Parallel Computational Acoustics Library - Audio Functions Developper's Reference Manual

Frédéric Magoulès, Peter Ivànyi

Parallel Computational Acoustics Library
Audio Functions Developper's Reference Manual
by F. Magoulès and P. Iványi

June 13, 2002

Contents

1 Introduction

The *PCA-Lib Audio Functions Developper's Reference Manual*, is the manual for the PCA-Lib functions library and defines the PCA-Lib audio file format. This work was motivated by the need to integrate new tools for the numerical simulations for acoustics problems which will be more easily perceptible than by a simple analysis by means of a traditional graphic interface.

Indeed, when considering acoustic phenomena in frequential mode, it is very current to visualize the acoustic pressure (in Pascal or in decibel) in the form of zones of colors. The zones of red color, for example, correspond to a high noise level, those of color blue, for example, on a lower noise level. In the same way, when considering acoustic phenomena in temporal mode, it is current to visualize the evolution of the acoustic pressure in a point given in the shape of two-dimensional curves, which represent the amplitude of the pressure according to time. Still here, the amplitude of the curve makes it possible to deduce the associated amplitude and the distance between two peaks makes it possible to know if the sound is in a low or high frequency. The research undertaken in acoustics recently established that the perception of a sound by a person is a strongly subjective phenomena. Thus the gene caused by a sound is closely related to the idea that the person associates with this sound. Squeaking, even light (i.e. of low amplitude) of a chalk on a table in is a perfect example. So, the analysis of the amplitude and of the frequency of sound, even if they make it possible to know if this sound is dangerous or not for the tympanums (amplitude higher than 90 dB), in general does not make it possible to know if this sound will be perceived in "pleasant" or "unpleasant" way by a person. By having audio functions, we thus hope in the future to improve the numerical methods which we in addition develop in acoustics (like optimization of form or absorbing materials), not only on one reduction of the amplitude of the noise level, but also on a subjective perception of the sounds. A person will be able to then choose the shape or the absorbing material which gets a "pleasant" sound to him.

In this book, the PCA-Lib file format is detailled and a complete analysis of the WAV file format initialy developped by Microsoft is performed. The methodology to be followed in order to program a basic sound interface in a code of numerical simulation is presented. Then, a module in language C++ is provided, and the WAV file generated by the execution of this program is analized and commented on. Then the PCA-Lib library is presented with the description of the library functions.

2 The PCA-Lib Files Format

The PCA-Lib format supports audio signal of real and/or integer type. The PCA-Lib file format takes the form of a series of keywords followed by one or more data items. For example,

```
TIME_IN_SECOND 2
```

denotes that the signal has a total duration of 2 seconds.

`IMPORTANT` All keywords are mandatory, and must be in a strict, predefined order. This order automatically accounts for dependencies between the data.

The format also allows the use of file comments. If the first non-white character on a line is the # character, then all remaining text on that line is ignored. Blank lines are also permitted.

There are currently two types of data file – the free format audio (**.aud**), the WAV format audio (**.wav**). The free format file contains all the basic description of the signal and the WAV format file correspond to a Microsoft WAV file format and can be listen on all computer under Microsoft Windows systems.

The following sections describe the keywords and keyword data for the above data files. Each section contains a table which lists the keywords in the order they must be declared and defines the associated keyword data. The data type – whether it is an integer (i), real (r) or a character string (s) – is also given, where, for example, 'i 3*r' denotes that the data consist of an integer followed by three reals. Units, where relevant, are enclosed in square brackets ([...]).

IMPORTANT Keyword data consisting of a single data item must follow the keyword on the same line. For vectors and matrices, each vector component must start on a new line and each line must begin with an integer index. Unless otherwise stated, this index is either the vector component number or matrix row index. All keyword data strings must be enclosed in double quotes.

3 The Free Format Audio File

The free format audio file (**.aud**) contains the description of the audio signal.

An example free format audio file is shown in the following

```
# An example audio file

TITLE "An example signal"
TIME_IN_SECOND 2
SAMPLE_PER_SECOND 44100
DATA_CODING_IN_BITS 16
NUMB_CHANNEL 2
MAXIMUM_AMPLITUDE 110.0

LEFT_LOUDSPEAKER
  1 15.156
  2 16.189
  3 18.954
  ...
  88200 14.567

RIGHT_LOUDSPEAKER
  1 15.156
  2 16.189
  3 18.954
  ...
  88200 14.567
```

The first keyword **TITLE**, is compulsory and specifies the title of the signal. The title can be used to annotate output such as the display in a audio program and, as with all string arguments, must be enclosed in double quotes.

An audio signal is defined in term of a set of value which are linked by some relations in order to represent correctely the discretisation of the continuous associated signal, i.e. the time in second, the number of samples per second, etc.

