Generative Art 2002

pCM (pure C Music): a real-time music language

Leonello Tarabella
computerART Project, Istituto A.Faedo ISTI/C.N.R, Area della Ricerca di Pisa
e-mail: l.tarabella@cnuce.cnr.it
http://www.cnuce.pi.cnr.it/tarabella/cART.html

Abstract

In order to put to work the facilities offered by the gesture interfaces realised at cART project of CNR, Pisa, I started writing basic libraries for processing sound and for driving the gesture interfaces. In the long run the framework became a very efficient, stable and powerful "music language" based on pure C programming, that is "pure-C-Music", or pCM. This programming environment gives the possibility to write a piece of music in terms of synthesis algorithms, score and management of data streaming from gesture interfaces. The pCM framework falls into the category of the "embedded music languages" and has been implemented using one of the most popular C compilers or better, multiplatform development systems: Metrowerks' Code Warrior. As a result a pCM composition consists of a CW project which includes all the necessary libraries, including a DSP.lib consisting of a number of functions able to implement in real-time the typical synthesis and processing elements such as oscillators, envelope shapers, filters, delays, reverbs, etc. The composition itself is a C program consisting, mainly, of the *Orchestra*() and *Score*() functions. Everything here is compiled into machine code and runs at CPU speed.

1. Introduction

The activity of cART (computerArt Project of C.N.R., Pisa) is characterised by the design and the realisation of systems and devices for gesture control of real-time computer generated music in order to give expression to interactive electro-acoustic performances[1], as it happens in traditional music. The "wireless technology" (or "touchless technology") paradigm has been taken into consideration [2]. Main targets of the research consist of the implementation of models and systems for detecting gestures of the human body that becomes the natural interface able to give feeling and expressiveness to computer based multimedia performances. The term "multi-modality" [3] is often related to these typologies of interfaces just for emphasising that combining different modes of perception, becomes relevant for the performer and for the audience that, at the end, is the final "user" of the performance.

At cART, attention has been focused in designing and developing new original general purpose man-machine interfaces taking into consideration the wireless technologies of infrared beams and of real-time analysis of video captured images [4]. Specific targets of the research consist of studying models for mapping gesture to sound; in fact, in electro-acoustic music nothing is pre-established as in traditional music and instruments where there exist precise and well consolidated timbric, syntactic (harmony) and fingering systems of reference. The basic idea consists of remote sensing gesture of the human body considered as a natural and powerful expressive "interface" able to get as many as possible information from the movements of naked hands. A brief description of systems and devices we developed, follows. Video clips of performances realised with our gesture interfaces can be

found at the reported web page.

1.1 Twin Towers

This device consists of two sets of sensing elements that create two zones of the space, i.e. the vertical edges of two square-based parallelepiped, or virtual towers. At the time the first prototype was realised (1995), the shape of the beams suggest us the profile of the late Twin Towers in New York (we chose to keep the name in tribute to the victims of the tragedy). The measurements of distance of the different zones of the hands are performed by the amount of reflected light captured by the receivers (Rx's) and are quite accurate in respect to the irregularity, in shape and colour, of the hands palms. Voltage analog values coming from the Rx's are converted into digital format and sent the computer about 30 times/sec. The computer then processes data in order to reconstruct the original gesture [5,6]. It's so possible to detect positions and movement of the hands such as height and side and/or front rotations.

1.2 Imaginary piano

In the Imaginary Piano a pianist sits as usual on a piano chair and has in front nothing but a CCD camera few meters away pointed on his hands. There exists an imaginary line at the height where usually the keyboard lays: when a finger, or a hand, crosses that line downward, proper information regarding the "key number" and a specific messages issued in accordance of "where" and "how fast" the line has been crossed are reported. Messages are used for controlling algorithmic compositions rather than for playing scored music.

1.3 PAGe system

Another application based on the video captured image analysis system is PAGe, Painting by Aerial Gesture, which allows video graphics real-time performances. PAGe has been inspired and proposed by visual artist Marco Cardini (www.marcocardini.com) who suggested the idea of a system able to give the possibility of "..painting in the air.." and to introduce a new dimension to painting: time. PAGe has been designed by L.Tarabella and developed by Davide Filidei. PAGe allows Cardini, who also performs on stage, to paint images projected onto a large video screen by moving his hands in the air [7].

