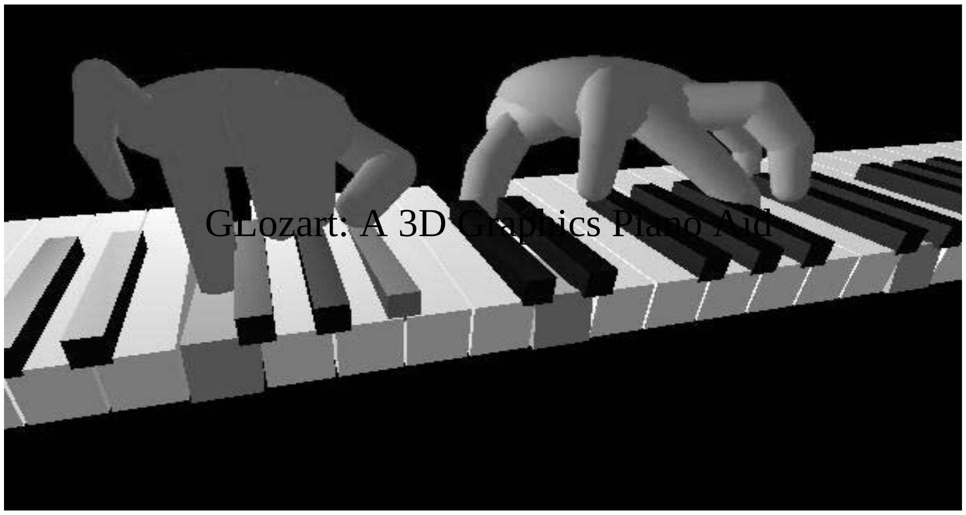


GLozart: A 3D Graphics Piano Aid

James Delos Reyes
CSC 491
Project Advisor: Zoë Wood
Spring Quarter 2007

Introduction

GLozart is a three-dimensional graphics program that graphically simulates hands playing a piano. It can be thought of as a virtual player piano with the addition of hand simulation. The program works as an educational tool for amateur pianists to learn how to play piano by hearing and seeing a demonstration of how a piece is played. The application is written in C++ and combines the utilities of OpenGL, GLUT, Qt, MusicXML, and MIDI. Some of the challenges addressed during the implementation process include how to show a real-time animation sequence with graphic and sound synchronization, parsing a given music file into animation frames, how to dynamically select finger positioning, and supply an easy-to-use user interface.

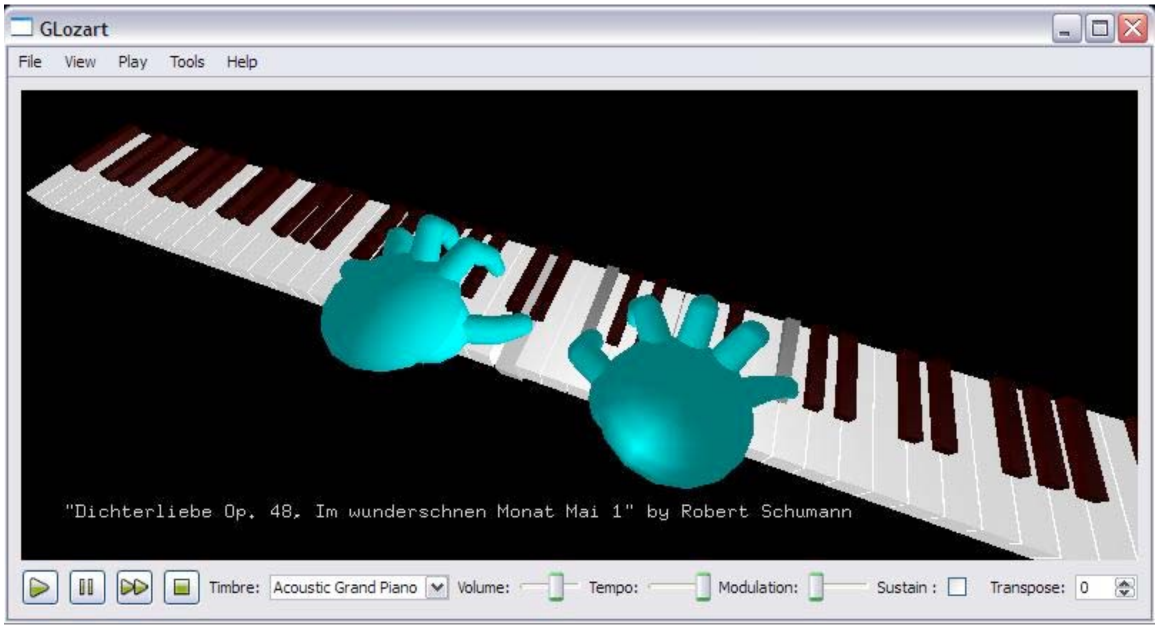

GLozart
File
View
Play
Tools
Help
"Dichterliebe Op. 48, Im wunderschnen Monat Mai 1" by Robert Schumann
Timbre: Acoustic Grand Piano
Volume:
Tempo:
Modulation:
Sustain :
Transpose: 0

Figure 1.1: GLozart Screenshot

Application Features

GLozart features a full 88-key keyboard which can be rotated for the user to see the simulation from different angles (Figure 1.1). It has typical keyboard functionality including volume controls, tempo controls, pitch modulation effects, pitch sustain, transposition control, timbre control, and play, pause, fast forward, and stop functions for playback. It also features a dynamic side-by-side notation allowing users to see the standard music notation associated with the current chord positioning. In addition, a chord constructor is in place so that users can construct their own chords and inspect how they are positioned.

The 3D Component Using OpenGL

The 3D graphics component of this program was written with OpenGL and GLUT. OpenGL is a cross-platform open graphics API developed by Silicon Graphics Inc. in 1992. GLUT is a utility toolkit built on top of OpenGL to supply useful function calls including drawing graphics primitives and window management. Both OpenGL and GLUT are widely used in many 3D applications and became the immediate graphics library candidate for the program thus providing the name, GLozart (pronounced glōt'särt), a combination of Open**GL** and the Classical pianist M**ozart**.

