SOUND IN LANDSCAPE OUT (SILO)

CHRISTOPHER PHILIP RIDGERS

Using Every Day Sound Files to Produce 3D Assets for use in Graphics Industries

School of Computing and Maths
Keele University

May 9, 2014 – version 1.0

Christopher Philip Ridgers: *Sound In Landscape Out (SILO),* Using Every Day Sound Files to Produce 3D Assets for use in Graphics Industries,

SUPERVISORS:
Adam Stanton

LOCATION:
Keele, Staffordshire, UK

TIME FRAME:
May 9, 2014

ABSTRACT

This project was the implementation of Fourier Analysis in order to generate 3D models based on music input's spectra.

The application visualises the process on screen through the use of the Allegro5 game engine library, and saves the 3D object to disk in the plain text Wavefront .obj format. The object saved is regular 2D plane in 3D space which then has its height values altered.

The FFT was implemented using the GPL licensed FFTW3 library, developed at MIT.

The end result is a small application that is capable of producing consistent results, total time to calculate landscape approx 2-5 seconds when compiled without window drawing support.

Recommended future works include the development of GUI interface to combine output objects, and further research into filters to create more interesting objects from sound spectra. Other works could include the creation of abstract data from the spectra to describe tiling patterns and combinations of objects.

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth (Knuth, 1974)

ACKNOWLEDGEMENTS

Thanks to my parents, *Sarah Davies* and *John Ridgers*, for encouraging me every step of the way, and never saying no, except for when it didn't matter. To my girlfriend and best friend *Sam Shone*, for her endless patience as I sat for hours staring at the computer without typing.

Thank you to my supervisor *Adam Stanton*, for allowing me to bombard his email box in the weeks leading up until the deadline. For the responses you gave. And thank you for the supervisor meetings. They helped guide me through the maze that is third year.

Thank you to *Keele University* and the *School of Computing and Mathematics* for the support over the years. To *Nikki Williams* for direction when lost, and *Steve Linkman* for perspective.

A Very Special Thanks: To my Grandfather,

Philip 'Opa' Quirk,

who sadly didn't get to see me graduate, but never doubted that I would.

Travelled the World, Resting in Peel.

CONTENTS

LIST OF FIGURES

ACRONYMS

DRY	Don’t Repeat Yourself
API	Application Programming Interface
UML	Unified Modeling Language
FFT	Fast Fourier Transform
FFTW	Fastest Fourier Transform in the West
DSP	Digital Signal Processing
RIFF	Resource Interchange File Format
WAV	Waveform Audio File Format
VST	Virtual Studio Technology
GTA	Grand Theft Auto
LTS	Long Term Support
GPL	General Public License
LGPL	Lesser General Public License
GNU	GNU’s Not Unix
DTFT	Discrete Time Fourier Transform
SILO	Sound In Landscape Out
OO	Object Oriented
MIT	Massachusetts Institute of Technology
FLAC	Free Lossless Audio Codec

Part I

INTRODUCTION

An introduction to the domains of which my project is focused, and a *very* brief introduction to the maths involved.

1 INTRODUCTION

1.1 OPENING STATEMENT

I am implementing the generation of a planar landscape object generated from data processed using a Fourier Transform. The rapid creation of such objects from arbitrary data, such as readily available music files, is important because of the highly competitive nature of the targeted industries.

It is also important academically as it serves to demonstrate the applicability of Fourier Analysis across domains and applications in such a way that has not (that I have been able to find) been examined by academics at a formal institution.

Hopefully papers such as this will raise further awareness of the technique and its flexibility, and its potential for future research.

1.2 PREVIOUS WORK

Musgrave, Kolb and Mace (1989); Hnaidi, GuÃ©rin, Akkouche, Peytavie and Galin (2010); Pickover (1995); Frade, Vega and Cotta (2008) all detail various approaches to generating natural terrain. Of those however, none make reference or preference to the Fourier Analysis and Synthesis Technique that is described by Bourke (2014).

Smith (1997, ch 24) describes the process of analysing an image's spectra and Smith (1997, ch 22) describes the Fourier Transform of audio while earlier chapters in the book detail the theory in great detail.

The knowledge of Fourier Transforms and Digital Signal Processing has been around for many years, however Smith (1997, ch 24) claims that a lack of processing power has been responsible for holding back on the use of these techniques. That was 17 years ago now, and there has been a lack of DSP Terrain Generation research going on beyond individuals such as Bourke (2014).

This may be because of the development of wavelet synthesis; an alternative technique that over comes some of the limitations of Fourier Transforms, such as the low spectral resolution at higher frequencies.

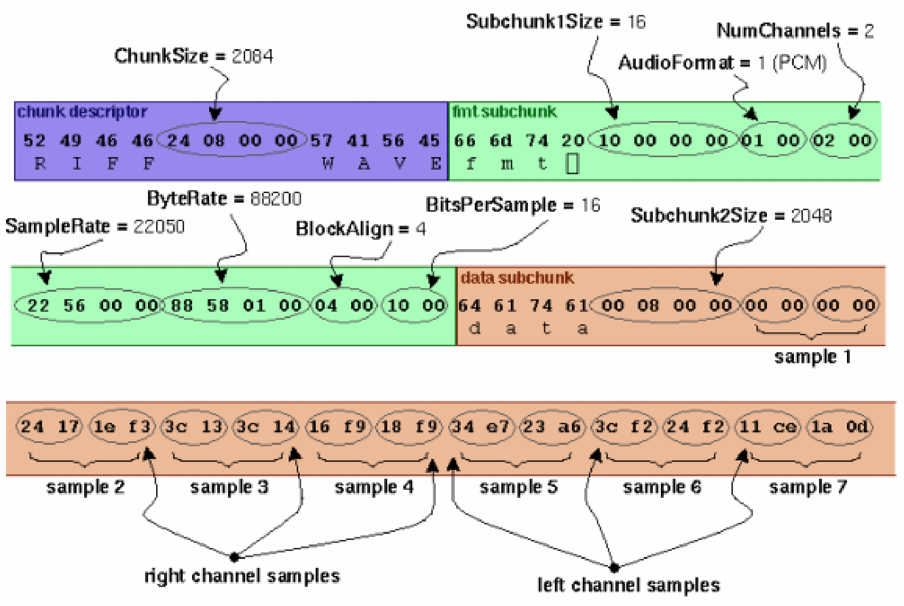


Figure 1: Description of sound data and layout of multi channel audio (Codeproject.com, 2014)

1.3 BACKGROUND KNOWLEDGE

.wav Specification

It's quick and easy to verify this by running the following command on any system with the hexdump utility: hexdump -C -n4 file.wav

The .wav format takes on from the RIFF file format, with the leading chunks describing the following audio data. The leading four bytes are 52 49 46 46 making it very easy to identity, as they are the ASCII representation of 'RIFF'.

