munger1~: TOWARDS A CROSS-PLATFORM SWISS-ARMY KNIFE OF REAL-TIME GRANULAR SYNTHESIS

Conference Paper · January 2007

CITATIONS
5

READS
73

2 authors:

Ivica Ico Bukvic
Virginia Polytechnic Institute and State University
49 PUBLICATIONS **118** CITATIONS
SEE PROFILE

Thomas Grill
Universität für Musik und darstellende Kunst Wien
26 PUBLICATIONS **570** CITATIONS
SEE PROFILE

munger1~: TOWARDS A CROSS-PLATFORM SWISS-ARMY KNIFE OF REAL-TIME GRANULAR SYNTHESIS

Ivica Ico Bukvic, D.M.A.
Virginia Tech
Department of Music
DISIS, CCTAD, CHCI
ico@vt.edu

Ji-Sun Kim
Virginia Tech
Computer Science, CHCI
hideaway@vt.edu

Dan Trueman, Ph.D.
Princeton University
Department of Music
dan@music.princeton.edu

Thomas Grill
University of Music and Performing Arts, Vienna
gr@grrrr.org

ABSTRACT

munger1~ is a new and enhanced version of a powerful *Max/MSP* real-time granular synthesis external found in the *PeRColate* library. Apart from added features and optimizations, including ability to generate theoretically unlimited number of grains per second using real-time input, *munger1~* also offers GPL-licensed platform-transparent code which works both under *Max/MSP* and *Pure-Data* without any alterations to the source. This platform-agnostic design is made possible through the use of *flext* library. Due to object's complexity, as well as a significantly less common porting path from the *Max/MSP/C* to a platform-agnostic *C++/flext*, this project has resulted in a number of improvements in *flext* build scripts and supporting documentation. It has also generated an invaluable list of caveats which are presented in this paper in hope to foster more platform-agnostic object design and porting efforts. Since its release, *munger1~* has been featured in two interactive multimedia creations whose technical and artistic impact are also addressed in this paper.

1. INTRODUCTION

Originally introduced in 1947 as a theory by Dennis Gabor [13], sonified by Iannis Xenakis in 1971 [27], and made real-time by Barry Truax in 1988 [24], granular synthesis is by no means a new technology. Yet, in part due to its inherent versatility, granular synthesis remains a prominent technique in contemporary digital audio vocabulary. Its recent adoption into the mainstream audio software, such as *Propellerhead's* introduction of *Maelstrom* synthesizer in *Reason 2.0* [19], certainly attests to this ongoing trend.

Late 1990s have introduced proliferation of portable computing, DSP-oriented programming languages and ensuing popularity of audio-visual environments geared towards interactivity, most notably *PD/Gem* [20, 12] and *Max/MSP/Jitter* [10]. As a result, there was a growing need for an integrated real-time granular synthesis object which would offer balance between versatility, and ease of use. Although only a peripheral object of the *PeRColate* library authored by Dan Trueman and R. Luke DuBios [4, 25, 26], *munger~* external to this day remains arguably one of the most powerful granular synthesis objects for the *Max/MSP* environment. *PD* community quickly realizing the importance of the *PeRColate* collection had made a genuine effort towards a native port. As a result, in 2002 Olaf Matthes had generated a near complete *Pure-Data* conversion. Unfortunately, in this port *munger~* failed to operate properly, maxing out the CPU usage in a seemingly random fashion and ultimately producing garbled and therefore unusable audio output. Another unfortunate shortcoming of Matthes's formidable efforts was the fact that original authors of the *PeRColate* library had no resources and/or interest in maintaining two concurrentcode bases, in part since their work revolved predominantly around the *Max/MSP* platform. More so, multiple *#ifdefs* in Olaf's code had made the source cumbersome to read and even more difficult to maintain. For these reasons, Matthes's contributions had fallen by the wayside, leaving *PD* community without an ability to tap into the vast potential of the *munger~* object.