The keyboard and hands are drawn using simple GLUT polygon primitives (Figure 1.2). Each key of the keyboard is a GLUT cube scaled into a rectangular key. Display lists are used to improve rendering performance when drawing each key. Display lists store and precompile the commands used to draw the key. Each key is then drawn simply by calling the display list and applying the correct translations and rotations. This improves performance since there are many keys to draw (88 in all) and less code. The code is stored in the graphics card if the graphics card allows avoiding the performance bottleneck of pushing vertices from the CPU to the GPU. As keys are drawn, they are rotated vertically according to which notes are currently stored in the animation frame to simulate being pressed and unpressed.

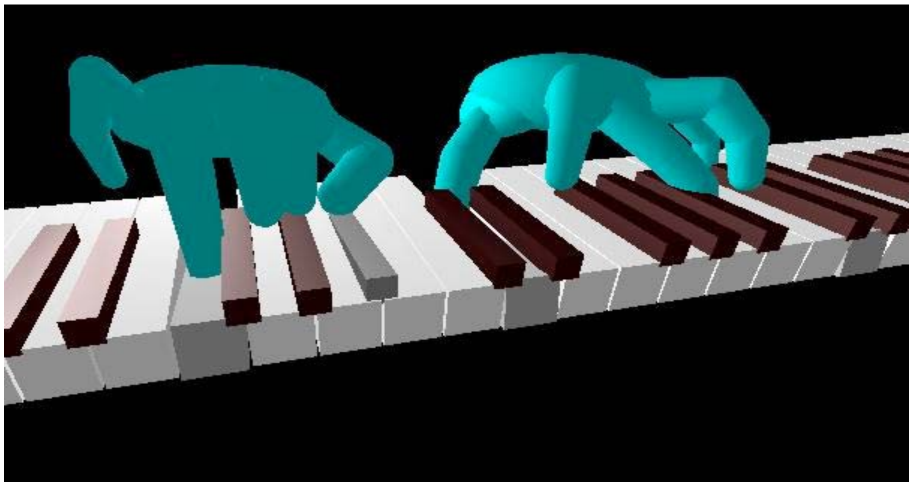


Figure 1.2: Hands and Keyboards drawn using GLUT primitives

To draw the hands, each section of the fingers is a GLUT cylinder. The palm and joints are made using GLUT spheres. The finger animation uses a hierarchical modeling design in order to bend and behave similar to human movement. Each section of the finger is attached at pivot joints such that when one section of the finger is translated or rotated that the other sections of the finger are translated and rotated appropriately and remain attached. Hierarchical modeling along with complex trigonometry is ideal for the hand animation since the fingers need to be told which keys to play and where those keys are. The fingers are implemented such that the developer can specify any coordinate in world coordinates, and the tip of the finger will try to "touch" the specified point. This, of course, is based on wherever the hand is located. Therefore, if a finger is told to touch a point it cannot reach, it will disappear graphically due to non-defined trigonometric values. This is a limitation to GLozart since if two notes are spread too far apart, there are fractures in the animation sequence. To address this issue, the fingers are given default value points based on where the fingers currently are in the animation frame. If the note is too far away for a finger, the finger does not attempt to play it. This is a modest way of hiding a hard problem. At the same time, there are many pieces composed by people with larger hands than others, and not every piece is playable to those with small and even average hand spans. For now, GLozart can be thought of as someone with an average hand span.

GLozart also features a wired meshed view which uses wired meshed polygons to draw the hands (Figure 1.3). This allows users to see which keys are being played through the hands in case the user does not prefer to rotate around the hands to see what the fingers are pressing. The user can also choose for the hands not to be drawn leaving only the piano to be animated.

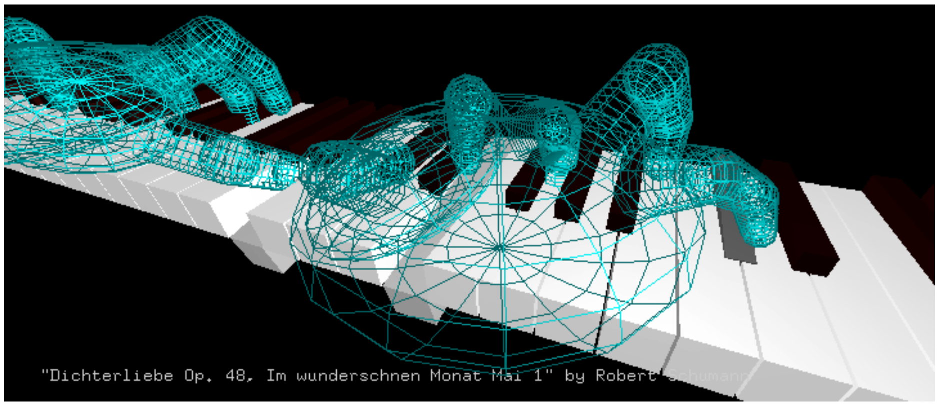

"Dichterliebe Op. 48, Im wunderschnen Monat Mai 1" by Robert Schumann

Figure 1.3: Wired mesh view

Lighting in GLozart is done in OpenGL via the Phong model—a combination of ambient, diffuse, and specular light. A spotlight is used to bring out the hand's specular components to give more definition to the hand's depth and shape.

To improve runtime performance, backface culling was implemented. Backface culling removes vertices not drawn in the viewing frustrum. Because of this, fewer vertices get pushed down the graphics pipeline greatly improving performance.

GLozart's main viewing widget also displays the name of the composition piece and composer using GLUT's bitmap text capabilities. The text is written using OpenGL's raster positioning calls so that it does not rotate, scale, or translate along with the rest of the viewer.

Importing Music Using MusicXML

Users can import music into GLozart using MusicXML. The MusicXML format file was developed by Recordare LLC (http://www.recordare.com/) as a bridge to translate Western musical notation between various program applications. The first version was released in 2004 and although it is still a new file format exchange, it is used by over 75 different commercial products today. This includes leading programs for score-writing such as Finale and Sibelius.

