this is not the case often tend toward one of two categories: information whose security status requires physical layers of protection and information which, due to human physiology, is most easily consumed in their physical state. In the past, Novels belonged squarely in this latter category, and it was largely in an attempt to imitate a book’s ease of use that the Amazon Kindle and similar products were designed. Another type of print media that has suffered a somewhat diminished incarnation in digital form is sheet music.

A musician’s relationship to their sheet music is a critical one, largely built around the act of page turning; practiced musicians can fluidly turn a page then continue playing in fractions of a second, and sheet music is written to synchronize breaks in the music with page turns as a matter of course. However, much of this ease of use is lost when reading off of a screen, and for many musicians, particularly those less financially privileged, a library where digital sheet music represents the dominant portion is normal. Moreover, they may be stored as pdfs, word or text documents, images, or any number of other file types, for which the few, and often expensive, solutions available may be ineffective. The products that do exist tend to focus on replicating the traditional page for a touch screen. However, other possibilities seem to go largely unexplored, such as a timed discrete interval page change, or page turns triggered by audio feedback, or physical input approaches that do not rely on a touch screen or using a proprietary file format. Practical solutions for the average musical hobbyist are not readily available, affordable ones less so.

It is worth noting as well, that, while Kindle-like products are effective at reproducing the process of reading a physical book, there are speed reading programs, such as Spritz, that claim to demonstrate digital media can actually be superior to traditional media with regards to certain metrics. In addressing the problem presented by digital sheet music collections, this project presupposes that the necessity of the page turn may be a holdover from print media that musicians could perform more easily without and opts for a continuous page scroll approach, with respect to the particular geography and time signature(s) of the piece.

**2. BACKGROUND**

Despite my familiarity with using digital sheet music, or even because of it, it is of particular importance to investigate and understand the empirical realities of how sheet music is generally read and played, to avoid designing for personal biases and eccentricities. If a program is to scroll in time with the music on a page, it must keep pace with the reader’s focus with regard to the tempo and time signature of the piece, representing the play rate, as per measure, which can be calculated algorithmically from those two inputs. However, a sight reader will actually be looking a variable distance ahead of the notes being currently played. This distance is referred to as the eye-hand span (EHS) by Rosemann, Altenmüller, and Fahle [1], who attempt to quantify this value inductively given different variables, such as piece complexity and the subject’s familiarity with it. A scrolling function would need to wait an initial amount of time consistent with this value before matching the pace of the piece. They also give a comprehensive review of other researchers’ findings with regard to the EHS, including Lechmann and Kopiez [2], who describe that a sight reader’s focus will frequently travel back from the EHS’s current locale to the notes currently being played, as a kind of error checking. Given this behavior, it is important that the previous notes, going back at least one EHS, remain in view during scrolling.

Wallace, Savage, and Cockburn [3] experimented with what page scrolling speeds, in terms of most comfortable and maximum tolerable, still allowed subjects to subjectively identify important features in a document, and what effect different magnifications had on these speeds, a relationship which was inversely proportional. While this experiment involved mixed text and image documents, their findings are especially important to us. Higher magnification necessarily allows for clearer element resolution, but moreover it decreases the amount of image blur. If a user is to read a page that is actively scrolling, image blur must be kept at a minimum, but this must be balanced against keeping a wide enough magnification view to include the EHS with tolerance as well as maintaining the correct speed for the piece of music. Quinn, Cockburn, Casiez, Roussel, and Gutwin [4] attempted to reverse engineer complex scrolling transfer functions, which transform human input into on-screen scrolling, with various drivers and devices. They provide a useful background on how scrolling is implementing, its inherent relationship with device drivers, and the Windows messages that can execute it without physical operation of the mouse or trackpad.

Taele, Barreto, and Hammond [5] built an educational tool which, among many other features, can recognize hand-written measure breaks with a high degree of accuracy, using line tests. A similar, albeit exceedingly simpler, line test will be implemented in order to detect, locate, and document measure breaks in the selected piece of music.

**3. APPROACH**

**3.1 TARGETS**

A cursory review of digital sheet music available for download, either via sharing or purchase, will reveal the complete dominance of Adobe’s Portable File Format. Therefore, building a solution that will work for the average musician presents a challenge, as PDFs are notoriously difficult to data mine.

This version of MusicScroller is only intended to run on the Windows 10 operating system, with Google Chrome installed in the default directory.

