Music Based Procedural Content Creation

Alberto Fustinoni  
BSc (Hons) Computer Games Technology  
2008/2009

University of Abertay Dundee  
School of Computing and Creative Technologies

# Abstract

With ever increasing development costs due to the adoption of new hardware platforms and the expectations they create among the game buying public, there is a pressing need to automate as much of a title’s production process as possible. While procedural content creation techniques are increasingly seen as an answer to the trend and are consequently gaining widespread diffusion, they are limited in their applications by their general inability to establish emotional connections with the user.

The current body of work attempts to improve procedural content creation techniques’ ability to emotionally target the user by relying on musical tracks selected by him as inputs, in turn broadening their range of possible usages.

Research was carried out into defining a small set of numerically expressible parameters synthesizing key audio perceptual features as well as into implementing a method for their computation from digitized representations of audio signals. Subsequently, a proof of concept background music reactive application was created to show the feasibility of their usage as seed for procedural content creation.

By providing a method for audio perceptual feature calculation and proving its applicability to game scenarios, the development of new procedural content creation techniques that can be used in more prominent ways is facilitated.

# Acknowledgements

While undertaking this thesis required tremendous effort on my part, I am extremely grateful to the following people for the support and guidance they provided along the way; were it not for their help, the end result would have been much less complete and polished.

My technical supervisor Matthew Bett, who in addition to always being available to provide support is the originator of the idea this project is based on.

Dr. Kenny Mc Alpine, whose in-depth knowledge of music, mathematics and signal processing proved determinant on multiple occasions.

The module supervisor Dr. Ozveren C. Suheyl, whose guidance and organizational tips ensured timely project completion.

Abertay Dundee University’s lecturers, particularly Dr. Henry Fortuna, Dr. T. Nigel Lucas and Dr. Colin Miller, for everything they taught me in the past four years.

All my family and friends, both in Italy and the UK, for their love and support.

# Table of contents

[Abstract 2](#_Toc230540002)

[Acknowledgements 3](#_Toc230540003)

[Table of contents 4](#_Toc230540004)

[Table of figures 6](#_Toc230540005)

[List of tables 7](#_Toc230540006)

[Introduction 8](#_Toc230540007)

[Motivation 8](#_Toc230540008)

[Aim 9](#_Toc230540009)

[Literature review 10](#_Toc230540010)

[Literature research methodology 10](#_Toc230540011)

[Existing commercial implementations 10](#_Toc230540012)

[Related applications 12](#_Toc230540013)

[Music theory 12](#_Toc230540014)

[Physics of musical sound and psychoacoustics 13](#_Toc230540015)

[Signal processing theory and Fourier analysis 15](#_Toc230540016)

[A suitable experimental reference 17](#_Toc230540017)

[Tempo detection as basis for input analysis 18](#_Toc230540018)

[Methodology 20](#_Toc230540019)

[Overview 20](#_Toc230540020)

[Data structures 21](#_Toc230540021)

[Input conversion to LPCM 22](#_Toc230540022)

[Windowing 23](#_Toc230540023)

[Fourier analysis 25](#_Toc230540024)

[Rhythmic tracks computation 27](#_Toc230540025)

[Need for an alternative to particle filters 30](#_Toc230540026)

[Rhythmic tracks interpolation 30](#_Toc230540027)

[Beat detection 32](#_Toc230540028)

[Determining tempo in BPM 33](#_Toc230540029)

[Proof of concept game 35](#_Toc230540030)

[Results 38](#_Toc230540031)

[Conclusion 40](#_Toc230540032)

[Summary 40](#_Toc230540033)

[Future work 41](#_Toc230540034)

[References 43](#_Toc230540035)

[Bibliography 46](#_Toc230540036)

[Appendix A: Project Proposal 50](#_Toc230540037)

[Introduction 50](#_Toc230540038)

[Research Topic 51](#_Toc230540039)

[Research Question 52](#_Toc230540040)

[Project Execution 53](#_Toc230540041)

[Project Evaluation 54](#_Toc230540042)

[Possible Issues 54](#_Toc230540043)

[Resource Requirements 55](#_Toc230540044)

[Appendix B: Accompanying CD 56](#_Toc230540045)

[CD contents listing 56](#_Toc230540046)

[Demo applications usage instructions 57](#_Toc230540047)

# Table of figures

[Figure 1: Piano middle C periodogram 14](#_Toc230540048)

[Figure 2: Time-frequency resolution trade-off 16](#_Toc230540049)

[Figure 3: Section of rising sine wave periodogram 26](#_Toc230540050)

[Figure 4: Hann windowing function graph 27](#_Toc230540051)

[Figure 5: Rhythm tracks graph section 29](#_Toc230540052)

[Figure 6: Rhythm tracks interpolation graph section 32](#_Toc230540053)

[Figure 7: BPM graph 35](#_Toc230540054)

# List of tables

[Table 1: Analysis results comparison 39](#_Toc230540055)

# Introduction

## Motivation

The continuous evolution in the capabilities of gaming hardware, coupled with consumer expectations of its optimal usage, has dramatically increased the minimum requirements a commercial videogame needs to meet in order to be successful in the marketplace.   
These additional demands have determined a progressive increase in the amount of art assets required for a game’s viability, driving production costs up and making content creation their most significant part.  
Procedural content creation, having software algorithms instead of humans create the required art assets, has been gaining popularity as a consequence, with developers trying to take advantage of it as much as possible in order to reduce the amount of man-hour required for projects completion.

While procedural generation can deliver on its cost-cutting promise, it does have downsides limiting its usefulness: amongst these, one of the most prominent is the fact that procedurally generated content, by its very nature, tends to be remarkably homogeneous.  
Assets created by most techniques are variations of a single base scheme: randomly or pseudo-randomly generated numbers are usually used to “seed” the generation algorithm, specifying how to modify or extend the base scheme to create the output.  
As such, procedurally generated content is most useful for decorative, non gameplay related entities, assets adding value to the player experience only once and then ignored, but which presence is required for the game world to be believable: examples of this range from trees and foliage to the individual buildings forming a city’s landscape.

When it comes to key gameplay elements, however, procedural techniques fall short: the difficulty or outright impossibility of fine tuning them to the specific needs of single game sections and the failure to replace human aspects of content or establish emotional connections with the user push them to the sidelines. At most, procedural generation techniques can complement or act as the base for the work of human artists tasked with the creation of the game’s main elements (Danc, 2007).

One possible way of bridging the aforementioned emotional gap, allowing procedural content generation techniques to take a major role in the overall user experience creation, could be to base them on external input which has an existing emotional connection with the user, while taking steps to make the relationship between the input and resulting output as recognizable as possible. Audio tracks supplied by the player itself should be prime candidates for serving as input, given how people relate to music.

## Aim

Focus of the project is demonstrating the feasibility of using audio tracks as input for the procedural creation of game content, while trying to establish relations as easily discernible as possible by the end user between musical input and generated output.

In other words, this thesis will attempt to answer the following question:  
“To what extent can a musical audio signal be used as an input for procedural content generation techniques in a computer game environment?”

An answer will be reached via the creation of a “background music reactive game” or “musically controlled game” as defined by Aallouche et al. (2007): an application that reacts to background music with game event creation instead of adapting the soundscape to the game’s state; as such, the procedurally created output will be the game level itself, as opposed to any particular asset it will contain.

# Literature review

## Literature research methodology

While the project’s ultimate goals had been clearly defined since the idea’s conception, understanding of a method suitable to achieve them was tenuous at best: nothing other than that the input signal would be analyzed in order to compute a set of numerical values descriptive of its musical content, which would be used as seeds for the created game’s procedural creation techniques was defined. What the previously mentioned values could be was entirely unknown, as were the effects they would have had on the game.

Literature research activity was therefore carried out with the goal of finding suitable models to follow in fleshing out these ideas: the process has been described in the present section in chronological order.

## Existing commercial implementations

Research started by analyzing existing and widespread applications performing, at a first glance, functions similar to the ones to be implemented: games belonging to the “rhythm game” or “music game” genre.   
While not all-including (notable absents include the likes of Pop’n’Music, DJ Max, O2Jam and Dance Factory), Shane Patterson’s The History of Music Games (2008) lists and describe most of the games belonging to the genre, thus providing a detailed yet concise survey.

The first inference that can be drawn from the article is related to the degree of homogeneity of the genre, or, more specifically, the lack of it when compared to most other ones (first person shooters or driving games, for example): different music games often have vastly different gameplay and, taken as a group, use a much bigger array of input peripherals than any other videogame genre.

The observations most interesting and relevant to the current project, however, can be drawn by analyzing the games comprising the genre building on Aallouche et al’s theory.  
The authors divide videogames into two distinct functional categories: “background music reactive games”, applications using music, usually while playing it in the background, to modify game parameters in real time, and those modifying the background music according to game events or having no relation between BGM and game state, which will be referred to as “background music adaptive games” from now on for brevity.

Applying this categorization to the previously mentioned genre review sees a minority of titles (Otocky, Mario Paint, Simtunes, MTV Music Generator, Electroplankton, and Hiite Utaeru DS Guitar) unquestionably categorizable as background music adaptive, while the rest would seem to qualify as reactive: the user selects a track, with gameplay involving the performance of a sequence of actions within specified time windows bearing intuitive resemblance to the tune being played in the background.

A more in-depth analysis reveals this to be false, however, as hinted by how the user selects from a predefined list of tracks: as opposed to procedurally analyzing the audio stream, such applications rely on human created metadata describing the sequence of actions, as well as their timing, required from the player during playback, and sometimes instructions on how to modify audio output as a function of player’s success or failure.  
While discussion on what category such games belong to (Is the game reacting to the audio track even if a human instructed it on how to do so or is the audio track at least been selected, if not modified to complement the gameplay?) is only marginally relevant, their reliance on human intervention for level creation made them unsuitable as models from which to draw ideas for the current project’s implementation.

