Sun Yat-sen University

Bachelor Thesis

Guitarist: a sheet music viewing and playing application for classical guitar learners

Author:
Zhanrui Liang

Supervisor:
Prof. Chenyin Gao

A thesis submitted in fulfilment of the requirements
for the degree of Bachelor of Engineering

in

School of Software

March 2014

Declaration of Authorship

I, Zhanrui Liang, declare that this thesis titled, 'Guitarist: a sheet music viewing and playing application for classical guitar learners' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

SUN YAT-SEN UNIVERSITY

Abstract

Faculty Name
School of Software

Bachelor of Engineering

Guitarist: a sheet music viewing and playing application for classical guitar learners

by Zhanrui Liang

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgements and the people to thank go here, don't forget to include your project advisor...

Contents

List of Figures

List of Tables

Abbreviations

LAH **L**ist **A**bbreviations **H**ere

Physical Constants

Speed of Light $c = 2.997\ 924\ 58 \times 10^8\ \mathrm{ms}^{-\mathrm{S}}$ (exact)

Symbols

a	distance	m
P	power	W (Js^{-1})
ω	angular frequency	rads^{-1}

For people who enjoy music and guitar.

Chapter 1

Introduction

1.1 Background

Guitar is a popular musical instrument that can be used perform a variety of musical genres. Typically, a guitar has 6 strings and 17 or more frets, dividing about 100 positions on it's fretboard. When each position is pressed by the player's left hand and the corresponding string is plucked by the right hand, a sound is produced. Without special techniques, each position played by this manner will produce a sound with specific pitch. Different from the piano, which has a one-to-one correspondence between each pitch and each key on the keyboard, a sound with specific pitch can be produced on different positions on the guitar fretboard. From another point of view, a guitar can produce about only 50 level of pitches while it has about 100 positions, there must be some positions producing same pitch, though there may be difference in the sound effect. Therefore, we need to choose some proper positions for the notes that we want to play.

Typical musical scores is written on the five line staff, from which we can know each note's pitch easily. However, few scores has fingering information. It's somewhat difficult for beginners to determine the proper position for each note on the score.

Tablature is a form of musical score that solve this problem to some extent. It's widely used for string instruments. It adopt a different format to denote music, by telling which position(both string and fret) to use for each note, instead of telling the note's pitch. However, tablature is not always available, since many guitar pieces have only the score version, especially pieces for classical guitar. As [1] mentioned, one reason for this is that once a guitar player become experienced enough, he/she don't need it that much.

1.2 Previous Works

1.3 Objectives

1.4 Contribution

1.5 Structure of this Thesis

This thesis is divided into 3 main chapters.

Chapter 2

Processing Sheet Music

2.1 Introduction

MusicXML[**?**] is a music notation interchange format. Nowadays, most commercial music softwares support this format so we can use it to interchange sheet music between different softwares. Before MusicXML, the universal music notation interchange format is MIDI. However, a midi file don't contains enough information for rendering sheet music. For example, we won't know from midi the direction of note stems. Another example is that a note with specific pitch can be represented differently, with the alternating symbols such as sharp, flat and natural, etc.

As the website of MusicXML states, there are more than 170 applications include MusicXML support. Also, more and more scores in MusicXML format are available in digital music databases. All input data comes from the MuseScore website.

2.2 Parsing MusicXML

MusicXML, as its name implies, is a XML based format, so we has plenty of libraries in different programing languages available for manipulation. Though we can use XML libraries to load the XML file, there are still some works to put on the semantic analyzing of the musical content.

2.2.1 Semantic Parsing

Semantic parsing is a process that converts data from the storage format to internal format.By internal format, we means a format that is easy to use in the application.