**3.2 REQUIREMENTS**

The most critical requirement of MusicScroller is that it is able to accurately analyze user-selected score and transform it into an actionable scroll speed map. It must account for variations in music notation such as bar thickness and length, the number of lines on a page, and measure length. From there, it must execute an intelligent scroll from the beginning to the end of the piece. The perspective transition should be as smooth as possible, maximizing readability by reducing motion blur.

Another important criterion is MusicScroller’s ease of operation. Hard to navigate command switches, complicated installs, and anything else that might depend on specialized technical knowledge should be avoided. The objective here is to engineer a hands-off tool that the typical musician can pick up and use almost instantly. Relatedly, the tool should require no user interaction at all after receiving the necessary inputs through the end of the scrolling operation. Should the user wish to terminate the scroll prematurely, however, that must be allowed for. Even with perfect functionality from MusicScroller, the musician can become out of sync with the scroll due to mistakes or distractions. They should not have to redo the input process in either this scenario or one where they simply wish to replay the same piece of music.

The tool must give comprehensible feedback if an error is encountered, and it must clean up after itself upon termination.

**4 IMPLEMENTATION**

**4.1 IMAGE HANDLING**

When the program executes, the main method calls a handler method. This method then presents the user with a Windows open file dialog box, from which they can navigate to the desired sheet music file and select it. Thanks to Windows File Explorer properties, this dialog box will start in the previously selected directory on subsequent executions. The user is afterwards prompted to type the tempo, in beats per minute, and then the time signature, in beats per measure, into the console.

For the next step, the handler utilizes Magick++, a powerful API which allows C++ calls to the ImageMagick image-processing library. The API allows conversion of PDFs, otherwise nearly impossible to rasterize, into a manageable BMP format with a uniform bit depth and with the alpha channel removed if present. The new file is then saved temporarily into a predetermined project directory. If the file contains multiple images, as will almost always be the case with the targeted PDFs, they are saved as separate BMP files, the distinction between which will be the basis for our Page class.

Owing to ImageMagick’s unique versatility, this same code will work equally well for JPGs, PNGs or any other standard image format. Unfortunately, Magick++ is a somewhat arcane API, and some of the documentation has become unreliable with updated versions, which has concerning implications for future refactoring or updating of MusicScroller. This, however, is unavoidable, as alternative open-source image processing libraries that can handle PDFs are limited and share this deficiency.

Each newly created BMP file is read into an input buffer, after stripping the header and reserving the relevant metadata. An unsigned short two-dimensional matrix is then constructed from the pixel data, in accordance with BMP format particularities, such as the inverted vertical axis and atypical padding. The resultant matrix dimensions and elements match the pixel data of the image, with a simple one or zero representing opacity or lack thereof respectively.

A page object may then be instantiated and its member, EnumMeasures(), can be passed the matrix. The newly created Page object is then pushed into a storage vector and the next BMP file can be read.

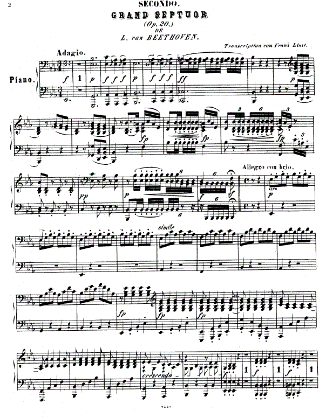
**4.2 ELEMENT TRIAGE**

The bulk of the Page Class is devoted to the EnumMeasures() method, which is responsible for enumerating and locating every line and measure in the page of sheet music.

The method implements nested for loops to iterate through each “pixel” in the matrix, checking for a zero. A Checker variable will preempt the next step if a measure divider was encountered recently on the same horizontal plane, as this will represent a thick or double bar rather than that of a new measure. Next. a while loop iterates down the y-axis as long as it encounters a zero, incrementing the height variable for the opaque segment. If the height of the traversed segment is near enough that of the bar factor variable, it is presumed to be a bar. The bar factor is initially underestimated, as a function of image height, but is replaced if a longer bar is found. Since a bar is unerringly encountered first in a musical score, this prevents abnormally tall note stems from throwing off the algorithm.

