MAXIMILIAN: AN EASY TO USE, CROSS PLATFORM C++ TOOLKIT FOR INTERACTIVE AUDIO AND SYNTHESIS APPLICATIONS

Dr M Grierson
Dept. of Computing
Goldsmiths
University of London
m.grierson@gold.ac.uk

Mr Chris Kiefer
Dept. of Computing
Goldsmiths
University of London
c.kiefer@gold.ac.uk

ABSTRACT

Maximilian is a free, open source, MIT licensed C++ audio synthesis and signal processing library, designed to be cross platform and simple to use. The syntax and program structure have been designed to enable it to easily integrate into openFrameworks, being partly based on the approach taken by the popular Java-based environment, 'Processing'. Complex DSP operations have been masked as much as possible to facilitate the use of the library by artists and creatives who are learning to program, or those who are attempting to rapidly prototype audio applications. The library provides classes for standard waveforms, envelopes, sample playback, filters with resonance, delay lines, FFTs, granular synthesis and low-level feature extraction. In addition, equal power stereo, quadraphonic and 8-channel ambisonic support is included. The library can be used on its own, or in combination with other tools such as the Steinberg VST SDK, and runs well on embedded devices such as the iPhone. Maximilian will form part of the core synthesis library in the next release of openFrameworks.

1. INTRODUCTION

Maximilian is a freely available audio synthesis library written in C++. It is designed to work alongside either Gary P. Scavone's RtAudio 4 library [5][6], or Ross Bencina's PortAudio [2], providing a framework for compatibility across Windows, Linux and OS X. Maximilian's principle design aim has been to simplify the process of learning and implementing computer music approaches including digital signal processing and synthesis in C++. In addition, it has been designed to integrate well with the popular C++ creative coding toolkit, openFrameworks [www.openFrameworks.cc]. It is intended that this library be used by those who have less experience with textual programming languages, and by those with good programming skills who wish to develop and deploy audio applications quickly on any platform, including mobile devices. The API has a simple syntax, for example playing a sample takes only a few additional lines of code:

```
/*load a sample – normally in void setup()
or similar. */

beat.load(ofToDataPath("beat.wav"));

/* openFrameworks audio method */

void app::audioRequested(float *output,
                         int bufferSize,
                         int nChannels) {

for (int i = 0; i < bufferSize; i++){

sample=beat.play(0.5, 0, beat.length);
mymix.stereo(sample, outputs, pan);
output[i*2] = outputs[0];
output[(i*2) + 1] = outputs[1];
       }
}
```

The structure and syntax of the library has been designed along similar lines to the popular Java based programming environment for artists, Processing [7]. openFrameworks adopts a similar approach, allowing users to concentrate on the rapid development of platform independent interactive applications. These approaches to programming are a central part of the Goldsmiths BSc in Creative Computing, Masters in Computational Arts, and PhD in Arts and Computational Technologies. These programmes fuse interdisciplinary computing and creative arts approaches. Students on these courses come from a variety of arts and science backgrounds. At undergraduate level, students are taught Processing in year 1, and openFrameworks in year 2. This teaching method has proved successful, impacting significantly on student learning. Maximilian enhances this approach, allowing students to transition to C++ from Java, while giving them the power to create cross platform audio and music applications more easily.

2. RELATED WORK

The openFrameworks library (OF), also uses RtAudio. Maximilian has been deliberately designed to integrate well with OF. OF is an extremely powerful tool for the creation of a wide range of interactive applications. However, it has limited functionality with respect to audio and synthesis (this is also the case with Processing, with the exception of Ollie Bowns Java audio library, 'Beads' [1]). As Maximilian has been released using an open source, MIT license, it is a useful alternative to other existing solutions for creative audio application development in C++, including FMOD and the Synthesis Tool Kit (STK) [3][4]. The MIT license allows any programmer or artist the right to use the library to create commercial applications free of charge; there are few comparable libraries that give the user this freedom.

As Maximilian is written in C++, it can be easily used to develop software using a variety of different audio programming toolkits, including the iPhone and Steinberg VST SDK's. This makes it more widely portable than other text-based computer music platforms, and although it is not intended as a competitor to SuperCollider, Chuck[8], PD, STK or MaxMSP, it has proven very useful in learning/teaching creative software development to both undergraduates and postgraduates studying arts, computation and creative coding. In addition, the library greatly facilitates the creation of commercial audio software, and is being used by gaming and audio companies, as well as installation artists. As there are no significant restrictions with respect to the commercial use of Maximilian, students, researchers and creatives can generate income without restriction via their own Maximilian-based software projects.