Leading chunks describe (among other things) the file length the sampling rate, the number of channels, and in later version of the Wave Extensible format, speaker positions to resolve ambiguity issues in the earlier version. The audio data itself is encoded sequentially with each channel interleaved with the others, as Figure 1 shows.

.obj Specification

Reddy (unknown) Explains the full .obj specification in far greater detail then is required for simple planar surfaces. The structure of a 3D object is described in space relative to the object (local space), first by defining vertices, then the faces connected to them.

Chapman (2013) further discusses de facto rules about describing 3D Objects within space, notably mentioning the topics of unnecessary vertex duplication and face orientation through vertex ordering.

As well as the practice of subdividing complex shapes into triangles for representation. See Figure 2.

```
v       -1.0     1.0     1.0
v       -1.0    -1.0     1.0
v        1.0    -1.0     1.0
v        1.0     1.0     1.0
v       -1.0     1.0    -1.0
v       -1.0    -1.0    -1.0
v        1.0    -1.0    -1.0
v        1.0     1.0    -1.0
f       1       2       3       1
f       3       4       1       3
f       4       3       7       4
f       7       8       4       7
f       8       7       6       8
f       6       5       7       6
f       5       6       2       5
f       2       5       1       2
f       5       1       4       5
f       6       7       3       6
f       3       2       6       3
```

Figure 2: The above table shows the obj. specification format for defining an object in 3D space. vertex texture coordinates and it's associated faces are not shown; neither are .mtl definitions

Vertex number are assigned in the order that they appear in the document. By using these references unnecessary vertex duplication of vertices can be prevented in order to define the separate faces that make up the cube.

Imaginary Numbers and Complex Number Theory

Pierce (2014*b*) states that imaginary numbers are numbers that squared give a negative result. They are useful in mathematics because they allow us to solve equations that would otherwise be unsolvable with real numbers. Similar to real numbers, imaginary numbers have a unit i, which represents : $i = \sqrt{-1}$

Through the use of i other imaginary numbers, such as the square root of -9, can be represented Figure 3.

This is useful in several different disciplines, including physics, engineering, pure maths etc.

Complex number theory is the representation of values through a combination of real numbers and imaginary numbers. Real numbers

$$\sqrt{-9} = \sqrt{(9x-1)} = \sqrt{9}x\sqrt{-1} = 3x\sqrt{-1} = 3i$$

Figure 3: Using i to work with powers of negative values

are in fact complex numbers that have an imaginary part of 0 (Pierce, 2014*a*). The true value of a complex number is the summation of the two parts. In this sense, a complex number is no different from an ordinary pair of bracketed values in real maths, albeit one of the values is now imaginary. Similarly the manipulation of complex numbers follows the same rules maths involving bracketed numbers. For example, multiplication Figure 4:

$$\begin{aligned}(1+1i)(2+2i) &= 1*2+1*2i+1i*2+1i*2i \\ &= 3+2i+2i+2i \\ &= 3+6i\end{aligned} \tag{1}$$

Figure 4: Multiplication of of complex numbers

This shows that complex numbers can be manipulated and combined in such a way that the imaginary part becomes 0, and in which case the result is a real value.

This result is of great importance to the Fourier Series, and the Trigonometric $\sin\theta\cos\theta$ functions when represented in the form of a complex number.

Fourier Series / Transforms

The Fourier Series / Transformation is the analysis of the structure of complex structures and phenomena and their component structures. A study of periodicity in both time and space, together and apart depending on the nature of the application, the transformation is derived from the concept that any generalised complex structure is comprised from the sum of it's component parts (Osgood, 2008).

Originally developed as a means to solve a specific problem (the heat equation), the series later applied to many different disciplines, including spectral analysis of sound data.

The formula can be written in several different forms, but often due to the relationship between the trigonometric functions (sin, cos etc.) and the complex exponential, a commonly written and used version of the series is written as Figure 6.

Where $f(t)$ is a complex function and C_k represents the frequency and phase of the function in the form of the complex number. T is the period of the function. The real part represents the phase of the function through the trigonometric cos() function while the imaginary part represents the amplitude through the sin() function. The magnitude is the absolute value of the two combined, and in respect to the Cartesian coordinate system, is the value of the distance from 0. Most applications assume a periodicity of unit circle (or 1), resulting in the version of the formula displayed above. For other periodicities, additional steps must be taken.

The Inverse Fourier Transform (IFFT) is simply the summation of frequencies, resulting in the synthesis of a complex function from simple component parts.

Sound Theory

A minimum amount of domain knowledge is needed to justify some of the design choices made in this project. Most significantly, there are consequences of Nyquists's Theorem to account for. Marshall (2001) describes the problem, and the solution Figure 7.

The problem still exists when selecting our own samples for a Fourier Transform however. When an audio signal is recorded, prior to the samples being taken it passes through a high stop filter set to +22050Hz. This removes all the frequencies that in the event of inadequate sampling would result in a nasty feedback effect within the audible spectrum (20Hz - 22050Hz). Then, because of Nyquist's theorem, the signal is sampled at approximately double the frequency of the upper limits, usually 48000Hz or 44100Hz (typical for an Audio CD distribution).