Once a bar is confirmed, the method checks if the bar is on a new line by comparing the height of the topmost pixel of the current bar with the height of the bottommost pixel of the pervious bar. If successful, a new row is recorded and the last measure of the previous row is deleted to account for line ending bars. Otherwise, the current row’s measure count is incremented. In either case the bar coordinates are recorded. By the end of function execution, all pertinent data has been stored in member variables that can be retrieved from the scrolling procedure in the handler method.

Despite attempts to provide tolerance for slanted lines and photocopy marks, this algorithm does not perform well for low quality scans, and can only be relied upon with perfect accuracy in the case of computer-generated sheet music. Fortunately, the latter is one of the most common forms sheet music will take, especially in the realm of sharing original arrangements.



An Ideal Image A Poor Image

**4.3 SCROLL EXECUTION**

The scroller portion of the handler method takes the data stored in Page objects and translates it into an ideal scroll for each section of the piece.

The handler will first open the original PDF file in a new, maximized Google Chrome window, chosen for its ubiquity, fine scroll resolution, and its PDF capability. The scroll is optimized for two-hundred percent magnification. Unfortunately, Google has not yet provided a command switch which can set this value, so the handler will first wait for confirmation from the user before commencing the scroll. This gives them time to set the magnification, if needed, as well as to prepare themselves to play. The confirmation can be given independent of the command console by simply pressing the enter key.

The program calculates the seconds per measure from the tempo and time-signature provided by the user. This represents the amount of time the musician will spend playing each measure. By accounting for the number of lines and the number of measures in each individual line, it can accurately be judged how long each row should be in the center of the screen. After confirmation, the handler will wait one eye-hand span, as judged by the tempo, before commencing the scroll.

seconds per measure = [ (60 / tempo) x (beats per measure) ]

By testing singular mouse wheel scrolls at WHEEL\_DELTA, WM\_MOUSEWHEEL = 120, over different files, I have discovered a reliable factor, 0.24, which describes how much scrolling, in units of WM\_MOUSEWHEEL/30, is required to completely clear a PDF page when multiplied by the page height. This amount of scrolling is represented by the ticksPerPage variable. Based on the seconds per measure and measures per row, the procedure calculates the milliseconds each small tick should take.

To handle these many small delays the handler calls a helper class that creates a synchronous thread to sleep for the required milliseconds for each tick. After the delay, mouse\_event is called to execute a scroll equal to one tick.

However, the vertical distance between lines is not uniform, so the program must also account for the height of each new line, relative to the page height. This factor is multiplied by ticksPerPage to determine how many ticks it will receive, a greater distance between lines means a smaller factor and a faster pace).

Testing revealed that using a factor that accounted only for the current measure was insufficient, resulting in too many sharp speed-ups and slow-downs when either the distance between lines or the measures per line, and therefore the scroll speed, changed. Instead it now implements an algorithm that takes into account: the vertical distance from the previous, current, and next line and the measure count of the previous current and next line, including between pages. This also means the scroll will gradually speed up when approaching a line with fewer measures, giving the musician time to register the change.

row factor = averageOf(vertical distance between current line and [last line, next line]) / height of current page

seconds per line = ((seconds per measure x measures on current line) x .75) + ((spm x measures on next line)\*.25)

ticks = row factor x ticks on this page \* 30 seconds per tick = seconds per line / ticks

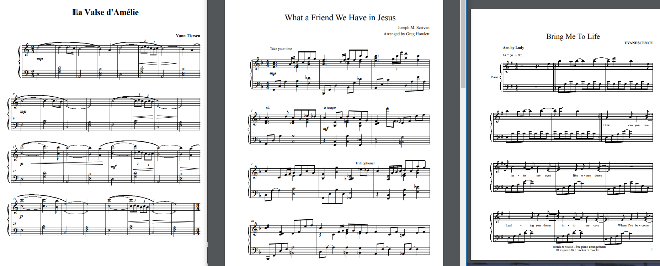
An asynchronous key listener insures the user can interrupt the scroll at any time and from any window of focus by pressing ESC.

**4.4 ENDING CONDITIONS**

After scrolling is completed or interrupted the user is given the options of closing MusicScroller, rerunning with a different file selection and inputs, or rerunning with the same selection and inputs, in the form of a simple integer response. The handler returns this choice to the main method, which will in turn call the handler with the appropriate parameters. An object reference to the Session class is always passed to the handler, allowing it to store the user selections, should they be required on rerun. This means that two key presses are the only barriers for a musician wishing to replay the arrangement.