One might expect GLozart to read in MIDI files to follow suit of most music applications. MIDI (Music Instrumental Digital Interface) and MidiXML are industry standard music exchange formats. MIDI is widely popular among music applications and digital music instruments and has remained nearly unchanged since its release in the early 80's. The decision to read MusicXML, a comparably younger format exchange, versus MIDI is based on the fact that MusicXML and MIDI are not necessarily competing format exchanges. This is because MIDI specializes specifically with music while MusicXML specializes specifically with music *notation*. MIDI ignores standard notation since it is verbose and unnecessary to play a piece. Instead, MIDI is merely made of music events. This is where MIDI does not fulfill GLozart's requirement to parse and devise an animation sequence for piano. For example, MIDI cannot tell whether an instrument is a piano or not. Although during playback MIDI can generate piano sounds, it only has knowledge on the timbre of the instrument and not the instrument itself. MIDI may specify a note to play outside the typical 88-key spectrum pianos are limited to. This is because MIDI does not consider physical limitations of instruments. For example, GLozart would have a hard time parsing a MIDI file with more than ten notes at a time since most piano players only have ten fingers. The core reason why MIDI is not read, assuming the file was playable by a pianist, is that a very complex algorithm would be needed to decide which hand should play which notes since there are no strong restrictions without notation. This is not a practical problem to solve since users would want to play pieces where notation and hence hand position is already provided. Also, it is not necessarily solvable since most composers divide hand position based on melody in one hand and rhythm in the other, and there are many pieces where this overlaps. Furthermore, rhythm and melody can be intertwined where distinction between the two is often debated philosophically not algorithmically. Thankfully, MusicXML does tell GLozart which hand plays which notes by denoting staffs in the music notation.

MusicXML also has knowledge of whether a piece is specifically for piano. If the MusicXML specifies more than one piano, GLozart simply parses the first piano in the file (although allowing the user the ability to choose does sound like a possible future direction). This is true even if there are other instruments specified in the piece (e.g. voice or guitar). For now, GLozart plays MusicXML pieces written with at least one "Piano," "Grand Piano," or "Acoustic Grand Piano" as specified by the Part ID of the MusicXML, although other types of instruments are not out of the question for future releases. MIDI is reserved for the audio component of GLozart which will be discussed later.

Parsing a MusicXML File

The XML is parsed using Microsoft's XML API. MusicXML is formatted such that each musical measure is divided into divisions. The divisions are as small as the shortest note duration throughout the piece. A piece where the shortest duration is a quarter note will have less divisions per measure versus a piece where the shortest duration is a sixteenth note (i.e. four versus sixteen divisions respectively). In GLozart, each division is parsed into its own animation frame. This is a performance bottleneck since the animation sequence can only be as fast as the number of divisions per measure. For example, if there is only one sixteenth in a very large piece while every other note is a quarter note, GLozart will spend most of its frames doing nothing but waiting for every sixteenth frame where the next quarter note begins. Despite this limitation, MusicXML divisions work well as animation frames where each frame in GLozart is simply stored as a collection of pitches and on/off flags. Frames are also precomputed before the actual animation sequence so that parsing does not affect the runtime performance of the animation sequence.

Due to time constraints, GLozart is limited as to what information is parsed from the MusicXML. Certain elements when parsing are completely ignored due to the fact that GLozart would have to perform extra computation just to translate them into frames. Piano articulations (e.g. trills, glissandos, grace notes), dynamics (e.g. crescendos, fortissimos), repeats, and codas are completely ignored by GLozart.

Some files cannot be parsed at all by GLozart. The first obvious MusicXML that cannot be parsed is pieces that have no piano instrument. For now, hand position is specified by whether a note is played in the bass clef or treble clef. For this reason, pieces that have multiple clefs (e.g. two treble clef's, or a bass clef that changes in the middle of a piece to treble) cannot be parsed including voices that cross over staffs. To prevent fingers from playing notes it cannot reach, any chord for one hand that spans over an octave cannot be parsed.

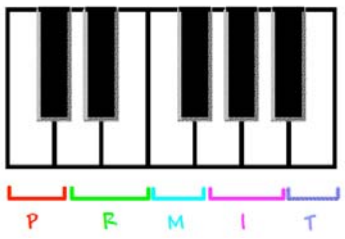
Despite its limitations, MusicXML is a still a great format exchange to read in GLozart. There are many MusicXML pieces and resources online. This means there is a lot of music available for sale and for free. Users can even create their own MusicXML using commercial programs like Finale and then import them into GLozart. This allows a great deal of flexibility over what pieces users can import into the application.

Finger Positioning Importance and Implementation

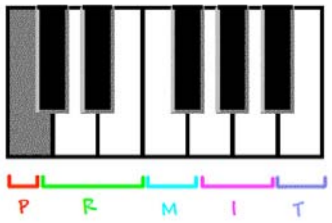
GLozart provides users an angle at reading music that standard notation often does not provide—finger positioning. Most of the time finger positioning is not specified in standard notation for many reasons. It can make notation too busy and hard to read if there are many notes. Also, advanced players are expected to show enough discipline to choose their own finger positioning. There is hardly ever one way to play any piece, since finger positioning is mostly based on what is comfortable to the player. However, bad fingering at an early level in the learning stage can lead to bad fingering habits for the future. GLozart merely provides users one acceptable solution for finger positioning though there can be many. The difference between this and deciding hand position is that there are many acceptable fingering positions with the exception of a few business rules. GLozart's algorithm for choosing fingers is based on the closest note to a finger. One limitation is that it does not accurately define how humans prioritize certain fingers over others based on finger strength. For example, the pinky is the least used for any instrument simple because it is the weakest finger and hardest to control. Humans usually pick a more comfortable position based on finger strength rather than pure range approximations. GLozart might pick finger positions that would be seen as uncomfortable to most people. This can, however, encourage users to exercise finger strength and distribute finger positioning among all fingers. Another limitation is that there are certain business rules that classically trained pianists follow. For example, pianists are not "supposed" to use their thumbs on black keys when ascending and descending scales. This behavior is not featured in this version of GLozart but does not necessarily mean it cannot show up in future releases.

A greedy algorithm approach is used to determine finger positions. The algorithm is inspired in part by how advanced pianists sight-read music notation. To sight-read, a player plays a piece that they for the most part have never played before. As they play the piece, they are constantly looking ahead to see what the notes are. This in turn determines their finger positioning based on where their hands are currently. GLozart emulates this behavior by reading ahead a designated amount of frames before choosing the finger positions. This makes sense since notes will be distributed to all fingers in a succession of notes. For example, a *scale* is an ordered succession of single notes. (The first melody, "Joy to the world/The Lord has come" in the Christmas Carol, "Joy to the World," is a scale.) If GLozart did not read ahead when determining finger position, it might pick the thumb to play every single note in the scale since no two note overlaps. Ideally, this does not make sense since the other fingers are closer to the other notes than the thumb. By looking ahead like sight-reading, GLozart is able to distribute the keys to all the fingers.

