

台灣二〇〇五年國際科學展覽會

科 別：電腦科學

作品名稱：音材施教-音高與音色辨識之探討

得獎獎項：大會獎第一名

新加坡正選代表:參加新加坡第 28 屆青年科學節

英特爾電腦科學獎:第一名

學 校：臺北市立第一女子高級中學

作 者：李雨霜、陳昀詩

評語與建議事項：

此作品主要利用電腦程式做音高與音色之辨識。作品具創意，且研究完整，方法，結果均佳。

作者簡介

姓名：李雨霜

性別：女

自我簡介：

身為一名理組學生的同時，我也很喜歡文學閱讀，以及音樂欣賞。

在研究這個題目的過程中，我們理解到美好樂音背後的物理及數學意義，更能夠以歸納的想法來處理音訊。



姓名：陳昀詩

性別：女

自我簡介：

我是一個興趣廣泛的人，不論任何科目都不排斥，而最喜歡的是文學。另外，我也很喜歡幾何、網頁製作等設計方面的事情。

製作這個專研讓我收穫良多，也作的滿快樂的，可以多了解聲波的一些特性，有關聲音數據的處理，也解開了電腦如何處理聲音資訊的疑惑。



音材施教-音高與音色辨識之探討

摘要

我們使用 C++ 撰寫了一個音準練習程式：使用者輸入聲音後，經由音頻辨識方法求出其頻譜中最高能量之頻率，以之為基頻，再將其與目標音高相較，得到誤差率及走音程度。此外程式還可秀出所唱的樂譜，和發出對應的鋼琴或正弦波的聲音，方便使用者校音。而為了做音高辨識，我們也收集了許多聲音檔案，觀察其特色，研究不同音色的頻譜或是波形特性，並利用其特徵完成一個音色自動辨識程式。

首先我們測試了各種演算法，並且選用了快速傅利葉轉換作為主要製作的演算法；接著我們利用 Microsoft Visual C++ 撰寫我們的程式。這個程式主要可分為錄音、辨識以及樂譜繪製三大部分，皆會在此份報告中詳細說明。

文中將說明音頻及音色辨識的方法，一些關於音樂的基本知識，微軟公司的 wave 檔格式，此系統之應用，以及音色的波形頻譜分析。我們使用 FFT 求得聲音的頻譜，且將針對此部分演算法做簡單的說明，並探討如何達到所需之頻率準確度，如何以較高效率辨識，及如何找出不同音色的特性。

壹、研究動機

音樂課時，老師時常會帶領大家唱些歌曲，但是在唱歌的過程中，難免會有走音的狀況，為了解決走音的問題，勢必要回家練習，但練習時很難確定是否唱得正確，因此我們想到撰寫一個程式，能幫助有心改善自己音準的人自行在家練習。

另外研究進行到一半時，我們也發現所使用的演算法並不適用於所有的樂器，而就算是能測得較準確的樂器，仍是有些許誤差，因此我們想進而做音色方面的研究。

貳、研究目的

我們希望製作出一個能練習音準的程式：使用者在藉由麥克風輸入聲音後，程式會對所輸入的聲音作辨識，告知使用者所唱的聲音之音高為何，並同時顯示原本與後來所唱的樂譜，方便使用者對照何處需要改進，以達到練習唱歌的功能。除此之外，使用者也可以選擇不同之音高，並聽其對應的鋼琴或正弦波聲音，以方便校對音準。另外也希望觀察各種不同音色的聲音，分析其波形與頻譜，以找出各個音色的特徵，進而撰寫一些程式對不同的聲音做音色判斷。

參、文獻探討

一、國內相關軟體

首先，我們搜尋關於音樂教學軟體的資料，發現國內在此方面十分缺乏：僅有一些以鍵盤當鋼琴的幼教軟體，而缺乏讓人練習唱歌的 CAI 程式。

二、基本背景知識

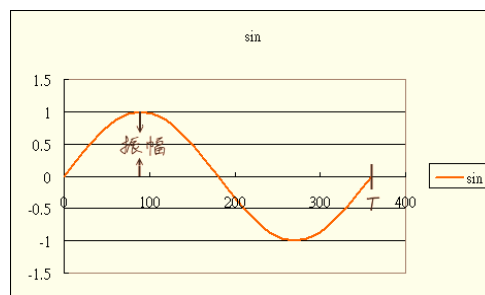
〈一〉音高[參考資料 1,3]

音高即聲音的高低，取決於物體振動的快慢；視振動頻率穩定與否而產生固定音高及不定音高，像鋼琴及各樂器的音即是固定音高；而打擊樂器大部分則產生不定音高，而各種音高間的變化，便產生了旋律。聲音的頻率影響音高，在一個八度之間，依照十二平均律劃分為 12 個音，十二平均律是由明朝的朱載堉所提出，經由十二平均律的分割，每兩個半音之間的頻率商大約為 2 的 $1/12$ 次方，相當於 1.05946309 Hz，且每個音各有其對應的頻率（如表 1）

	1	2	3	4	5	6	7
C	32.70	65.41	130.81	261.63	523.25	1046.50	2093.00
C#	34.65	69.30	138.59	277.18	554.37	1108.73	2217.46
D	36.71	73.42	146.83	293.66	587.33	1174.66	2349.32
D#	38.89	77.78	155.56	311.13	622.25	1244.51	2489.02
E	41.20	82.41	164.81	329.63	659.26	1318.51	2637.02
F	43.65	87.31	174.61	349.23	698.46	1396.91	2793.83
F#	46.25	92.50	185.00	369.99	739.99	1479.98	2959.96
G	49.00	98.00	196.00	392.00	783.99	1567.98	3135.96
G#	51.91	103.83	207.65	415.30	830.61	1661.22	3322.44
A	55.00	110.00	220.00	440.00	880.00	1760.00	3520.00
A#	58.27	116.54	233.08	466.16	932.33	1864.66	3729.31
B	61.74	123.47	246.94	493.88	987.77	1975.53	3951.07

[表 1. 十二平均律分割音高頻率對照表]

我們可以藉由觀察聲音的頻譜與波形來得知音高：聲音是由許多弦波組成，具有弦波的許多數學及物理特性，所以我們利用其特性來做處理。首先我們用以時間為橫軸，信號大小為縱軸的波形圖來觀察聲音的週期及振幅；在這樣的圖形中，波的高度為其振幅，週期的倒數為其頻率，波的形狀為其音色。（如圖 1.）

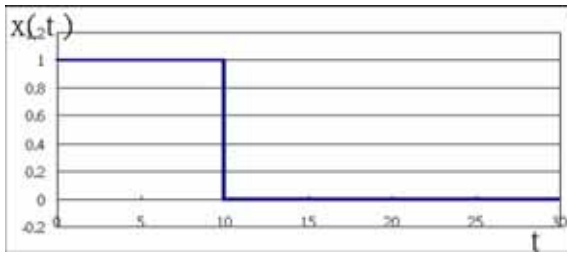


[圖 1. 基本正弦波圖形]

將聲音的波形資料通過傅利葉轉換〔後敘〕得出頻譜；在這樣的圖形中，縱軸為振幅，橫軸為頻率，音樂訊號的第一高峰則在大多數情況下為其基頻，各個高峰，也就是泛音之間的比例為其音色。以下是一個方形波訊號經過頻譜轉換後所得到的