At present all the basic informations described Table 1 are needed in this file. The **TIME_IN_SECOND** is the keyword for the total time of the signal in second, and **SAMPLE_PER_SECOND** defines the number of samples per second. The

keyword **DATA_CODING_IN_BITS** indicates the type of coding of the signal and **NUMB_CHANNEL** defines the number of channel. The keyword **MAXIMUM_AMPLITUDE** is the maximum absolute value taken by the signal in the file. **LEFT_LOUDSPEAKER** and **RIGHT_LOUDSPEAKER** are two vectors of dimension **TIME_IN_SECOND * SAMPLE_PER_SECOND** and contains all the value of the signal.

The different types of these keyword are described Table 1.

Keyword	Keyword Data	Data Type
TIME_IN_SECOND	Time in second	i
SAMPLE_PER_SECOND	Number of Samples per second	i
DATA_CODING_IN_BITS	Type of formatting	i
NUMB_CHANNEL	Number of channel for mono or stereo recording	i
MAXIMUM_AMPLITUDE	Maximum amplitude of the signal	r
RIGHT_LOUDSPEAKER	Value whiwh are sent to the left loudspeaker	i 1*r
LEFT_LOUDSPEAKER	Value whiwh are sent to the left loudspeaker	i 1*r

Table 1: The free format audio file keywords and keyword data

The example free format audio file presented before represent a stereo recording, with a duration of two seconds, using 44100 samples per second.

4 The WAV Format Audio File

The audio formated file (**.wav**) contains the digital signal as stored in a WAV file format. The results of this format can be listen in the loudspeakers of all computer using a Microsoft Windows system or an equivalent emulator.

This file can be open trough an hexadecimal editor, but the description of the format is rather difficult to understand without a full analysis. This format is presented and commented in the following chapter.

4.1 Description of the Format

The WAV files are probably the simplest of the common formats for storing audio samples. Contrary to other type of files which are compressed, files WAV store the data "in the raw", without any different operation than the formatting of these data (see for example [Web 00], [Gun 95], [Dir 00], [Mag 01]).

A WAV file is divided into three principal blocks of information: the RIFF block which identifies the file like a WAV file; the FORMAT block which identifies the parameters of the file as sample rates; the DATA block which contains the data themselves. Each block is composed of a series of Bytes (1 Byte = 8 Bits, 1 Bit = a primary data) formatted according to methodology indicated Table 2.1, 2.2 and 2.3.

Byte Number	Data Stored	Description	Formatting
0 - 3	"RIFF"	Beginning of RIFF Block	ASCII Characters
4 - 7	—	Total length of package (file) to follow	Binary 'little endian'
8 - 11	"WAVE"	Identify the file as a WAV file	ASCII Characters

Table 2: RIFF Block (total length 12 Bytes)

Byte Number	Data Stored	Description	Formatting
0 - 3	"fmt "	Beginning of FORMAT Block	ASCII Characters
4 - 7	0x10	Length of FORMAT Block	Binary 'little endian'
8 - 9	0x01	Microsoft format	Binary 'little endian'
10 - 11	0x01 (= mono) 0x02 (= stereo)	Number of channels	Binary 'little endian'
12 - 15	—	Number of sample per second (Hz)	Binary 'little endian'
16 - 19	—	Number of Bytes per second	Binary 'little endian'
20 - 21	1 (= 8 Bit mono) 2 (= 8 Bit stereo) 2 (= 16 Bit mono) 4 (= 16 Bit stereo)	Number of Bytes per samples	Binary 'little endian'
22 - 23	—	Number of Bits per samples	Binary 'little endian'

Table 3: FORMAT Block (total length 24 Bytes)

4.2 Analysis of a Simple WAV File

The simplest approach to study a format WAV consists in looking at how the data are stored in the file. For this purpose, we look at the file DING.WAV which is a standard file, provided with all the implementations of Microsoft Windows. The edition of the file by means of a hexadecimal editor reveals on each line the values of the stored Bytes.

```
52 49 46 46 D0 3B 01 00   57 41 56 45 66 6D 74 20
10 00 00 00 01 00 02 00   22 56 00 00 88 58 01 00
04 00 10 00 64 61 74 61   7C 3B 01 00 F8 FF 01 00
FF FF FE FF 05 00 02 00   05 00 03 00 03 00 FD FF
...
```

- **Line 1 (total length 16 Bytes)**

As indicated in the preceding section, the file begins with the **four Bytes** corresponding to ASCII Characters "RIFF". Follow **four Bytes** indicating the total length in Bytes of the continuation of the file (integer value 80848), i.e. the size of the file on the disc (integer value 80856) minus the size of word "RIFF" (integer value 8). The next **eight Bytes** correspond to ASCII Characters "WAVE", and to the ASCII Characters "fmt ".