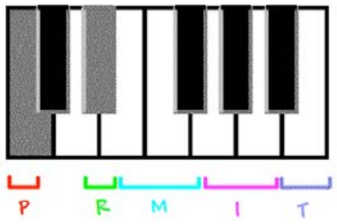
The algorithm starts by reading ahead a fixed amount of frames. The minimum pitch for those frames is then stored. For each frame, the minimum and maximum pitch is calculated to obtain the span of the notes updating the minimum pitch if necessary. The fingers are then given a minimum and maximum pitch to cover. If a pitch in the frame appears within a certain finger's span, the finger is assigned the pitch and the span for the other fingers are recomputed. Figure 1.4 shows a full iteration through a frame assigning notes to each finger in ascending order.



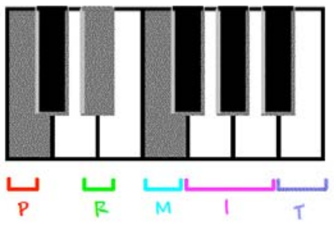
Step 1: Spans assigned for each finger



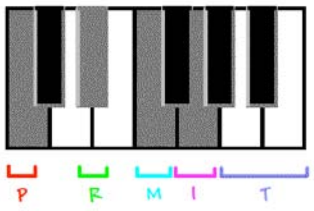
Step 2: After iterating for the first note, the pinky is assigned the pitch and the ring span is adjusted.



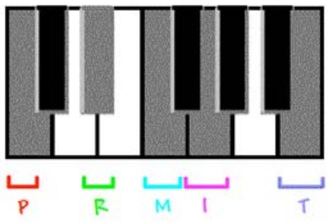
Step 3: After iterating for the second note, the ring is assigned the pitch and the middle span is adjusted.



Step 4: After iterating for the third note, the middle is assigned the pitch and the index span is adjusted.



Step 5: After iterating for the fourth note, the index is assigned the pitch and the thumb span is adjusted.



Step 6: After iterating for the fifth note, the thumb is assigned the pitch.

Figure 1.4: Applying the finger algorithm to a frame for the left hand.

Here is a more detailed explanation of the greedy algorithm used to obtain finger positions for the right hand in pseudo-code:

```
// Iterate through each scan in the animation sequence
for each sightReadScan in AnimationSequence
        // Store the minimum pitch for the division scan
        int minPitch = getMinPitchForScan(sightReadScan);

        //Store current low
        currentLow = minPitch;

     // Iterate through each frame in the scan
     for each frame in sightReadScan
         // Get the frame’s minimum pitch
         int minFramePitch = getMinPitchForFrame(frame);

         // Update the current low if the mainframe
```

```
                // is lower
                currentLow = min(minFramePitch, currentLow);

                // Get the frame’s maximum pitch
                int maxFramePitch = getMaxPitchForFrame(frame);

                // Center the hand between the currentLow and high
                centerHand(currentLow, maxFramePitch);

                // Dynamically assign spans for each finger based
                // on the currentLow and high
                fingerSpan thumbSpan = getThumbSpan(currentLow, maxFramePitch);
                fingerSpan indexSpan = getIndexSpan(currentLow, maxFramePitch);
                fingerSpan middlSpan = getMiddlSpan(currentLow, maxFramePitch);
                fingerSpan ringSpan = getRingSpan(currentLow, maxFramePitch);
                fingerSpan pinkySpan = getPinkySpan(currentLow, maxFramePitch);

                // Iterate through each pitch in the frame in
                // ascending order and assign them greedily to
                // whichever fingerspan it fits in.
                for each pitch in frame order by asc
                    // Check if the thumb is used and see if
                    // the pitch fits in the thumbSpan
                    if(thumb.getPitch() == -1 &&
                          thumbSpan.containsPitch( pitch)){
                       // Set the thumbs pitch
                       thumb.setPitch(pitch);

                       // Adjust the index now that the thumb is taken
                       indexSpan.adjustSpan();
                    }
                    // Check if the index is used and see if
                    // the pitch fits in the index Span
                    else if(index.getPitch() == -1 &&
                          indexSpan.containsPitch( pitch)){
                       // Set the index pitch
                       index.setPitch(pitch);

                       // Adjust the index now that the index is taken
                       middleSpan.adjustSpan();
                    }

                    ...

                    // if there are still notes left over, shift the pitches
                    // down a finger and let the last finger get the leftover pitch
                    else{
                       // Shift the pitches down a finger
                       swapPitches(thumb, index);
                       swapPitches(index, middle);
                       swapPitches(middle, ring);
                       swapPitches(ring, pinky);

                       // Let the last finger get the left over note
                       pinky.setPitch(pitch);
                    }
                } // end frame loop
            } // end sightReadScan loop
       } // end animation loop
```

It is important to note that finger positioning is precomputed before the actual animation sequence. This prevents the position processing from interrupting the synchronization and timing of the sequence.

Dynamically Computing the Animation Sequence

Not all the properties of a rendered frame are precomputed. The actual animation sequence takes the current division frame and next division frame and dynamically generates animation subframes in between them to give smoother finger and key movements. The benefit of this is that less memory is needed to store the sequence. Its major drawback is that it forfeits the ability to rewind during playback since the implementation only traces forwards. Also, dynamic computation is a bit of a performance dragger. GLozart might have benefited from full precomputation but the implementation was avoided since the architecture was already deeply embedded and would require a time-consuming, full-scale refactoring of code.

