UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Progetto di Programmazione ad Oggetti

Marco Uderzo, matricola 1201290

Anno Accademico 2019/2020

QVisualizer

Real-Time Music Visualization

Sommario

QVisualizer è un'applicazione desktop che permette la gestione di file musicali e la loro visualizzazione real-time.

Istruzioni di Compilazione ed Esecuzione

Come concesso dal Professor Ranzato, prima della compilazione è necessario lanciare nel terminale il comando

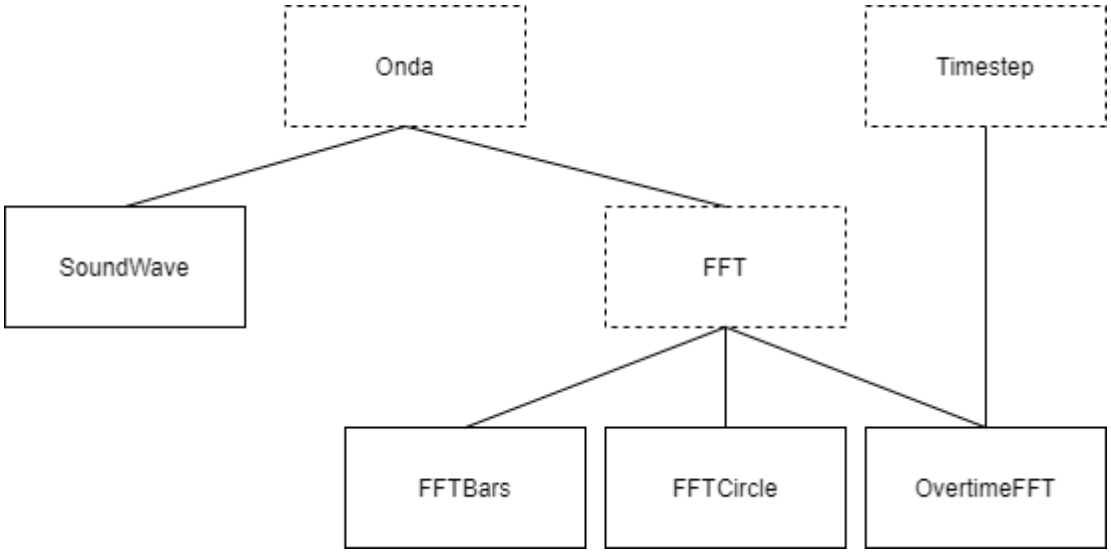
```
sudo apt install qtmultimedia5-dev libqt5multimedia5-plugins
```

che procederà ad un download di 1.4Mb e quindi all'installazione del modulo multimedia di Qt. Questo modulo, infatti, non è originariamente stato installato nella macchina virtuale ufficiale del progetto, ma è indispensabile per il funzionamento di QVisualizer.

Si procede quindi alla compilazione del progetto utilizzando il .pro fornito insieme agli altri file.

Gerarchia

Il modello di QVisualizer comprende due gerarchie distinte. La prima è composta dalle classi:



Onda è la classe base astratta di questa gerarchia ed offre un'interfaccia per la gestione del buffer audio in arrivo dal controller. SoundWave è la prima classe concreta. Implementa il metodo virtuale puro **run(**std::complex<double>*, unsigned int, unsigned int, double) della classe Onda. Elabora l'onda grezza attenuandola tramite un fattore moltiplicativo deciso dall'utente.