This will be reflected in the output of the FFT, regardless of sample size.

Following this, samples are selected to perform the Fourier Transform; practically a finite number of samples has to be defined. In a pure audio domain, this is crucial choice; the result from an FFT is a series of frequency bins each of which is $1_s/N$ with 1_sHz wide. The larger the sample size the smaller the bins, and as a result greater accuracy in the frequency domain. On the other hand, a larger sample size is required, and thus the result is less accurate in the time domain.

Analog.com (2014) gives a comprehensive run down of the FFT process, including the differences between the various transformations in the FFT family, explanations on their efficiency, and a more in depth discussion of the different representaions of the FFT, including the polar and cartesian forms, also demonstrated by Osgood (2008).

We actually select a sample size of 512 by 512; 65536 samples. For mathematical reasons and optimization, Fourier Transforms work best with sample size N as a power of 2 (Analog.com, 2014). Also, typical video game textures tend t scale in powers of 2 for optimization(Katsbits.com, 2014).

1.4 CONCLUDING STATEMENT

Attempts are made to reference at any opportunity an outside work at the mention of the specifics of the workings of Fourier Analysis and its transforms, as the mathematics involved are beyond anything I've been officially taught and tested against. Every effort has been made to ensure that the information is correct, but I feel that the fact needs stating when talking about the numbers and processes.

The project product will be a demo application that demonstrates the ability to create landscape objects of a similar nature to those described by Bourke (2014), but with none random data. The ideal result would to demonstrate that audio frequencies within a complex function (such as a music track) would carry their properties over to visual objects consistently, thus allowing potential for future development of landscape categorisation and further manipulation.

One of the key defining traits of Bourke's generated landscapes are their tile-ability in both the X and Y dimensions, opening them up for a lot work post generation compared to the none tillable pieces generated by the diamond-square technique (also described at Bourke (2014)).

Ideally, the end result outputs would retain this characteristic.

Finally the product software will be well structure for future development, with potential for integration both as VST plugin, and as a landscape tool within a package such as Blender.

And division:

$$\begin{aligned}\frac{(1+i)}{(2+2i)} =& \frac{\overline{(2+2i)}}{\overline{(2+2i)}} * \frac{(1+i)}{(2+2i)} \\ =& \frac{(2-2i)}{(2-2i)} * \frac{(1+i)}{(2+2i)} \\ =& \frac{2+2i-2i-2i^2}{4+4i-4i-4i^2} \\ =& \frac{2-2i^2}{4-4i^2} \\ =& \frac{2+2}{4+4} \\ =& \frac{8}{4} \\ =& 0.5\end{aligned} \tag{2}$$

Which can be better represented with:

$$\begin{aligned}(1+1i)/(a+bi) =& \frac{\overline{(a+bi)}}{\overline{(a+bi)}} * \frac{(1+1i)}{(a+bi)} \\ =& \frac{(2-2i)(1+1i)}{a^2+b^2}\end{aligned} \tag{3}$$

$$\begin{aligned}(1+1i)/(2+2i) =& \frac{\overline{(2+2i)}}{\overline{(2+2i)}} * \frac{(1+1i)}{(2+2i)} \\ =& \frac{(2-2i)(1+1i)}{2^2+2^2} \\ =& \frac{4}{8} \\ =& 0.5\end{aligned} \tag{4}$$

Figure 5: Demonstrating the basics of complex number division theory

$$f(t) = \sum_{k=-N}^{N} C_k e^{2\pi i k t} \tag{5}$$

$$C_k = 1/T \int_{-T/2}^{T/2} e^{-2\pi i (k/T) t} f(t)\, dt \tag{6}$$

Figure 6: The Fourier Transform and Complex Component

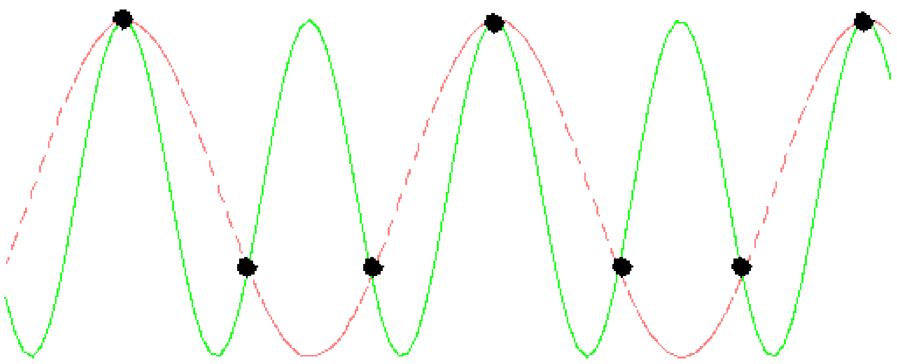


Figure 7: Result of insufficient audio sampling Marshall (2001)

Part II

THE PROJECT

A written report on the readings and work undertaken to produce the software demo, along with a commentary on the techniques used and their origins. Also, an overview of the target industry in regards to requirement elicitation is is also provided, as well as extracts of code needing special mention.

2 REQUIREMENTS ENGINEERING AND EVALUATION

Requirements elicitation is a key aspect of any Software Development Process; Belle and Thayer (1976)'s "*Software Requirements: Are They Really a Problem?*" found that inadequate, incomplete, inconsistent or ambiguous requirements are numerous and have a critical impact on the quality of the resulting software.

This project follows a set of requirements self-elicited, for the purpose of producing a piece of software, which have then undergone a process of evaluation and documentation (at multiple levels) demonstrating an understanding of real word application of the Requirements Engineering process, including aspects such as inconsistency management , risk management, and formal documentation.

2.1 INFORMAL REQUIREMENTS GATHERING - SELF ELICITATION

The who, what, why dimensions defined in Lamsweerde (2009)'s "*Requirements Engineering: From System Goals to UML Models to Software Specification*" are made in reference to systems requirements engineering, specifically regarding the development of a 'system-to-be', but are equally applicable to this project; who is using my software, why they are using it, and what it is that my software is/ will be doing is key to defining its requirements.