A few rare instances of actual background music reactive games were, however, discovered in Audiosurf, Vib-Ribbon and Dance Factory: all these titles share the common trait of allowing the user to supply any musical track (as long as its format is supported, which means audio CD tracks for the last two) and proceed to create a level without any human intervention.  
Research was carried out in order to find information about their working, but, predictably, nothing significant was found: considering the rarity of these implementations, the usefulness of their function and the complexity involved, it wasn’t really surprising to find that they were kept proprietary and closely guarded.

## Related applications

A few related fields, graphical music visualizers and automatic music categorization, were also considered as possible research areas: unfortunately, neither of them proved viable. Music visualizers, while creating visual patterns based on the audio stream entirely algorithmically, do not require that their output be linked to the input signal in any obvious way. Automatic music classification systems, while providing potentially useful ideas for features to extract from the signal, were also discarded due to the influence of categorization algorithm, usually some form of neural network or other AI trained on huge databases, on the choices made by the designers.

## Music theory

Initial failure in finding information on existing approaches to solving the research question led to research on the fundamental disciplines it related to: music as well as signal processing theory, psychoacoustics and the physical bases of musical sound.

An initial overview of the basics of music theory seemed promising for the project: the subset of music referred to as “western music”, the target source material, is highly structured and well codified.  
The realization of western music’s reliance on notes as its fundamental building block, in particular, appeared an essential breakthrough, as it seemed to allow treating the input as a sum of discrete frequency components only allowed to belong to a subset of values, the notes’ frequencies.

This simplistic assumption implied that generating a representation of the input signal equivalent to music notation (MIDI is such an example) by using Fourier analysis would have been a relatively straightforward process; subsequent input analysis would have entirely focused on the music notation equivalent to identify and classify higher order musical components (from chords, bars and phrases all the way up to the whole piece) to be mapped to specific game events.

From this standpoint, the main obstacle to project completion consisted of the massive amount of musical components and their combinations, making their detection and association to game events in meaningful ways an arduous and time consuming task, likely to require much more time than was available.

## Physics of musical sound and psychoacoustics

More in depth research on the nature of musical sound and the way it is perceived revealed a much more complex picture than imagined: while musical notes are associated to a single frequency, the pitched sounds produced by real world musical instruments are not.

The waveform produced by playing a note on a musical instrument isn’t a simple sinusoid, a pure tone; rather, it is the summation of several harmonic partials, sinusoids having frequencies that are integer multiples of a fundamental frequency corresponding to the one of the note being played. The number and relative strengths of these harmonic partials, along with other factors such as the nature of transients, the behaviour of the produced waveform at the beginning and the end of the note being played, are what gives musical instruments their characteristic timbre.

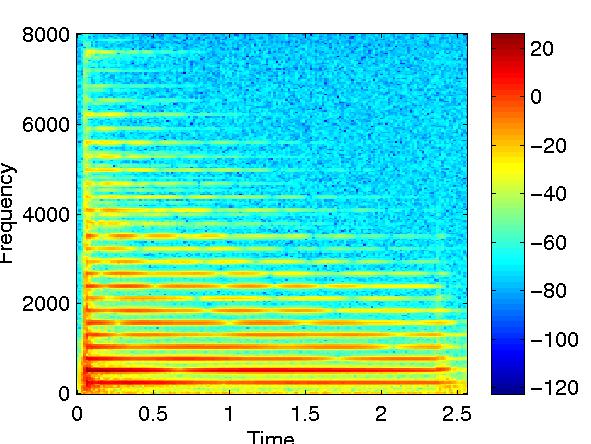


Figure : Piano middle C periodogram

Fundamental frequency 261 Hz. The harmonic partials can be clearly seen as equally spaced horizontal lines. Jarno Seppänen (1999)

It is important to notice that not all musical instruments conform to this model exactly: in the case of some, the first harmonic, having equal frequency to the fundamental one, is absent from the created waveform (the brain is still capable of recognizing the remaining partials as multiples of the fundamental frequency, thus allowing us to perceive the sound as having the expected pitch); others, chiefly gongs and some bells, exhibit partials having frequencies not integer multiples of the fundamental one, while others still (some percussive instruments) produce non-pitched sounds, waveforms entirely devoid of periodicity. (Loy, 2007)

All these factors made accurate and reliable detection of what notes were being played at any given time in a musical performance an arduous task, all but rendering the previously devised approach to answering the research question unworkable.

## Signal processing theory and Fourier analysis

While the previous lines of enquiry had failed to provide viable methods of audio input analysis, they strongly hinted that any such procedure would have had to be based on both the time and frequency domain representations of the signal; Fourier analysis, the determination of a function’s frequency composition, and related signal processing techniques became therefore the new focus of research activity.  
While providing an understanding of the bases and workings of the Fourier transform, as well as the meaning behind the usage of complex numbers in conjunction with it, the information most useful to the project gleaned in the process related to the limitations of applying Fourier analysis to a digitized representation of a signal.

The inevitable information loss occurring as a result of signal digitization (a continuous function in time, the original analogue signal, can only be sampled a finite number of times, all its non sampled values being irreversibly lost) poses a hard limit to the resolution of its Discrete Fourier Transform (DFT): the bandwidth between 0hz and the signal’s Nyquist frequency (half the sampling rate) is divided into n/2+1 frequency bins, where n is equal to the number of samples being transformed.  
The need to not only know the frequency composition of the input signal, but to also track its evolution in time, means the input has to be divided into sections of equal duration, or “windowed”, in turn reducing the resolution of each window’s DFT, as every window comprises less samples than the whole; time and frequency resolution are inversely proportional.

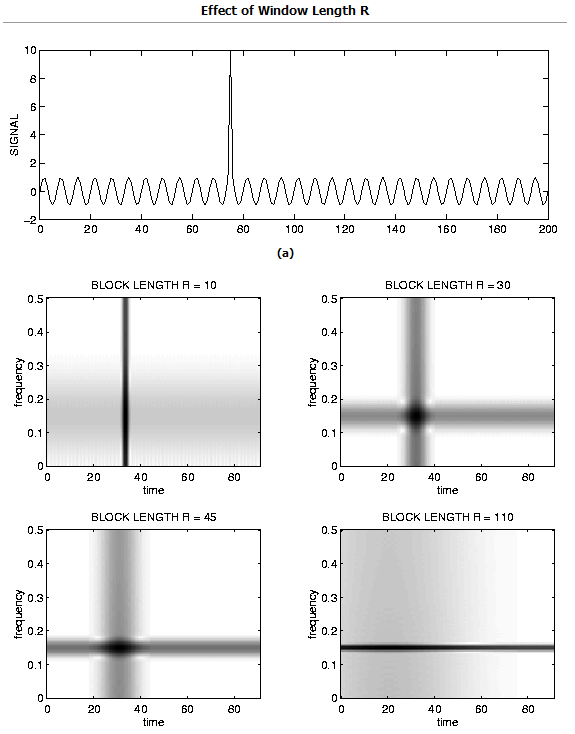


Figure : Time-frequency resolution trade-off

Illustration of window size’s effect on DTFT output. Screen cap from Ivan Selesnick’s DTFT explanation (2005)

In addition to this, applying Fourier analysis, intended for periodic signals, to real world ones carries its own side effects: as the set of samples used by the DFT is interpreted as a period of an infinite signal, any discontinuity at its ends will result in noise, spurious frequencies being added across the spectrum; to counter this, the signal can be multiplied by a windowing function to diminish discontinuities at the ends. Many such functions exist, however there is a significant trade-off to be made: usually, the better a windowing function is at suppressing noise, the more it will increase spectral leakage, making it impossible to distinguish between signal components having similar frequencies.

Having to deal with a much coarser than expected view of the evolution of frequency components of an input signal over time was yet another confirmation of the impracticality of the method originally devised to answer the research question, suggesting instead an approach based on much lower level signal features.

## A suitable experimental reference

The previous conclusion led to a step back in the research process, in order to start again from a clean slate. This new line of investigation caused the rediscovery of the paper by Aallouche et al. (2007) which, in addition to providing a solid theoretical framework for the evaluation of musical games, outlined a viable method for input analysis not relying on the acquisition or generation of a music notation equivalent, going as far as to give specific suggestions on which low level signal features to consider as the most easily discernible by listeners.

In spite of its effectiveness, the method was ultimately rejected due to its heavy reliance on the MP3 psychoacoustic model: the research question was more general in scope than the provided implementation, concerning itself with feature extraction from digital audio signals in general as opposed to any particular encoding. While popular, MP3 is far from a universal standard for digital audio: the prospect of transcoding any input into said format for analysis was unappealing for both the computational cost it incurs and the degradation in audio quality and fidelity to the original signal intrinsic to the process.

The basic analysis principles outlined by Aallouche et al. (2007), and even more so their choice of noteworthy signal features, were however found to be a viable basis for an original implementation; the suggestion of tempo being the most easily discernible feature of musical signals to listeners, in particular, single-handedly determined the course of subsequent research.

## Tempo detection as basis for input analysis

Investigation in the fields of procedural tempo detection and beat tracking revealed no single theory as the universally accepted standard, turning up several completely unrelated analysis methodologies instead.