The Qt GUI Component

The GUI component of GLozart was implemented with Qt, a cross-platform API developed by TrollTech. Qt is widely used in numerous applications including Google Earth and Skype. The biggest advantage of using Qt in GLozart is that it integrates well with OpenGL using QGLWidgets. The other feature of Qt is how it handles GUI events. This is done through signals and slots. Each GUI component with an action (e.g. buttons, sliders, menu items) can be registered as signals. Developers then associate those signals to slots where the code is implemented to react to the signal. In GLozart, each menu item and tool bar action is registered as a signal. The associated slots are registered in the QGLWidget where the piano and hands then react to each registered signal. Besides the easy interface, Qt also has great documentation and numerous forums where developers can discuss challenges, address bugs, and advise tips. The major drawback using Qt is that Microsoft Visual Studio developers have to engage in a large amount of configuration details in order to get Qt to compile. The steps are recorded in a tutorial at http://qtnode.net/wiki/Qt4_with_Visual_Studio. Even after Qt is integrated, there is a significantly larger amount of compile time. This makes the application harder to debug if the developer wants to go back and forth between running the application and coding. There is a little speed improvement if the developer addresses all warnings immediately. Qt is also limited since the OpenGL timer callback QTTimer is a large performance bottleneck. This is the standard way of calling the updateGL calls needed to repaint the OpenGL viewing component. However, the smallest time increment is one millisecond when GLozart could benefit from a smaller time unit. Other GUI API's were used but proved much more inferior to Qt's capabilities. MFC can be very complex and has a steep learning curve that GLozart's development cycle could not afford. GLUI, a GUI API made just for OpenGL applications, had many configuration issues and was not very flexible to begin with.

The MIDI Audio Component

The audio component of GLozart was implemented using MIDI (Musical Digital Instrument Interface). MIDI works by inputting and outputting MIDI event messages. The audio signals are not generated by MIDI directly. MIDI merely sends the event messages to an output device that generates the sound. The output device can vary between a musical instrument or MIDI device driver and might not even be an audio component but a visual output like a laser light show. GLozart takes advantage of the fact that almost all of today's PC's include a default MIDI device driver. GLozart sends MIDI event messages to the PC’s device driver where the audio is generated and pushed through the speakers.

A short MIDI event message is only 8 bytes long so setting parameters is implemented using bit operators. To send simultaneous messages, short messages are queued into one long event message. The messages can tell the driver to turn a note on or off and may set other parameter controls including volume, modulation, timbre, etc.

To synchronize the audio with the animation sequence, GLozart streams the event messages to the output device using the MIDI Stream API. The API works first by opening the MIDI stream device in the default driver before GLozart’s animation sequence. As the sequence runs, the application checks for keys that have been pressed down all the way and sends “Note on” event messages for those pitches. As keys are unpressed, GLozart sends “Note off” messages for those pitches. In one animation rendering pass, the messages are queued in one long message for playback and then sent to the driver. This approach gets rid of synchronization issues whereby the device driver messages are triggered by what the user sees graphically. Another approach might have been to queue MIDI event messages for the entire animation and start playback at the beginning of the animation sequence using the MIDI Sequencer. However, this does not synch with the animation well since the OpenGL component has limited timing control with rendering passes. If multiple processes are open, then the animation sequence may slow down in the middle of a song and it would be hard for the MIDI Sequencer to be on time. This would also limit playback since pause and fast forwarding would trip the sequencer up. By using MIDI Stream API, GLozart also does not have to deal with memory management of long message queues that would result from the previous approach. Developers should take caution as the default driver gets left open for a long period of time. This means that every note turned on by an event message should be followed with an event message to turn the note off. Otherwise, too many notes left open can build up and be ear splitting to the developer.

The MIDI Mapper allows GLozart to have control on where to send event messages within the device output driver. The mapper can map messages to either a channel, a patch, or a key. The “Note on” and “Note off” messages triggered by the animated keys are event messages sent to the key map. This is how GLozart is able to turn some notes on while leaving other ones on at the same time. Messages sent to a patch affect a certain instrument although this is not explored much in GLozart since it only works with one MIDI instrument number at a time. Events sent to a channel map can affect all the notes within the specified channel. Since all the audio in GLozart only uses one channel, certain events like volume, sustain, and modulation are applied to that channel affecting all the notes in play.

The diagram below (Figure 1.5) shows how GLozart uses all the above-mentioned MIDI components to communicate with the output device driver using Window’s Multimedia Software Extension. GLozart lies in the top Application Level. Since it is using the Stream API, the MIDI sequence driver is not used at all. The sequence driver would be used to playback a long sequence of MIDI events whereby every message is queued before the output device driver is opened. Instead, GLozart can stream messages to the device driver via the MIDI API function calls. It can send messages directly to the driver in order to open and close it, or it can map event messages to certain channels, patches, or keys through the MIDI mapper. Each message is finally output by the default MIDI device driver where the actual sound is generated into the computer’s speakers.

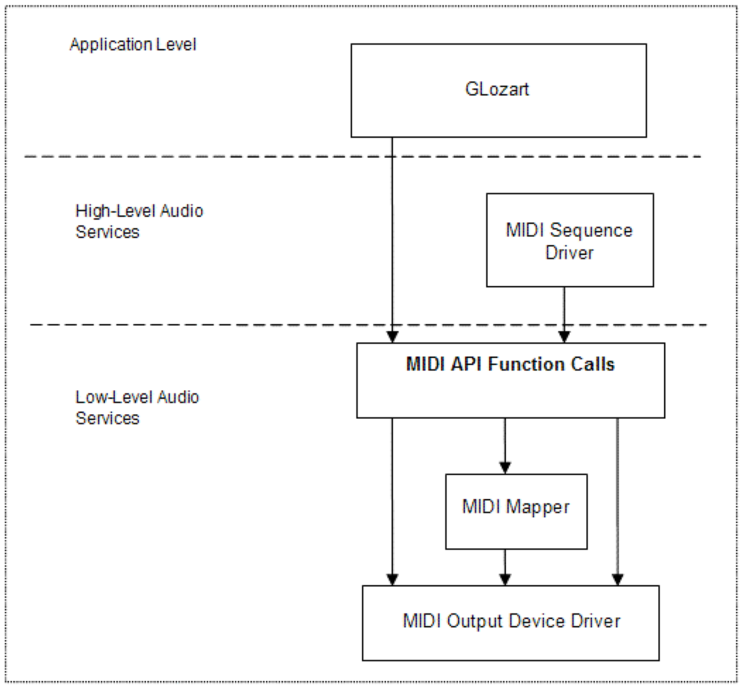


Figure 1.5: Application level using MIDI services

Timbre Controls

Timbre in GLozart can also be controlled through MIDI channel messages. This allows us to pick different sounds that the piano will generate. The user can select other piano timbres like a Rhodes piano and a Honky-Tonk piano or non-piano timbres like a guitar, synthesizer, brass, bird tweets, and even helicopters. GLozart accesses 128 default timbres that are in all MIDI device drivers. It should be noted that the quality of the timbre can be affected by the quality of the MIDI device driver. In the early 80’s, MIDI was synonymous with cheesy blip noises, but that was because all that was available at the time were poor quality MIDI drivers.