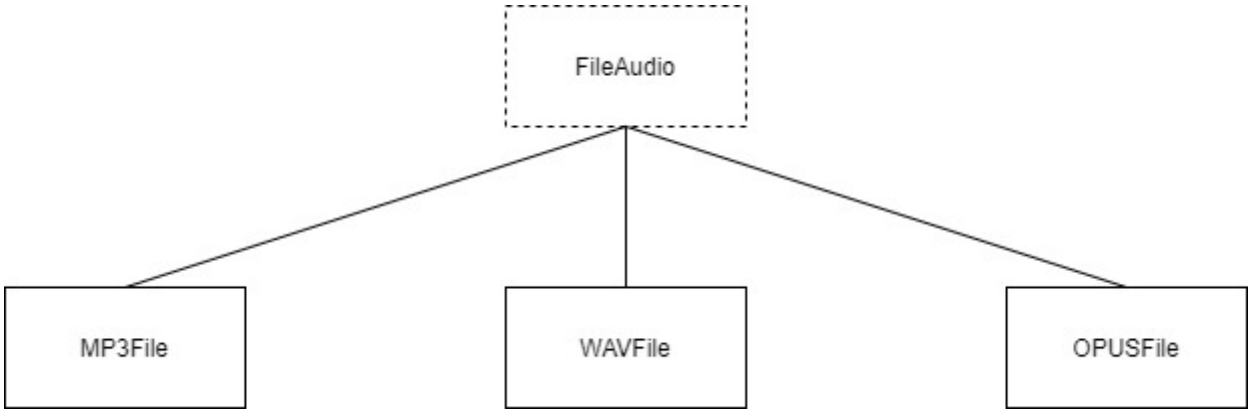
La classe FFT eredita da Onda, ed è una classe astratta che si occupa di computare la Trasformata di Fourier (Fast Fourier Transform) sull'onda, elaborando una trasformazione della stessa dallo spettro temporale allo spettro delle frequenze. L'algoritmo scelto è quello di Cooley-Tukey, implementato nei metodi **runFFT(**std::complex<double>* f, unsigned int N, double d**), reverse(**unsigned int N, unsigned int n**), sort(**std::complex<double>* f1, unsigned int N**)** e **transform(**std::complex<double>* f, unsigned

int N**)**. Si può trovare l'algoritmo della FFT al link. Il metodo **zeroPad(**std::complex<double>***)** utilizza la tecnica dello Zero Padding per assicurarsi che a **runFFT(**std::complex<double>* f, unsigned int N, double d**)** venga passato un buffer con dimensione pari ad una potenza di 2, nel nostro caso fissata a 4096, fondamentale per il corretto funzionamento dell'algoritmo. E' il metodo di interpolazione standard nel campo dell'analisi dei segnali. Inoltre è presente un metodo di smoothing del risultato della trasformazione, **smoothFFT**(int) e il metodo virtuale puro **remapFFT**(const std::vector<double>& ,double, double, double, double) per rimappare i valori in uno specifico range deciso dall'utente.

Da FFT derivano tre classi concrete, FFTBars, FFTCircle e OvertimeFFT. Queste classi implementano i metodi virtuali puri **run**(std::complex<double>*, unsigned int, unsigned int, double) della classe Onda e **remapFFT**(int) della classe FFT.

Di particolare importanza è la classe OvertimeFFT, in quanto sfrutta l'ereditarietà multipla. Infatti essa deriva sia da FFT che da una classe astratta Timestep. Sapendo che un'onda sonora è rappresentata nelle ascisse dalla componente temporale, che si perde nella FFT, si è deciso di aggiungere una "terza dimensione" alla FFT. Quindi, OvertimeFFT rappresenta lo spettro delle frequenze nel corso del tempo. Per fare ciò, Timestep utilizza un *std::chrono* che regola la frequenza di campionamento delle FFT. La classe OvertimeFFT implementa i metodi virtuali puri **tick**(), di TimeStep, **remapFFT**(int) di FFT e **run**(std::complex<double>*, unsigned int, unsigned int, double) di Onda. Il metodo di OvertimeFFT **smoothTimeBuffer**() utilizza invece un algoritmo di smussatura particolarmente diverso rispetto agli altri.

La seconda gerarchia è composta dalle seguenti classi:



E' una gerarchia di tipi che permette la gestione di file audio e i relativi metadati, se presenti e supportati. FileAudio è la classe base astratta. Le classi che ne derivano sono concrete e rappresentano i più comuni formati di file audio. Implementano dei setter e getter per la gestione dei metadati, e il metodo virtuale puro **clone**() della classe base FileAudio. La scelta iniziale dei formati comprendeva anche formati più diffusi, quali FLAC, ALAC e AAC. Purtroppo questi non sono supportati dall'implementazione del metodo **getPeakValue(**const QAudioFormat &format**)**, metodo presente negli esempi di Qt al link. La decodifica di tali formati, quindi, avrebbe richiesto una considerevole quantità di tempo. Al loro posto si è scelto di supportare il formato OPUS.