1.4 Mapping

The different kinds of gestures such as continuos or sharp movements, threshold trespassing, rotations and shifting, are used for generating sound event and/or for modifying sound/music parameters. For classic acoustic instruments the relationship between gesture and sound is the result of the physics and mechanical arrangement of the instrument itself. And there exist one and only one relationship. Using a computer based music equipment it's not so clear "what" and "where" is the instrument. From gesture interfaces, such as the infrared beam controller or image processing based systems, to loudspeakers which actually produce sound, there exist a quite long chain of elements working under control of the computer which performs many tasks simultaneously: management of data streaming from the gesture interfaces [8], generation and processing of sound, linkage between data and synthesis algorithms, distribution of sound on different audio channels, etc. This means that a music composition must be written in term of a programming language able to describe all the components including the modalities for associate gesture to sound, also said how to "map" gesture to

sound. The “mapping” makes therefore part of the composition [9].

2. A new real-time music language

In order to put at work the mapping paradigm and the facilities offered by the gesture interfaces we realised, at first we took into consideration the most popular music languages: MAX/DSP and Csound. Unfortunately, both languages resulted not precisely suited for our purposes mainly because Csound was not so real-time as declared and Max was not so flexible for including video captured images analysis code. Using two computers (first one for managing interfaces, second one for audio synthesis) connected via MIDI, resulted awkward and inefficient. We started writing basic libraries for processing sound and for driving the gesture interfaces bearing in mind the goal of a programming framework where to write a piece of music in terms of synthesis algorithms, score and management of data streaming from gesture interfaces. On the long run the framework became a very efficient, stable and powerful “music language” based on pure C programming, that is “pure-C-Music”, or pCM. “pCM” falls in the category of the “embedded language” and needs a C compiler in order to be operative; at first this may appear an odd news, but consider the good news of getting a compiled code of a synthesis algorithm which generate and process sound running many times faster in respect to Csound which, on the other hand, is an interpreted language.

2.1 pCM, pure C Music

pCM has been implemented using one of the most popular C compiler or better, multiplatform development system: Metrowerks Code Warrior. As a result a pCM composition consists of a CW project which includes all the necessary libraries including, once again, a DSP.lib (Digital Signal Processing library) consisting of a number of functions (at the moment more than 50) able to implement in real-time the typical synthesis and processing elements such as oscillators, envelope shapers, filters, delays, reverbs, etc.. The composition itself is a C program consisting of four void functions: *Init*(), *Orchestra*(), *Score*() and *End*() properly invoked by the main program which controls the whole “machinery”.