Tempo Controls

Tempo in GLozart is controlled by how fast the animation sequence is rendered. This is done by associating the OpenGL component with the QtTimer class. Every time the associated QtTimer finishes a specified time interval, the OpenGL viewing frustrum is queued to be drawn. The tempo controls in the tool bar essentially sets the specified time interval in QtTimer. Tempo can vary between sessions based on how many processes are fighting for CPU time and how fast the combination of the computer and its graphics card is in the first place. This creates a problem since it is hard to control consitency of tempo between computers. Tempo is therefore relative to how fast GLozart can render each frame and not based on any actual tempo measurement like beats per minute (bpm). GLozart defaults to the fastest tempo possible so that the user immediately hears the application's maximum tempo capacity.

Transposition Controls

The transpose tool in GLozart is not a direct MIDI event message. The transpose function simply increments or decrements a class variable that is added to the pitch when generating "Note on" and "Note off" events in the animation sequence. The problem with this is that if a pitch is transposed after the note is turned on then the next "Note off" message will be sent to a entirely different pitch. This will leave many notes orphaned where some pitches may trail until the application is closed. To fix this, the function also sends an event message to turn all the notes off to the channel. This makes sense audibly as well since the old transposition might conflict with the new one throwing off the user's sense of tonality (assuming the piece is not atonal).

Dyanamic Side-by-side Notation

GLozart also features dynamic side-by-side notation in a separate Qt widget during the animation sequence (Figure 1.6). This feature is based on an LCD version offered by many of today's keyboards targeted towards amateurs. This helps users quickly establish the correlation between the notes being played on the virtual piano and the actual music notation. The notation is displayed in a staff while notes are displayed corresponding to notes that are pressed on the virtual piano.

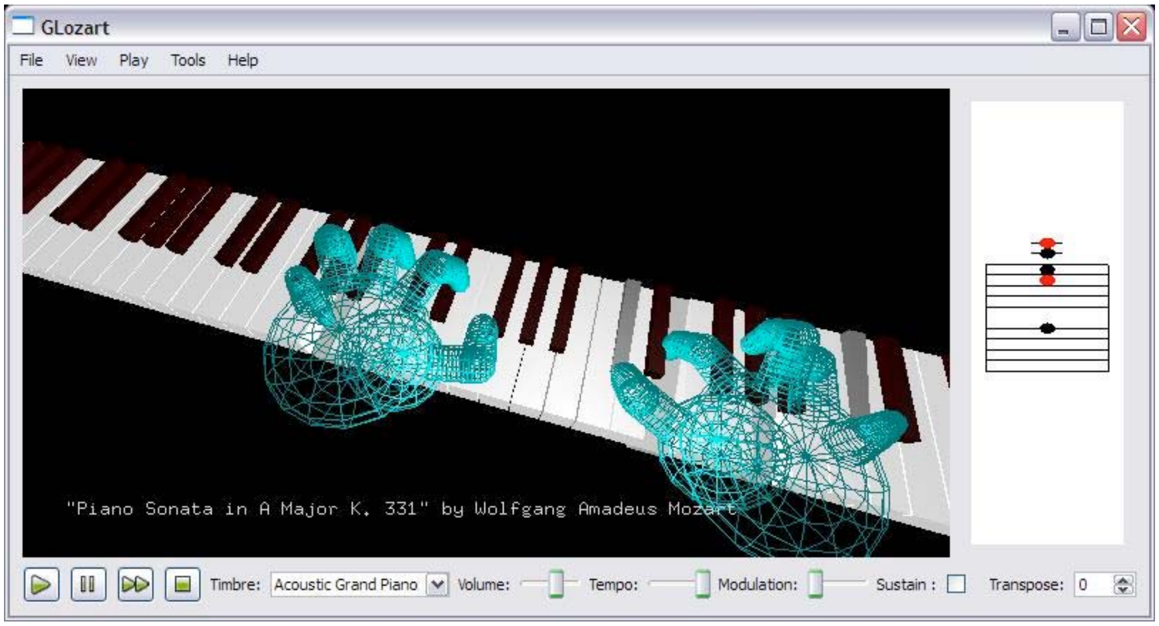


Figure 1.6: Side-by-side dynamic notation.

GLozart is limited in that the notation does not display exactly the same notation as the input MusicXML. GLozart's notation does not include any information on timing and only displays pitch. Also, enharmonic notation is not used. Instead, every black piano note is denoted by a sharp note instead of its flat spelling. This limitation also appears in many keyboards with the LCD feature. This is because choosing enharmonic spelling is usually based on key signature. Since the current version of GLozart only parses pitches and divisions from the MusicXML, certain data like key signature are not stored. Hence, GLozart is unable to distinguish the correct enharmonic spelling although this is a possibility for future versions. Also, because of timing constraints and bugs with reading in TGA textures, a sharp note is not denoted with its typical '#' symbol. Instead, it is denoted by the color red.

Chord Constructor

A chord constructor is also supplied by GLozart to allow users to choose their own chords to be played directly from the interface (Figure 1.7). This is much quicker, convenient, and practical method than creating and importing a MusicXML with only one chord for the entire piece. This feature seemed necessary since keyboardists often stumble over finger positioning over specific chords. The constructor works by allowing the user to add notes to a typical staff with treble and bass clef. Due to time constraints, the constructor does not feature enharmonic spelling. Users may only choose between natural and sharp notes. This is reasonable since enharmonic spelling does not change how chords are played on the piano. Users may also delete notes currently on the staff. GLozart takes a beginner's approach to deciding whether a note is for the left or right hand by checking if the note is drawn in the tremble or bass clef. Notes below Middle C are played by the left hand while notes above are played by the right hand. Middle C itself is played by whichever hand is free with prioity given to the left hand.