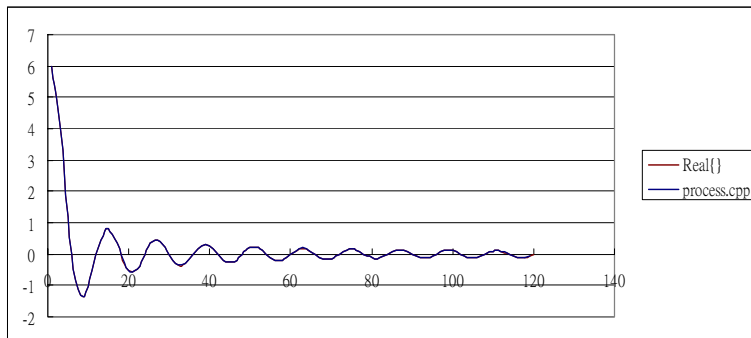
結果範例：

1. 以方形波為例



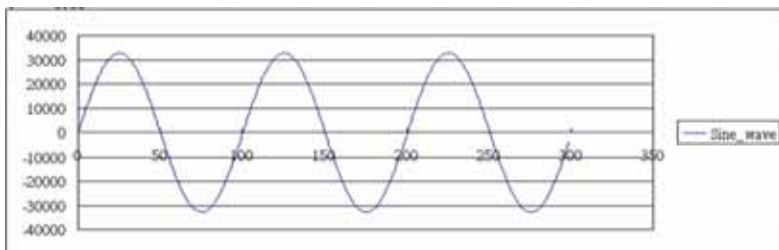
[圖 2. 方形波波形圖]

↓Fourier Transform



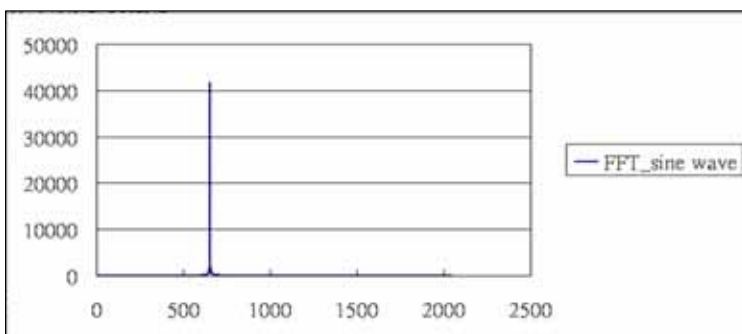
[圖 3. 方形波頻譜圖]

2. 以正弦波為例



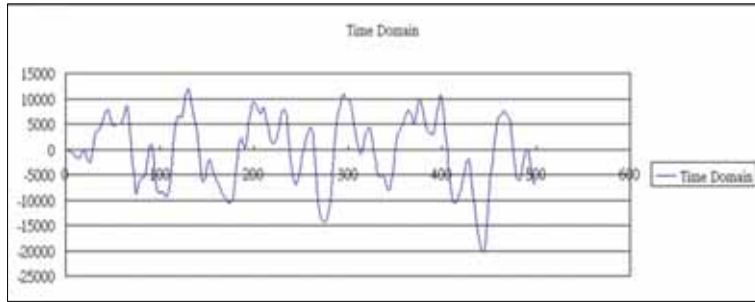
[圖 4. 正弦波波形圖]

↓Fourier Transform



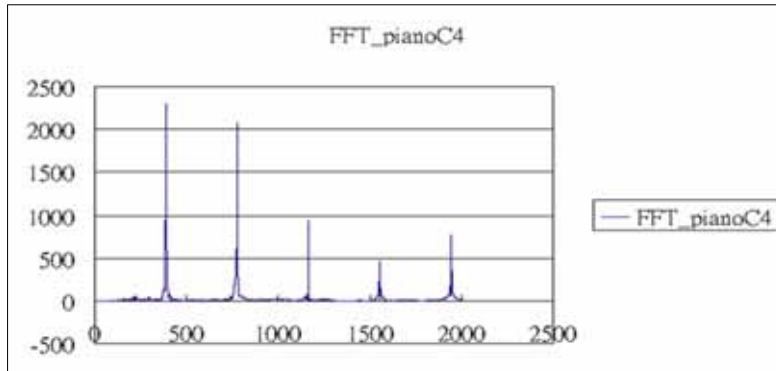
[圖 5. 正弦波頻譜圖]

3. 以鋼琴音為例



[圖 6. 鋼琴波形圖]

↓ Fourier Transform



[圖 7. 鋼琴波頻譜圖]

〈二〉 Wave 檔案格式 (Waveform Audio File Format) (參考資料2)

wave 檔分三區：控制訊息區、錄音資料區及文字說明區，每一區最開始有四個位元組記載著該區名稱：控制訊息區的名稱為“_fmt”，錄音資料區的名稱為“data”，文字說明區的名稱則為“LIST”。接著是標記大小的另四個位元組，最後是此區的資料內容。

1. 檔頭訊息區塊 (_fmt)：大小固定為 16 bytes。
 - (1) 資料的格式形態 (wFormatTag)：通常為 1
 - (2) 錄音軌數 (nChannels)：1 表示單音、2 為立體音
 - (3) 取樣頻率 (nSamplePerSec)：每秒取樣數(常為 44100)
 - (4) 每秒平均取樣位元數 (nAvgBytesPerSec)：每秒位元數
 - (5) 排列區間數 (nBlockAlign)：頻道數 x 取樣大小 ÷ 取樣頻率
 - (6) 每個樣本所佔位元 (nBitsPerSample)：取樣大小，8 或 16 位元
2. 語音資料區塊 (data)：原始語音資料存放處〈立體音左聲道在前〉
 - 取樣大小 8 位元：所錄得的語音資料以不含正負號的數值表示
 - 取樣大小 16 位元：每個樣本以 2 bytes(單音) 或 4 bytes(立體音) 表示，以含正負號的數值表示。

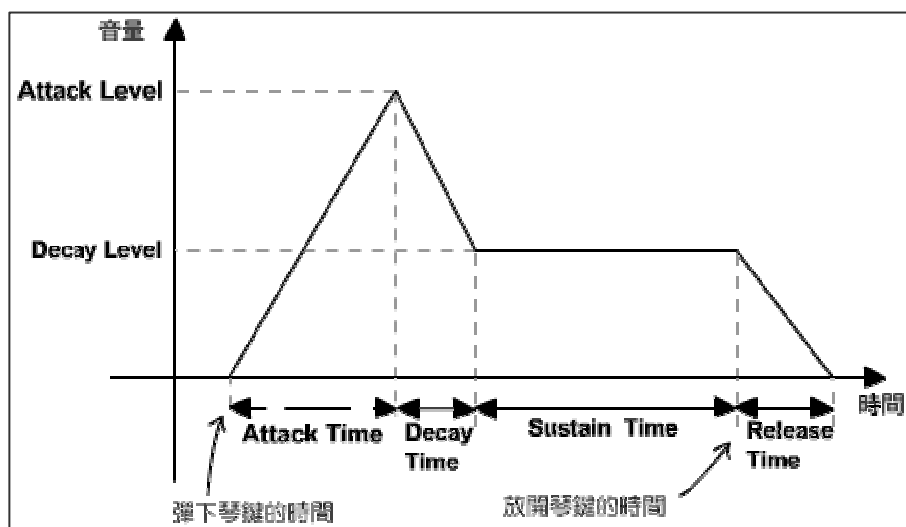
3. 文字說明區塊(LIST)：版權、製造日期等，與檔案的播放無直接關係。

〈三〉音色的概念[參考資料 5]

音色是聲音的三大特性之一。人們能夠明確分辨不同樂器的聲音，而不會混淆，這是由於它們的音色、波形包迹不同。音色決定於樂音的泛音（諧波）頻譜，也可以說是樂音的波形所確定的。如果沒有泛音成分，單純的基音正弦信號是較無音樂感的。因此，樂器樂音的頻率範圍，決非只是基頻的頻率範圍，應把樂器樂音的各次諧波都包括在內，甚至很高頻率的泛音，對樂器音色影響仍很大。