3. THE MAXIMILIAN API

In order to compile software using Maximilian, RtAudio.h and RtError.h must be included, or in the case where PortAudio is being used, the PortAudio precompiled library is required. Both the RtAudio main function and callback function reside in maximilian.cpp, alongside the PortAudio callback, and Maximilian's core functions. Additional classes featuring FFT, MFCC extraction and granular synthesis functionality reside in separate files.

Although Maximilian can easily be used on its own alongside RTAudio or PortAudio, there now exists a fully supported openFrameworks add-on, ofxMaxim, which is far simpler to use. This add-on comes with a small number of examples, described below. The library itself can be downloaded from http://www.maximilian.strangeloop.co.uk. This download contains both the standalone software and ofxMaxim. The website also includes a number of simple code examples that explain how to use Maximilian's core functions. However, in order to make these examples work in openFrameworks, the object names must be prefixed with ofxMaxi - so the maximilian osc class is known as ofxMaxiOsc when used with ofxMaxim. Use with the most recent version of openFrameworks is recommended. Further to this, the ofxMaxim addon is fully compatible with ofxiPhone, the iPhone distribution of OF. Significantly, the ofxMaxim addon will be distributed as part of the next release of OF, greatly simplifying the creation of interaction audio and synthesis applications.

At present, maximilian.h contains 6 main classes: osc, envelope, delayline, filter, sample and mix (for equal power panning in stereo, quad and 8 channel (ambisonic sphere)). In addition, there are separate classes from spectral analysis and resynthesis, MFCCs and granular synthesis. The feature set is continuing to grow as the library is used for more projects.

3.1. OSCILLATORS

The osc class is a generic oscillator object, capable of producing phasors, sawtooth waves, sine waves, cosine waves, square waves, triangular waves and noise. In addition, there is a buffer-based oscillator object capable of playing back arbitrary wavetable data from a 514 point buffer using either linear or cubic interpolation. The osc.phasor is a central general purpose object, and can produce a continuous signal between any two values (specified as a pair of doubles), providing a means to control any number of object types or functions, In addition, the phasor function of the osc object can be used to index arrays of musical information, or buffer data, which in turn can be used to control new objects. The phasor function is also the object which defines the central approach that all other Maximilian objects follow. The object maintains its own phase, updating it every time it is called. For this reason, the samplerate must be known to all objects. Further to this, there is no buffer loop as such, although each class can easily be wrapped in a loop of any buffersize.

3.2. ENVELOPES

The envelope class at present has two main functions, line and trigger. The line object takes an array as input and produces a sample accurate ramp with up to 1000 segments. The input array must have two values for each segment – target value, and (double) duration in milliseconds. In this way, the line function is similar to the line~ object in PD and MaxMSP. However, the trigger function allows the envelope to be triggered from any point along the ramp. In addition, envelope::line() is powerful and fast enough to be used for waveshaping and sample playback.

It has been pointed out by some users that the envelopes are somewhat tricky to use. A generic ADSR is being created as part of the upcoming release which is far simpler to use, but consequently not as flexible.

3.3. DELAY LINE

The delay line is a simple buffer object with a variable size buffer. The buffer size can be altered at signal rate, making this object very flexible for the creation of a number of effects, including delay, flange and chorus etc. In addition, the object can be used as a string for elementary physical modelling. When combined with a filter (detailed below), the delay line is even more powerful, and can be used to create elementary reverbs. The main delay line function is called dl, and operates by specifying the input signal and the delay time.

3.4. FILTER

The filter object contains a number of useful filters, including low pass, high pass, low pass with resonance, high pass with resonance, and bandpass. These greatly expand the power and flexibility of the library. The lowpass filters are as elementary and as fast as possible. Below is the design for the low pass and high pass functions, followed by the resonant low pass filter function. Importantly, the standard lowpass and

highpass filters take a cutoff frequency input scaled between 0. and 1., whereas the resonant filter functions (such as filter::lores()) take a frequency value in Hz, and a resonance value between 1 and 100 respectively.

```
double maxiFilter::lopass(double input,
double cutoff) {
      output=outputs[0] + cutoff*(input-
      outputs[0]);
      outputs[0]=output;
      return(output);
}

double maxiFilter::hipass(double input,
double cutoff) {
      output=input-(outputs[0] +
      cutoff*(input-outputs[0]));
      outputs[0]=output;
      return(output);
}

double maxiFilter::lores(double
input,double cutoff1, double resonance) {
      cutoff=cutoff1*0.5;
      if (cutoff<10) cutoff=10;
      if (cutoff>(sampleRate*0.25))
      cutoff=(sampleRate*0.25);
      if (resonance<1.) resonance = 1.;
      z=cos(TWOPI*cutoff/sampleRate);
      c=2-2*z;
      double r=(sqrt(2.0)*sqrt(-pow((z-
      1.0),3.0))+resonance*(z-1))/
      (resonance*(z-1));
      x=x+(input-y)*c;
      y=y+x;
      x=x*r;
      output=y;
      return(output);
}
```