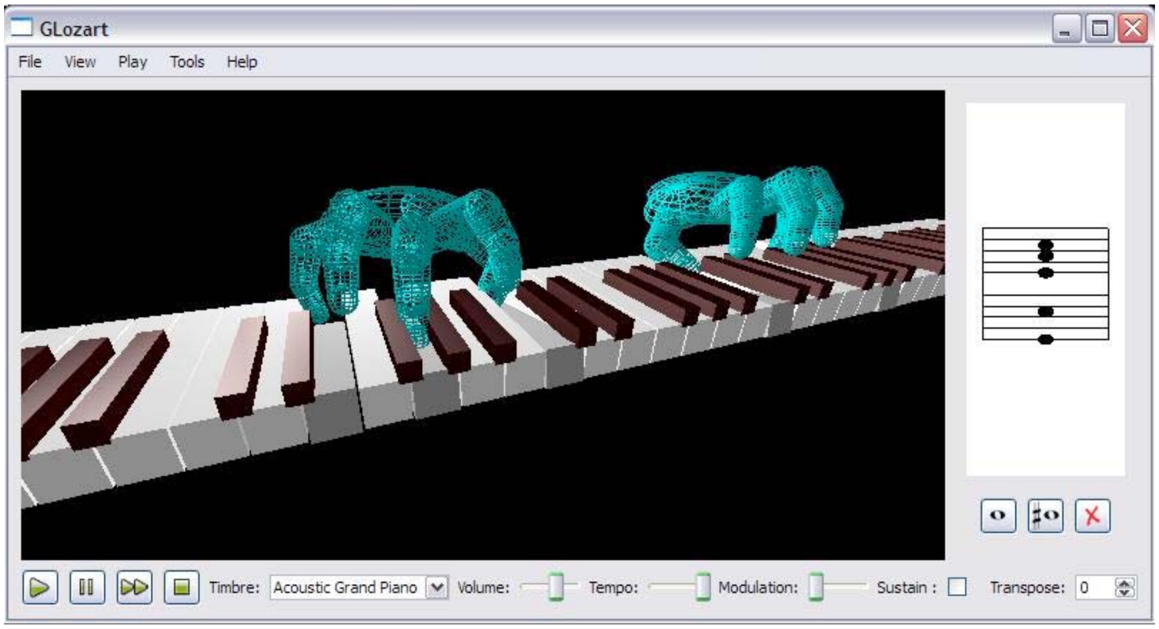
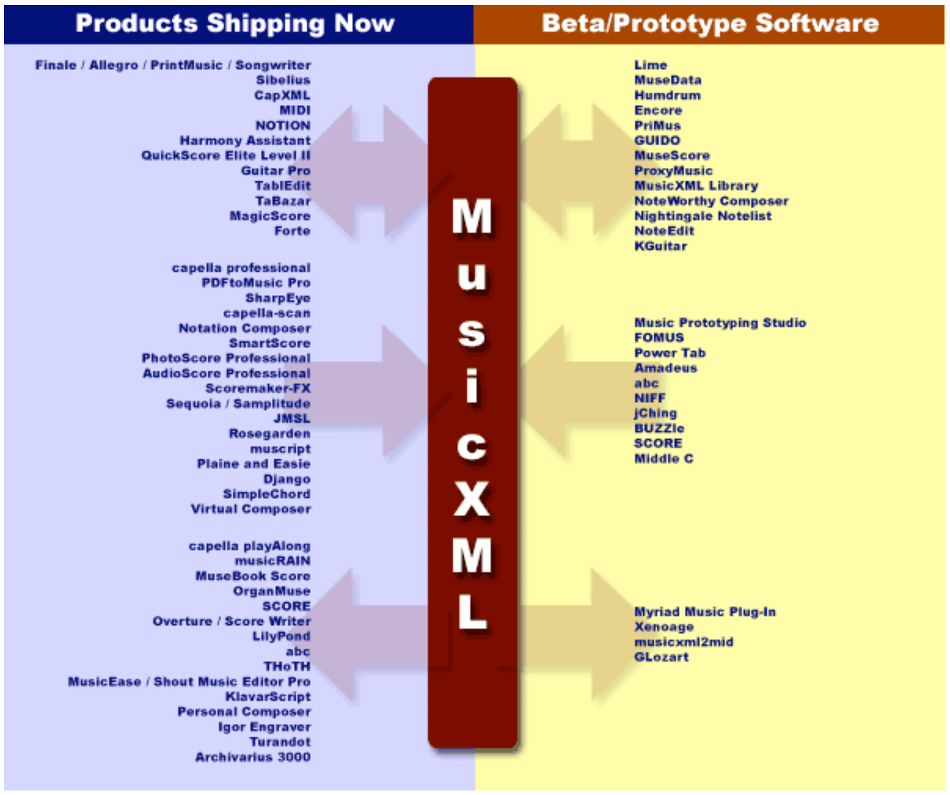


Figure 1.7: Chord constructor.

The chord constructor also enforces that the chord be playable in GLozart. This means that only a maximum of five notes can be constructed per hand (one note per finger). The minimum and maximum span per hand cannot be wider than an octave. If the user tries to add a note that invalidates a chord’s playability, the note is not added to the staff. Once a proper chord is constructed, the user can playback the chord with the same play and stop functions in the tool bar and can apply the same controls allowed during regular MusicXML playback.

Application Potential

Future, more complete versions of GLozart may have a large potential for dissemination. In reference to an earlier prototype version of GLozart, the developers of MusicXML, Recordare, commented, “We hope this is the first of many programs that use MusicXML data to create interesting and useful animations and visualizations.” GLozart is listed on Recordare’s web page along with several commercial and prototype products at http://www.recordare.com/xml/software.html. The link contains a bio of GLozart and a table (Figure 1.8) of both shipping and prototype software that uses MusicXML including GLozart on the bottom right of the table under the Beta/Prototype Software.