In 2007, after having an opportunity to attain deeper understanding of the *flext* library [15], an effort was made to port *munger~* to the *flext* framework and thus make the object platform-transparent with practically no added overhead to its code maintenance. Several steps had to be taken before port could commence, one of them being permission to redistribute source under a *flext*-compatible license, in this case GNU General Public License [14]. After two weeks of dealing mostly with platform-specific idiosyncrasies which were ironically tied to the environment variables otherwise considered to be platform agnostic, *munger1~* 1.0.0 was released, offering new featuers and total backwards compatibility. Since its initial release in March 2007, the object has seen several updates. Latest version 1.2.1 provides fully modular memory allocation model, code optimizations, and virtually unlimited number of real-time grains per second which are restricted only by the raw CPU power.

2. PORTING CODE OUTSIDE THE BEATEN PATH

Although *flext* library is a very stable and mature cross-platform framework, its adoption has been associated predominantly with the *PD* community. As a result, apart from few exceptions (i.e. *fftease* [17] port initiated by the *flext* author Thomas Grill) most of the porting efforts have been either from *PD* to *flext*, or were entirely new objects designed from scratch within the *flext* environment (such as Thomas Grill's *xsample*, *pool*, *py*, etc., Tim Blechman's *chaos* and *tbext*, Frank Barknecht's *fluid~*, *syncgrain~*, etc., all of which are a

part of the *PD/flext* CVS repository [21]). Consequently, documentation regarding porting objects from the *Max/MSP C*-based source to the flext *C++* environment had been sparse.

munger~ is a relatively complex external with its own concurrently moving and rotating buffer as well as ability to interface with external buffers. Its latest version also relies upon the *Synthesis Toolkit* (STK) in order to provide ADSR [23] grain envelope. *munger~* in and of itself provides up to sixteen outputs, ability to specify grain content by manually traversing the current buffer, accessing random points from the buffer, as well as read the same in different directions. The object allows for up to fifty simultaneous grains per sample. The resulting absolute grain density per second depends upon grain separation and rate variation (1st and 2nd inlets respectively), individual grain size and size variation (3rd and 4th inlets respectively), absolute minimum allowable grain size ("minsize" message), and the delaylength message (whose size reflects the size of the moving sub-buffer and consequently largest possible grain). Grains can be transposed using random as well as pitch-based transpositions. Each grain is also spatialized using customizable random deviation. Unlike stereo spatialization which has its own dedicated inlet, multichannel diffusion requires "spatialize" message entry into the leftmost inlet in order to specify random amplitude deviation and the default channel amplitude gain.

As can be observed from its features, *munger~* has proven to be a formidable test of *flext*'s robustness.. Although the initial port was relatively quick, its clean-up and testing was encumbered by slowdowns stemming from the platform-specific compiler and engine peculiarities which for the most part fell outside *flext*'s domain.

2.1. Unexpected Caveats

Apart from the expected and relatively well documented API alterations which were necessary for the code to conform to the *flext* framework and thus become transparent to both *Max/MSP* and *PD*], there were several unexpected considerations whose scope will likely have a universal impact. They are listed below.

2.1.1. From C to C++

Apart from endless and in most cases dubious argument which programming language offers better performance, *C++* as an object-oriented environment provides superior scalability and has an easier path to memory management, in part due to its inherent ability to deal with freeing of the allocated memory through the class destructor [16]. While working on this project, we have discovered several peculiarities which had slowed our testing phase.

Another, more pronounced code deviation is related to the fact that *Microsoft Visual C++* (a.k.a. MSVC) [18], rather than *Cygwin* [11] environment was used on the Windows platform. As a result, the following compiler idiosyncrasies had to be addressed:

rand() function which is often critical component of algorithmic systems produces different value range in MSVC (namely 0-32767) from that of gcc (used by *Linux* and *OSX*). As a result an additional *#ifdef* was necessary to standardize value range of the original *munger~* object across platforms:

```
#ifndef __GNUC__
#define RANDOM (rand())
#else
#define RANDOM (random()%32768)
#endif
```

MSVC has also exhibited a peculiar frailty in respect to HEAP and STACK management. STK's ADSR class appears to require a considerable amount of memory. Using envelopes as an array (in this case variable `gvoiceADSR[numvoices]`) would inevitably result in crashing application whenever a total number of voices exceeded a moderate size (exact number proved to be an elusive target). Adjusting HEAP and STACK sizes, although allowing for greater sizes, had failed to attain sought stability. As a result, the code was converted into a vector of ADSR pointers which were then dynamically generated by the constructor using "new" operator embedded in the "for" loop.