2.1.1 *Who*

The targeted users of my software are individuals working within the creative arts/ graphics industries. They would be in-actively interacting with the software to produce 3D object/ assets for further use in production pipeline. They would be responsible for providing access to a batch of input data, but have minimal responsibility as to how 'correct' that input is. The software itself would be responsible for verifying input, and reacting appropriately to produce an output, and reacting to inform the user of progress/ errors it encounters.

2.1.2 *Why*

Every year creative industries generate billions of dollars in revenue around the world. In 2013, Karmali reported that Rockstar North's video game Grand Theft Auto V had broken all records, earning $1 billion faster than any other game, or entertainment property, in history.

GTAV though is one of the few games that makes such a sizeable revenue; commercial development titles is a long expensive process, with individual titles comparing to feature films in regard to the development work and time schedule: GTAV was in development for over 9 years, with a team of over a 1000 individuals (Hill 2013).

Even while being developed, dozens of studios and teams have shut down and disbanded, as reported by Plunkett (2012). Loss of revenue through poor games sales are not the only problem plaguing developers. Over the years, the PC platform has been notorious for piracy; in 2008 40,000 copies of the game 'Assassins Creed' were sold across the US, while illegal downloads of a copy released two months prior topped 700,000 (Sinclair 2008), resulting in law suits at further cost to the developer/ publisher Ubisoft.

The high-risk nature of the industry results in an extremely competitive work environment, and through it stories of employee exploitation have emerged. 'Crunch' is part of the staple diet of the business, pushing working hours of entire teams to the limits; 'easpouse' (2004)'s "*EA: The Human Story*" told of the personal story of an EA development team through the eyes of a developers partner. More recently Miller and Buckley (2013) conducted a survey regarding quality-of-life within the industry, finding that factors such as unpaid overtime and layoffs were still relevant; with 23% responders expecting to be laid off at the end of their current project, and split opinions on job satisfaction and value of compensation.

'easpouse' (2004) stated concerns over leaving, '...one crunch to join another.' Moving between teams and studios is project to project concern for several artists/ developers, particularly those employed under freelance contracts during crunch phases. As a result, individuals tend to build their own personal libraries of resources in an attempt to increase their efficiency, and as a result their attractiveness as a future employee. Artists will use stock images and models to remove the need to repeat iterations of content job to job. This particular pattern though has its drawbacks: often the content the artist is using belongs to the company that it has been developed for, and there are laws and policies in place that restrict an individuals ability to keep/ take content for use in other projects.

2.1.3 *What*

My project is to create an application that can input an audio file, evaluate it's content and produce a 3D object based on the information. The full "concept" is that the software would run as a background daemon analysing the audio files within a media library or directory, storing appropriately categorised landscape objects to disk categorised by terrain type. This would allow artists to quickly identify suitable stock objects to work from to aid with landscape environment production. The nature of the output should be such that if the application was to scan over the same files, it would produce a similar but not identical output. This would greatly aid a 3D artist moving between projects: even if his former parent company were to hold his previously produced environments under contract, they could very quickly and with minimal effort produce a large amount of stock objects to create future work with, simply by continuing to run the application.

Requirements of the software would include (but are not limited to):

- Output object files compatible with most standard 3D modelling environments; specifically Blender and 3DSMax.
- Allow input music files to be of multiple format types; notably uncompressed audio in the wav/ aiff extension, and common formats distributed by major vendors i.e. .m4a files as part of iTunes store purchases.
- Platform compatibility between common environments used in the target industry: Windows and Mac OS / OS X;
- Accessible to individual artists as well as companies: affordable.
- Should be efficient; defined as not stealing processor cycles away from a users other processes.

In regards to this project however, there are a number of restrictions regarding submission and assessment: most notably the requirement that the software run on the standard lab computers the university provides, and the lack of access to popular commercial softwares such as 3DS Max and Maya for testing compatibility. As a result, the target platform for this project is the Linux Ubuntu environment provided on the project lab computers, and the software compatibility target will be the open source Blender application. I'll also be focusing on inputs of .wav for the purposes of demonstration.

When given the option, I will always opt for OS X compatibility in addition to Ubuntu Linux, however it is not a stated goal

2.2 REQUIREMENTS EVALUATION

2.2.1 *Summary of requirements*

So our project requirements are:

- Blender output compatibility
- .wav input files
- Ubuntu target platform
- Affordable to individuals
- Efficient

2.2.2 *Blender Output Compatibilty*

Blender‘s supported file formats for 3D objects include Autodesk 3DS (.3ds), COLLADA, Filmbox (.fbx), Autodesk (.dxf), Waverfront (.obj), DirectX (.x), Lightwave (.lwo), and Motion Capture (.bvh) among others (Blender.org, 2014).

Of these, Wavefront’s OBJ specification is well suited as an output format: Murray, James, Ryper and William (1996) and Reddy (unknown) details the open documented format of the specification, which is written in plain text ASCII making it particularly accessible to amateur artists / individuals as well as professional programmers and design teams.

In addition LeFebvre (2013) states that not only is 3DS Max compatible with the .obj format, but that it,

> “...is one of, if not the, most common transfer format for 3D models as it can be read by nearly every industry-standard 3D application...”

Thus extending accessibility not just to Blender and Unix-like platforms, but also Windows systems.

2.2.3 *.wav Input Format*

The Waveform Audio File Format (WAVE / WAV) is an uncompressed storage format for the storage of sound data. An implementation of the Resource Interchange File Format (RIFF), the design “chunk” based, with leading chunks describing the following data in the audio “chunk”. Boulanger and Lazzarini (2013, p. 189-193) *“The Audio Programming*

Book" describes the standard layout of an audio file and its data in relation to initialising and retrieving it via the literatures accompanying educational API. This text in particular is very useful as a reference tool regarding audio files and their manipulation, and leads the user through a number of in depth examples for different techniques resulting in the .wav / .aiff formats being very accesible.