After careful evaluation, the beat tracking method described by Sethares and Morris (2005) was chosen over the alternatives: the choice was made on the grounds of its intuitiveness and comparative ease of implementation as well as its usage of many of the features shown by Aallouche et al. (2007) to be the best candidates for feature extraction and gameplay events mapping. Determining the song’s tempo this way allowed a significant reduction of computation time, in turn diminishing the amount of time users would have to wait to start playing after providing the input track.

The method outlined by the authors comprises several steps: first, the source digitized audio signal is divided into a number of fixed length windows; each window is analyzed in the time and frequency domains to compute four individual measurements: power, spectral centroid, spectral spread about said centroid and group delay.

The first three features are highly representative of the way the audio signal is perceived in their own right: power is an approximate measure of signal loudness, centroid frequency relates to the sound’s perceived brightness (Schubert and Wolfe, 2006), while spectral spread can be meaningfully interpreted as a measure of the number of notes or instruments playing concurrently.

Feature values are stored into four separate arrays, each element corresponding to a window of the original signal and having their same order; the four arrays are analyzed to compute four more arrays holding the delta values between adjacent elements in the original ones (the first element of such arrays is assumed to be zero). Following this, the power delta, frequency centroid and spread delta and group delay arrays are considered as discrete time functions: these are taken to represent independent individual observations of the input signal’s underlying rhythmic structure; in particular, a spike in any of these functions is taken as chance of a beat occurring at the corresponding time.

The four rhythmic tracks are interpolated using a particle filter, an advanced statistics technique, in order to predict initial beat location, time between beats and any change in its value as time passes. Using the average time between beats, estimating the number of beats per minute for each window is trivial.

By merely following the method outlined above, a program computes more than enough features for each signal window to feed procedural generation techniques with values representing highly discernible features of the input.

# Methodology

## Overview

Finding a suitable theoretical framework for the creation of a musical signal analysis algorithm concluded the literature review based research phase; before work could be started on implementing a demonstration, however, how to evaluate its performance according to the research question had to be considered.

The original idea of rating the created program on its gameplay was, on further analysis, rejected on two different grounds: first of all, doing so would have been impractical, as it would have required running an experiment where a suitably sized sample was allowed to play with the game and polled for opinions on its efficacy in creating discernible relations between music and created content. Far more significant, however, was the realization of such an evaluation method being intrinsically flawed: arbitrary game design choices, on which the procedural generation routines would inevitably be based, would more than likely have a greater effect on players’ perception of the created application than its efficacy in analyzing the input.

As a consequence, two separate applications were created: one an analysis tool capable of graphing values and evolution over time of the signal features deemed to be interesting in an easily readable way, the other a bare bone 2D scrolling shooter based on the same analysis code which uses the results to procedurally generate enemy waves as the song progresses. With the game left as a mere proof of concept, discussion of results focused entirely on the analysis tool’s output.

Due to the analysis process’s complexity and interdependence of its components, it was implemented incrementally: any functionality would be debugged and its output checked until reasonable certainty of its correctness could be established before going forward; in many cases, this meant creating debugging and visualization tools for the purpose. Starting the proof of concept game implementation after completion of the analysis tool and designing both to share parts of their source code can be seen as an extension of this practice.

## Data structures

Four main data structures are used throughout the audio analysis algorithm: in order to facilitate explaining its workings they are described here.

The PCMWAV class holds a complete representation of a digitized audio waveform in 16 bit uncompressed PCM format: the analog waveform is sampled at regular intervals and each sample has its value quantized to the closer of the 2^16 values possible and stored in an array. PCM is usually used for audio manipulation by computer programs, playback by audio devices and is output by audio codecs when decompressing other formats.  
Other than storing sample data in a dynamically allocated array, the class holds information about the format of the sound itself, such as its sample rate and number of channels; even when multiple channels are present, all data is stored in a single array by interleaving samples (ex. ch1sam1, ch2sam1, ch1sam2, ch2sam2...).  
While Windows provides its own PCM manipulation and storage facilities, a choice was made to create an original implementation in order to better audit decoding stage output, to have greater control over sample data and to be able to programmatically create playable sounds.  
The class also provides functionality to load from and save to standard .wav files in order to play back its contents from any compatible, known to be working correctly, media player software (such as Windows Media Player).

The FFTWOutput class holds a complete representation of a periodogram, a sequential series of discrete Fourier transforms, each corresponding to a particular window of a digitized signal. Each discrete Fourier transform is stored as a series of as many complex values (structs holding two floats) as there are frequency bins corresponding to the number of input samples.  
As with the previous class, FFTWOutput also stores additional information such as sample rate, number of windows and samples per window (needed to retrieve specific complex values, as they are all stored in a one dimension dynamic array), window length in milliseconds and samples and window overlap.

The AudioFeatures class holds the values of signal features of interest (Power, group delay, frequency centroid and spread about it) for every window the signal has been divided into, together with the variation of their value, the “delta”, compared to the one of the window coming before it.  
Again, the class also holds information about window length in samples and in milliseconds, together with the signal’s nyquist frequency (half the sample rate): this is needed as frequency centroid and spread are stored in normalized form, varying between 0 and 1, the nyquist frequency itself.

As typical BPM values for music can reach around 300, multiple signal windows need to be considered for tempo determination; such window groupings have been called “audio sections” in the current implementation.  
All information related to audio sections, chiefly each section’s BPM value together with the average power, frequency centroid, spread and beat location windows, is stored in the homonym class; again, metadata detailing the number of sections and their length is also stored.

## Input conversion to LPCM

Given the audio analysis algorithm’s independence from any specific encoding, an effort was made to create an implementation supporting as broad a range of media formats as possible, in order to both increase its usefulness and ease testing.

Realizing the futility of attempting an original implementation of multiple media codecs and container formats, a decision was made to use existing software libraries for the purpose.  
FFMpeg, in particular its libavcodec and libavformat libraries, were the obvious choice, due to their open source (GPL) licensing, proven quality (the library is the de facto standard for open source media playback and encoding), active community, good documentation availability and constant development.  
The two advantages offered by FFMpeg were the capability of decoding a huge range of media formats using a single, relatively short, code path and its native outputting of decoded audio stream in 16 bit LPCM, requiring no further conversion.

Audio decoding formed the first functional stage of the analysis tool: the program would prompt the user to input the path of a media file encoded in one of the supported formats, use FFMpeg to fully decode it in system memory and then write the LPCM stream to disk, with appropriate headers, as a standard .wav file. Existing media playback software known to work correctly (Windows Media Player or Media Player Classic in my testing) was then used to compare input and output for decoding errors; such an approach ensured that analysis was carried out on an actual, accurate representation of the original input signal.

## Windowing

As the DFT computes the frequency composition of the entirety of its input, it is necessary, in order to determine the evolution over time of a signal’s frequency components, to divide it into parts to be individually transformed, a process known as windowing. While straightforward at a first glance, a number of factors have to be considered when performing the process, given their huge influence on the output of subsequent Fourier analysis.

First in order of prominence is each window’s temporal size, directly proportional to the amount of samples to be fed to the DFT at any sample rate and therefore to its output’s frequency resolution; obviously, this is also inversely proportional to the temporal resolution of the set of DFT outputs for the input signal, so a balance must be struck between the two.

Closely following window size is the choice of windowing function: as the DFT considers its input as one period of an infinite periodic function, any discontinuity at its ends (as is the case for most real world signals) results in spurious frequency components, essentially noise, being added to the output. To minimize this, the input signal can be multiplied by a windowing function to have it fade to silence at its ends whatever its actual behaviour, removing discontinuities introduced by windowing; the trade-off for such a gain is an increase in spectral leakage, where individual input frequency components contribute to bins other than their corresponding ones. Many different windowing functions have been created, each representing a different compromise between noise suppression and spectral leakage.

While only essential in applications where the DFT’s output needs to be reconstructed as a signal, window overlap, the number of samples included both by a window and the following one over window size in samples, still deserves consideration. Higher overlap provides more data points, discrete Fourier transforms of different windows, to be interpolated to achieve greater analysis accuracy at the cost of computation time.

Finally padding, appending a number of zero values to a window when using it as input for the DFT algorithm, influences both its output and speed. Fast Fourier Transform (FFT) algorithms achieve maximum performance when operating on inputs consisting of numbers of elements that are a power of two, often being dramatically faster than in other cases: appending enough zeros to a window to grow its size to the closest power of two for analysis cuts down on computation time while leaving the signal’s frequency content unchanged, if an appropriate windowing function is used.  
As said previously, increasing the number of samples fed to the DFT increases its output resolution: in the case of padding, however, no additional information is given to the DFT algorithm; as a consequence, the output is merely a higher resolution interpolation (a rather expensive one in computational terms) of the one that would be obtained without padding, frequency components too close to be distinguished originally still irresolvable.

As no specific guidance on suggested values for these parameters could be found, the analysis algorithm was designed so that they could be easily modified, to allow tweaking in order to improve results.

## Fourier analysis

As was the case with format decoding, a decision was made to rely on existing, tested libraries to perform the discrete Fourier transform; after some research, FFTW appeared to be the best choice due to its GPL licensing, performance and widespread use.

While the library’s usage is straightforward, with relatively little that could go wrong from a programming perspective, its reliance on the output of windowing procedures, impractical to debug on their own, required extensive testing and output sanity checking.

To do so, a library for image manipulation, EasyBMP, was used: as the output of DFTs of multiple signal windows, a periodogram, can be thought of as a 2D matrix of values (individual columns storing the values of individual frequency bins for one window and individual rows storing the values of a frequency bin over time or vice versa), visualizing it as a picture, using a one to one mapping between matrix elements and pixels and mapping pixel colors to element values, seemed the quickest and most effective choice.