2.1.2. flext, SndObj, and STK

flext has an inherent ability to interface with *SndObj* and *STK* libraries. As a result, compiling these libraries for developers may require some additional overhead despite the common building environment. One of the very useful features which would clearly benefit from additional documentation is an ability to build *Universal Binary* (UB) [3] external. STK by default is not UB-friendly and therefore needs to be built with UB explicitly enabled. This step, however, requires only a minor alteration to the STK's makefile:

```
CFLAGS += -isysroot
/Developer/SDKs/MacOSX10.4u.sdk -arch ppc
-arch i386
```

Once *flext* is compiled, the process of building an external is relatively straightforward. The *flext* build script relies upon the `package.txt` file which is for the most part self-explanatory.

flext external uses the building process identical to that of *flext* itself. The newly generated external is by default statically linked against supporting libraries, making its distribution and/or integration seamless. It is however worth noting that dynamic linking is also available and in this case preferred as it minimizes resource duplication by multiple concurrent instances of *flext* objects.

2.1.3. *Optimizing Code*

Due to message translation inherent to *flext*, we expected the platform-agnostic external to introduce larger CPU footprint than its native counterpart. Our experiments have shown, however, that *munger1~* when compiled with optimization flags provides near identical performance to that of the native *munger~* object. For this reason, we treated the anticipated overhead as negligible.

3. WHAT'S NEW?

The new cross-platform port *munger1~* introduces several enhancements over its precursor, some of which provide new ways of shaping sound, others that enhance internal object data and feedback monitoring, and finally those which are strictly internal overhauls for the purpose of streamlining the code.

3.1. New Features

munger1~ introduces two new user-controllable features which have a direct impact on the audio output. One of them is "discretepan" option (passed into the first inlet) which can be toggled on or off (default). Unlike default *munger1~*'s behaviour which projects the same grain onto every channel with varied amplitude (provided the random amplitude deviation set via "spatialize" command is other than zero), "discretepan" sends each grain to one particular channel generating a granular swarm whose full spectral composition can be discerned only from a location where all channels are equally perceivable. In this respect, *munger1~* provides spectral diffusion of grains (albeit with limited control over grain location).

The second feature is a modular number of output channels and concurrent voices (with arbitrary ceiling imposed at 64 and 1000 respectively for the sake of limiting human error and potential memory overflow). This alone empowers *munger1~* to deliver densities easily beyond 100,000+ grains per second. This number, however rapidly drops off with increased grain lengths and proportionally increased CPU overhead. For this reason it is not uncommon for *munger1~* to overload even the most contemporary hardware. The two options are set via optional 2nd and 3rd arguments (1st backwards compatible argument is reserved for internal buffer size). Another architectural peculiarity retained from the original *munger~* prevents the external from generating more grains than there are samples per second. This is due to the MINSIZE variable which determines absolute minimum grain size possible. Therefore, the absolute grain density per second is currently limited only by the sampling rate and the raw CPU power.

3.2. Object Monitoring and Feedback

munger1~ is designed to provide easy monitoring of multiple instances within the same session. For this purpose, *munger1~* offers an ability to add unique name to every instance. This name is consequently reflected in all of the object's output generated in the console. Object name can be set via optional 4th argument which also ignores "_" [underscore] entries, interpreting them as means of extending object's width in *PD*.

Due to improved, potentially high-volume data monitoring feedback, *munger1~* also offers "verbose" option which allows for setting the following four verbosity levels:

0 = all messages (including errors) off
1 = only errors and warnings (default)
2 = all messages
3 = all messages plus number of grains per second

munger1~ with its ever-growing complexity also includes near-complete documentation for both platforms.