3.5. SUBMIXING

The mix class provides functions for equal power panning in stereo, quad or eight channel ambisonic scenarios. In each case, it is important that the sound hardware attached to the machine is capable of utilising the specified number of outputs. These functions are declared (double *) and take three arguments: the input signal, the output array (with either 2,4 or 8 elements), and the desired position in either one, two or three dimensions. The eight channel ambisonic version of this function is used in the newly created Embodied AudioVisual Interaction (EAVI) lab.

```
/* 8 channel bus (first order ambisonic)
*/
double*maxiMix::ambisonic(double
input,double eight[8],double x,double
y,double z) {
       if (x>1) x=1;
       if (x<0) x=0;
       if (y>1) y=1;
       if (y<0) y=0;
       if (z>1) y=1;
       if (z<0) y=0;
       eight[0]=input*(sqrt((1.0-
       x)*y)*1.0-z);
       eight[1]=input*(sqrt((1.0-x)*(1.0-
       y))*1.0-z);
       eight[2]=input*(sqrt(x*y)*1.0-z);
       eight[3]=input*(sqrt(x*(1.0-
       y))*1.0-z);
       eight[4]=input*(sqrt((1.0-x)*y)*z);
       eight[5]=input*(sqrt((1.0-x)*(1.0-
       y))*z);
       eight[6]=input*sqrt((x*y)*z);
       eight[7]=input*sqrt((x*(1.0-y))*z);
       return(eight);
}
```

Importantly, the mix class is not as simple to use as the other classes. The function returns an array which then must be written back to the appropriate output. At present, it appears that this is the most complex piece of code a user will have to write.

3.6. SAMPLE PLAYBACK

The sample loading and playback functions at present load WAV files only. This process is entirely universal, however, and works as well on the iPhone as it does on the desktop. This is very useful for prototyping applications on one platform that can then be transparently ported to another. Sample playback can be specified with speed defined as a pitch ratio, or alternatively by frequency / start sample / end sample. This design is useful for implementing sample manipulation, segmentation / beat cutting and granular synthesis techniques. The sample player takes negative speed / frequency values for reverse playback. In addition, it is capable of linear or cubic interpolation although the cubic interpolation is necessarily expensive in computational terms.

4. APPLICATIONS : SPECTRABRUSH

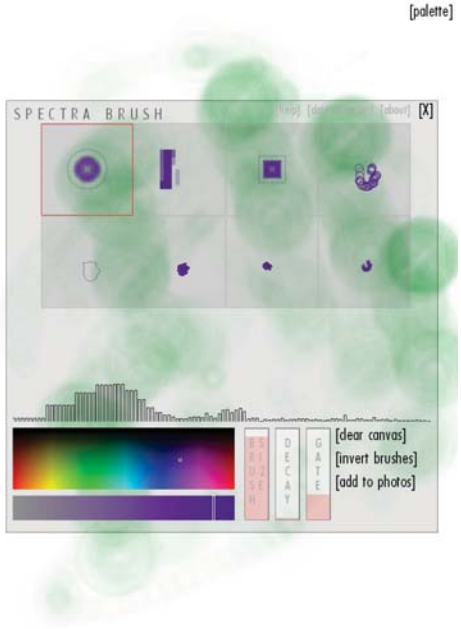

Figure 1. Spectrabrush Screenshot showing the main interface.

Spectrabrush is an expressive painting application, built for the iPad with openFramework's ofxiPhone library and ofxMaxim. The program allows the user to control brushes with both their fingers and with audio input, the shape of the brush being controlled by spectral analysis of the signal from the iPad's microphone. This allows the user to paint expressively using, for example, their voice or environmental sounds to change the brush shape as they paint. Eight different brushes are

available, all controlled by varying interpretations of the audio input. The program uses ofxMaxim's FFT class to analyse the incoming audio, and then uses the octave analyser class to create a perceptually matched representation of the spectral content, splitting the spectrum into 100 bins. The mean bin size and position of the largest bin are used as additional control features. Each brush uses this data in varying ways to create it's shape. For example, one brush draws circles in increases size from a centre point; the alpha value of each circle is controlled by the amplitude of a frequency bin. Another brush creates a shape using the amplitude of each bin as the distance value in equally spaced polar coordinates, and rotates this shape proportional to the maximum frequency bin. To increase the creative possibilities, the user can modify the colour, brush size, amplitude decay and gate values, and can invert the brush shapes. Spectrabrush is available free from the app store; to date there have been around 1100 downloads.