〈四〉波形的包迹(包絡)：[參考資料 8]

樂音的波形包迹指樂音演奏（彈、吹、拉，撥）每一音符時，單個樂音振幅起始和結束的狀態。有些樂器，在演奏開始一瞬間，振幅馬上達到最大值，然後振幅逐步衰減，有的樂器則增加地較為平緩。我們可將波形的變化情形分為分別是 Attack(起音)、Decay(衰減)、Sustain(延持)、與 Release(釋音)，也就是一般稱的 **ADSR**。



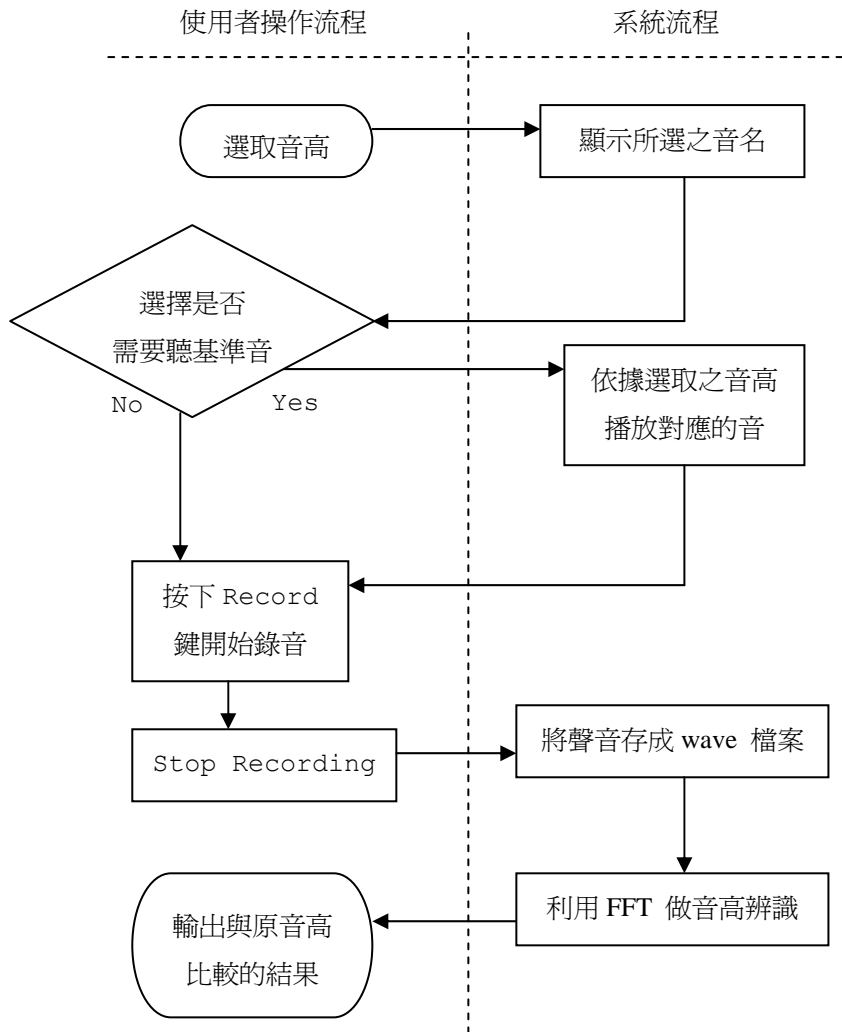
[圖 8. ADSR 示意圖]

肆、研究過程或方法

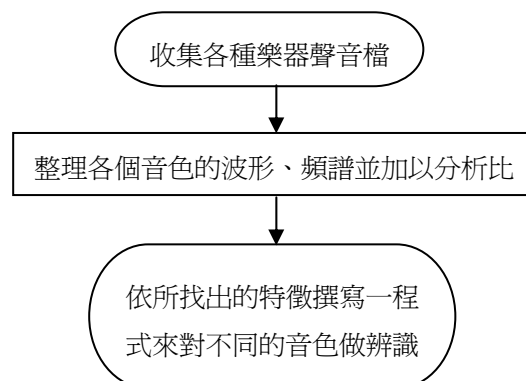
一、系統設計及流程

〈一〉系統流程

1. 音高辨識程式



2. 音色分析研究



〈二〉錄音程式的製作[參考資料 7]

爲了紀錄使用者所唱入聲音的資訊，撰寫一個錄音程式是必要的。數位錄音最大的目標就是解決傳統錄音〈紀錄類比訊號〉品質不佳的問題。主要用到的技術是將類比式的聲音轉化爲數位聲音符號的「類比 - 數位轉換」〈Analog-to-Digital, D/A〉。

所謂數位訊號指的是 0 與 1 的訊號，除了能精確紀錄各種聲音之外，也能避免雜音干擾或能量的散失而造成訊號模糊不清等現象。轉換的過程中尚牽涉到聲音品質，也就是轉化後能否逼近原聲的問題，這與聲音的取樣頻率及取樣大小有關。

我們主要運用在撰寫錄音程式的技術有聲音的輸入、輸出成 wave 檔，以及按下 replay 按鈕後的聲音撥放部分。我們引入 winmm.lib 這個函式庫。並利用其中關於 waveIn 的函數來進行實作。首先產生一組 wave 檔頭的資料：

```
monWave.BuildFormat(1, 8000, 16);
```

接下來將 wave 檔頭資料輸入到一個 wave 檔中：

```
waveIn.SetWaveFormat( monWave.GetFormat() );
```

利用 WaveInOpen 內建的函式將麥克風輸入的聲音訊號紀錄下來，最後存檔即完成錄音程序。

上面的程式碼主要是用 waveInOpen 來引入麥克風所輸入的聲音資料，並用 SetBuffer() 將資料存入陣列中，而當使用者按下 stop 按鈕時，waveInClose() 函數執行，停止資料的輸入。並利用上一節提到的方式將陣列資料寫成一個 wave 檔。接著，我們利用 PlaySound() 函數來播放使用者剛剛輸入的聲音，如此便完成一基本的錄音流程。

〈三〉Wave 檔製作[參考資料 1]

我們的程式還有一項功能，便是可讓使用者在唱入聲音之前先聆聽基準音，也就是所謂的標準音高，以方便使用者判別音準。以下是我們的以 MS Visual Basic 表現的虛擬碼：

```
-----  
Load length, Hz  
Output header  
For i = 1 to length  
Data[i] = sin(2*PI*I*deltaT*Hz)  
// deltaT = 1/44100  
Output data []  
Output 00 // The end mark of wave file  
-----
```

[程式碼 1. wave 檔案製作]

在這裡，我們製作了一個結構來定義檔頭，之後將聲音資訊存入陣列中。之後

將檔頭跟資訊以二進位方式輸出，最後補上檔尾(00)，便成爲一個 wave 檔案。

這個方法是使用單純開讀檔的方式，程式簡單而容易改造，缺點是這個結構必須自己定義來製作。實際上，MFC 的函式庫中已經有了相關的結構來幫助我們輸出 wave 檔。以下爲 MFC 函式庫中定義的檔頭格式以及製作方式：

首先我們先含入 winmm.lib 來幫助我們從事這個工作。在這之中，定義了以下跟 wave 有關的結構

```
-----  
typedef struct {  
    WORD    wFormatTag;  
    WORD    nChannels;  
    DWORD   nSamplesPerSec;  
    DWORD   nAvgBytesPerSec;  
    WORD    nBlockAlign;  
    WORD    wBitsPerSample;  
    WORD    cbSize;  
} WAVEFORMATEX;  
typedef struct {  
    LPSTR    lpData;  
    DWORD    dwBufferLength;  
    DWORD    dwBytesRecorded;  
    DWORD_PTR dwUser;  
    DWORD    dwFlags;  
    DWORD    dwLoops;  
    struct wavehdr_tag * lpNext;  
    DWORD_PTR reserved;  
} WAVEHDR;  
-----
```