Freestockmusic.com (2014) describes aiff and wav as 'interchangeable' meaning that either format would be appropriate to focus on.

2.2.4 *Ubuntu target platform*

As a restriction of the available operating systems available on the project lab computers, I am limited in choice of target platform to Canonical's Ubuntu platform and Microsoft Windows 7. As I am far more comfortable developing on and for a Unix-like environment, I have opted for Ubuntu.

I also have access to an Apple OS X machine, allowing me to test compatibilty

The version of Ubuntu is 12.04LTS.

2.2.5 *Affordability*

Commercial / Professional design software often tends to cost individuals and companies large sums of money to use; particularly for commercial use (Autodesk, 2014*a*,*b*; Pixologic, 2014). A notable exception to this is the open source GPL licensed Blender software.

In order to ensure potential future users would be able to afford the product, I will have to ensure that all technologies involved are freely available to be re-used and distributed without incurring licensing costs.

2.2.6 *Efficiency*

My original concept for the software is that in a fully developed product, it would repeat continuously on a multitude of different files. As a demo, my software would have to demonstrate potential for future scalability, and can be measured in a total and average measure of completion of multiple run cycles. I can measure this through the use of a start time and end time value and measuring the difference to calculate execution time. Over a large sample set, I can gather an indication of how long my program takes to complete.

3

SOFTWARE DESIGN AND IMPLEMENTATION: METHOD

3.1 DESIGN SPECIFICATION

3.1.1 *Tools - Libraries*

A number of different tools have been used to create this application. Figure 8 shows the different API's have been combined to successfully create a height output.

```
#include <complex.h>
#include <string>
#include <iostream>
#include <sstream>
#include <fstream>
#include <vector>
#include <fftw3.h>
#include <sndfile.h>
#include <allegro5/allegro.h>
#include <getopt.h>
#include "globals.h"
#include "landscape.h"
#include "silo.h"
```

Figure 8: The various libraries used to produce output

Some of these libraries are very widely used by computer scientists and need no introduction. Others however are more specialist and deserve specific mention, along with information pertaining to their licensing rules in order to meet the affordability requirement.

The bottom 3 includes are local includes written for this project; globals.h defines a data structure to hold variables that would otherwise be declared in global "space" and initialised in the main() function. Landscape.h describes the functions and interface for the landscape class. Object orientated designs have been followed to allow future scaling and improved flexibility for future development. Silo.h forward declares the functions called and defined within silo.cpp.

The use of globals.h and a data structure to hold global variables is a personal design choice. It is also however a common practice / requirement for designing VST plugins for audio software. With potential development in mind, it makes sense to meet this requirement from the start.

getopt.h

License: GNU LGPLv2

GNU's getopt library is distributed as free software, and its license states that it may be linked to as part of free and not free (closed source) applications.

In this regard, it is allowed to be used without incurring financial cost which would then have to be passed on to the user, aiding affordability.

The library' focus is the standardised parsing of input from the user through the command line and the argv array. The use of a regular interface such as getopt vastly reduces the chances of overlooking a potential erroneous error by the user, while keeping input validation to a specific area makes any future developments much simpler to integrate.

complex.h

C++ does actually provide it's own complex class which is theoretically bit compatible with fftw_complex. At a later point it may prudent to use it instead of FFTW's complex type / C implementation

Complex numbers are part of the C99 standard, and as such are well defined and their use requires no extra licensing on the developers part.

Complex numbers are well described in section 1.3, but it is this library header file that specifies exactly how the computer should work with them.

fftw3.h

Licence: GNU GPL

FFTW3 (Fastest Fourier Transform in the West (3)) is a C library for computing the Discrete Fourier Transform in N dimensions. It is free software, distributed under the GNU General Public License. This means it is free to distribute for none commercial purposes, similar to the LGPL, however marketing derivatives or works using it as a commercial product (without distributing the source would require a separate license. This is suitable for use as an academic project, however would imbue a cost upon the user should the result not be distributed as free software (with the source). Such a release would meet our affordability requirement, but limit profitability.

FFTW (2014*a*,*b*) claim that FFTW is quick and efficient, and details it's benchmarks and testing setup, meeting the efficiency requirement (in regards to design choices).

allegro5/...h

Licence: As Is

Allegro5 is a game development / multimedia library focused on cross platform implementation of ‘low level’ tasks such as window management. Within the project it is used to draw the result of transforms to a window in order to quickly provide visual feedback and further develop the source. In an actual release of a finished application, it is likely that the use of this library and the window drawing feature in general could be dropped in favour of efficiency.

sndfile.h

Licence: GNU LGPLv2 or GNU LGPLv3

Libsndfile is a cross platform library supporting the reading / writing to and from audio files. Boulanger and Lazzarini (2013) includes a small library called Portsf to aid in educating a programmer in working with audio files, however they openly state the limited feature set and it’s lack of suitability to replace a full fledged library (such as libsndfile).

Under either license terms, it is possible to use libsndfile within close source projects under the condition that the libraries are linked dynamically, thus meeting the affordability requirement.

3.2 TOOLS - OTHERS

The development tools for this project also include the Vim text editor, GNU / Apple Development toolkits; specifically XCode 5.1, GNU’s g++ 4.8, gdb and lldb.

Furthermore, the project has been stored and distributed to various machines for testing via version control software Git, and the entire project history can be downloaded and viewed by running the command:

libfftw3, libsndfile, and allegro5.1 will also need to be installed locally in order to compile the software.

```
git clone https://github.com/chrisRidgers/silo
```

By executing the makefile, the application should compile without issue.

3.3 METHOD

User Input Processing

User input is handled through the GNU getopt library: the use of a long option type and an index value to loop through **argv and compare each flag to that of a defined string to test for matches. This can be seen in Figure 9.

```
int c;
while(( c = getopt_long(
global->argc,
global->argv,
"i:o:s",
global->long_options,
&global->option_index)) != -1)
```

Figure 9: The string "i:o:s:" defines three known flags, and states that they must all have an associated argument when used.