Initial tests relied on the ability of the PCMWAV class to procedurally generate sounds: pure tones having known frequency behaviour over time would be analyzed, their periodograms saved do disk and opened with an existing image editor to ensure they conformed to expectations (single horizontal lines for stationary tones, diagonal ones having inclination proportional to the frequency change over time otherwise). After the most serious issues were found and corrected, real world musical signals were analyzed and the periodograms compared to the output of WaveSurfer, a program performing a similar function.

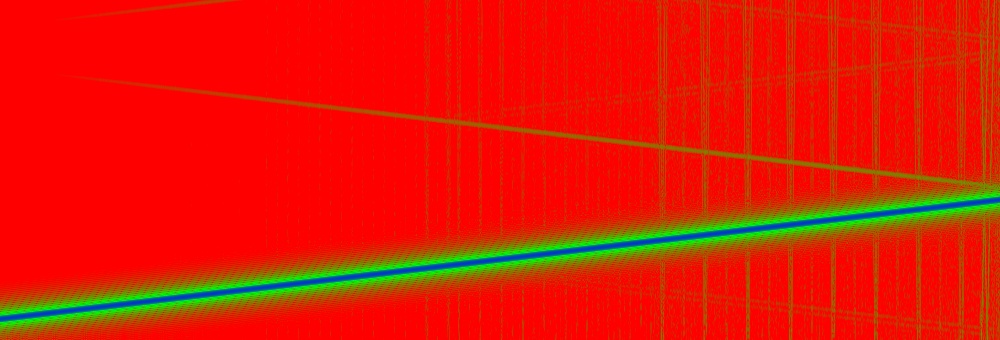


Figure : Section of rising sine wave periodogram

Blue and green diagonal line represents the wave’s frequency rising as time passes; other green patterns are a result of quantization noise. Red signifies no energy is present in the corresponding frequencies

It was during this process that fine tuning of windowing parameters took place: the most suitable window size was found to be 10ms with a 50% overlap and the Hann windowing function was chosen over the alternatives for its good noise suppression capability and modest impact on spectral leakage.

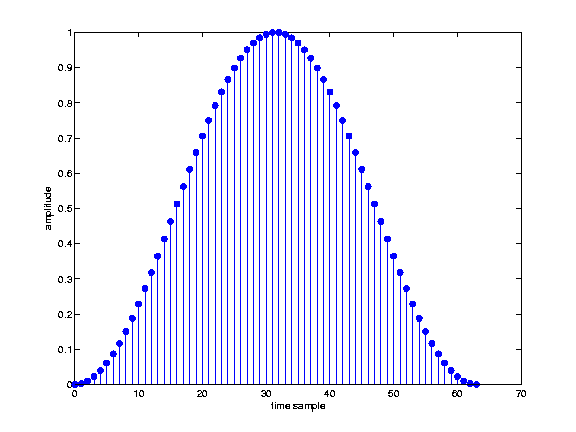


Figure : Hann windowing function graph

It is important to notice that the function can be arbitrarily scaled on the x axis to fit with the window length used. Jarno Seppänen (1999)

## Rhythmic tracks computation

After verification of the DFT’s output reliability to a satisfactory degree was completed, work began on calculating the rhythm tracks described by Sethares and Morris: rate of change in signal power, position of spectral centroid and spread about it as well as group delay. As both input functions, the signal in time domain and its periodogram, are discrete, said rates of change are satisfactorily approximated by the variation of the relevant parameters between adjacent windows.

The computation process is therefore divided into two separate steps: the values of the parameters of interest are calculated and then their deltas are determined.

Signal power for a window is calculated as the sum of the squared values of all samples comprising it before any windowing function is applied; in addition to being used for beat determination, the value is taken as an approximation of the perceived loudness of the window.

Spectral centroid was found to be an ambiguous concept: while always representing a measure of central tendency, it can either be defined as the “center of mass” of a spectrum or as the frequency dividing it into two regions having equal power.  
In the first case, the centroid’s position is calculated as the weighted mean of all frequency bins’ center frequencies with their power as the weight, while in the second it is the frequency F for which ∑ x(n) 0<=n<F is equal to 0.5\*∑ x(n) 0<=n<NF, where x(n) is the power of the nth frequency bin and NF is the signal’s nyquist frequency.

With the authors not suggesting any particular centroid definition, both were tried and their impact on the output observed before deciding in favour of the first one.  
As suggested by Schubert and Wolfe (2006), centroid frequency is taken as a measure of the window’s perceived brightness.

Spectral spread is calculated as ∑ |Cf(n)-Fc|\*x(n) with 0<=n<NumBins, where Cf(n) is the centre frequency of the nth bin, Fc is the frequency of the centroid and x(n) is, again, the power of the nth frequency bin. The value is taken as an indirect and very approximate measure of the number of notes and/or instruments being played at the same time.

The variation of the three parameters between a window and the one preceding it is then calculated for the whole signal: the resulting sets of values are treated as discrete time functions, independent observations of the input’s underlying rhythmic structure. Spikes in the functions’ values denote the occurrence of high energy events which, observing that most of such events occur at or near the beat in musical signals, is in turn taken as an indication of a beat occurring at that instant (Sethares and Morris, 2005). Being derived from the same input, the observations are used to corroborate each other, with beats only acknowledged in case of significant agreement (multiple functions peaking at roughly the same time).

In spite of all efforts made, the fourth rhythm track, group delay, could never be properly computed: research into its meaning as well as phase unwrapping, the process on which it depends, failed to provide a clear enough understanding of either. An attempt at its determination was nonetheless made, but results were so far out of line with the other rhythm tracks and Sethares and Morris’s results that the whole implementation was deemed structurally flawed and its products ignored.

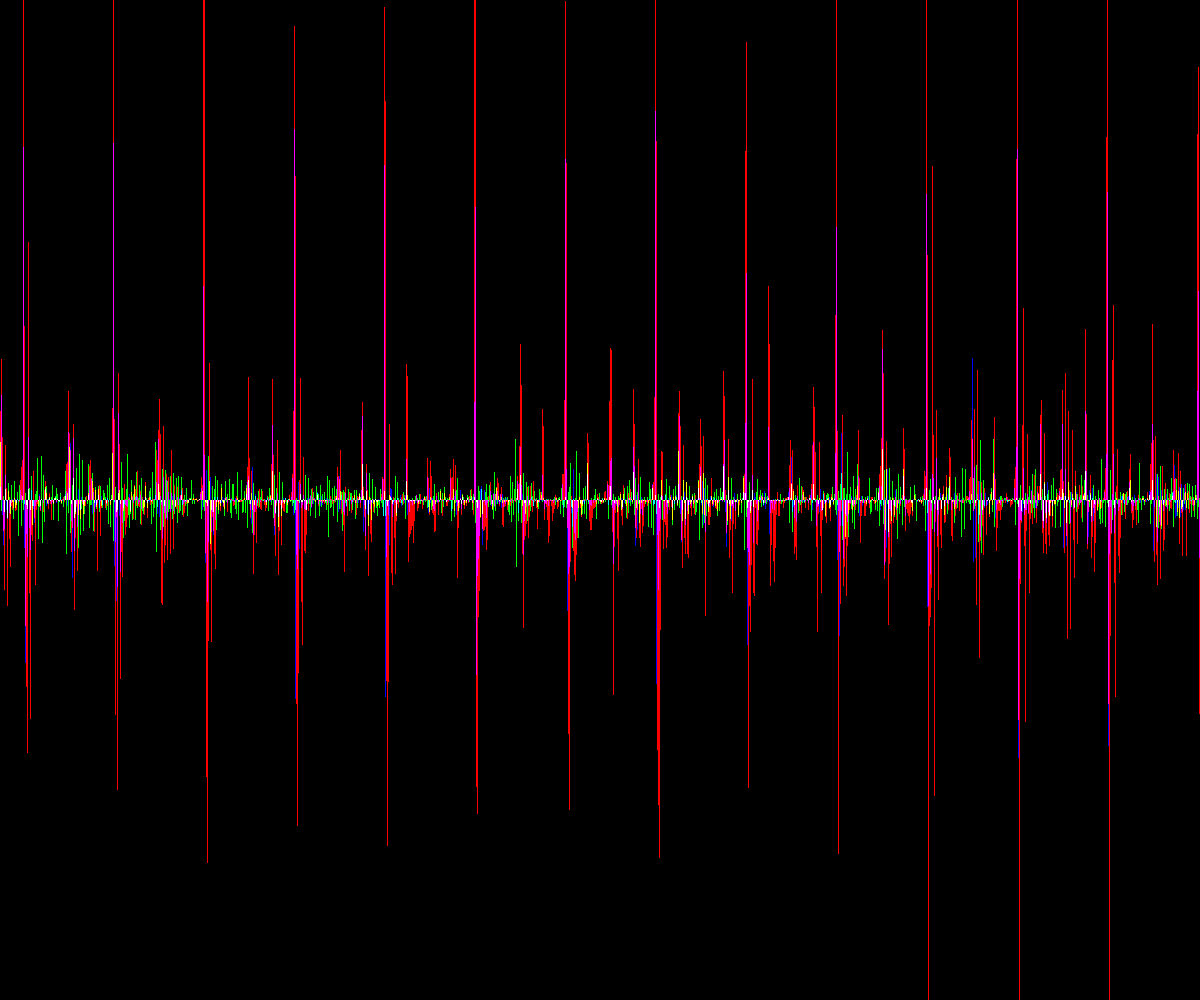


Figure : Rhythm tracks graph section

Computed from track Route 66 by Depeche mode. Red: Power delta. Green: spectral centroid delta. Blue: spectral spread delta. The three tracks tend to pike concurrently at beat locations

## Need for an alternative to particle filters

After completion of the rhythmic tracks computation algorithm, a significant hurdle preventing further development was encountered: the paper by Sethares and Morris (2005), which up to that point was used as a reference, invoked the usage of particle filters as the final step for tempo determination. Research into the technique failed to make its meaning and usage any clearer: in contrast to previous occurrences of such a situation, however, this wasn’t due to an inability to find enough relevant information; rather, it was due to its reliance on notions far outside my knowledge.