Products Shipping Now

Finale / Allegro / PrintMusic / Songwriter
Sibelius
CapXML
MIDI
NOTION
Harmony Assistant
QuickScore Elite Level II
Guitar Pro
TablEdit
TaBazar
MagicScore
Forte

capella professional
PDFtoMusic Pro
SharpEye
capella-scan
Notation Composer
SmartScore
PhotoScore Professional
AudioScore Professional
Scoremaker-FX
Sequoia / Samplitude
JMSL
Rosegarden
muscript
Plaine and Easie
Django
SimpleChord
Virtual Composer

capella playAlong
musicRAIN
MuseBook Score
OrganMuse
SCORE
Overture / Score Writer
LilyPond
abc
THoTH
MusicEase / Shout Music Editor Pro
KlavarScript
Personal Composer
Igor Engraver
Turandot
Archivarius 3000

Beta/Prototype Software

Lime
MuseData
Humdrum
Encore
PriMus
GUIDO
MuseScore
ProxyMusic
MusicXML Library
NoteWorthy Composer
Nightingale Notelist
NoteEdit
KGuitar

Music Prototyping Studio
FOMUS
Power Tab
Amadeus
abc
NIFF
jChing
BUZZle
SCORE
Middle C

Myriad Music Plug-In
Xenoage
musicxml2mid
GLozart

Figure 1.8: Recordare’s listing for MusicXML software

Today, in a globalized environment via Internet technologies, there is more of a demand for people to learn things on their own. This can stem from people’s inflexibility with time schedules. Also, it is much cheaper to learn on one’s own than hire private tutors. GLozart addresses those needs by supplying a different angle to learning than your typical “how to” online tutorial. GLozart is a great tool for amateur pianists who want to play challenging pieces in limited amount of time. This is true especially because reading standard notation can be very time-consuming and difficult. With GLozart, some users might even be able to ignore the standard notation entirely and rely just on imitating the graphics. Experienced pianists and even non-musicians can use GLozart for entertainment purposes like watching a player piano. Furthermore, the application logic of GLozart can be extended to future applications for other musical instruments. This leaves room for a possible guitar application (GLendrix?) or even a saxiphone application (Kenny GL?).

Music Glossary

Articulation—The manner in which notes are performed (Harnsberger 14).

Atonal—Music without a tonal center or key (Harnsberger 14).

Bass Clef—The F clef on the fourth line of the staff (Harnsberger 18).

Chord—Three or more notes sounded simultaneously (Harnsberger 31).

Clef—The symbol written at the beginning of a staff that indicates which notes are represented by which lines and spaces (Harnsberger 33).

Coda—An ending section of a movement or piece (Harnsberger 33).

Crescendo—Gradually becoming louder (Harnsberger 38).

Dynamics—Symbols that indicate varying degrees of volume (Harnsberger 47).

Enharmonic—Two notes that sound the same, but are spelled differently. For example, B-flat and A-sharp are enharmonically the same (Harnsberger 49).

Flat—The symbol 'b' that indicates a note to be lowered by one pitch (Harnsberger 55).

Fortissimo—Very loud (Harnsberger 56).

Glissando—To slide from one note to another; on a piano, a rapid scale produced by sliding fingers over the desired keys (Harnsberger 60).

Grace Note—A small note played quickly before the beat (Harnsberger 61).

Key—The tonal center of a composition (Harnsberger 72).

Key Signature—The groups of sharps or flats that appear at the beginning of a staff which indicate the key (Harnsberger 73).

Measure—The notes and rests between two bar lines in notation (Harnsberger 81).

Melody—A succession of single notes (Harnsberger 81).

Middle C—The note C that is near the middle of the piano keyboard. It is notated between the treble and bass clef of a staff (Harnsberger 83).

Modulation—To change key within a composition (Harnsberger 84); attuning to a certain pitch or key; varying in volume of tone (Dictionary.com)

Natural—A note that is neither sharp or flat (Harnsberger 87).

Octave—Distance from one note to its nearest note above or below it with the same name (Harnsberger 90).

Pitch—The location of a note related to its highness or lowness (Harnsberger 99).

Quarter Note—Note that specifies duration is a fourth of the measure.

Repeat—Signs that indicate a musical section should be repeated (Harnsberger 108).

Rhythm—The organization of music in time using long and short note values (Harnsberger 109).

Scale—The arrangement of notes in a specific order (Harnsberger 113).

Sharp—The symbol '#' that indicates a note to be raised by one pitch (Harnsberger 117).

Sixteenth Note— Note that specifies duration is a sixteenth of the measure.

Staff—The horizontal lines on and between which notes are written. Normally there are five lines (Harnsberger 124).

Sustain—The amount of time the strings of the piano are allowed to vibrate (Harnsberger 127).

Tempo—The speed of a section of a composition or the speed of a complete composition (Harnsberger 130).

Timbre—Quality of sound of a voice or instrument (Harnsberger 132).

Tonal—Pertaining to a key (Harnsberger 133).

Tonality—Having a tonal center to a composition (Harnsberger 133).

Transposition—To change a composition from one key to another (Harnsberger 135).

Treble Clef—The G clef on the second line of the staff (Harnsberger 135).

Tremolo—Alternating rapidly between two notes or chords (Harnsberger 135).

Voice—A part or melody line of a piece (Harnsberger 144).

Resources

Music Glossary Resources

http://www.dictionary.com.

HarnsBerger, Lindsey C. Essential Dictionary of Music. Los Angeles, Ca: Alfred Publishing Co., Inc., 1997

GUI Resources

http://trolltech.com. (Qt's Developers' Official Site)

http://doc.trolltech.com/4.3/tutorial.html. (Qt 4.3 Tutorial)

http://doc.trolltech.com/4.3/classes.html (Qt 4.3 Class API Reference)

http://qtnode.net/wiki/Qt4_with_Visual_Studio. (Configuration Turtorial for Qt Integration with Visual Studios)

MusicXML Resources

http://www.recordare.com. (MusicXML Developers' Official Site)

http://www.recordare.com/xml/software.html. (Recordare's Listing With Link to Previous Version of GLozart).

MIDI Resources

http://msdn.microsoft.com/library/. (Microsoft's MSDN Library including MIDI API)

http://www.midi.org/ (MIDI's Official Website)

http://www.borg.com/~jglatt/tech/stream.htm. Glatt, Jeff. (MIDI Stream API Tutorial)

http://www.borg.com/~jglatt/tutr/miditutr.htm. Glatt, Jeff. (MIDI Tutorials)

http://www.skytopia.com/project/articles/midi.html. White, Daniel. 2005 (MIDI Byte Layout)

http://www.cs.uccs.edu/~cs525/midi/midi.html. Chow, Edward. (MIDI Tutorials)

Miscellaneous Resources

http://www.csc.calpoly.edu/~zwood/teaching/csc471/finalproj26/jdelosre/. (Previous Version of GLozart).

http://www.csc.calpoly.edu/~zwood/teaching/csc471/finalproj24/cpasilla/ (A similar project without hand animation)