3.3. Under the Hood

With its modular voice and channel output, *munger1~* offers fully dynamic memory allocation of its resources. This is especially important due to its potentially large memory footprint. This dynamic model is in part achieved through the vector-based implementation of larger and more memory demanding data arrays which also designed to simplify their deallocation.

Finally, new object has a built-in *flext*-based method which, when coupled with external buffer, continually checks for external buffer's validity. While generally unlikely, and mostly limited to human error (i.e. by explicitly erasing the external buffer while audio engine is turned on), were it not for the aforesaid implementation, such an occurrence would inadvertently bring down the entire application with irreversible data loss.

4. NOTES TO END-USERS

munger1~ has several idiosyncrasies which require end-user attention. Although it is our aim to address many of these in object's future iterations, currently the best way of dealing with them is through awareness.

When generating a multichannel instance of *munger1~* no audio will be outputted until object is given a "spatialize" message followed by an array of numbers reflecting channel-specific amplitude gain and random amplitude deviation, a.k.a. spread. For instance, "`spatialize 0.1 0.5 0.1 0.5 0.1 0.5`" message to the object would map an amplitude gain of 0.5 and a gain spread of 0.1 to the first three output channels).

Internal and external buffers can be used interchangeably via the "buffer <buffer_name>" message. "buffer" message by itself reverts back to internal buffer. It is important to note that in Max/MSP, when external buffer mysteriously disappears or is explicitly deleted, *munger1~* will cease all output until

its buffer is manually set to another buffer (internal or external). *PD* in such a situation automatically reverts to internal buffer.

"maxvoices" message which was used originally to remap the number of used voices out of a hardwired maximum of 50 is now deprecated and is retained only for legacy purposes. Although it can to some extent optimize the main loop (i.e. in situations when not all allocated voices are needed, which is controllable via "voices" command), its impact is negligible. Due to its backwards compatible implementation, however, "maxvoices" can have an impact on the number of available voices. For instance, if an object is initiated with 70 voices (using object's optional 3rd argument), and is given "maxvoices 50" command, "voices" message will only allow for up to 50 voices. For this reason, use of maxvoices in *munger1~* is considered deprecated and its use is discouraged.

5. REAL-WORLD PERFORMANCE

5.1. Benchmarks

In order to assess object's performance two benchmarks have been conducted on *Linux*, *OSX*, and *Win32* platforms. For this purpose we used the *munger1~* help file with two different settings. Since internal *munger1~*'s sample buffer is hardwired to 64 bytes, it remained constant in all tests. Max/MSP's I/O Vector Size was set to 512 while Signal Vector size was 8. *PD*'s internal buffer was calculated to the closest millisecond (namely 512x8/48,000 which yielded approx. 9 milliseconds). All tests were using internal laptop soundcards. The two scenarios were as follows:

Test 1: 100ms grain size, 50ms grain size variation, 0 grain separation and grate variation, 8-channel output with default spatialization values, minsize 1, delaylength 300, and a sampling rate of 48,000Hz.

Test 2: 0ms (or minimum internally hardwired) grain size, 0ms grain size variation, 0 grain separation and grate variation, 8-channel output with default spatialization values, ,minsize 1, delaylength 300, and a sampling rate of 96KHz.

An AMD64 3000+ (1.8GHz) notebook running 32-bit Windows XP and Wuschel's *ASIO4ALL* driver [5] with 512x4 internal buffering (lower internal latencies were not reliable even at very low CPU utilization, in part likely due to the lack of native ASIO support by the embedded sound chip), in the first test was able to generate 135 simultaneous voices per sample, bringing the total number of grains per second to 6,750. The second test's reliable ceiling was at 82,500 grains.

A Macbook Pro 1.83GHz with a Core Duo processor running *OSX* 10.4.9 in the first test generated 150 simultaneous voices, with a total of 7,500 grains per second. In the second scenario, the 96,000 sampling rate upper limit was reached with Max/MSP's DSP Status panel showing a peak 79% CPU utilization, suggesting that output was limited strictly by the sampling rate.