Any flags and their associated inputs that aren't recognised will be printed out to the terminal, though won't affect the running. If however, one of the defined flags is specified but no argument accompanies it, the program will exit gracefully.

Getopt also supports the setting of flags directly (rather than relying on an if/ else or switch statement. Rather than pass an identifier for testing, a reference to the variable and the desired flag is passed instead as part of the option structure.

Variable Definitions and Library Initialisations

Within the SILO application, user input parsing is the second event to take place. Initially, the program initialises a struct of type global to hold various variables, including the getopt option structure and optindex, argc and argv location, some boolean ints used to verify input and the allegro display to handle drawing to the screen.

Not all of the variables are initialised at once, but they all exist within the same structure for the sake of tidy code and future developments.

Figure 10 shows the initialisation of the struct itself.

A reference to this struct is passed to all variables that may need any of these variables. It should be noted that no variables should be stored in a struct publicly such as this that need to remain secure or

```
int main(int argc, char **argv)
{
  struct global global{
    {
      {"input",         required_argument, 0, 'i'},
      {"output",        required_argument, 0, 'o'},
      {"smoothness",    required_argument, 0, 's'},
      {0, 0, 0, 0}
    },
      0,
      argc,
      argv
  };
```

Figure 10: Values primarily initialised: option struct, option index, argc and argv

anonymous from functions that do not require them. C++ classes with access functions are far better suited to that task.

Landscape Object

SILO takes advantage of the OO nature of the C++ programming language by using objects of classes where appropriate. The landscape class holds all data and variables linked to the creation of an individual output. The use of a class-object design choice is out of deference to the initial concept where the application would be generating multiple outputs in one runtime cycle. section A.2 shows the structure of the class while Figure 11 shows the constructor and deconstructor to handle memory allocation of buffers.

Throughout the existence of the landscape object, the object is represented in several different formats. The fftw_complex represents the object described as raw sound data, in the form of complex values ready for / as a result of a transform while the 3D coordinates of the object are represented in the form of a vector of vectors of doubles:

```
vector<vector<double>> object;
```

All of the landscapes variables are private, and as such have access functions associated with them. In this they are all self contained, not just from functions that shouldn't have access to them, but also from future programmers/ developers unfamiliar with the source attempting to access them from else where in the program.

Libraries

Allegro's initialisation varies from platform to platform in order to take advantage of particular features. On OS X, the development platform, it works by re naming the main() function and initialising its own (starting a Cocoa window) before running the newly renamed 'main' function. This poses a few technical issues in regards to compilation and linking, but the Makefile should accommodate for this.

Allegro is also particularly stringent about the form of which the user's main must be written. It's not uncommon to see main() functions written both as

```
int main(int argc, **char argv){
..
}
```

or

```
int main(int argc, char argv[]){
..
}
```

But Allegro will throw errors on compilation if the main() function does not take the formers syntax.

FFTW's API requires no formal initialisation, however the buffers required for its transform functions and objects are initialised on a landscape by landscape basis, as each object of landscape is created. This simplifies reading and writing the source code as all of our variable initialisation happens in one place, and is standard OO methodology practice.

In addition to a number of *fftw_complex* buffers that are required, FFTW also requires a number of *plan* objects in order to carry out its functions. FFTW calculates the most optimum method of transforming data based on characteristics such as the N number of data samples being transformed, the number of dimensions the transform is taking place in and the type of data being transformed.

Indeed, a forward transform of 'real' data results in FFTW discarding half the data by means of optimizing due to the similarity between the two halves. While this may be beneficial as far as data is concerned,the result is less suited for image processing (including terrain generation), as half of the data set has been reduced to 0 in both the dimensions.

Sound Data

Validate Sound File

Getopt's validation process is well suited for parsing user input, but it does not guarantee that said input is correct. The user may specify an input file that does not exist for example, or one that is of an incompatible format.

In order to test for this, libsndfile is used to attempt to open a file, and return a value based on whether it was successful or not at doing so. In the event that it was not, SILO closes the file and exits gracefully Figure 12.

Memory Allocation

Use of *fftw_alloc_complex()* and *malloc* define the various memoy buffers used in SILO. The number of samples required is fixed at 256*256 (65536), so allocating the correct amount of memory is simply done. A float* buffer is used to read in the initial samples before they are re scaled and shifted into a fftw_complex* buffer for processing.

Because the configuration of FFT's transform plans are 'out of place' there are two fftw_complex* buffers, that are both used to pass the data back and forth during the forward / backward transform process.

Seek Setup

One of the advantages of SILO is that it randomly sets the location in the file from where it starts reading data. This means that on consecutive runs on an identical file, with similar settings, the potential is there for a different output. This supports future development plans for the initial concept where the application runs continuously processing a users music library and generating landscapes for later use.

The location is set by reading the number of samples of file and finding the difference between that figure and that of the number of samples we are reading (65536). Figure 13 shows this in greater detail.

The application is safe from overflow providing that that sound file is at least 65536 samples long, which at a sampling rate of 44100Hz, is 1.486 seconds, which accounts for the vast majority of sound files, in a music library, and can be accounted for else where as part of input validation.

Data Reading

The data samples get averaged into a single mono channel stream as they are read in from the file. Figure 14 demonstrates some code that will work for any n number of channels data. This sound data is then also re mapped from a range of -1.0 - 1.0 to 0.0 - 255.0. In order to do this, the actual range of values must be tracked, and then they are passed to the linear map function shown in Figure 15.

After this conversion, they are then read into the real part of a waiting fftw_complex* buffer.

Discrete Time Fourier Transform

Forward Transform and Frequency Domain

SILO uses FFTW's *fftw_plan_dft_2d()* to calculate a 'plan' type to create the forward and backward transforms. These plan types sit in memory and can be used at any point to perform a transform, regardless of whether or not the transform has already occurred. Repeated calls to *fftw_execute()* will carry out a transform repeatedly ad infinitum (provided the resources necessary are available i. e. the buffers defined in the plan).