After consultation with both the project’s technical supervisor, Matthew Bett, and other lecturers, the impossibility of implementing a particle filter as described in the paper was fully revealed: the technique relies on advanced statistics concepts which would have required more time than was available to be researched, especially considering the background learning such an endeavour would have entailed.

Attempting an original solution to the problems of interpolating between the three rhythm tracks and determining the input’s BPM, while risky and bound to be inferior to the published method, was seen as the only practical course of action.

## Rhythmic tracks interpolation

The first problem posed by deviating from the implementation Sethares and Morris described was how to interpolate between the three rhythm tracks previously computed.

A solution was reached by considering the nature of the tracks themselves, in particular their being representations of separate observations of the same rhythmic structure, detailing the probability of beat occurring at a certain time: a simultaneous peaking of all three could be considered a very strong indication of a beat occurring.  
Using this notion as a base, the interpolation algorithm was created: as a first step, each rhythm track is normalized by dividing its elements by the modulus of the one having greatest absolute value, effectively rescaling the function’s codomain to lie between -1 and +1, in order to ensure the equal weighting of rhythm tracks in the interpolation process.

While it seemed natural for the interpolation function to approximate the summation of the rhythm tracks, in order to have it spike at a beat’s occurrence, it was also clear that determining it as a simple member by member addition wasn’t a suitable approach: different tracks would peak at slightly different instants when a beat occurred, producing, instead of the desired spike, a lower and wider plateau much less distinguishable from background noise.

To compensate for this, each value of the interpolation function is calculated as the sum of the values of the three tracks over a small region surrounding the output’s location multiplied by a bell curve centred on it: for example, Interpolation[10] = A\*(Track1[8]+ Track2[8]+ Track3[8]) + B\*(Track1[9]+ Track2[9]+ Track3[9]) + C\*(Track1[10]+ Track2[10]+ Track3[10]) + D\*(Track1[11]+ Track2[11]+ Track3[11]) + E\*(Track1[12]+ Track2[12]+ Track3[12]), where A,B,C,D,E = a\*e^(x^2/c^2) for constant values of a, e and c (all greater than 1) and with x equal, respectively, to -2,-1,0,1,2.

The choice of a bell curve as well as its scale and variance parameters were partially arbitrary and partially empirically determined by tuning observing output behaviour; while suboptimal, such an approach was felt as the only feasible one and did provide a usable interpolation.

The final step taken while tuning was disregarding the spectral centroid frequency, due to it not being as clear an indicator of beat locations as the other two tracks: such a decision resulted in a clearer interpolation with more marked peaks and less background noise.

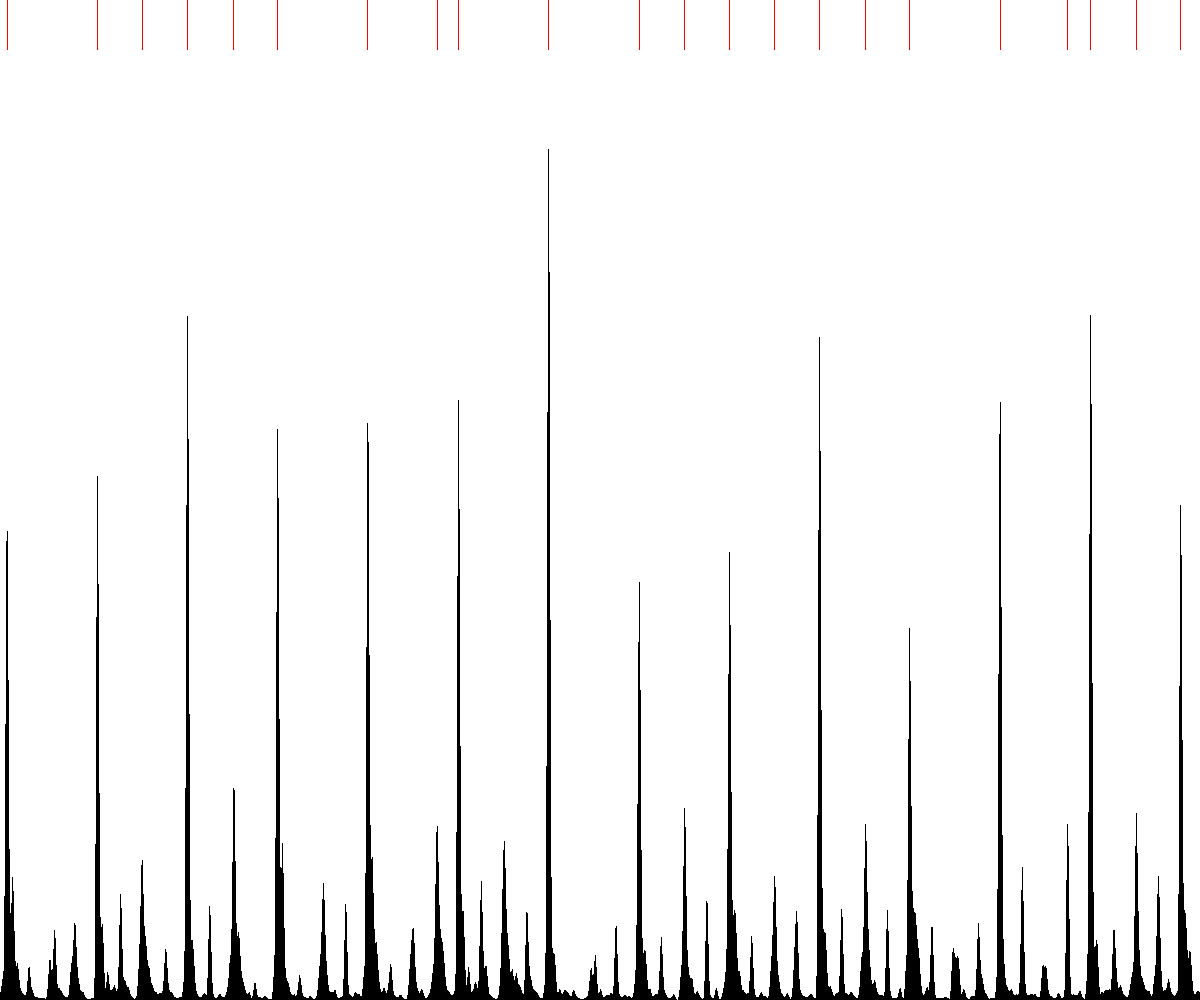


Figure : Rhythm tracks interpolation graph section

From track Route 66 by depeche Mode. Candidate beat locations are marked with red traits.

## Beat detection

The assumption underlying the whole tempo detection process, that beats be manifested as high energy events in the signal, implies that beat detection needs to be performed over a period of time, in order to compute an average energy level to which compare events for qualification as highly energetic or otherwise.

Individual 10ms windows, used as approximation of instants for frequency content determination, are therefore unsuitable for the beat detection process; the signal is instead divided into a number of much longer, non overlapping sections for which beat locations are determined.

As it is conceivable that tempo could change over the course of the song, the length of said sections is a compromise between having enough data, in the form of interpolation windows values, to compute an average and look for spikes reliably, and keeping them brief enough to assume tempo to be constant through their duration; in the current implementation, sections are 2 seconds long.

Starting from these assumptions, beat locations determination is rather straightforward: for each 2 seconds section, the mean and standard deviation of the values of the interpolation function that fall within it is calculated; peaks within the section (values of the interpolation function surrounded by lower ones) have their magnitude tested against the mean plus the standard deviation multiplied by a custom parameter: if they are greater, the corresponding 10ms window is marked as a beat location.

The custom parameter’s value effectively determines the beat marking algorithm’s sensitivity and is determined by tuning, with the goal of removing as many background noise peaks as possible while making sure all of the ones indicating beat occurrences are marked.

## Determining tempo in BPM

The tempo determination algorithm acts on each 2 seconds signal section independently using the previously determined peak locations as input: their list is analyzed and all intervals between members are calculated and stored (information about duration and start and end beat location is stored for each interval).

Following this, list of occurrences of different interval durations are computed by going through the interval list and, for each member having a duration not encountered before, creating a new occurrence and including all intervals having duration equal to the list originator plus or minus a tolerance factor (currently set at 10% of the originator's length). Every element of the duration occurrences list is then again analyzed to compute its longest interval chain (an interval chain is a series of intervals adjacent to each other or, in other words, where each apart from the first begins at the exact location the previous ends), its length in both samples and number of member intervals being recorded. While such an approach would seem to require the construction and traversal of a tree, a much simpler method is to use two std::vectors (an unsigned int vector and a vector of unsigned int vectors) and go through each branch to the very end before moving back one level and test other ones without ever having the whole tree constructed.

The focus on chain determination and evaluation stems from an analysis of the method used to attempt at determining tempo by looking at a graph of the rhythm interpolation for the track: rather than just focus on the highest spikes, attention would be paid to chains of interval having roughly equal size repeating one after the other; the number of BPM could be inferred by measuring the pixel width of said intervals.

Following this the algorithm determines a score for each duration occurrence based on the length of the longest chain in both elements and intervals; the duration occurrences list is then arranged using this value from highest to lowest, and tempo in BPM is determined by a function of the average of the durations of the occurrences sharing the highest value.