In MP3File si include una classe di Qt, *QImage*, in modo da avere una *QImage* come campo dati nella classe derivata per salvare la Cover Art del file audio, se il formato lo prevede. Questa scelta permette di avere già pronta la Cover Art fin da subito e salvarla in un MP3File, evitando ulteriore overhead nella creazione "*on the fly*" di una *QImage* nel controller. Inoltre, la soluzione adottata rende il codice più chiaro.

Il contenitore MediaVector contiene oggetti della gerarchia FileAudio.

Descrizione delle Chiamate Polimorfe

Nella gerarchia di FileAudio sono presenti i seguenti metodi virtuali puri:

- virtual ~***FileAudio***() = default
- virtual **FileAudio*** ***clone***() const = 0 : usato come costruttore di copia polimorfo.

e il seguente metodo virtuale:

- virtual bool ***isLossless***() const : ritorna *false* di default, viene reimplementato nelle classi WAVFile e OPUSFile per ritornare *true*, poiché tali formati sono lossless.

Nell'altra gerarchia sono presenti i seguenti metodi virtuali puri:

Nella classe base Onda:

- virtual ~***Onda***() = default;
- virtual **std**::**vector**<double> ***run***(**std**::**complex**<double>*, unsigned int, unsigned int, double) = 0 : questo metodo virtuale puro è implementato da SoundWave, FFTBars, FFTCircle e OvertimeFFT.

Nella classe FFT:

- virtual void ***remapFFT***(const **std**::**vector**<double>& ,double, double, double, double)= 0 : questo metodo virtuale puro è implementato nelle classi FFTBars, FFTCircle e OvertimeFFT. La sua implementazione è identica a quella di **map()**, da processing.org.

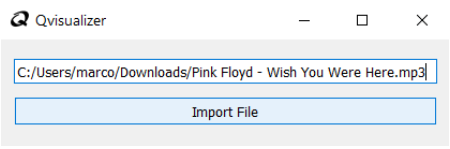
Nella classe Timestep:

- virtual ~***TimeStep***() = default;
- virtual void ***tick***(**std**::**vector**<double>) = 0 : questo metodo virtuale puro è implementato da OvertimeFFT.

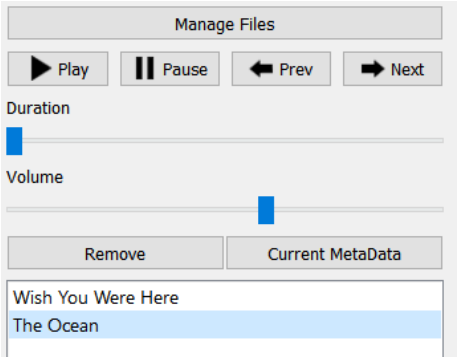
Alcune classi della View, quali SoundWaveWidget, FFTBarsWidget, FFTCircleWidget, OvertimeFFTWidget e MetaDataWidget, ereditano dalla classe QOpenGLWidget (e anche da QOpenGLFunctions) di Qt, reimplementando i metodi:

- virtual void ***initializeGL***() override : inizializza OpenGL.
- virtual void ***paintGL***() override : gestisce il rendering della scena.
- virtual void ***resizeGL***(int w, int h) override : gestisce il resize del widget

Manuale Utente della GUI



Il lato destro dell'applicazione è il pannello di controllo principale di QVisualizer. Per prima cosa, per caricare della musica, si prema il pulsante "Manage Files". Comparirà una schermata, si incolli il path completo del file: su Ubuntu, il path dovrebbe essere nella forma:



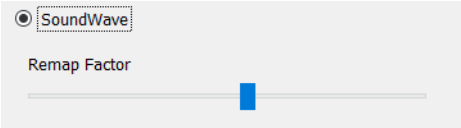
"`/home/path/to/file.estensione`".

Il carattere " \ " non è supportato in quanto è il carattere di escape di C++. Infine si prema Import File. Se il processo di importazione è andato a buon fine, si troverà il corrispondente file nella lista. Per selezionarlo, si faccia doppio click. A questo punto, il file è pronto alla riproduzione.