The actual transform used in SILO is a 2 dimensional Discrete Time Fourier Transform. What this calculates is the product of two separate transforms, one in the X dimension and one in the Y. This is a common enough process in regards to image analysis using Fourier transforms, but it should be pointed out that a 2d transform of 512 in X and 512 in Y *IS NOT* equal to a 1d transform of 65536. In regards to normalisation especially, one should be aware of the scaling factors that are accruing through the use of an unnormalized Fourier Transform (such as FFTW).

As Figure 16 shows, once the transform is complete, the values pertaining to the frequency domain are in the second fftw_complex* buffer, and this remains the case until the inverse transform back.

Scaling / Filtering

The next step is to apply a scaling filter of $1/f^p$ in order alter the real and imaginary parts of the image to create a surface map. In this filter, f represents the Cartesian distance from the origin of each value (the magnitude), while p represents a scaling factor that affects the overall 'ruggedness' of the terrain Bourke (2014).

Inverse Fourier Transform and Normalisation

The final step is to perform the inverse transformation to retrieve the terrain map. However, as a library FFTW computes non normalised transforms as a means of efficiency. The resulting output must be normalised to counteract the scaling factor that occurs as part of the process. This can be done either after the forward transform or after the inverse transform, as long as it is done. FFTW (2014*c*) that the amount of scaling is the size of the array (single), or in the case of multi dimensional arrays, the products of the length of the dimensions, which in the case of silo is 256 * 256. See Figure 17.

.obj Output

Defining Vertices

Vertices are defined in sequential ordering and treated as a single list. Figure 18 shows the initialisation of the vector that holds the values of the xyz coordinates. The loop uses two counters to track the positions in X and Y as it increments through the vertices. It sets those coordinates based on the position in the grid.

Any vertex position in the grid can be identified numerically with the formula $v = y * x + x$. This is important when deciding which vertices to draw faces between.

Similarly, when setting the Z coordinates, SILO loops through the object vector, setting the Z coordinate for each instance.

Defining Faces

Because vertices can be defined numerically, it's possible to define faces by specifying the plane by in the form of these vertices. As Figure 19 shows, for any vertex C, the connecting vertices making up the plane can be defined as; C + 1, C + width, C + width + 1.

This alone however will not give you a planar structure; it will also connect the vertices at the edge of the plane in a 'wrap around' like manner. There has to be a conditional check to prevent this, as demonstrated in Figure 20

Cleanup

As good practice, it's apporpirate to take care of all of the variables before gracefully exiting the program, which SILO does, using the landscape objects destructor to de-allocate the memory buffers assigned to it.

```
landscape::landscape(struct global *global, int
   width, int height)
{
  this->width   = width;
  this->height  = height;
  this->size    = this->width * this->height;
  this->output  = "output/";

  //Defines file output path based on time
  time_t t = time(0);
  struct tm *now = localtime(&t);
  stringstream ss;
  ss
    << now->tm_mday
    << now->tm_mon+1
    << now->tm_year+1900
    << now->tm_hour
    << now->tm_min
    << now->tm_sec
    << ".obj";
  output += ss.str();

  object.resize(size);
  for(int i = 0; i < object.size(); i++)
  {
    object[i].resize(3);
  }

  landscape::setupVerts();
}

landscape::~landscape()
{
  fftw_destroy_plan(p);
  fftw_destroy_plan(p2);
  fftw_free(image);
  fftw_free(image2);
}
```

Figure 11: Memory buffers are handled intelligently through the use of class constructors and destructors

```
test->setInfile(sf_open(test->getInput().c_str(),
    SFM_READ,
      test->getInInfo()));
if(!test->getInfile())
{
  fprintf(stderr, "Invalid input file\n");
  sf_close(test->getInfile());
  exit(0);
}
```

Figure 12: Graceful handling of incorrect file type/ questionable state of existence

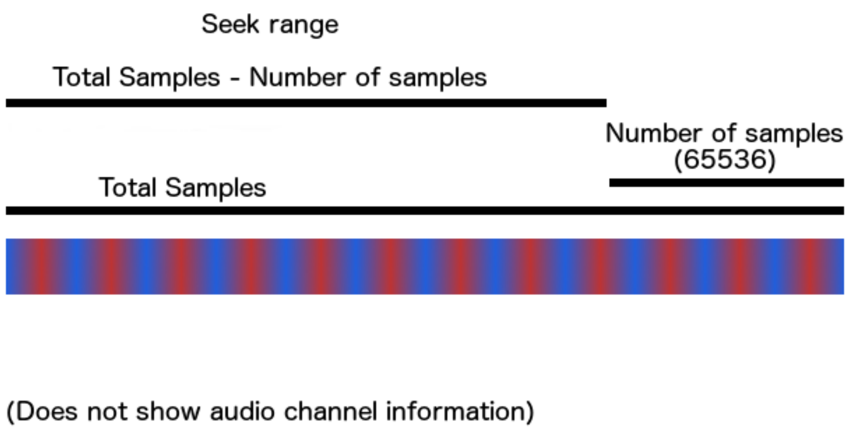

Seek range
Total Samples - Number of samples
Number of samples (65536)
Total Samples
(Does not show audio channel information)

Figure 13: The seek start location can be set to anywhere between 0 and seek range

```
int averageChannels(global *global, landscape *test
   )
{
  double biggest = 0.0;
  double smallest = 1.0;
  for(int i = 0; i < test->getHeight() * test->
     getWidth(); i++)
  {
    test->getImageBuffer()[i][0] = 0;
    test->getImageBuffer()[i][1] = 0;
    for(int ch = 0; ch < test->getInInfo()->
       channels ; ch++)
    {
      test->getImageBuffer()[i][0] +=
        test->getSoundSamplesBuffer()
        [i * test->getInInfo()->channels + ch];
    }

    test->getImageBuffer()[i][0] /= test->getInInfo
       ()->channels;

    if(test->getImageBuffer()[i][0] > biggest)
      biggest = test->getImageBuffer()[i][0];

    if(test->getImageBuffer()[i][0] < smallest)
      smallest = test->getImageBuffer()[i][0];
  }

  for(int i = 0; i < test->getHeight() * test->
     getWidth(); i++)
  {
    double value;
    linearMap(test->getImageBuffer()[i][0], value,
       smallest, biggest, 0.0,
        255.0);
    test->getImageBuffer()[i][0] = value;
  }

  return 0;
}
```