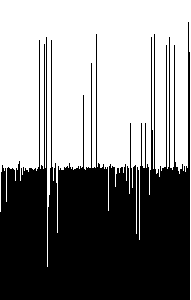


Figure : BPM graph

Track: Route 66 by Depeche mode. X axis represents time (one pixel per 2 seconds signal section), Y axis corresponds to BPM. Most independent BPM determinations agree in the case of this track, creating a plateau running its entire length

Finally, in order to have a more accurate estimation of the input’s tempo, the BPM values computed for every 2 seconds section are used as independent observations in order to determine their most recurring value, in a process similar to the one previously described. In addition to being a trade-off between detail and accuracy, the approach requires input tempo to be constant to be justifiable; due the huge importance placed on correctly determining input BPM and the fact most songs do, in fact, conform to that criterion, the method was adopted in spite of these drawbacks.

## Proof of concept game

After completion of the analysis algorithm implementation, work started on creating a background music reactive game based on it that could serve as proof of concept for audio based procedural content creation.  
To keep the task manageable it was decided to create a new game from scratch, basing all design elements decisions on their suitability to work with the technique, as opposed to attempting to modify an existing game to become background music reactive.

Two main decisions originated from this approach: first and foremost, to make the game a vertical scrolling shoot ‘em up; the genre was favoured over alternatives for its ease of implementation (few and basic game rules, limited physics modelling requirements and ability to cope well with rudimentary bounding circle collision detection) and highly structured gameplay (player interaction is limited to moving a floating ship and shooting, while enemies behave in predetermined ways regardless of user input, to the point of leaving the screen on their own accord after a while), factors contributing to more relaxed criteria to meet in order to ensure playability.  
Secondly, it was decided to apply procedural generation techniques to the creation of levels using fixed building blocks as opposed to any specific in-game art asset: in the context of a shoot ‘em up, this translated into defining where, when and how many enemies to spawn during a play session.

In practice, the created game prompts the user to choose a media file to open and, after a brief pause to allow for analysis and game event determination, puts it in control while playing back the opened audio stream: the W,A,S,D keys are used to move in the four cardinal directions while spacebar is assigned to firing; every few seconds new enemies to be shot down enter and move through the player’s field of view, occasionally firing rounds of their own. This is, however, where the analogies with standard shoot ‘em ups end, as other gameplay element such as player death, game over conditions or a scoring system could not be implemented.

As opposed to completely defining a number of enemy spawn modes and mapping those to arbitrary patterns in the evolution of the computed audio features over time, it was decided to have them play a more direct role by acting as modifying parameters of a single base enemy spawning event repeated at regular intervals (more specifically, the value of their average over a period of 0.25 seconds neighbouring the spawning time is used); while such a choice limited the amount of control that could be exerted over the created levels’ features, it was found to be the only practical one in the time available due to the huge amount of possible combinations.

In the final implementation, the time interval between enemy waves is inversely proportional to the song’s tempo, usually ranging between 1 and 2 seconds; each wave is comprised by a variable number of two kinds of enemies: blue triangular fighters, which appear on the top or bottom of the screen and quickly fly to the opposite direction, and green circular bombers, which can appear from any direction and move through the approximate centre of the screen to the opposite one firing bullets all around them every couple of seconds.

The number of fighters spawned is proportional to the average spread at the time, while average power is used to determine the number of bombers; both kinds of enemies’ flying speed is directly proportional to the song’s tempo in keeping with its significance as a measure of the song’s speed and to give the parameter as much prominence as possible. Enemy initial positioning is a pseudorandom function of the average value of the spectral centroid, with constraint in place to ensure fighters are placed in recognizable formations.

In spite of the chosen method’s apparent simplicity, significant issues were encountered when implementing and testing the procedural level generation algorithm. Due to the huge (more than 1 order of magnitude) variance of audio feature values across different tracks, algorithm tuning, mainly the choice of proportionality coefficients for the above described techniques, proved problematic, as a balance had to be sought between meaningful event creation from different inputs and creation of different game events over the course of the inputs themselves. This is manifested by the program generating entirely meaningless output or empty levels in response to certain input tracks.

# Results

Due to the issues encountered during development of the proof-of-concept game and to the impossibility of objectively evaluating procedurally generated levels on the grounds of their suitability as representations of musical content, a choice was made to focus on the analysis algorithm’s performance in input feature determination as a measure of project successfulness.

As tempo determination relies on the evolution of the other parameters over time and its product is the only value which can be checked against independent observations (in the current test, the output of MixMeister BPM Analyzer), its accuracy has been used as benchmark for the whole analysis algorithm.

BPM: beats per minute as determined by the created program  
REF: beats per minute as determined by the reference implementation  
ERR: absolute error, value of difference between BPM and REF  
ERR%: error percentage over REF

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Title** | **Artist** | **Genre** | **BPM** | **REF** | **ERR** | **ERR %** |
| Test 120 BPM cuetrack | NA | NA | 120 | 120 | 0 | 0.00 |
| Marble Zone | Sega | Chiptune | 133 | 133 | 0 | 0.00 |
| Route 66 (Beatmasters Mix) | Depeche Mode | Synthpop | 131 | 132 | 1 | 0.76 |
| Senor Burns | The Simpsons | Soundrack | 100 | 101 | 1 | 0.99 |
| Evangelion Op | Gainax | Anime | 126 | 128 | 2 | 1.56 |
| When I grow up | Pussycat Dolls | Pop | 116 | 118 | 2 | 1.69 |
| The real slim shady | Eminem | Hip-hop | 104 | 104 | 0 | 0.00 |
| Pop Star | Hirai Ken | J-Pop | 137 | 140 | 3 | 2.14 |
| Ghostbusters Theme | Ray Parker Jr. | Soundrack | 115 | 116 | 1 | 0.86 |
| Oakland | Charlie Hunter | Jazz | 65 | 86 | 21 | 24.42 |
| Nantes | Beirut | Folk | 68 | 131 | 63 | 48.09 |
| Into The Pit | Testament | Trash Metal | 202 | 100 | 102 | 102.00 |
| Same Axe | John Scofield | Jazz | 95 | 110 | 15 | 14.00 |
| High | Blue Nile | Pop | 130 | 131 | 1 | 0.76 |
| Everlong | Foo Fighters | Rock | 157 | 158 | 1 | 0.63 |
| We Love The Trance | DJ MaWi | Psytrance | 128 | 145 | 17 | 11.72 |
| I Like Birds | The Eels | Alternative Rock | 136 | 138 | 2 | 1.45 |
| Spiritual Groove | Antoine Dufour | Acoustic Alternative | 131 | 101 | 30 | 29.70 |
| Lakme-Flower Duet | Danielle Millet | Opera | 95 | 116 | 21 | 18.10 |
| Metal Slug Assault | Nazca | Game | 147 | 148 | 1 | 0.68 |
| Full Metal Cars | Daniel | Eurobeat | 317 | 157 | 160 | 101.91 |
| Balalaika Vodka & Rock 'n Roll | Mad Cow | Eurobeat | 160 | 162 | 2 | 1.23 |
| Gee | SNSD | K-Pop | 198 | 100 | 98 | 98.00 |
| Message in a bottle | The Police | Rock | 150 | 151 | 1 | 0.66 |
| Less Talk more rokk | Freezepop | Rock | 124 | 124 | 0 | 0.00 |
| Rock this Town | Stray Cats | Rock | 200 | 100 | 100 | 100.00 |
| Computer City | Perfume | J-Pop | 118 | 131 | 13 | 9.92 |
| Still Alive | GLaDOS | Game | 118 | 120 | 2 | 1.67 |
| Britannia's Hymn | Sunrise | Anime (classical) | 126 | 125 | 1 | 0.80 |
| Yuusha-oh Tanjou | Sunrise | Anime | 159 | 163 | 4 | 2.45 |

Table : Analysis results comparison

Comparison between the project’s BPM determination algorithm and MixMeister BPM analyzer, a third party application performing the same function.

In spite of the many approximations employed in developing the algorithm, its performance was found to be remarkable: more than its absolute success rate (the amount of tracks evaluated is too small to constitute a comprehensive, statistically significant test), its paralleling of human listener behaviour was noteworthy. The algorithm would flawlessly determine tempo for tracks obvious to a human listener (cue tracks as well as ones with a clear drum line), its accuracy decreasing as perceived difficulty in determining tempo increased; in particular, classical music, syncopation and tracks having most recognizable rhythmical elements differing from actual tempo would produce the worst results. The behaviour parallels that of a listener possessing no or limited musical knowledge (up to the point of sometimes detecting a tempo that is twice or half the actual one) and is the same (to a much greater extent, unfortunately) as the one exhibited by the other tempo determination algorithm.

# Conclusion

## Summary

The present research set out to determine to what extent musical audio signals could be used as input for procedural generation techniques in the context of a game application.  
To reach an answer, the problem was divided into a number of stages to be treated separately, each progressively building on the solutions of the previous ones and at the same time constituting a higher degree of audio signal utilization in the context of procedural content creation techniques.

First of all, whether a musical audio signal could be effectively described by a small number of numerically expressible features, and if so what these features would be was considered: literature research showed that to be the case and led to the choice of tempo, power, spectral centroid and spectral spread to synthetically represent musical content.

Following this, how to compute the chosen features’ values from a digitized representation of an audio waveform was considered: further literature review coupled with a degree of experimentation and original approaches led to positively answer the question with the creation of an audio analysis algorithm, the efficacy of which has been discussed in the previous section, for the purpose.