Linux tests were conducted on the same hardware as Windows, namely a dual-boot AMD64 3000+ laptop. However, due to the fact that Linux's default audio driver (ALSA) [1] performs vastly better in native hardware sampling rates, instead of dealing with added buffering of asoundrc-based virtual devices [2], we opted for conducting tests using 48KHz, the native sampling rate of the onboard *Realtek* sound chip. We expected that this choice would thus generate near identical CPU overhead induced by the low-latency operation: since ALSA's direct access to hardware introduces virtually no additional buffering overhead, its performance was deemed equivalent (or as equivalent as possible) to that of an ASIO and *Core Audio* drivers on Windows and Apple platforms respectively. For this test *PD* was run in realtime mode (-rt flag) with pd-watchdog reniced to ensure elevated priority in the case of CPU overload. *Linux*'s measured grain density in the first test was identical to that of its Windows counterpart suggesting that *flext*, STK, and consequently *munger1~* enjoy equivalent optimizations across platforms. The data produced by the second test, was generated using 48KHz sampling rate and is provided here purely for referential purposes as it lacks uniform (or near uniform) testing conditions. Nonetheless, ensuing data showed *PD* plateau at 48,000 grains per second with approx. 50% CPU utilization by *PD*, suggesting equivalency to that of its Windows counterpart, once again reaffirming external's CPU-footprint equivalency across platforms.

As can be observed from the aforesaid benchmarks, *munger1~* exhibits near identical platform-agnostic performance. The supporting data also suggests that *munger1~* could benefit from multithreaded optimizations which would warrant quantum leaps in performance on dual-core CPUs available in the Macbook Pro and other modern portables. Due to the absence of the aforesaid multithreaded design, its performance appears to be strikingly proportional to the CPU clock speeds. Preliminary tests have also shown that CPU footprint increases with a larger number of output channels. Therefore it is not unreasonable to expect attainability of greater densities using stereo output, than those presented above.

5.2. Art

Since its milestone 1.0.0 release *munger1~* has been utilized in at least two performance-based multimedia creations authored by Ivica Ico Bukvic, both of which are covered here in order to provide but a hint of its real-world performance. One of them is *Pandora* interactive multimedia work for color-based gesture-tracking hyperinstrument, interactive visuals, voice, laptop and a quad output. *Pandora* relies heavily upon *munger1~* in order to generate sustained reverb-like textures whose amplitudes are in part controlled via

grain density, as well as produce dense spectral layers built from the captured vocal material and its concurrent pitch, length, and amplitude permutations. Given that the majority of *Pandora*'s CPU overhead stems from real-time video processing, 3D rendering, and motion tracking, only one instance of *munger1~* was used, Perhaps more importantly this one instance has proven more than adequate in generating the aforesaid dense aural textures. An audio-visual recording of *Pandora*'s performance is downloadable from http://ico.bukvic.net/Video/.

Second work titled *Soul* for baritone, audience, laptop, and quad output which was also premiered in April 2007 relies almost exclusively upon *munger1~*'s diverse signal processing potential as well as its newfound ability to generate dense real-time granular textures. In part to attain the aforementioned goal, the work calls for three concurrent instances of *munger1~* with discrete buffers. Each instance is connected to a simplified on-screen version of the gesture interface utilized in *Pandora* and is used at specific points throughout the piece. The work's closure reengages all three instances with their respective buffers intact in order to produce a dramatic cumulative effect, climaxing in a timbrally rich wall of sound. This particular gesture utilizes 120 (40 per instance) concurrent grains per sample. In addition to DSP techniques inherited from *Pandora*'s interface, *Soul* also deals with a more subtle pitch detuning associated with ideas and motives inherent to the Emily Dickinson's poetry which had inspired the work. A recording of *Soul* is available at http://ico.bukvic.net/Audio/.

6. FUTURE DEVELOPMENT

While *munger1~* is already a stable, production-ready external, its real-world deployment has isolated several potentially useful additions as well as occasional bugs. As a result, we have generated a roadmap with an aim to implement these in the near future. The following fixes are listed according to their priority.