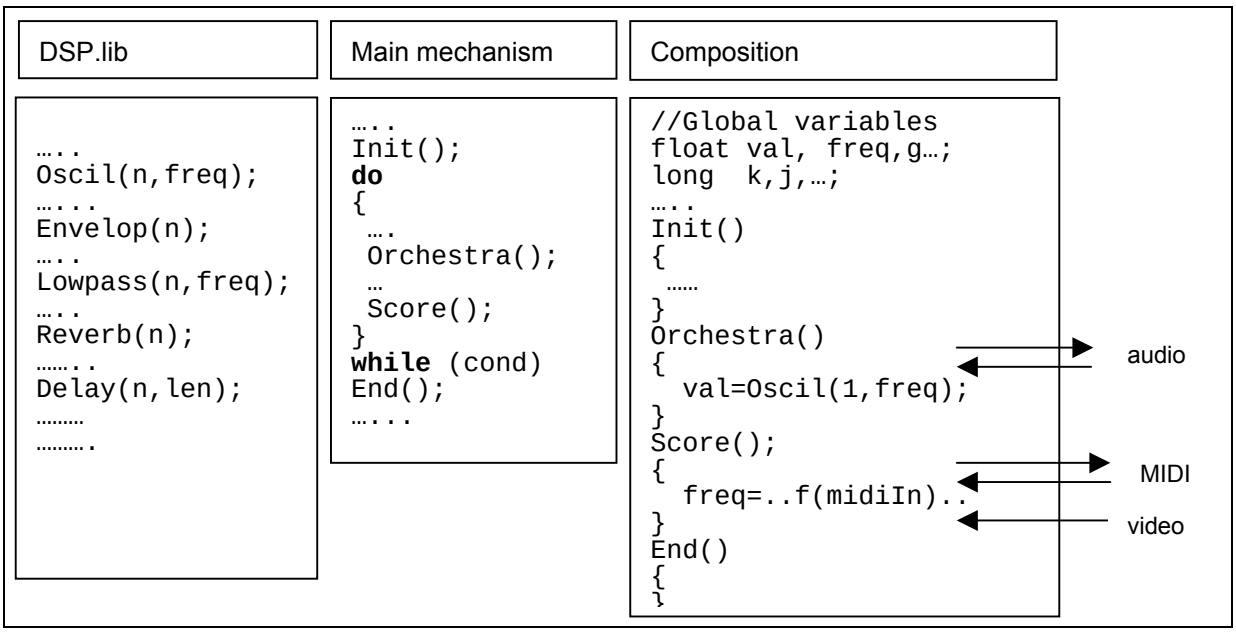

Fig.1 pCM principle of operations

For practical reasons of consistency it's a good idea the four functions make part of the same file which also includes declaration of the variables visible by the all functions and especially from *Orchestra*() and *Score*(). Everything here is written following the C syntax: synthesis algorithms, score and declaration of variables and data structures. Everything here is compiled into machine code and runs at CPU speed. The *Init()* function includes everything regarding initialisation and/or loading such as envelopes, tables, samples, delays, reverbs, variables and data structures. Usually it also includes calls for opening Midi, TCP/IP, Audio and/or Video Input channels. As a counter part the *End*() function is called at the end and is used for closing channel and disposing previously allocated memory.

2.2 Orchestra and instruments

An instrument is defined in the *Orchestra()* function and consists of code for sound synthesis and processing; it is continuously called at audio sampling rate, that is 44100 times per second. An instrument is defined in terms of an ordinary C program (with all the programming facilities such as *for*, *do-while*, and *if-then-else* control structures) which calls functions belonging to the DSP.lib; assigning the results of the whole computation to two predefined "system variables" *outL* and *outR,* sound is generated. Look at the following simple example.

```
….
  sig  = ampli*Env(1)*Oscil(1,freq);
  outL = sig*pan;
  outR = sig*(1.-pan);
```

where *sig* is a local variable, *ampli*, *freq* and *pan* are global variables loaded by *Score*(), Env(1) and OscSin(1,freq) belong to the DSP.lib and sounds like that:

```
float Oscil (int nOsc, float freq)
{
      float pos;
      pos = oscPhase[iN][nOsc] + freq;
      if (pos>=tabLenfloat) pos=pos-tabLenfloat;
      if (pos<0) pos=pos+tabLenfloat;
      oscPhase[iN][nOsc] = pos;
      return Tabsen[(long)pos];
}

float Env (int nEnv)  // One-shot envelope
{
  float  vval,vv,pos;
  long   ntabEnv =envNum[iN][nEnv];
  pos  = envPos[iN][nEnv];
  vval = *(envTable[ntabEnv] + (long)pos);
  if((long)pos<envLenght[ntabEnv]) envPos[iN][nEnv]=pos++;
  return vval;
}
```

2.3 The score

Once again the *Score*() is a C function which prepares parametric values and loads the global variables (*ampli*, *freq* and *pan* in the example) used by the active instrument in Orchestra(). There exist different modalities for writing a score: following the algorithmic composition

approach, writing sequences of predefined events, getting values coming from the external gesture interfaces and, finally, combining in different ways these techniques.

Suppose we want to control in real-time a very simple instrument using movements of the mouse by linking the vertical position to frequency and the horizontal position to left-right panning. In the MacOs environment, the mouse position is returned invoking the GetNextEvent(..) tool-box function which leaves the x,y position values in the "Event.where.v/h" variable and used as follows:

```
horiz  = Event.where.h/1023.;
freq   = Event.where.v+200.0;
```

These two lines make part of the *Score*() which is automatically and repeatedly called by the main mechanism. Since the mouse spans between 0 and 1023 horizontally and from 0 to 767 vertically, the variable *pan* and *freq* communicate proper values to the instrument for changing frequency and panoramic position. The following example explains the dynamic relationship between the Orchestra and the *Score*.