[程式碼 2. wave 檔頭設定]

然後我們就可以利用這些結構從事程式的寫作，從而使程式的語法較爲精簡(因爲無須自行定義)，但是含入一個函式庫的檔案頗爲巨大，所以就單純的儲存而言，自己做一個簡單的程式是比較理想的作法。

由上面的虛擬碼可以發現，整體而言差距並不大，都是在準備完檔頭之後，讀入檔案資訊，而後輸出。

我們的程式在 console 版中，採用第一種的方式。而在視窗版中，則採用了第二種也就是 MFC 的方式來在錄音的時候存檔，實際上，由於第二種的函式等均包含在同一個檔案之中，所以在資料傳輸的方面較爲便利，而整體的擴充性也比較大。但若是單純的寫正弦波檔等，則以第一種較爲理想。

二、演算法實作[參考資料 1,2,4,6]

這整個程式的核心，便是音高辨識的演算法：傅利葉轉換，亦即一個能將一段聲音資訊轉換爲頻譜的演算法。以下將說明傅利葉轉換的應用原理與我們實作的部

分：

傅利葉轉換 (Fourier Transform)：

聲音最初的資料是隨著時間轉變，在這種情況下，X 軸為時間，Y 軸為信號強度(音壓)。而要進行音高的辨識則必須要對數位的訊號作分析，因此經由 Fourier Transform 取 X 軸為頻率 (frequency)，Y 軸為信號強度(振幅)的圖，稱為該音的頻譜 (Spectrum)。大多數的聲音轉換成頻譜後，在某個頻率位置會有一凸起 (代表此處能量最為集中)，這最高振幅所對應的頻率通常便是該音的頻率。

〈一〉離散傅利葉轉換 (Discrete Fourier Transform; DFT)

其定義為：

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \quad \text{其中 } k = 0, 1, \dots, N-1; n = 0, 1, \dots, N-1$$

亦即

$$X[k] = \sum_{n=0}^{N-1} \{ \operatorname{Re}\{x[n]\} \operatorname{Re}\{W_N^{kn}\} - \operatorname{Im}\{x[n]\} \operatorname{Im}\{W_N^{kn}\} \\ + j\{ \operatorname{Re}\{x[n]\} \operatorname{Im}\{W_N^{kn}\} + \operatorname{Im}\{x[n]\} \operatorname{Re}\{W_N^{kn}\} \}$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-kn}$$

$k = 0, 1, \dots, N-1$ 其逆轉換之定義則為

(*由於某些數學特性，離散傅利葉轉換在計算上可以有取巧的地方，這也就是所謂的快速傅利葉轉換 (FFT, Fast Fourier Transform)，關於這點於後面再說。)

離散傅利葉轉換可以將信號由時間關係訊號轉換到頻率關係圖，即是將時間關係圖轉換到頻譜 (Spectrum)；而離散傅利葉反轉換則可以將信號由頻譜轉換為時域圖，這一點可從傅利葉轉換的積分式來看，在積分式中時間的數值及單位，由於代入積分的上下值而消去，因此即變成了頻率單位，而傅利葉反轉換則反是。

以下是我們製作傅利葉轉換的程式虛擬碼：

```
-----  
Define Frequency 44100  
FOR M=0 to Frequency {  
    SUM = 0;  
    FOR N = 0 to Frequency {  
        ARG = 2*PI*N*M/Frequency;  
        C = Complex (cos(arg), -sin(arg));
```

```

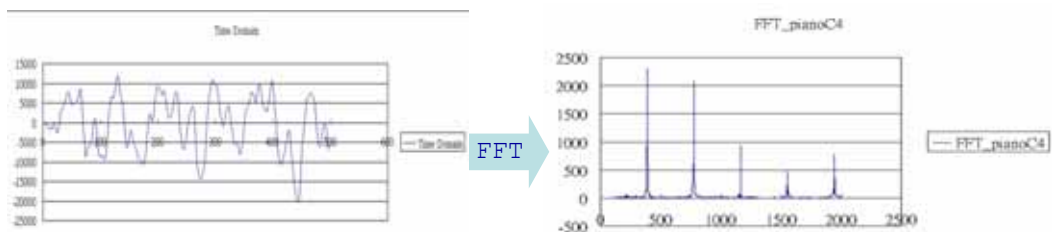
SUM += ARRAY[N]*C;
}
SUM*= DELTA_T;
OUTPUT SUM.real() , SUM.imag();
}

```

[程式碼 3. DFT (離散式傅利葉轉換)]

〈二〉快速傅利葉轉換 (Fast Fourier Transform; FFT)

傳統的DFT 做轉換時，處理速度過慢，但是若使用快速傅利葉轉換 (FFT) 便可節省許多時間。由於不斷有各種新的快速傅利葉轉換的計算方式被開發出來，目前快速傅利葉轉換的樣本個數已不限只是 2^n 個點，例如在 MATLAB 中，其快速傅利葉轉換是求出最接近樣本個數的 2^n 個做快速傅利葉轉換，其他則套入別種計算方式分別求出再與前者合併，而若樣本個數是 2^n 個點的話，花費的計算時間將節省許多。此處不詳述FFT的原理，詳細說明可參考[參考資料 2]



[圖 9. 鋼琴波經由傅利葉轉頻譜圖]

三、音色辨識[參考資料 8,9]

我們首先收集了幾種不同音色的聲音檔案，經過傅利葉轉換後求出各音的頻譜，然後加以分析比較，找出不同音色頻譜的差異性。

〈一〉我們發現，不同的音色其波形和頻譜變化各有其規律：

1. Attack Time

不同音色的聲音其 Attack 一段達到最高點的時間不盡相同，我們可以利用偵測其 Attack 一段所佔的時間來分析其為何種音色。

例如鋼琴是用槌子敲擊弦發出聲音，因此會很快達到最高峰，笛子則較為平緩地上升，而由其斜率(最高點對到起始點的線段)，我們可以找出各種我們找出各種不同樂器在此點上特色的不同。(詳見程式碼 4)

2. 頻譜各泛音的能量變化

不同音色的頻譜，其各個泛音的振幅大小變化不同，而且相同的樂器其各泛音的能量變化有一定的規律，因此靠著比對各泛音的能量變化也可以判斷音

色的不同。

在此，我們計算各個泛音對基頻的比值，並以此得到各種樂器在此項特質中的特色，將此設為一個陣列。當讀入一段聲音時，我們運用內積的方法，將第一筆資料與第二筆資料分別對應的各點作內積相乘，內積值越大則相關程度越高，以此來判斷該音跟每個紀錄特色的陣列的差異度，從而得知跟哪個樂器的音色在此項特質上最為相近。(詳見程式碼 4)

3. 過零率

過零率 (Zero Crossing Rate) 是在每個音框中，音訊通過零點的次數。一般而言，雜訊的過零率大於氣音的過零率，而氣音的過零率又大於有聲音的過零率。通常用在端點偵測，估測氣音的啓始位置及結束位置。可用來預估訊號的基頻，但很容易出錯，所以必須先進行前處理。

4. 包迹變化

所謂的包迹就是波型圖的描邊，在此我們可以看到振幅的強弱變化，例如敲擊的樂器會很快達到高點後下降(如鋼琴)，而吹奏樂器則較為平穩(如笛子)，我們可藉由此變化來判斷各種樂器的特色。