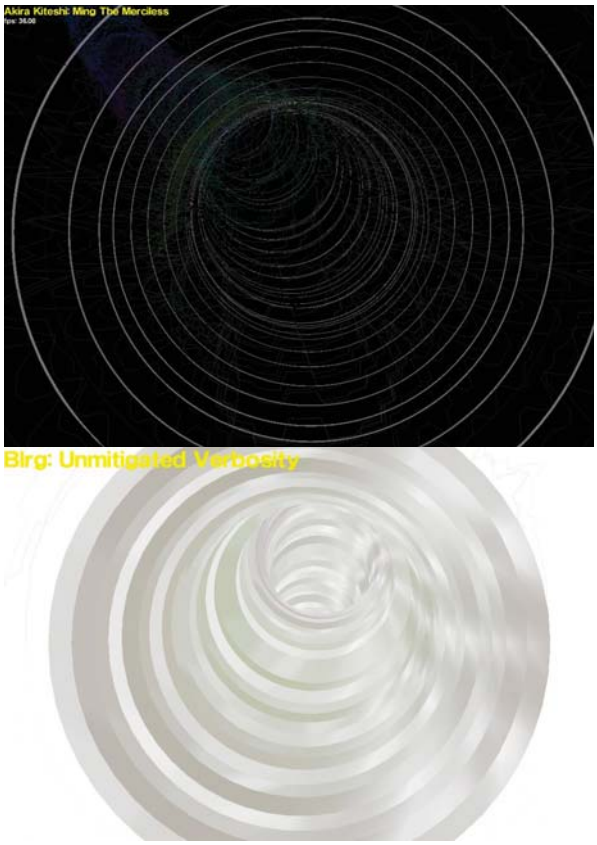

Figure 2. Earworm, an audio generated tunnel game for the IOS platform.

5. OTHER PROJECTS

The UK's premiere organisation for new music and sound, soundandmusic.org have commissioned a range of desktop and iPhone applications that make use of ofxMaxim for audiovisual interaction and sonic arts. These include *iWax*, a wax cylinder gramophone emulator for the iPhone, and *Oramics*, an emulation of the graphic sound synthesiser created by the electronic music pioneer and founder of the BBC Radiophonic Workshop, Daphne Oram. These commissions are being realised by staff and students working within the EAVI group at Goldsmiths. Further to this, ofxMaxim has been used to create psychophysical experiments used by Lottolab at the science museum [lottolab.org]. Finally, games company Roll7 are using the library to create a range of mobile games based on audiovisual interaction. The first of these games is Earworm, a game for the iOS platform that generates animated tunnels from music on the users iPod. It uses Maximilian's spectral and MFCC analysis functions to extracts features from the music and map them to tunnel generation parameters.

6. CONCLUSION

Maximilian is currently used to deliver year two and three undergraduate DSP and creative software development as part of the Goldsmiths Creative Computing Degree program. In addition, it has been adopted by the growing openFrameworks community, and as such is being used to create a huge number of interactive softwares and installations, including those that use multichannel audio. It is the basis of a number of independently commissioned iPhone and iPad applications, including some that form the basis of a three-year AHRC funded knowledge transfer partnership in embodied interactive games development. It can be used to rapidly prototype a wide range of complex audio and music software that can be exploited commercially free of charge. It is designed to be easily extensible and very easy to use. As such we hope to continue to develop Maximilian as part of our approach to the creation of better audiovisual interaction software.

7. REFERENCES

[1] Bown, O., (2009) “Ecosystem Models for Real-Time Generative Music: A Methodology and Framework”, in Proceedings of the International Computer Music Conference, Montreal, Canada.

[2] Bencina, R., (2001) “PortAudio–an Open Source Cross Platform Audio API” In Proceedings of the International Computer Music Conference, La Habana, Cuba.

[3] Cook, P. and G. Scavone (1999). The Synthesis ToolKit (STK). In Proceedings of the International Computer Music Conference, Beijing.

[4] Cook, P. R. (2002). Real Sound Synthesis for Interactive Applications. A K Peters, Ltd.

[5] Scavone, G. and P. R. Cook (2005). RTMidi, RTAudio, and a Synthesis Toolkit (STK) Update. In Proceedings of the International Computer Music Conference, Barcelona, Spain.

[6] Scavone, G. P. (2002). RtAudio: A Cross-Platform C++ Class for Realtime Audio Input/Output. In Proceedings of the International Computer Music Conference, Göteborg, Sweden, pp. 196–199. ICMA.

[7] Shiffman, D (2009). Learning Processing: A beginners guide to Programming Images, Animation and Interaction, Morgan Kaufmann Series in Computer Graphics.

[8] Wang, G. and P. R. Cook (2003). ChucK: A Concurrent, On- the-fly Audio Programming Language. In Proceedings of the International Computer Music Conference (ICMC), Singapore.