Since this kind of internal format varies from different application, there are few libraries available for this job. Having such a general library for semantic parsing is actually a reinvention of a general format. However, MusicXML is already a general format and is intuitive enough so having a new one is not necessary. We explored some open source computer music projects which have MusicXML support and found such internal format. The Music21 library [**?**] can convert between MusicXML and its own format, but it expose its MusicXML class as public API. The Lilypond [**?**] project has a script, namely "musicxml2ly.py", for converting from MusicXML to Lilypond's ".ly" format. The relevant classes used by this are also not part of the public API and are nearly undocumented, so it's not easy to reuse in our application. Combining these facts, we decide to write our own semantic parser for converting data from MusicXML to a format that can be used other parts of our application, such as rendering, sound playing and fingering arrangement.

MusicXML is strictly defined by a XML schema, which is also well documented. We can study the file format directly from the schema. In MusicXML, a musical score is stored as a tree structure[1]. It has the following main hierarchy, listed from the root:

- score
- part
- measure
- note/rest/direction

Before we introduce these element types, one thing worth to mention is that MusicXML use tenths as unit. A tenth is 1 / 10 of the distance between two staff lines.

In the following, we will introduce these element types briefly. More details will be explained when a element is used in the following sections in this chapter.

2.2.2 Score

The score element is the root element. It provides the information about page layout such as page size and page margins. The correspondence of tenths and real world length unit is also specified here.

[1] MusicXML has two types: time-wise or part-wise. We discuss only part-wise here

2.2.3 Part

A part element contains and only contains many measure elements. There may be several parts in a score, however, our implementation only supports one part. This is because most guitar pieces can be expressed using one part and they are actually expressed using only one part. Using one part does not mean that we can't have polyphony music, since MusicXML chords in a single part. This will be explained in 2.2.4.

2.2.4 Measure

The measure element is the most complex part of MusicXML. The most notably elements that it contains are notes, rests and direction. It also contains information about time signature, key signature and tempo.

The timing mechanism inside a measure works by maintaining a time counter with time changing instructions. When a measure is entered, the time counter is set to zero. Then when each note(or rest) is added, the time counter increase the amount of the note's duration. From this aspect, a note's occurrence can be viewed as a time changing instruction. To support polyphony music, we have the "backup" and "forward" instructions. When a backup Instruction is processed, the time counter decrease an amount stated by the instruction. A forward instruction is processed in a similar way, but make the time counter goes forward instead.

2.2.5 Note

A note element tells us the pitch, duration and display style of the note. Unlike the MIDI format, which represent a pitch by an integer proportional to the logarithm of the physical pitch value, pitch in MusicXML is represented by a (step, octave) pair. A step can be one of (C, D, E, F, G, A, B) and octave is an integer.

A note element also tells about accidentals, dots, stems, beams, etc. Further details will be explained in 2.5.

2.2.6 Other Symbols

There are some other types of symbols on a musical score, such as time signature, key signature, text, etc. These symbols is usually straightforward to handle since they do not depend on each other.

2.3 Internal Data Representation

We adopt a Object Oriented Style in our internal data representation design. As mentioned above, MusicXML has the score-part-measure-note hierarchy. Instead of following the MusicXML hierarchy strictly, we decide to have a sheet-page-measure-note hierarchy. Miscellaneous symbols is attached to measures and notes.

One challenge of dealing with MusicXML is that some information is optional, so we need to infer it's default value from the context. For example, the distance of a measure to the measure below it may be omitted, then we need to infer this distance value from the previous measures.. Sometimes we even have to hard code default values in side our application. For example, tempo value is optional in MusicXML. However, without this value we cannot convert the music time to actual physic time.

2.4 The Parsing Procedure

In the parsing procedure, a ParseContext is maintained. A ParseContext consist of these values:

- sheet: Current sheet.
- page: Current page.
- measure: Current measure.
- beams: A lookup table for beams parsing.
- endings: A lookup table for endings parsing, similar to beams.

To parse a MusicXML file, we follow the steps below:

1. Load the MusicXML schema file.
2. Load the MusicXML file.
3. If the file is compressed, decompress it.
4. Validate the file using the loaded schema.
5. Initialize the parse context.
6. Iterate through each measure element in the only part element.