Finally, the feasibility of meaningfully expressing the computed values through procedural content creation in a game context was considered: in an attempt to answer the question, a proof of concept background music reactive top down scrolling shoot ‘em up was created. While the process failed to unequivocally confirm this last hypothesis due to the impossibility to objectively evaluate its output, it constitutes a strong hint at its practicality. At the same time, attempting to create a background music reactive game highlighted the main obstacles to its realization: difficulty in balancing gameplay and a need to compensate for the huge variance of parameter values across tracks while retaining enough sensitivity to distinguish between different regions of the signals themselves.

To summarize, it is believed that the present research proved the suitability of audio signals as input for procedural generation techniques by demonstrating the existence of a small set of numerical features summarizing their key perceptual aspects and by developing a method for their determination. That these values could be meaningfully expressed via in game content could only be suggested, however a number of areas having margin for improvement with further investigation were identified in the process.

While basing procedural content creation techniques on user supplied musical tracks has the potential to broaden their usage spectrum, in turn allowing a reduction of development costs, the process was found to be more complex than anticipated: even with a set of descriptive numerical features significant design effort is required to produce useful results.

## Future work

Project development required a number of compromises: the areas outlined below in particular could be improved or expanded upon.

While the current tempo detection algorithm postulates the input track as having constant BPM, this needs not be the case: in particular, following Sethares and Morris’s (2005) method and implementing a particle filter should overcome this limitation and allow for much more accurate results.

The current implementation uses power as a measure of loudness: while true in principle, this fails to take into account the different sensitivity of the human ear across the audible spectrum of frequencies; using more accurate loudness estimation would be preferable.

A significant hurdle encountered when implementing the proof of concept procedural generation algorithm was its tuning: a comprehensive analysis of as large a sample of tracks as possible, evaluating at a minimum average value and standard deviation for audio features within single tracks, within tracks belonging to the same musical genre and within the whole sample would constitute invaluable guidance in the process.

The FFMPEG file handling routines only support multi-byte string encoding, preventing files having paths including international characters from opening; enhancing the program to fully support Unicode would greatly increase its usefulness.

# References

Fabrice Bellard and the FFmpeg project. 2008  
FFmpeg Revision 15986 [Software library]  
[Source link](http://ffmpeg.arrozcru.org/builds/)

Matteo Frigo and Massachusetts Institute of Technology. 2003  
FFTW Version 3.2.1 [Software library]  
[Source link](http://www.fftw.org/download.html)

Paul Macklin and The EasyBMP Project. 2006  
EasyBMP Version 1.06 [Software library]  
[Source link](http://easybmp.sourceforge.net/download.html)

Khalid Aallouche, Homam Albeiriss, Redouane Zarghoune, Juha Arrasvuori, Antti Eronen, Jukka Holm. 2007  
Implementation and Evaluation of a Background Music Reactive Game  
Paper presented at: The 4th Australasian Conference on Interactive Entertainment  
Melbourne, 3-5 December

Loy, Gareth. 2007  
Musimathics, Volume 2: The Mathematical Foundations of Music  
Cambridge, MA, MIT Press

William A. Sethares, Robin D. Morris, James C. Sethares. 2005  
Beat Tracking of Musical Performances Using Low-level Audio Features [Online]  
[Source link](http://eceserv0.ece.wisc.edu/~sethares/paperspdf/beatrack2005.pdf) [Accessed 06/02/2009]

Robin D. Morris, William A. Sethares. 2002  
Beat Tracking (or, can the computer clap along with the music?) [Online]  
[Source link](http://eceserv0.ece.wisc.edu/~sethares/paperspdf/bayes2002) [Accessed 16/02/2009]

Emery Schubert, Joe Wolfe. 2006  
Does Timbral Brightness Scale with Frequency and Spectral Centroid? [Online]  
[Source link](http://www.phys.unsw.edu.au/~jw/reprints/SchubertWolfe06.pdf) [Accessed 19/02/2009]

Danc. 10/02/2007  
Content is Bad [Online]  
[Source link](http://lostgarden.com/2007/02/content-is-bad.html) [Accessed 11/11/2008]

Shane Patterson. 2008  
The History of Music Games [Online]  
[Source link](http://www.gamesradar.com/pc/f/the-history-of-music-games/a-2008060393437857045/g-20080228162145956077): [Accessed 10/12/2008]

Dylan Fitterer. 2008  
Audiosurf [Online]. PC. Valve Corporation.

NanaOn-Sha. 1999  
Vib-Ribbon [Disk] Sony Playstation. Sony Computer Entertainment

Codemasters. 2006  
Dance Factory [Disk]. Sony Playstation 2. Codemasters

Pentavision Entertainment. 2004 – 2008  
DJ Max (series). [Disk, Online] PC, Sony Playstation Portable. Pentavision Entertainment

Konami. 1998 – 2009  
Pop'n Music (series). [ROM, Disk, Cartridge] Arcade, Sony Playstation, Sony Playstation 2, Sega Dreamcast, Nintendo Game Boy Color.

O2Media Inc.  
O2Jam (series). [Online] PC. 9you, e-Games

MixMeister Technology, LLC. 2007  
MixMeister BPM Analyzer. [Online] PC  
[Source link](http://www.mixmeister.com/bpmanalyzer/bpmanalyzer.asp) [Accessed 26/04/2009]

Jarno Seppänen. 1999  
Piano middle C spectrogram Graph [Online]  
[Source link](http://www.cs.tut.fi/sgn/arg/intro/basics.html) [Accessed 30/03/2009]

Ivan Selesnick. 2005  
Short Time Fourier Transform [Online]  
[Source link](http://cnx.org/content/m10570/latest) [Accessed 31/03/2009]

Dryicons. 2008  
Croquette part 3 icon set [Online]  
[Source link](http://dryicons.com/free-icons/preview/coquette-part-3-icons-set) [Accessed 17/04/2009]

# Bibliography

Scott Jarrett, Holly Day. 2008  
Music Theory For Dummies  
Hoboken, NJ, Wiley Publishing Inc.

Michael Pilhofer, Holly Day. 2007  
Music Composition for Dummies  
Hoboken, NJ, Wiley Publishing Inc.

John R. Pierce. 1992  
The Science of Musical Sound  
New York, W.H. Freeman and Co.

Niall Muldoon. 2008  
The Animation of Speech via Signal Processing  
BSc (Hons) Dissertation, University of Abertay Dundee

Geoffroy Peeters. 2005  
Tempo Detection and Beat Marking for Perceptual Tempo Induction [Online]  
[Source link](http://www.music-ir.org/evaluation/mirex-results/articles/tempo/peeters.pdf) [Accessed 23/01/2009]

George Tzanetakis. 2005  
Tempo Extraction using Beat Histograms [Online]  
[Source link](http://www.music-ir.org/evaluation/mirex-results/articles/tempo/tzanetakis.pdf) [Accessed 23/01/2009]

Chris Duxbury, Juan Pablo Bello, Mike Davies, Mark Sandler. 2003  
Complex Domain Onset Detection for Musical Signals [Online]  
[Source link](http://www.elec.qmul.ac.uk/dafx03/proceedings/pdfs/dafx81.pdf) [Accessed 23/01/2009]

Simon Dixon. 2006  
Onset Detection Revisited [Online]  
[Source link](http://www.dafx.ca/proceedings/papers/p_133.pdf): [Accessed 26/01/2009]

Geoffroy Peeters. 2005  
Time Variable Tempo Detection and Beat Marking [Online]  
[Source link](http://recherche.ircam.fr/equipes/analyse-synthese/peeters/ARTICLES/Peeters_2005_ICMC_Tempo.pdf) [Accessed 24/01/2009]

Paul M. Brossier 2005  
Fast Onset Detection Using AUBIO [Online]  
[Source link](http://www.music-ir.org/evaluation/mirex-results/articles/all/short/brossier2.pdf) [Accessed 28/01/2009]

Introversion Software. 02/06/2007  
Procedural Content Generation [Online] Gamecareerguide.com  
[Source link](http://www.gamecareerguide.com/features/336/procedural_content_.php?page=1) [Accessed 11/11/2008]

Andrew Doull. 14/01/2008  
The Death of the Level Designer Part One [Online] Blogspot.com  
[Source link](http://roguelikedeveloper.blogspot.com/2008/01/death-of-level-designer-procedural.html) [Accessed 11/11/2008]

Andrew Doull. 14/01/2008  
The Death of the Level Designer Part Two [Online] Blogspot.com  
[Source link](http://roguelikedeveloper.blogspot.com/2008/01/death-of-level-designer-procedural_14.html) [Accessed 11/11/2008]

Andrew Doull. 14/01/2008  
The Death of the Level Designer Part Three [Online] Blogspot.com  
[Source link](http://roguelikedeveloper.blogspot.com/2008/01/death-of-level-designer-procedural_2464.html) [Accessed 11/11/2008]

Andrew Doull. 15/01/2008  
The Death of the Level Designer Part Four [Online] Blogspot.com  
[Source link](http://roguelikedeveloper.blogspot.com/2008/01/death-of-level-designer-procedural_15.html) [Accessed 11/11/2008]

Andrew Doull. 19/01/2008  
The Death of the Level Designer Part Five [Online] Blogspot.com  
[Source link](http://roguelikedeveloper.blogspot.com/2008/01/death-of-level-designer-procedural_19.html) [Accessed 11/11/2008]

Andrew Doull. 28/01/2008  
The Death of the Level Designer Part Six [Online] Blogspot.com  
[Source link](http://roguelikedeveloper.blogspot.com/2008/01/death-of-level-designer-procedural_28.html) [Accessed 11/11/2008]