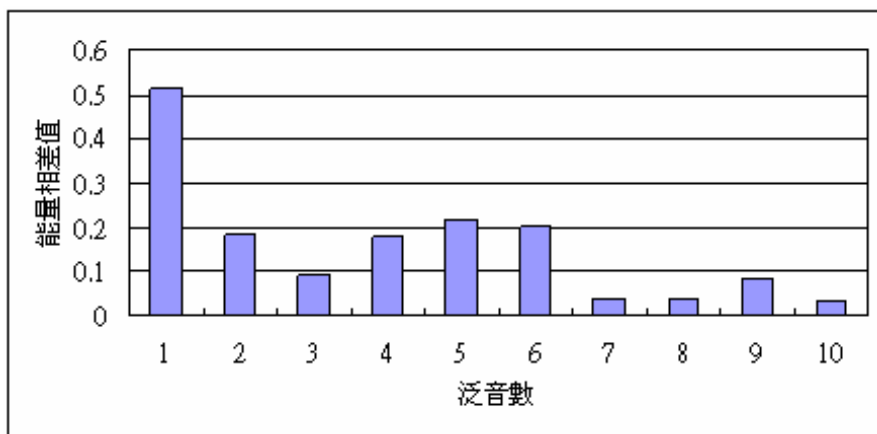
〈二〉在音色分析的方面，我們取得了許多聲音資料做分析：

1. 鋼琴

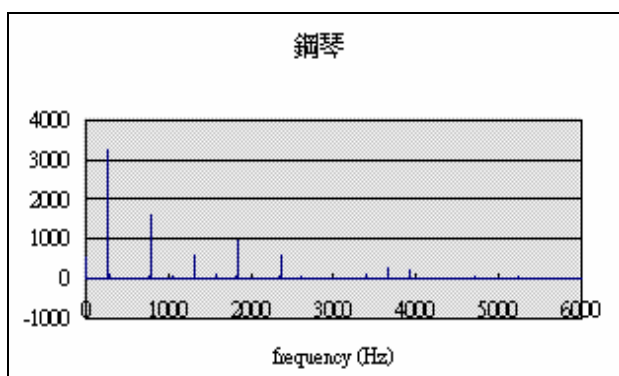
我們轉換出鋼琴的頻譜，並且測得其泛音間比例的關係(如表 2)

Timbre	種類	第一泛音	第二泛音	第三泛音	第四泛音	第五泛音	第六泛音	第七泛音	第八泛音
Piano C4	2638.48	1350.19	486.105	242.44	467.129	577.992	535.752	97.3417	97.2338
	0.51173	0.184237	0.091886	0.177045	0.219062	0.203053	0.036893	0.036852	0.082084
Piano D4	2021.82	1334.71	467.797	269.199	382.278	519.93	463.966	76.9417	84.2022
	0.660153	0.231374	0.133147	0.189076	0.257159	0.229479	0.038056	0.041647	0.110603
Piano E4	2086.56	1125.86	445.247	194.595	391.32	468.171	438.321	87.1115	76.7479
	0.539577	0.213388	0.093261	0.187543	0.224375	0.210069	0.041749	0.036782	0.094401
Piano A4	2788.07	1766.5	602.865	307.112	614.817	573.155	674.657	96.8929	112.664
	0.633592	0.21623	0.110152	0.220517	0.205574	0.24198	0.034753	0.040409	0.102011
Piano C5	2638.48	1350.19	486.093	242.428	467.142	577.99	525.749	97.3327	97.2466
	0.51173	0.184232	0.091882	0.17705	0.219062	0.199262	0.03689	0.036857	0.09054

[表 2. 鋼琴各泛音間能量相差表]

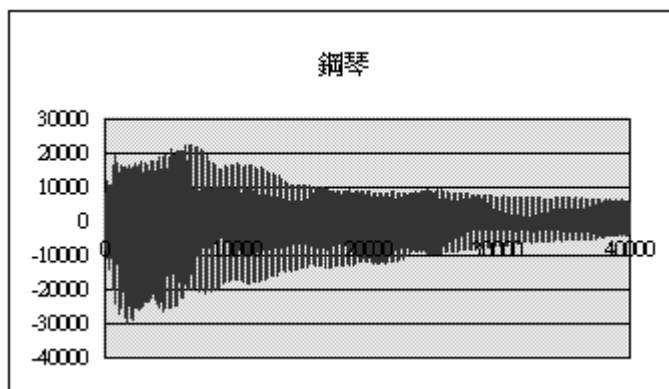


[圖 10. 鋼琴各泛音間能量相差圖]



[圖 11. 鋼琴的頻譜圖]

於是我們得出一個鋼琴泛音的特色陣列，也就是我們程式碼 4 中的 piano[]，另外我們也觀察了鋼琴的波形：



[圖 12. 鋼琴的波形圖]

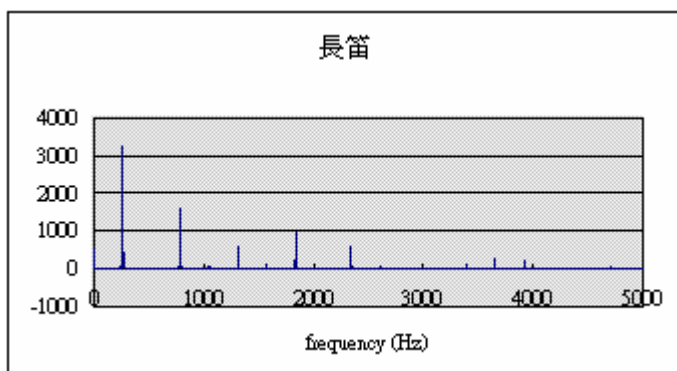
我們可以發現鋼琴由於是用槌子敲擊的關係，整個波形呈現快速上升的狀態，使得 attack time 較短，而達到最高峰的斜率的絕對值較大，根據我們觀察的結果，我們亦得到幾組數據來代表鋼琴，詳見附錄程式碼部分。

2. 長笛

下表是長笛各泛音的能量值：

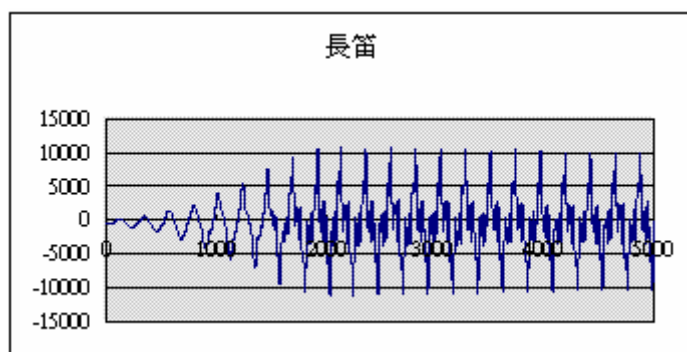
Timbre	積頻	第一泛音	第二泛音	第三泛音	第四泛音	第五泛音	第六泛音
Flute C4#	10852.2	1511.57	2243.93	690.731	421.198	11.4846	37.418
	0.139287	0.206772	0.063649	0.038812	0.001058	0.003448	
Flute A4#	1114.45	4679.39	1208.7	166.516	87.2852	19.1161	44.037
	4.198834	1.084571	0.149415	0.078321	0.017153	0.039515	
Flute A5#	9590.09	613.757	246.882	16.1076
	0.063999	0.025743	0.00168				

[表 3. 長笛各泛音間能量相差表]



[圖 13.長笛的頻譜圖]