3. An example of composition

This listing reports a real working example.

```
//======================== FMouse =======
//  global  variables

float    freq,pan,vert,val;
int      enva, samp;
bool     mousePressed;
Point    position;
//-------------------------------------------------------
int Init ()
{
  float env[]={3, 0.0,0.0, 0.1,1.0, 4.0,0.0};
  enva  = NewLinEnv(env);
  mousePressed=false;
  OpenAudio();
}//-------------------------------------------------------
void Finish()
{
  DisposeEnv(enva);
  CloseAudio();
}//-------------------------------------------------------
void  Orchestra() // simple FM
{
  iN = 1;
  if (noteon[1]) TrigEnv(1);
  val  = Env(1)*Oscil(1,freq+Oscil(2,freq*1.5));
  outR = val*pan;
  outL = val*(1.-pan);
}
//-------------------------------------------------------
void Score ()
{
  if (Button() && !mousePressed)
  {
     iN=1;
     noteon[iN]  = true;
     position    = theEvent.where;
     pan         = position.h/1023.;
     freq        = position.v+200.0;
     mousePressed= true;
```

```
  }
  if (theEvent.what==mouseUp) mousePressed=false;
}//--------------------------------------------------------
```

A composition consists of five different sections: -1: global variables declaration for communication between *Orchestra* and *Score*; -2: *Init*(), for opening channels, declaring envelopes and delay lines, loading samples, etc..; -3: *Orchestra*() where sound synthesis and processing algorithms are defined; -4: *Score*() defined in terms of algorithms and management of input data stream coming from external gesture controllers –5: *End*() for closing channels and disposing envelopes and samples. Functions in bold belong to the DSP.lib. *Orchestra* is automatically called at 44.100 Hz sampling rate and has the priority on the CPU request; Score is called 50÷100 times per second depending on the complexity of the synthesis and processing algorithm defined in *Orchestra*.

Each time the mouse button is depressed (*Score* has the control) the predefined boolean variable *noteon[iN]* is set to true; then, the *Orchestra* trigs the envelope defined in Init and sound is produced with pitch related to the mouse vertical position and pan position related to the mouse horizontal position. The "system" predefined *iN* variable allows to select different instruments inside the same *Orchestra*: what follows the *iN* variable assignment is related to that instrument. Actually, the layout of both *Score* and *Orchestra* is more complex and consists of a number of " *case N: break;* " blocks which define different situations at micro level of sound processing and at macro level of events control.

3.1 The DSP.lib (Digital Signal Processing library)

This is the library of predefined functions which perform the micro level computation for generating and processing sound. What follows is a very small excerpt of the current collection of functions at the moment developed and upgraded when requested.

```
float  Noise();                                    This set of functions performs signal
float  Oscil (int nOsc, float freq);               generation in accordance to the name
void  TrigKarplusStrong(int nOsc);                 and parameters reported.
float  KarplusStrong (int nOsc, float freq);

int    NewLinEnv (float v[]);                      Usually called in the Init
void  TrigEnv (int nEnv);                          This starts the scanning of the envelope
float  Env (int nEnv);                             Used in Orchestra.

int    LoadSample (char nomesmp[]);                This function load a sample with the specified
void  TrigSample (int nSmp );                      name (for example: sn=LoadSample("whistle"));
float  Sample(int nSmp );                          This scans and reporst the current sample

float  Bandpass(int nFilt, float inp, float freq, float q); This is the set of usual filters.
float  Reson(int nFilt, float signal, float freq, float band);   Used in Orchestra
float  Lowpass(int nFilt, float signal, float cutfreq);
float  Hipass(int nFilt, float signal, float cutfreq);

void  NewDelay(int nDly, float dur);               Define a new delay line of specified duration.
void  PutDelay(int nDly, float val);
float  GetDelay(int nDly);
void  NewReverb(int nRev);                         Such as NewDelay, called in Init or Score.
float  Reverb(int nRev, float sign);               Used in Orchestra
```

```
void  SetFader( int nFad, float startval, float endval, float time);     A Fader is an asynchronous task
void  TrigFader(int nFad);                                              which starts when TrigFader is called and
float  Fade(int nFad);                                                  repeatedly used in Orchestra and in Score.

void  OpenAudio();                                                      Opens and starts the available audio channel
```

3.2 Other facilities

It's also possible to generate (offline or in real-time under control of data streaming from gesture interfaces) sequences of events automatically activated by the Scheduler(), a special mechanism which triggers sounds at the right times and change parametric values in the instruments of an *Orchestra*. Other facilities supported by the pCM framework are:

```
void     CDtrackSearch (short nTrack);          activation of sound tracks of  CD
void     CDplay ();                             with complete control of track selection
void     CDpause();                             pause command,
void     CDstop ();                             stop command
void     CDvolume (short vol);                  and volume control.

Void     Record("soundFilename.aiff",60)        This is called in the Init and starts the recording  of the
                                                global sound result onto memory with no loss of quality.
                                                The file is saved onto disk  in  .aiff or   .wav format.

UDPSend((void *) &udp_out, sizeof(long));       Communication via TCP/IP
UDPReceive((void *) &udp_in, &theUDPDataSize);

void     OpenMidi();                            Communication via MIDI
void     GetMidiData();
void     SendMidiMessage(int status, int data1, int data2);
void     NoteOn(int chan, int num, int vel);
etc..
```