- **Line 2 (total length 16 Bytes)**

Byte Number	Data Stored	Description	Formatting
0 - 3	"data"	Beginning of DATA Block	ASCII Characters
4 - 7	—	Length of data to follow	Binary 'little endian'
8 - end	—	Data samples	Binary 'little endian'

Table 4: DATA Block

The **four Bytes** represent the length of the block FORMAT (integer value 16), followed by **two Bytes** indicating that it is a Microsoft file (integer value 1). The next **two Bytes** show that it is about a stereo recording (integer value 2). The next **four Bytes** indicate the number of samples per second (integer value 22050). The next **four Bytes** indicate the number of Bytes per second (integer value 88200) since we have (88200 = 22050 * 4), where 4 represents the number of Bytes per sample.

- **Line 3 - end**

The next **two Bytes** represent the number of Bytes per sample (integer value 4), since we have a 16 Bits stereo recording. The next **two Bytes** represent the number of Bits per sample, (integer value 16) and the next **four Bytes** correspond to the ASCII Characters "data". Come then, **four Bytes** indicating the number of Bytes of the data which follow (integer value 80764). The **following Bytes** which extend until the end from the file correspond to the coding of the samples.

4.3 Programmation of an Audio Function

In this section, we present a demonstration program to store two vectors in a WAV file format. The C++ program is tiny room to its bare minimum, i.e. a function *long main ()* which calls another function *void Demo ()*. The last one initializes two vectors *lhs* and *rhs* which respectively contains the signal which will be sent in the left loudspeaker (resp. in the right). The handling of files requires the inclusion of two files heading of the standard bookshop to knowing the files *iostream.h* and *fstream.h* . The function *void Demo ()* opens a file **sound.wav** in binary mode, and the data (made up here of the two vectors *lhs* and *rhs*) are formatted in format WAV describes with the preceding section. Moreover fuller details on the instructions used and handling of the files can be consulted in [Ker 92], [Str 99], [Fer 94].

The file **sound.wav** obtained after execution, is a stereo audio recording file, coded on 16 Bits, with a total duration of 2 seconds, and composed of 44100 samples per second. The aural beep sent in the right loudspeaker corresponds to the musical note LA of frequency 220 Hz and the signal sent in the left loudspeaker to the musical note LA of frequency 440 Hz.

```
#include <iostream.h>       // standard stream
#include <fstream.h>        // file stream

// ************************< Demo >**********************************
// *  Purpose : demonstration code for WAV file                      *
// *  Warning : to be used on a processor using little endian format *
// *            (INTEL or RISC)                                      *
// *  Details : if the processor use little endian format, this      *
// *            function run without bug, even when the C++ compiler *
// *            do not respect the ANSI C++ norm for the size of the *
// *            basic type (char, short, int, long)                  *
// ******************************************************************
void Demo ( )
  {
  long i;
  long n;
  double tmp;
  short sval;

  // WAV parameters initialization
  long time_in_second = 2;
  long sample_per_second = 44100;
  long data_coding_in_bits = 16;
  long numb_channel = 2;
  double maximum_amplitude = 110;

  // datas initialization
  long size;
  double * lhs;
  double * rhs;

  size = time_in_second * sample_per_second;
  lhs = new double [size];
  rhs = new double [size];
  for (i = 0; i < size; i++)
    {
    lhs[i] = 110 * sin(i*2*3.1415927*220/44100.);
    rhs[i] = 110 * sin(i*2*3.1415927*440/44100.);
    }

  // open file
  ofstream output("sound.wav", ios::out | ios::binary);

  //  RIFF chunk
  output.write("RIFF",4);
  n = (data_coding_in_bits/8);
  n *= numb_channel*time_in_second*sample_per_second;
  n += 44 - 8;
  output.write((char *)&n,4);
  output.write("WAVE",4);
```

Table 5: (a) The **demonstration code**

```
  //  FORMAT chunk
  output.write("fmt ",4);
  n = 16;
  output.write((char *)&n,4);
  n = 1;
  output.write((char *)&n,2);
  n = numb_channel;
  output.write((char *)&n,2);
  n = sample_per_second;
  output.write((char *)&n,4);
  n = (data_coding_in_bits/8)*numb_channel*sample_per_second;
  output.write((char *)&n,4);
  n = (data_coding_in_bits/8)*numb_channel;
  output.write((char *)&n,2);
  n = data_coding_in_bits;
  output.write((char *)&n,2);

  //  DATA chunk
  output.write("data",4);
  n = (data_coding_in_bits/8);
  n *= numb_channel*time_in_second*sample_per_second;
  output.write((char *)&n,4);
  for (i = 0; i < time_in_second*sample_per_second; i++)
    {
    tmp = lhs[i];
    sval = short ( short(tmp) * 32767/maximum_amplitude );
    output.write((char *)&sval,(data_coding_in_bits/8));

    if (numb_channel == 2)
      {
      tmp = rhs[i];
      sval = short ( short(tmp) * 32767/maximum_amplitude );
      output.write((char *)&sval,(data_coding_in_bits/8));
      }
    }

  // close file
  output.close();
  }

// ************************< main >**********************************
// *                                                                  *
// *                                                                  *
// ********************************************************************
long main ( )
  {
  Demo();
  return 1;
  }
```