於是我們也得到了一個長笛的特色陣列，也就是我們程式碼 4 中的 `flute[]`，另外我們也觀察了長笛的波形：



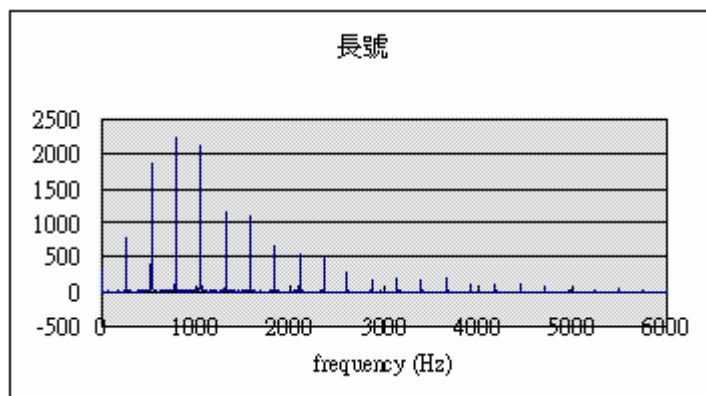
[圖 14.長笛的波形圖]

我們可以發現它的波型和鋼琴有顯著的不同，除了斜率較為和緩外，它的 `attack time` 也比較長，同樣地，我們把這些數據紀錄之後做平均然後用於分析。

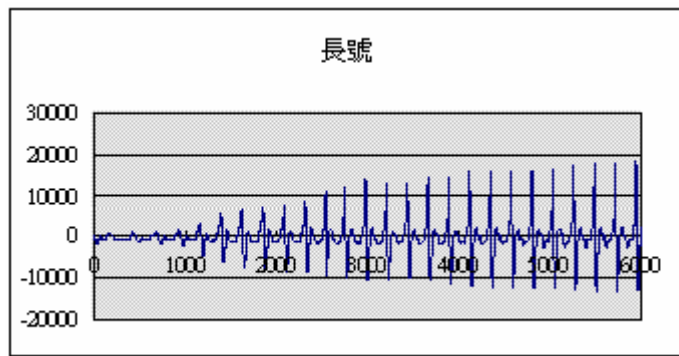
3. 小提琴及其他

至於其他樂器，我們同樣也觀察了它們的波形，並且得出斜率的關係，但是泛音的部分，因為有的樂器差異太大，或是泛音的能量跟基頻相差太大，有的泛音能量甚至都在 1% 以下，太小而不具有顯著意義，因此我們將這些聲音到最高峰的斜率絕對值整理並且紀錄，在程式中一起比對，但是並沒有加入它們的泛音特色。

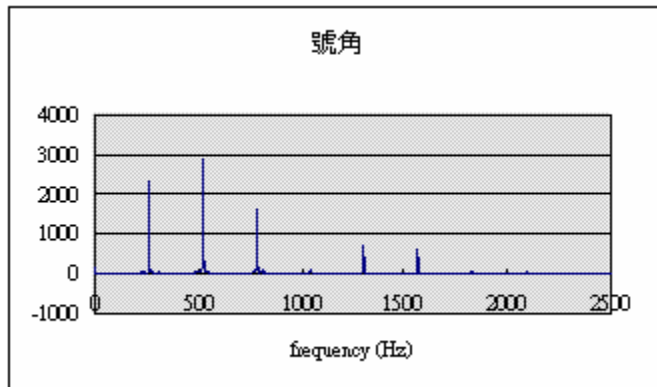
以下是我們對其他樂器所做的波形及頻譜分析：



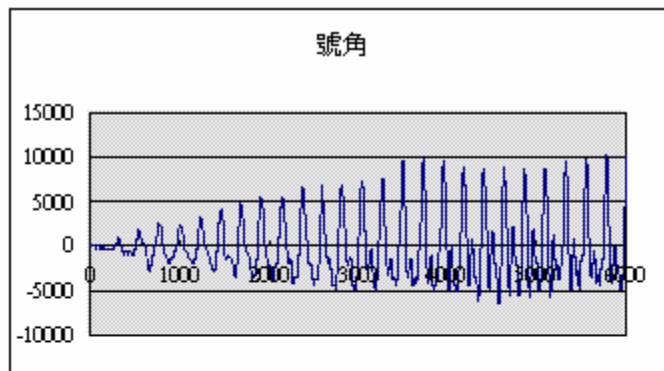
[圖 15.長號的頻譜圖]



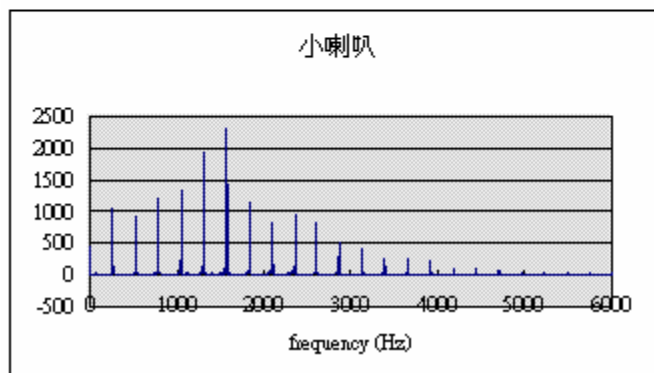
[圖 16.長號的波形圖]



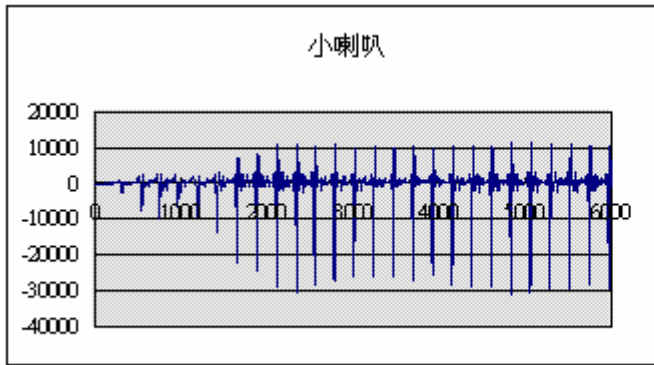
[圖 17.號角的頻譜圖]



[圖 18.號角的波形圖]

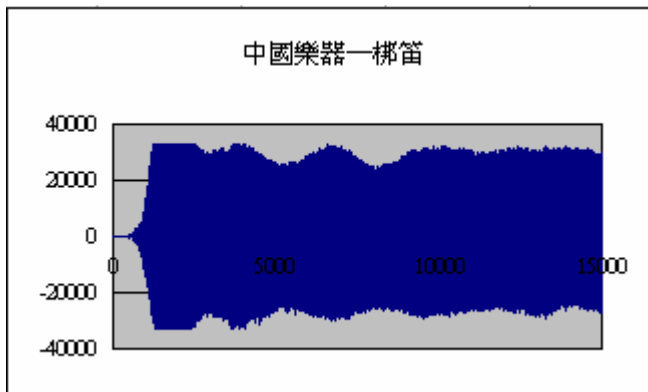


[圖 19.小喇叭的頻譜圖]

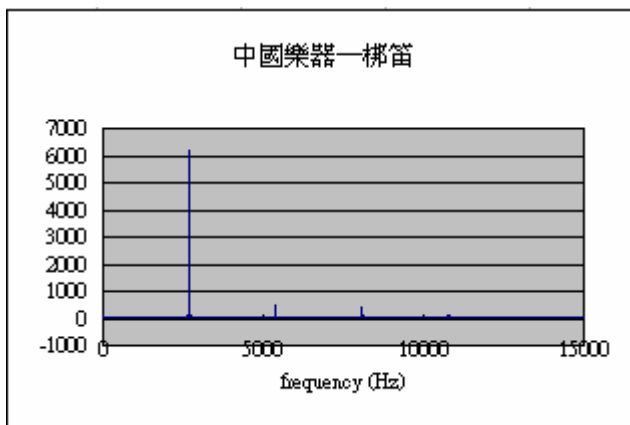


[圖 20.小喇叭的波形圖]