6.1. Fixing memory leaks

Both *Max/MSP* and *Pure-Data* require proper destructor implementation in order to avoid potential memory leaks. In our tests, however, we have encountered unexpected behaviour with *munger1~*'s destructor. Namely, while *PD* accepts munger1~ destructor's memory deallocation syntax without any errors and/or warnings, the same code approach triggers an unknown exception in *Max/MSP*. As a result, *Max/MSP* version currently has known memory leaks, whereas in *PD* munger1~'s memory footprint appears to be for the most part stable with few unaccounted lingering memory allocations. This problem is currently unresolved and as such is on top of our priority list.

6.2. ADSR Overhaul

Although current ADSR envelope implementation generates default values for all channels and its grain-specific settings are alterable via "oneshot" events, currently there is no facility which would alter ADSR shape globally nor revert individual ADSR "oneshot" alterations back to the global setting. There is also consideration to enhance the ADSR model to include more than just four points inherent to the classical ADSR model. We aim to address the aforesaid deficiencies with the ADSR overhaul, including an "adsr" message which will provide the aforesaid global envelope alteration.

6.3. Spectral and Duration-Based Diffusion

One of the new features is expansion of "discretepan" paradigm. In addition to the existing two methods of spatialization, we will also provide the following two modes:

Mode 2 will generate spectral diffusion based on pitch

Mode 3 will focus on spatializing all content according to the data received from a likely new inlet which will take two values: current grain swarm center channel (provided as a float-point virtual source), and the grain swarm spread surrounding the aforesaid center.

Similar to the aforesaid mode 2, a "durationpan" and "amplitudepan" will be implemented to spatialize grains according to their duration and amplitude respectively.

6.4. Built-in Initialization of Multichannel Diffusion

As reported in the chapter 4, *munger1~* when instantiated as a multichannel object will not output any audio until it receives "spatialize" command. Its future iterations will provide safe default spatialization values in order to enable immediate audio output.

6.5. Absolute Minimum Grain Size

In order to attain greater resolution in the main DSP loop we will assess decreasing the absolute minimum grain size and its impact on the aural as well as CPU output.

6.6. Multithreaded model and Vectorization of Code

Series of tests will be conducted to assess feasibility of a multithreaded design that will be capable of utilizing advantages of multi-core CPUs. This, coupled with code vectorization via SIMD [22] instructions should provide a considerable boost in performance. Currently there are no plans to pursue explicit Altivec optimizations.

7. OBTAINING munger1~

munger1~ is a free open-source GPL-licensed external which is currently freely downloadable from

http://ico.bukvic.net/Max/munger1~_latest.tar.gz. In its latest version 1.2.1 released on May 7th 2007, it comes with source, build packages for gcc and MSVC environments, help files, and prebuilt binaries for *Max-Win32-i386*, *Pd-Linux-i386*, and *Max-Mac-UB*. As always, contributions to code as well as submission of pre-packaged binaries for other platforms are most welcome.

8. CONCLUSION

munger1~ is an open-source versatile, scalable, and platform-agnostic real-time granular synthesis external for *Max/MSP* and *PD*. Currently limited only by the raw CPU power, barring any fundamental API changes, it is future-proof while warranting minimal increase in the code maintenance overhead over its *Max/MSP*-native precursor.

9. ACKNOWLEDGMENTS

Special thanks go to Dan Trueman and R. Luke DuBois for making and open-sourcing this great external, Thomas Grill for the incredibly useful and vastly underused *flext* layer, Perry R. Cook and Garry P. Scavone for STK, and obviously the entire *PD* and *Max/MSP* communities for making these tools arguably the most modular and versatile creative multimedia environments available today.

10. REFERENCES

[1] ALSA, "Advanced Linux Sound Architecture", Cited 2007; Available from http://www.alsa-project.org.

[2] ALSA, "asoundrc file", Cited 2007; Available from http://www.alsa-project.org/alsa-doc/doc-php/asoundrc.php.