FIGURE 2.1: A sample score for illustration purpose. Elements are highlighted and labeled with names. This sample is rendered by our rendering program and then processed with Inkscape.

7. For each measure, if it indicates that a new page should be created, create one.

8. For each child of the measure node, get the type of this child and use the corresponding handler to handle.

Most of these steps are straightforward: We just read data from the elements and store them to the objects.

2.5 Rendering Sheet Music

2.5.1 Overview

Sometimes a picture is worth a thousand words. Figure 2.1 is a overview of the elements on the score.

The fundamental part of a score are the staff lines. Each group of staff lines are 5 parallel long horizontal lines, forming a grid system for other musical elements. At the beginning of a staff, there's usually a clef followed by the key signature. For guitar pieces, usually the G clef is used. A staff is split into measures by the vertical bar lines. A measure helps the reader to group notes and to calculate the time. Each measure has its index number, counting from 0 or 1. It counts from 0 when the measure is partial. Although every measure has an index number, not all of them is printed and usually only the leftmost measure's index number is printed. With the index number, people can describe and locate measures efficiently.

The vertical bar lines splitting the measures may have different style. They can be thin or thick, single or double. Thin bar line is the most used one. They are used to split measures. Thick bar lines are used at the end of a score. Double bar lines, which companied double vertical dots, denote start or end of a repeat.

Musical notes, along with the stem and beams connecting them, is the most complex part. Details about them will be covered in the following discussion.

2.5.2 Primitive Types

Before we continue on discussing how to render different musical elements, it's worthwhile to classify different primitives.

In musical scores, we only cares about the shape of elements and don't care about the color.So in our model all primitives are rendered using the same color, black.

In our model, all musical elements can be decompose into these 4 primitives:

- Lines: Both vertical and horizontal.
- Beams: Beams is a special kind of line, with their two ends slanted.
- Texture: For example, clefs, quarter note heads, full note heads, tails, rests, accidentals.
- Text.

A line is actually drawn as a long thin rectangle. Formally, a "line" here should be called a segment, since it's not infinite. However, we call it a "line" for simplicity's sake. We use 5 variable to describe a line, they are:

$$(x_1, y_1, x_2, y_2, \text{thick})$$

This is illustrated by Figure **??**. One remarkable thing is that we found it important to keep the line ends sharp instead of bevel. This is because bevel ends are hard to align with other elements, causing artifacts in the final render result.

Beams look like lines, except that their ends are slanted. Also, they are attached to the stems most of the time. If we keep using $(x_1, y_1, x_2, y_2, \text{thick})$ to represent a beam, it results that we keep subtracting and adding thick/2 when calculating its position. To solve this inconvenience, we use the following representation instead:

$$(x_1, y_1, x_2, y_2, \text{height})$$

The meanings of these variable is illustrated by Figure **??**.

To render other elements with irregular shape, we use the template method. First we decompose a complex element symbol into different parts, based on the following rule:

- If the symbol has long vertical or horizontal lines, remove it.
- For the remaining parts, create a template for each of them.

FIGURE 2.2: Packed textures. Each template comes from the Lilypond project. They are rendered into SVG format and then converted to 500dpi PNG image.

FIGURE 2.3: The G clef template.

With this method, we get the templates as shown in Figure 2.2. Since we use OpenGL as the rendering backend, keeping all templates in a single texture image is quite a common way to go. However, packing different templates manually is not easy to maintain, so we make a script using a greedy algorithm to do this. With the help of the script, we can just make each template a single file with its own name, so that we can easily add or remove templates later.

Each template is represent by:

$$(x, y, w, h)$$

Each template has its pivot point (x, y). This is the coordinate on the final packed image. w stands for "width", and h for "height". Figure 2.3 is an example on the G clef about these variables.

Texts are rendering in a similar way like textures.