4. 國樂的梆笛



[圖 22.梆笛波形圖]

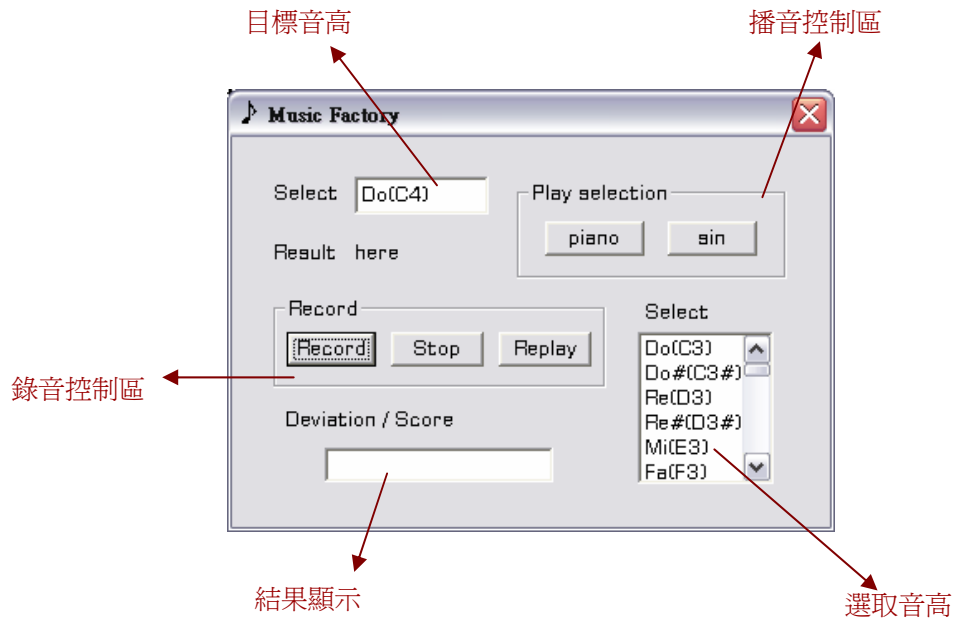


[圖 21.梆笛頻譜圖]

另外我們也用中國的樂器—梆笛做了些分析，主要在於比較其與西洋長笛的不同之處。梆笛的材料是竹子。我們觀察其波形及頻譜後發現，笛子所測出的音比實際上吹奏的音還要高，波形跟長笛不同，長笛是緩緩的橢圓形，此則是迅速地升高(及 Attack time 很短)，雖然其中有些音波形不穩定(中間的音強凹凸不同)，但影響音高不大，就波形來看，比較像鋼琴(因為現象 3)。

伍、研究結果

一、單音辨識程式



[圖 23. 單音辨識程式介面]

圖 23 是我們的程式使用介面，以下說明每個按鈕的功能：

列式盒(select)：選取音高

錄音控制區

Record：開始錄音

Stop：停止錄音(聲音輸入完成)

Replay：聽剛剛自己唱的聲音

播音控制區

piano：播放目標音高之鋼琴音

sin：播放目標音高之正弦波音

顯示資訊一覽

左上角的 Select：目標音高



左下方的 Deviation：使用者聲音跟目標音高的差距

二、秀譜程式



[圖 24. 單音辨識與秀譜程式介面]

此程式的使用流程：

首先選擇一段樂譜，所選擇的樂譜會顯示在下方。接著按下  按鈕，打開節拍器，〔按下  則可以停止節拍器〕程式便會按照此曲的速度規定打拍子。接著開始錄音，使用者可聽著系統節拍器所發出之規律節奏，以合乎此曲的節拍唱入聲音。唱完後按下 “stop” 按鈕，結果顯示區便會顯示剛才輸入的聲音與該曲比對後之結果。最後可按下 ”replay” 聽剛才所唱入的歌聲，或者也可選擇別首曲調來練習。

三、音色辨別程式：

```
int main() {
char *p = "piano4.wav";
ifstream fin (p, ios::binary);
//-----
// Wave Head
int head[5];
short channels[2];
int sample_rate[2];
short bytes_per[2];
int data_length[2];

// Read Wave Head
fin.read(reinterpret_cast<char *>(head),5*sizeof(int));
fin.read(reinterpret_cast<char *>(channels),2*sizeof(short));
fin.read(reinterpret_cast<char *>(sample_rate),2*sizeof(int));
fin.read(reinterpret_cast<char *>(bytes_per),2*sizeof(short));
fin.read(reinterpret_cast<char *>(data_length),2*sizeof(int));

// Read Datas
short *data = new short[data_length[1]];
fin.read(reinterpret_cast<char *>(data),data_length[1]*sizeof(short));
fin.close();
//-----

const int HIGH = 1;
const int LIMIT = 10;
int i, mag, old, index, max, first = 1, len;
int flag = 0;

index = max = mag = 0;
```

```

for (i=0;i<data_length[1];i++)
    if (abs(data[i])>=max) {
        max = abs(data[i]);
        len = i;    }

// Find the alp
for (i=0;i<len;i++) {
    old = mag;
    mag = abs(data[i]);
    if (old && mag/old>HIGH)
        flag = i;
        if (flag && mag && old/mag>HIGH && old>1000) {
            first = i;
            break;    }
        if (i>flag) flag = 0;    }

len -= first;
double slope = (double)max/(double)len;
//-----
int leng;

for (i=0;i<data_length[1];i++)
    if ((int)pow(2.0,(double)i)>data_length[1])
        break;
leng = pow(2.0, (double)(i-1));

Complex* fftarr = new Complex[leng];

for (i=0;i<leng;i++)
    fftarr[i] = (double)data[i]/32767.0;

FFT::transform(fftarr, leng);

int alps[10], where = 0;
const int ALPS = 1;
bool f = 0;
double fea[10];

for (i=mag=0;i<leng-1;i++) {

```

```

    old = mag;
    mag = abs(fftarr[i]);
    // find alps
    if (i&&!f) {
        if (old&&mag/old>ALPS)
            f = true;
    }
    if (f) {
        if (mag&&old/mag>ALPS)
            alps[where++] = mag;
        f = !f;
    }
    // 最多取十個高峰。
    if (where>9) break;
}

for (i=1;i<where;i++) {
    fea[i-1] = alps[i]/alps[0];
    //cout << fea[i-1] << endl;
}

const double flute[] = { 4.198833505, 1.084570865,
0.149415407, 0.078321324, 0.017152945, 0.039514559 };
const double piano[] = { 0.213388065, 0.093261157,
0.187543133, 0.224374569, 0.210068726, 0.041748859,
0.036782024, 0.094401311, 0.035835442 };
const double flen = 4.34014;
const double plen = 0.444037;
double dot1, dot2;
int loop;
loop=(where>6)?6:where;

// 取內積
for (i=0,dot1=0.0;i<loop;i++)
    dot1 += flute[i]*fea[i];
dot1/=flen;
loop=(where>9)?9:where;

for (i=0,dot2=0.0;i<loop;i++)
    dot2 += piano[i]*fea[i];
dot2/=plen;