[3] Apple, "Universal Binary Programming Guidelines, Second Edition", Jan. 2007, Cited 2007; Available from http://developer.apple.com/documentation/MacOSX/Conceptual/universal_binary/universal_binary.pdf.

[4] Arslan, B., Brouse, A., Castet,J, Filatriau,J.J., Lehembre,R. Noirhomme, Q. and Simon, C. "Biologically-driven musical instrument," *Proceedings of eNTERFACE'05 Summer work-shop on multimodal interfaces*, Mons, Belgium, 2005.

[5] ASIO4ALL, "Universal ASIO Driver For WDM Audio", Cited 2007; Available from http://www.asio4all.com.

[6] Bulka, D. and Mayhew, D. *Efficient C++: Performance Programming Techniques*, Addison-Wesley Professional, 1st edition, 1999.

[7] Cadiz, R. and Kendall, G. "Fuzzy logic control toolkit: real-time fuzzy control for Max/MSP and Pd", *Proceedings of International Computer Music Conference*, New Orleans, Lousiana, USA, 2006.

[8] Cook, P. R. and Scavone, G. "The Synthesis Toolkit (STK)." *Proceedings of the International Computer Music Conference*, Beijing, China, 1999.

[9] Cycling '74. "Max/MSP: A graphical programming environment for music, audio, and multimedia", Cited 2007; Available from http://www.cycling74.com/products/maxmsp.

[10] Cycling '74. "Jitter: A Brilliant Collection of Video, Matrix, and 3D Graphics Objects for Max", Cited 2007; Available from http://www.cycling74.com/products/jitter.

[11] Cygwin, "Cygwin", Cited 2007; Available from http://www.cygwin.com.

[12] Danks, M. "Real-time image and video processing in GEM." *Proceedings of the International Computer Music Conference*, Thessaloniki, 1997.

[13] Gabor, D. "Acoustical Quanta and the Theory of Hearing." Nature 159 (4044): 591-594, 1947.

[14] GNU, "General Public License (GPL)", Cited 2007; Available from http://www.gnu.org/copyleft/gpl.html.

[15] Grill, T. "flext - C++ layer for cross-platform development of Max/MSP and pd externals", Cited 2007; Available from http://grrrr.org/ext/flext/.

[16] Lippman, S. B. and Lajoie, J. "C++ Primer (3rd Edition)", Addison-Wesley Professional, 1998.

[17] Lyon, E. "FFTease: A collection of Max/MSP objects implementing various forms of spectral sound processing", Cited 2007; Available from: http://www.sarc.qub.ac.uk/~elyon/LyonSoftware/MaxMSP/FFTease.

[18] Microsoft, "Visual C++ Developer Center", Cited 2007; Available from http://msdn.microsoft.com/visualc.

[19] Propellerhead Software, "Reason,", 2003, Cited 2007; Available from http://www.propellerheads.se/download/files/whatsnew_rsn25.pdf.

[20] Puckette, M. "Pure Data." *Proceedings of International Computer Music Conference*. San Francisco, 1996.

[21] Pure Data. "Sourceforge.net: Pure Data Computer Music System." 2007, Cited 2007;

Available from http://sourceforge.net/projects/pure-data.

[22] Stewart, J. "An Investigation of SIMD instruction sets", 2005, Cited 2007; Available from http://noisymime.org/blogimages/SIMD.pdf.

[23] The Synthesis ToolKit in C++ (STK), "STK ADSR envelope class", Cited 2007; Available from http://ccrma.stanford.edu/software/stk/classADSR.html.

[24] Truax, B. “Real-time granular synthesis with a digital signal processor,” Computer Music Journal, vol. 12, no. 2, pp. 14-26, 1988.

[25] Trueman, D. and DuBois, R.L. “PeRColate”, Cited 2007; Available from http://music.columbia.edu/PeRColate.

[26] Trueman, D. and DuBois R. L., PeRColate manual, 2001, Cited 2007; Available from http://www.music.columbia.edu/PeRColate/PeRColate_manual.pdf, 2001.

[27] Xenakis, I. *Formalized Music: Thought and mathematics in composition*, Indiana University Press, 1971.