2.5.3 Note

Notes are the essential part of the score. The pitch of each note is specified by its vertical position and the duration is specified by both its head shape and its tail.

There exists 3 type of note heads. They're for full notes, half notes, and other notes, respectively. Full notes don't have stems connected, and the head is a bit larger than other notes. Half notes looks like quarter notes, except that their heads are hollow. Notes with duration less than quarter has the same note heads, since their duration is then not distinguished by their heads, but their tails instead. As a result, we need to prepare only 3 templates for note head rendering.

A note is usually connect by a stem, which heads up or down and has vary length. The direction of a stem is told by MusicXML. Different directions of a stem is used to distinguish between different voice parts. While the head position of a stem is determined by the note head and the stem direction, the tail of it is determined by the beam attached to it, if it has any. A stem without can be any visually proper length. We use 35 tenths as this default value in our application.

A single stem may be connected to multiple notes. This is often seen in chords. Examples can be found in Figure 2.1.

2.5.4 Beams

Beams are used to tell the duration of notes and to group notes. A note [2] without any beams but only a stem have the duration of 1/4. When one more beam is attached, the duration is divided by 2. It's not hard to get the conclusion that a note with n beams have duration of $1/2^{n+2}$.

Usually in musical score, beams has different slopes, which is determined by vertical positions of notes that they connect. However, the rules to determine the slope is seldom mentioned. Some software just let the slope be 0. This is, they render all beams horizontally, which make the score look weird and don't comply with the musical convention.

Here comes three observations that build up the fundamental rules of our method:

1. Beams that connected to common stems and have common horizontal span should have
2. A proper slope for a beam should make the length of stems connecting to the beam as uniform as possible.
3. There should be only several slope value used in rendering, since using too many slope value will mess up the score which is not ideal.

A beam can be partial or complete. Partial beams can then be divided into forward and backward beams. Partial beams cannot occur by it self, since a beam occur by it self is actually a tail, which is rendered differently. With this observation, we can assume that partial beams can always be drawn after complete beams.

A beam may be connected to up stems or down stems, but rarely both. We can assume that all stems that a beam connected to have the same direction. Then we can two types of beam, one with up stems and the other with down stems. Since the calculation between these two types are similar, from this point on we only discuss about the type that has up stems.

In our beam slope calculating algorithm, we have two important concepts:

[2] Here we are discussing notes with quarter note head, since full notes and half notes don't have beams.

- Length: the length of a beam is defined as the number of different stems that connected to it.
- Anchor: An anchor is a stem that its tail position is determined.

Note that if a beam's slope is calculated, the slope of beams below or above it should be the same value. So if we sort the beams by their length, from long to short, and then assign their slope, then we only need to calculate the topmost one in each group of beams. With the aid of anchors, sometimes we can determine the slope of a beam directly. Details is covered in the algorithm listing. **??**.

1. Sort the beams by their length, from long to short.
2. For each complete beam, find all anchors connected to it.
3. If the number of anchors is equal or more than 2, then set the beam's left end to the tail position of the leftmost anchor and set the right end to the tail position of the rightmost anchor. Make all stems connected to the beam an new anchor, that is, assign tail position for them. Finish this iteration.
4. Otherwise, sort all stems connected to the beam from left to right.

MusicXML do not tell about the slope.

2.5.5 Accidentals and dots

2.5.6 Key Signature

2.5.7 Rendering Order

2.5.8 Different layout Modes

2.6 Playing Sound

2.7 Results

Chapter 3

Fingering Arrangement

3.1 Introduction

3.2 Model

3.3 Cost Function

3.4 Decision Process

3.5 Optimizations

3.6 Tracing Solution

3.7 Generating Tablature

3.8 Results

Appendix A

Setup Instruction

Bibliography

[1] Masanobu Miura, Isao Hirota, and Masazo Yanagida. Constructing a system for finger-position determination and tablature generation for playing melodies on guitars. *Systems and Computers in Japan*, 35(6), 2004.