Finally, OpenVideo() allows to open a video channel and to grab frames coming from a video camera. A "videoCallBack" function is installed when the video channel is open and called as service interrupt routine when a new frame is digitalized and put into memory: analysis of images is performed by processing the planar x-y matrix of values corresponding to the pixels of frames. Extracted parameters values from the shape and positions of the performer hands are then used in real-time the Score part of a pCM project.

4. Conclusion and acknowledgements

The pCM framework has been efficiently used for composing and performing [10,11,12] many pieces of music under the control of the gesture tracking systems and devices realised at cART project in Pisa. It has been developed for Mac/Os environment and also included as special topic in the course of Computer Music I yearly teach at the Computer Science Faculty of Pisa University. Due to the great interest of the students towards pCM who really like programming and "put their hands" inside computers, a group of them, particularly skilled and hardworking, decided to port the pCM framework onto PC/Windows environment rewriting it following the Object Oriented paradigms and with the *obvious* name, they claim, pC++M. Thanks in advance to all of them.

Special thanks are due to Massimo Magrini who greatly contributed to set up the pCM main mechanism currently in use and the many facilities for data communication and audio processing. Also thanks to Roberto Neri who recently graduated in Electronic Engineering at Pisa University with a thesis regarding the upgrading of the pCM DSP.lib.

5. References

[1] Tarabella L., Bertini G., "Giving expression to multimedia performances" – ACM Multimedia 2000, Workshop "Bridging the Gap: Bringing Together New Media Artists and Multimedia Technologists" Los Angeles, 2000

[2] Tarabella L., Bertini G., "Wireless technology in gesture controlled computer generated music", Proceedings of Workshop on Current Research Directions in Computer Music MOSART2001, Audiovisual Institute, Pompeu Fabra University, Barcelona, nov.2001

[3] R. Bargar, "Multi-modal synchronization and the automation of an observer's point of view" in Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC'98), 1998.

[4] Tarabella L.,Magrini M., Scapellato G., "Devices for interctive Computer Music and Computer Graphics Performances", IEEE First Workshop on Multimedia Signal Processing, Princeton, NJ, USA - IEEE cat.n.97TH8256, (1997)

[5] Rowe R., "Machine Musicianship" Cambridge: MIT Press. ISBN 0-262-18296-8, pagg. 343-353, March 2001

[6] Tarabella L., Bertini G., Sabbatini T., "The Twin Towers: a remote sensing device for controlling live-interactive computer music". In Procs of 2nd international workshop on mechatronical computer system for perception and action, SSSUP, Pisa, 1997.

[7] Tarabella L., Magrini M., Scapellato G., "A system for recognizing shape, position and rotation of the hands" in in Proceedings of th Internationl Computer Music Conference '97 pp 288-291, ICMA S.Francisco, 1997

[8] Tarabella L., Bertini G., Boschi G. : "A Data Streaming Based Controller For Real-Time Computer Generated Music" Proc. Int'l Symposium on Musical Acoustics ISMA 2001, Perugia, Italy, Vol. 2 pp. 619-622, sept. 2001,

[9] Tarabella L., Bertini G., "The mapping paradigm in gesture controlled live computer music" in Procs of 2nd Conference Understanding and Creating Music, Caserta, Seconda Università di Napoli, November 2002,

[10] Tarabella L.: "Five minutes for Joseph Beuys", for electronic viola and Twin Towers – 49a Biennale di Venezia, Platea dell'Umanità, june 2001

[11] Tarabella L.: "Kite: for copper wire and TwinTowers" – Perfomance commissioned by the Guglielmo Marconi Foundation and Assindustria Bologna, for the centenary of the first transatlantic radio-telegraphic transmission. – Sala Farnese, 26.03.2002, Bologna, Sasso Marconi, 25.04.2001.

[12] Tarabella L.: "Suite for M" for TwinTower and Imaginary Piano – NIME2002, Media Lab, Dublin. May 25th, 2002.