```

```

//cout << dot1<< " " << dot2 << endl;

cout << "Judge by spectrum : ";
if (dot1<=dot2) cout << "piano" << endl;
else cout << "flute" << endl;

//-----

const int attack[7] = { 762, 48890, 51064, 47875, 48373,
3626, 46955};
const char intru[7][20] = {"piano", "flute", "clarinet",
"trombone", "horn", "trumpet", "violin"};
int dev, dit, intrument;
    for (i=0;i<7;i++) {
        dit = abs(attack[i]-len);
        if (dev>dit) {
            dev = dit;
            intrument = i;    }    }
cout << "Judge by attack time : ";
cout << intru[intrument] << endl;
//-----

fstream fout("record.txt",ios::outlios::app);
fout << p << endl;
fout << slope << endl;
double similar[7];
double feature[7] = {34, 0.625, 0.6, 0.64, 0.63, 8.53, 0.65};
double ans = 10.0, tmp;
for (i=0;i<7;i++) {
    tmp = fabs(feature[i]-slope);
    if (tmp<ans) {
        ans = tmp;
        intrument = i;    }    }
cout << "Judge by slope : ";
cout << intru[intrument] << endl;

cin.get();
return 0;    }

```

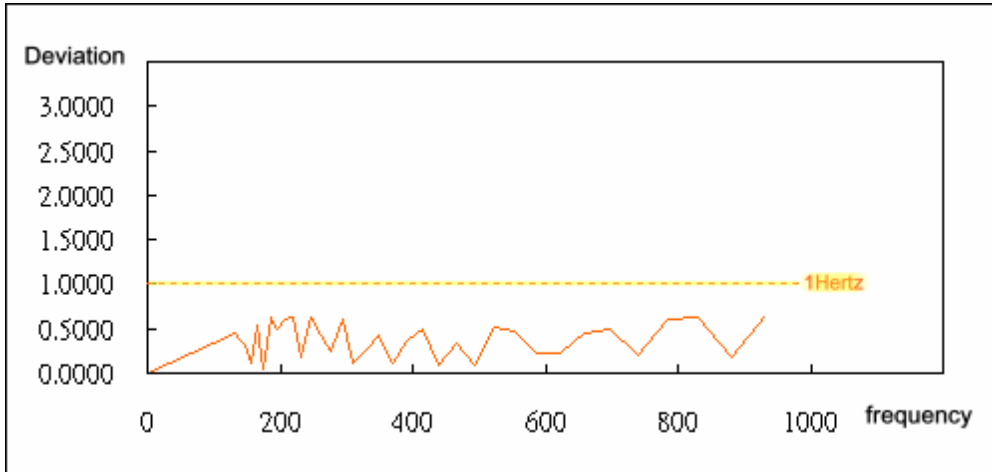
[程式碼 4. 音色辨別程式]

陸、討論

一、音高辨識實驗

爲了了解我們的程式做音高辨識的準確度，我們對不同音高的聲音做了辨識測試，並紀錄其誤差值。

圖 25 爲比較各對頻率做傅利葉轉換後所得之結果與原音頻之誤差：



[圖 25. 誤差比較]

理論上，越高頻率的聲音，其音高辨識越能準確，這是因爲音高的頻率是依據十二平均率劃分，音調越高的聲音其音頻值彼此之間差距越大，較容易判斷準確；低頻率的音高辨識其聲音訊號彼此之間的差異值很小，辨識時很容易因爲 1~2Hz 的誤差而落在鄰近音高的範圍內，而辨識不準確。

由上圖可看出，誤差範圍都小於 1Hertz，所以整個轉換的過程是很準確的。

二、不同音色對音高辨識準確度之影響

目前我們是利用傅利葉轉換將聲音訊號轉換成頻譜的方式來做音高辨識，但是在某些震動方式較特殊樂器如號角、喇叭等，其頻譜中能量高峰處卻不是其基頻，下頁表爲對各種樂器的 C4 這個音作音高辨識的誤差表：

Pitch Judging on Different Tone Colors		
Tone Color	Result	Deviation
sine wave	261.76 Hz	0.13 Hz
piano	261.09 Hz	0.54 Hz
clarinet	261.08 Hz	0.55 Hz
horn	522.19 Hz	260.56 Hz
violin	523.53 Hz	261.9 Hz
trombone	784.62 Hz	522.99 Hz
oboe	1305.45 Hz	1043.82 Hz
trumpet	1567.89 Hz	1306.26 Hz

[表 4. 不同音色做轉換之誤差表]

由上圖可以看出，有些樂器的聲音用傅立葉轉換來辨識其音高並不準確，因此我們也對不同音色聲音的頻譜和波形做進一步的觀察。

三、音色辨識實驗

下表(表 5)是我們對各種音色辨識結果所做的 Confusion Matrix：

	鋼琴	小提琴	長笛	豎笛
鋼琴	77%	1%	0%	22%
小提琴	33%	13%	0%	53%
長笛	29%	0%	71%	0%
豎笛	33%	0%	0%	67%

經由測試結果，我們可以發現鋼琴音的特徵較為明顯，而比較不容易跟其他兩者混淆。從程式中的三項特徵來看，鋼琴的波形具有快速上升的特性，跟笛子和小提琴的緩慢上升不同，所以在斜率的判定上，不容易誤判，而 attack time 也是一樣，因為鋼琴的 attack time 比笛子跟小提琴都小，也較不容易誤判；所以鋼琴音的準確度較高。

至於笛子跟小提琴則因為都是緩慢上升，使得斜率的判定較容易混淆，且 attack time 也比較難以分辨出究竟是哪一種樂器，只能依靠泛音的頻率比，因此使得這兩種聲音容易彼此誤判。而國樂的梆笛跟長笛更相似，僅在泛音頻率比部分有所不同：國樂的泛音在高頻率的部分比長笛能量要高，但二者的差異並沒有非常顯著，也使得要判定屬於何者較為困難。

柒、結論

目前為止，我們的音高辨識程式所能做到的功能有：錄音、播放該音之標準音(鋼琴及正弦波)，判斷該音之頻率並比對、輸出誤差率，以及樂譜的讀取與秀譜，節拍器的設定等。這對於一般要尋求起始音準的人而言已是相當便利，而我們整個程式經由包裝後，僅佔 124KB，若再包含鋼琴音的資料，整個程式也不到 1MB。可以說的確有做到我們一開始所提及的輕薄短小且方便使用。

在研究的過程中，我們發現所使用的演算法並不是對所有音色的聲音都能有效辨識其音高，也發現相同的聲音，其頻譜的分佈狀況大致上有固定的比例，因此我們將來希望針對不同音色，或針對單一樂器的聲音資訊做分析，找出其發聲原理與頻譜的相關性，以及比較不同音色的頻譜，找出各個音色的特徵，以對聲音的各方面特性有更進一步的了解。

目前我們的音色程式能夠從：泛音頻率比、attack time 以及到最高峰的斜率來判定不同音色的樂器，而目前我們收集資訊較為完備的音色特徵有：鋼琴、小提琴、長笛跟國樂的梆笛，其中前三者是用軟體產生，最後一種用錄音程式錄製。而經過觀察音色特徵後，我們發現許多樂器的最高峰未必是他的基頻，而找尋第一個高峰是較為準確的方法，於是我們同時改良了之前的音高辨識程式，使其能夠達到更高的準確度。

捌、參考資料及其他

- 一、李思毅 (1996)。音高辨識與 MIDI 及 Wave 之轉換與應用。中山大學碩士論文。
- 二、曾建誠、陳常侃 (2000)。離散時間訊號處理第二版。全華科技。
- 三、http://www.ringthis.com/dev/wave_format.htm
- 四、<http://www.sfu.ca/sonic-studio/handbook/index.html>
- 五、<http://cslin.auto.fcu.edu.tw/scteach/lego/mickya.htm>
- 六、<http://www.math.ncu.edu.tw/~guo/Games/menu2.html>
- 七、<http://wwwme.chit.edu.tw/~rthong/course>
- 八、<http://www.rolandtaiwan.com.tw/MI/glossary/e.htm>
- 九、<http://neural.cs.nthu.edu.tw/jang/>