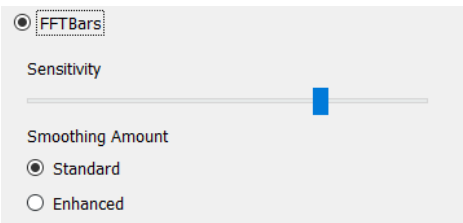
Sotto al pulsante Manage Files se ne trovano altri quattro, designati al controllo del player. Poi vi sono due slider, uno per visualizzare il progresso della musica e per spostarsi lungo la sua durata, l'altro invece per impostare il volume.

I due pulsanti sopra alla playlist servono, corrispondentemente, a rimuovere l'elemento e mostrare i metadati del file correntemente selezionato.

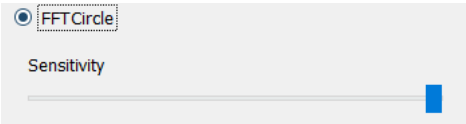
Premere Play e selezionare una modalità di visualizzazione. Si apriranno dei sottomenù per controllare le varie impostazioni.



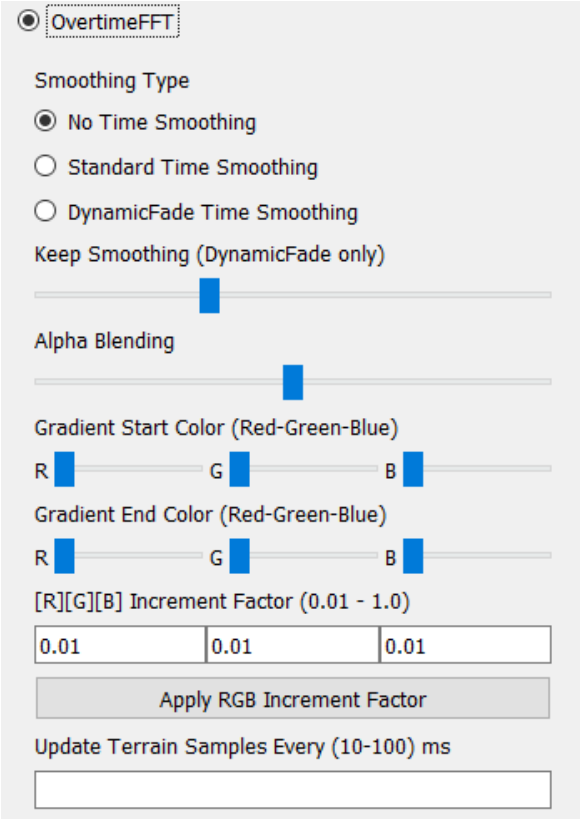
SoundWave ha un solo slider, permette di rimappare l'intensità dell'onda, per una migliore visualizzazione.



FFTBars permette di impostare la sensibilità del rimappamento della FFT. Usualmente, una via di mezzo equivale ad una visualizzazione più piacevole. Si può impostare la quantità di smussamento della FFT per ottenere picchi più uniformi.



FFTCircle ha un solo slider che permette di impostare la sensibilità, esattamente come in FFTBars.



OvertimeFFT offre una vasta gamma di impostazioni. Lo smussamento non è più nell'asse X, ma nell'asse simulato Z. Nel menù Smoothing Type si può scegliere il tipo di smussatura: nessuna, standard e DynamicFade. Quest'ultimo utilizza un algoritmo di smussatura diverso in grado di creare un disegno ancora più uniforme e piacevole. Usando DynamicFade, l'utente può scegliere il numero di iterazioni di smussatura tramite lo slider "Keep Smoothing". E' lasciata all'utente la decisione circa la frequenza di aggiornamento del "terreno" digitando nella casella "Update Terrain Samples Every (10-100) ms" e premendo il tasto invio. Il valore va espresso in millisecondi, e di default il valore è già di 10ms. OvertimeFFT offre molto controllo sulla parte grafica. E' possibile impostare la quantità di Alpha Blending desiderato, ottenendo un diverso effetto su come il "terreno" generato appare agli occhi dell'utente. Viene offerta la possibilità di impostare una palette RGB di due colori che comporranno una progressiva sfumatura dall'uno all'altro. E' sufficiente muovere gli slider. La velocità di sfumatura dal primo colore al secondo è determinata da un incremento, con range 0.01 - 1.0, impostabile nei relativi campi, e poi premendo "Apply RGB Increment Factor". Di default è già impostata a 0.01, quindi a meno che non si cambino quei valori, non serve fare nient'altro.