**5. RESULTS**

MusicScroller was tested with 3 downloaded sheet music PDFs it had never seen before. This selection was limited based on the following characteristics: apparently computer-generated, stave count cannot decrease from one line to another, bars must be continuous from the topmost stave to the bottommost stave. The test scores were then executed at 80, 100 and 140 beats per minute, while a browser metronome kept the time. If the current position of play left the screen, MusicScroller would receive a mark for that piece. If the current position left the screen and did not return, MusicScroller would fail.



Test Scores A, B and C

|  |  |  |  |
| --- | --- | --- | --- |
|  | 80 bpm | 100 bpm | 140 bpm |
| A | 3 | 3 | 3 |
| B | 0 | 0 | 0 |
| C | 0 | 0 | 0 |

Test score “A”, La Valse d'Amélie, by Yann Teirsen, kept tempo with the metronome on all but three occasions. These three occasions each occurred at the last measure on the page, and the occasions were consistent across the three test speeds.

Test Score “B”, What a Friend We Have in Jesus, by Joseph M. Scriven, Arranged by Greg Howlett, kept tempo with metronome for the entire piece.

Test Score “C”, Bring Me to Life, by Evanescence, Arranged by Ludy, also kept tempo with the metronome for the entire piece.

The application maintained a smooth, consistent scroll during each test, with no perceptible motion blur.

This result demonstrates MusicScoller’s utility for allowing sight readers to play from digital sheet music, within the limited scope that I have established.

**6. FUTURE WORK**

Numerous improvements could be made to make this program a more universal tool. A jump mechanism to skip past large stretches of empty page would solve the problem encountered in test “A” of this experiment. A line test capable of detecting and accommodating page slant, coupled with a higher tolerance for missing pixels, would make reading even low-quality scans a possibility. An ambitious endeavor would be to adjust the scroll speed based on microphone input to accommodate dynamics or mistakes, similar to what Hamanaka, Sakamoto, and Igarashi [6] have tried for the shamisen, but generalized to various instruments and traditions.

More generally, making this functionality available on a wider range of devices would allow musicians of every caliber to play and learn more. With an inexhaustible supply of public domain compositions and new arrangements being written and shared every day, there should be simple, open-source options for rendering that information in a usable way.

**7.REFERENCES**

1. Stephanie Rosemann, Eckart Altenmüller, and Manfred Fahle. 2016. The art of sight-reading: Influence of practice, playing tempo, complexity and cognitive skills on the eye–hand span in pianists. Psychology of Music 44, 4 (2016), 658–673. DOI=http://dx.doi.org/10.1177/0305735615585398
2. Andreas C. Lechmann and Reinhard Kopiez. 2009. Sight-reading. In *The Oxford Handbook of Music Psychology*, Oxford University Press, Oxford, United Kingdom, 344-351.
3. Andrew Wallace, Joshua Savage, and Andy Cockburn. 2004. Rapid visual flow: how fast is too fast?. In Proceedings of the fifth conference on Australasian user interface - Volume 28 (AUIC '04), A. Cockburn (Ed.), Vol. 28. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 117-122.
4. Philip Quinn, Andy Cockburn, Géry Casiez, Nicolas Roussel, and Carl Gutwin. 2012. Exposing and understanding scrolling transfer functions. In Proceedings of the 25th annual ACM symposium on User interface software and technology (UIST '12). ACM, New York, NY, USA, 341-350. DOI: <https://doi.org/10.1145/2380116.2380161>
5. Paul Taele, Laura Barreto, and Tracy Hammond. 2015. Maestoso: An Intelligent Educational Sketching Tool for Learning Music Theory. *In Proceedings of the Twenty-Seventh Conference on Innovative Applications of Artificial Intelligence*. Association for the Advancement of Artificial In Conference on Innovative the Network ([www.aaai.org](http://www.aaai.org)). Retrieved June 17, 2017 from <http://faculty.cs.tamu.edu/hammond/publications/pdf/2015TaeleIAAIMaestoso.pdf>
6. Takahito Hamanaka, Daisuke Sakamoto, and Takeo Igarashi. 2014. Aibiki: supporting shamisen practice with adaptive automatic score scroll. In Proceedings of the 11th Conference on Advances in Computer Entertainment Technology (ACE '14). ACM, New York, NY, USA, , Article 13 , 10 pages. DOI=http://dx.doi.org/10.1145/2663806.2663839