Eurogamer Network Ltd. 17/10/2006  
Games are already filling 25GB Blu-Ray discs – Harrison [Online] Gamesindustry.biz  
[Source link](http://www.gamesindustry.biz/articles/games-are-already-filling-25gb-blu-ray-discs-harrison) [Accessed 11/11/2008]

Jim Rossignol. 21/07/2008  
Indie Game Developers Enlist Algorithms to Do the World-Building for Them [Online] Wired.com  
[Source link](http://www.wired.com/gaming/gamingreviews/magazine/16-08/pl_games) [Accessed 11/11/2008]

Graham Whelan, Hugh McCabe  
Automatic Building Generation [Online] Graphics and Gaming Group  
[Source link](http://www.gamesitb.com/automaticbuildings.html) [Accessed 11/11/2008]

Graham Rhodes. 04/03/2008  
Procedural Data Generation in Far Cry 2 [Online] Gamedev.net  
[Source link](http://www.gamedev.net/columns/events/gdc2008/article.asp?id=1331) [Accessed 11/11/2008]

Jimmy Thang. 21/02/2008  
GDC 2008: Far Cry 2's Gamble [Online] Ign.com  
[Source link](http://uk.pc.ign.com/articles/854/854167p1.html) [Accessed 11/11/2008]

# Appendix A: Project Proposal

[References and Bibliography Above]

## Introduction

As the cost of developing game application rises with the power afforded by increasingly sophisticated platforms and end users’ expectation of their optimal use, the industry as a whole is looking at ways to automate more and more tasks, in order to reduce the amount of man-hours, and in turn the cost, required for the completion of any given project; as a consequence, research on the field of procedural generation techniques is a hot topic at this time.

While procedural generation can deliver on its cost-cutting promise, it does have downsides limiting its usefulness: amongst these, one of the most prominent is the fact that procedurally generated content, by its very nature, tends to be remarkably homogeneous.  
Assets created by most techniques are variations of a single base scheme: randomly or pseudo-randomly generated numbers are usually used to “seed” the algorithm, specifying how to modify or extend the base scheme to create the output.  
As such, procedurally generated content is most useful as “throwaway” content, assets adding value to the player experience only once and then ignored, but which presence is required for the game world to be believable: examples of this range from trees and foliage to the individual buildings forming a city’s landscape.  
On the other end of the content usage spectrum, procedural techniques fall short: the difficulty or outright impossibility of fine tuning them to the specific needs of single game sections coupled with failure to replace human aspects of content and establish emotional connections with the user push them to the sidelines, usually complementing or acting as a base for human artists tasked with the creation of the game’s key elements (Danc, 2007).

One possible way of bridging the aforementioned emotional gap, allowing procedural content generation techniques to take a major role in the overall user experience creation, could be to base them on input which has an existing emotional connection with the user, while taking steps to make the relationship between the input and resulting output as recognizable as possible. Audio tracks supplied by the player itself should be prime candidates for serving as input, given how people relate to music.

## Research Topic

Focus of the current project will be the creation of a “background music reactive game” or “musically controlled game” as defined by Aallouche et al, 2007: an application that reacts to background music with game event creation instead of adapting the soundscape to the game’s state.

To do so, a number of areas will need to be researched: a first line of inquiry will be the determination of numerically expressible values representative of the way musical audio signals are perceived by listeners, with the goal of singling out the ones bearing the most influence over the process.

A second area of research will relate to the procedures and algorithms needed to compute the values selected in the previous step from a digitized representation of an audio signal. While the capability of doing so in real time during gameplay isn’t part of the project’s design requirements, maximizing efficiency and computational speed will be of paramount importance. Including decoding algorithms for the most common audio codecs in use to offer as broad a file format support as possible will be a related but optional task.

Finally, experimentation will begin to find the mappings between patterns in the feature values extracted from the input signal and game events best suited to making the relationship between what the user sees and hears as clear as possible. To facilitate this process and reduce the chances of unplayable content being created, the application will be designed to minimize the impact of user input on game flow; to avoid generating frustration as a consequence it will conform to the canons of the scrolling shooter genre, its traditionally structured gameplay and other conventions having long been accepted by the gaming community.

In practice, the project’s desired outcome is the creation of a side scrolling shooter game application where the user is initially asked to provide an audio file and, after a brief wait, plays a level dynamically created from the chosen input track. The term “level” is here mainly used to indicate the sequence of game events, such as enemy types, appearance times and movement and shooting patterns occurring during a play through; procedural generation of graphic elements such as background or enemy sprites based on audio input could be included as an extra but will not be essential.

As said before, the overall goal will be for the user to discern a relationship between the audio track, which will be played in background during gameplay, and game events.

## Research Question

“To what extent can a musical audio signal be used as an input for procedural content generation techniques in a computer game environment?”

## Project Execution

As a first step, research will be conducted on the mathematical and physical nature of musical sound as well as on psychoacoustics, in order to find a small set of numerically expressible parameters representing major features of an audio signal.

Both time and frequency domains of the input will be considered: current candidate features for extraction are tempo, spectral distribution, taken as a measure of the number of instruments playing concurrently, balance between low and high frequencies, loudness and degree of consonance or dissonance. While potentially many more features could be considered, a conscious decision has been made to limit analysis to 3 or 4, to keep the amount of combinations used for pattern matching to manageable levels.

After conclusion of this review phase, research will be carried out to find algorithms suitable for computing these features of interest from a digitized representation of the input's waveform; following Aallouche et al’s example, the input signal will be segmented into multiple windows to be individually analyzed rather than treated as a whole. The result of this process will be the construction of a feature vector, a data structure holding the computed values of the selected features as a function of time.

Finally, a side scrolling shooter that uses the computed feature vector as the input for procedural generation techniques will be built: a number of high-level behaviors for enemy units will be coded and matched to specific patterns occurring in the vector, using additional values derived from the waveform as seeds to include some degree of pseudo-randomness in their execution.

## Project Evaluation

Given the intrinsic subjectivity in evaluations related to how closely or how appropriately the on screen action reflects the input background audio track, due to differing opinions and tastes amongst people, reaching an objective evaluation of the created program on these grounds would require huge effort and usage tests of significant scale.

As a consequence, effectiveness of the application will be assessed in operational and functional terms, such as creating functional levels using markedly different input sources, creating levels having similarity linked to the similarity in the original input sources (in its most basic interpretation, this means creating equal levels from equal or very similar sources and different ones from differing inputs), as well as following a set of basic rules not open to interpretation in level construction, such as not having any action on screen during silence intervals in the input.

The application should demonstrate the desired behaviors in a non ambiguous way, making evaluation possible without a need for extended testing.

## Possible Issues

While research conducted so far hasn’t revealed much in the way of best practices to solving the problems I am setting out to investigate, what was found so far seems to validate the chosen approach.

There are, however, several issues that could arise over the duration of the research project: first and foremost, mistakes in the evaluation of what numerical features describing the signal to focus on. This would probably be the most dangerous kind of error, as it would only manifest itself during testing, towards the end of the project, after considerable resources would have been spent devising algorithms to compute said features and integrating them in the application framework.

Less severe would be instances of a desired signal feature requiring algorithms too complex to be effectively implemented in the allotted timeframe or too computationally expensive to keep the time required to generate levels down to reasonable lengths. Such issues should manifest much earlier in the development cycle, allowing either the development of alternate methods of computation or the usage of different features without major averse consequences.

Problems might also arise in the mapping of patterns in the feature vector to game events: the number of permutations generated by variations in the selected signal features could prove too many to feasibly or meaningfully handle. These should be the easiest kind of problems to solve, doing so only entailing disregarding one or more of the least significant features computed.

Last but not least, all the usual pitfalls of software developments would, of course, still apply.

## Resource Requirements

As much of the project will revolve around research on the mathematical basis of sound, signal analysis, music theory and feature extraction techniques, access to relevant literature will be of the utmost importance.

Given the design decision of creating an application capable of running on most standard PC hardware using freely available software technologies, no additional resources other than access to the university’s computing facilities should be required.

# Appendix B: Accompanying CD

## CD contents listing

Root directory:   
Holds electronic copies of the current document and the slides used in its oral presentation, together with the dissertation proposal included as Appendix A and related slides, the table used in the results section in Excel worksheet format, a separate abstract document and the poster used in the demonstration session.

Executables folder:   
Holds the audio analyzer and proof of concept game binaries, together with the dlls they depend on: all these files are essential and need to be present in the same folder as the executables for them to run.

Research documents folder:   
Holds part of the material directly referenced or included in the bibliography; it doesn’t include copies of books, websites or computer programs.

Visual Studio 2008 solution directories folder:   
Holds the entirety of the created programs’ source code together with files required to open and compile them in the Visual Studio 2008 Express IDE. Unfortunately, back porting the projects to Visual Studio 2005 format proved impossible, the Musigen compiled binary crashing when compiled by that IDE. It is important to notice that the Musigen project requires the DirectX SDK to be installed in order to be compiled, due to its reliance on DirectSound.

## Demo applications usage instructions

Audio analyzer:   
This is a simple command line based application; detailed usage instructions are displayed by the program itself at runtime. It is important to notice, however, how the file paths the application accepts are relative to the run executable’s location as well as its need to be in a writeable location for graph generation. It is therefore highly recommended to copy it and all the dlls in the same folder in a location on the hard drive, together with the media files to be analyzed, for maximum ease of use.

Musigen – proof of concept game:  
It can be run from read only media and uses standard Windows dialogs for input file selection. When idle, choose to file->open to begin analysis and level creation (the application does become unresponsive during the process, as if it was crashing; this is, unfortunately, normal). Once gameplay begins, use the WASD keys to move the ship on screen and hold down the space bar to shoot.