Ripartizione Ore

- Analisi Preliminare del Problema: 1 ora
- Progettazione Modello e GUI: 3 ore
- Apprendimento della libreria Qt: conteggiato nella voce di codifica della GUI e controller.
- Codifica Modello: 17 ore
- Codifica GUI e Controller: 26 ore
- Debugging: 10 ore
- Testing: 3 ore
- Monte Ore Complessivo: 60 ore

Il monte ore complessivo più alto rispetto alle 50 ore richieste è dovuto soprattutto all'apprendimento di alcune classi del framework Qt, specialmente quelle multimediali, al debugging e al porting da Windows a Ubuntu, dal momento che i QAudioBuffer sono gestiti in maniera diversa nei due sistemi operativi e i metadati dei file multimediali non hanno un'implementazione unificata.

Suddivisione del Lavoro

- Contenitore: Kostadinov Samuel
- Controller: Uderzo Marco e Kostadinov Samuel
- Modello:
    - Onda: Uderzo Marco e Kostadinov Samuel
    - FFT: Uderzo Marco e Kostadinov Samuel
    - FFTBars: Uderzo Marco
    - FFTCircle: Kostadinov Samuel
    - OvertimeFFT: Uderzo Marco
    - SoundWave: Uderzo Marco
    - FileAudio: Kostadinov Samuel e Uderzo Marco
    - MP3File, WAVFile, AIFFFile, OPUSFile: Uderzo Marco e Kostadinov Samuel
    - TimeStep: Uderzo Marco
- GUI:
    - SoundWaveWidget: Uderzo Marco
    - FFTBarsWidget: Uderzo Marco
    - FFTCircleWidget: Kostadinov Samuel
    - OverTimeFFTWidget: Uderzo Marco
    - MetaDataWidget: Uderzo Marco
    - MediaPropertiesWidget: Uderzo Marco
    - MainWindow: Uderzo Marco e Kostadinov Samuel

File Musicali di Test

Al link di Google Drive è possibile trovare alcuni file musicali già pronti per testare QVisualizer. Si noti che è necessario scaricarli.

Note

Alcuni metadati, visualizzati correttamente su Windows, purtroppo, non sono disponibili su Linux, a causa della diversa gestione di Qt dei metadati dei file tra i due sistemi operativi e della mancanza di un loro corrispondente nel namespace *QMediaMetadata*. Si è deciso di mantenere comunque i metadati disponibili su Windows e differenziare il comportamento del programma a seconda del sistema operativo su cui gira.

Nella classe FileAudio, si può notare che il campo dati *duration* ha come tipo *long long*, nonostante sia immediato e naturale dichiararlo *unsigned*. Il motivo della scelta è che **QMediaPlayer->duration()** ritorna un *qint64*, ovvero un *long long*. Si è voluta evitare la conversione ed è stata introdotta un'eccezione che viene sollevata qualora la durata sia minore di 0, indicando un errore. La durata viene visualizzata da un QSlider, che accetta come range solo *int*. In questo caso, nel controller si è optato per uno *static_cast* da *long long* ad *int*, notando che tale valore rappresenta i secondi di una canzone e che quindi un overflow di *int* è virtualmente impossibile in un normale contesto di utilizzo.

Nella classe FFT, si noti che il metodo **getFFTInputData()** non è marcato *const* com'è buona pratica fare. Il motivo è che se tale metodo fosse *const*, anche l'intero array da ritornare dovrebbe essere *const*, e non lo può essere.

Ambiente di Sviluppo

Il sistema operativo utilizzato per lo sviluppo è Windows 10 64bit. La versione Qt utilizzata è Qt 5.13.0. Il compilatore utilizzato è MinGW 7.3.0 64bit.