Table 6: (b) The **demonstration code**

4.4 Analysis of a General WAV File

In this section we analyze file **sound.wav** obtained by the preceding program. The edition of the file by means of a hexadecimal editor reveals on each line the values of the stored Bytes.

```
52 49 46 46 44 62 05 00   57 41 56 45 66 6D 74 20
10 00 00 00 01 00 02 00   44 AC 00 00 10 B1 02 00
04 00 10 00 64 61 74 61   20 62 05 00 00 00 00 00
7D 03 7D 03 FB 06 FB 06   A2 0B A2 0B 20 0F 20 0F
...
```

- **Line 1 (total length 16 Bytes)**

As envisaged, the file begins with the **four Bytes** which identify the file as a Microsoft file, and which correspond to ASCII Characters "RIFF". The next **four Bytes** indicate the total length in Bytes of the continuation of the file (integer value 352836), since (352836 = 352844 - 8) where the size of the file on the disc is equal to 352844 Bytes, and the size of the word "RIFF" is equal to 8 Bytes. Appear then the **four Bytes** which correspond to ASCII Characters "WAVE", followed **four Bytes** corresponding to the ASCII Characters "fmt ".

- **Line 2 (total length 16 Bytes)**

Come then **four Bytes** representing the length from the FORMAT block (integer value 16, hexadecimal value 0x10). The next **two Bytes** indicate that it is a Microsoft file (integer value 1, hexadecimal value 0x01). The next **two Bytes** show that there are two channels i.e. stereo recording (integer value 2, hexadecimal value 0x02). The next **four Bytes** indicate the number of samples per second (integer value 44100). The **four Bytes** indicate the number of Bytes per second (integer value 176400) since we have (176400 = 44100 * 4), where 4 represents the number of Bytes per sample.

- **Line 3 - end**

The **two Bytes** (integer value 4) represent the number of Bytes per sample, since we have a 16 Bits stereo recording. The next **two Bytes** (integer value 16) represent the number of Bits per sample. Finally, appear the **four Bytes** corresponding to the ASCII Characters "data". Come then, **four Bytes** indicating the number of Bytes of the data which follow (integer value 352800) since we have (352800 = 2 * 44100 * 4) for a total of 2 seconds, with 44100 samples per second and 4 Bytes per sample). The **following Octets** which extend until the end from the file correspond to the coding of the samples.

Acknowledgements

The authors acknowledge partial financial support by the European Community under the Enhancing Access to Research Infrastructure action of the Improving Human Potential programme, contract number HPRI-1999-CT-0026.

5 References

[Dir 00] Dirand A., Houillon M., "Format de fichier WAV", Personnal Notes, 2000

[Far 00] Farhat C., Macedo A., Lesoinne M., Roux F.X., Magoulès F., Bourdonnaye A. de la, "Two-level domain decomposition methods with Lagrange multipliers for the fast iterative solution of acoustic scattering problems", Computer Methods in Applied Mechanics and Engineering, Vol.184, No.2, pp.213—240, 2000

[Fer 94] Ferraris B., "Scientific C++: Building numerical libraries - the object-oriented way", Addison Wesley, 2nd edition, 1994

[Gun 95] Gunter Born, "File Formats Handbook", ITP Boston, 1995

[Ker 92] Kernighan B.W., Ritchie D.M., "Le langage C", Masson - Prentice Hall, 2nd edition, 1992

[Mag 00] Magoulès F., Meerbergen K., Coyette J.P., "Application of a domain decomposition method with Lagrange multipliers to acoustic problems arising from the automotive industry", Journal of Computational Acoustics, Vol.8, No.3, pp.503—521, 2000

[Mag 01] Magoulès F., Calcul Scientifique et Informatique : Acoustique et Format de Fichiers WAV, Université Henri Poincaré, Research Report No.27, 11 pages, 2001

[Str 99] Stroustrup B., "Le langage C++", Campus Press Reference, 3 edition, 1999

[Web 00] "Wave file format description", www.technology.niagarac.on.ca /courses/comp630/WavFileFormat.html, 2000