Figure 14: Averaging channel data while tracking actual range

```
int linearMap(double value, double &result, double
   oldMin, double oldMax,
    double newMin, double newMax)
{
  result = (value - oldMin) * ((newMax - newMin) /
     (oldMax - oldMin)) + newMin;

  return 0;
}
```

Figure 15: Note the reference call of the second parameter

```
int landscape::setPlan()
{
  p = fftw_plan_dft_2d(width, height, image, image2
     , FFTW_FORWARD, FFTW_ESTIMATE);

  return 0;
}
```

Figure 16: Image2 is where the transformed values end up

```
  for(int i = 0; i < test->getWidth() * test->
     getHeight(); i++)
  {
    complex<double> v(test->getImageBuffer2()[i
       ][0],
        test->getImageBuffer2()[i][1]);
    test->getImageBuffer2()[i][1] *= 1.0 /
      sqrt((test->getWidth() * test->getHeight()));
    test->getImageBuffer2()[i][1] *= 1.0 /
      sqrt((test->getWidth() * test->getHeight()));
  }
```

Figure 17: Both the real and the imaginary values get normalised.

```cpp
int landscape::setupVerts()
{
  int countX = 0;
  int countY = 0;

  for(int i = 0; i < object.size(); i++)
  {
    object[i][0] = (double)countX / 32.0;
    object[i][1] = (double)countY / 32.0;

    countX++;
    if(countX >= width)
    {
      countY++;
      countX = 0;
    }
  }

  return 0;
}
```

Figure 18: Looping through verts sequentially and assigning values to X and Y coordinates

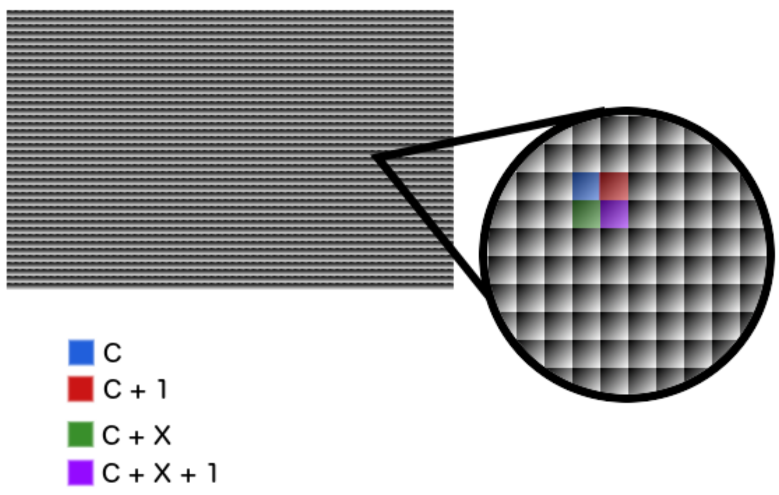


Figure 19: Simple planar surfaces are easy to define!

```
int landscape::saveLandscape(global *global)
{
  ofstream outputStream;
  outputStream.open(output);
  if(outputStream.is_open())
  {
  }else
  {
    return 1;
  }

  for(int i=0; i<object.size(); i++)
  {
    outputStream << "v\t" << object[i][0] << "\t"
       << object[i][1] << "\t"
      << object[i][2] << "\n";
  }

  outputStream<< "\n";
  for(int i=1; i<object.size()-width-1; i++)
  {
    if(i%width!=0)outputStream << "f\t" << i << " "
        << i+width<< " "
      << i+width+1 << " " << i+1 << " \n";
  }
  outputStream.close();
  return 0;
}
```

Figure 20: Notice the modulus check that prevents accidental wrap around of faces

4 TESTING AND EVALUATION

4.1 UNIT TESTING

4.1.1 *GDB and LLDB*

Syntax errors were flagged up by the compiler while runtime problems were identified with the use GNU's GDB / Apple's LLDB. The devleopment process itself was separated into to two main sections: testing and feature implementation. Most features of the program were tested in a separate smaller program before then being re worked as part of an OO design inside the application itself.

The majority of problems were in fact as a result of documentatin sparsity, particularly in regard to OS X and Allegro5. Code syntax was correct however debugging the crashes upon starting was made more difficult by the lack of progress at all through the application, and references to functions in the error code that were difficult to interpret / solve without corresponding documentation.

One of the key flaws in the code that lldb helped me to debug was the linearmap function shown in Figure 15. In a prior version of the function, the values were being normalised sample by sample, before the complete range was established.

In the not so rare event that the 'big' value equaled the 'little' value, a divide by 0 would occur and the entire fftw_complex structure fell apart as a result.

Without lldb or gdb, resorting to print statements to identify the exact fu function and print the value of all the variables would have been the only option.

4.1.2 *Print Statements and Allegro*

Semantic errors were harder to isolate as a result of the depth of the maths involved and the nature of the sound input. The most effective means of observing whether or not the application was working correctly was to draw it to the screen after each step.

Appendix B Show's the stages the visual output went through as the code developed. Many of them were quite interesting in their own right, but were unpredictable in regards to consistent reproduction.

Printing values before and after transformations helped to narrow down the exact errors, which were mostly incorrect normalisation values.

There was however an issue in an earlier version of the saveLandscape() function. Rather than using vectors to store the object vertices a valarray was used. The original reason for doing so was to efficiently set the similar values of the X and Y coordinates during vertexSetup(), as so many of them had the same value.

While the earlier tests were successful, moving the code into the main project resulted in strange error were the wrong coordinates were accessed. After some time spent attempting to resolve the problem, valarrays were dropped in favour of the better known vector class. This solved the problem immediately, so there may be a range of support between compilers for the valarray object.

Beyond the demonstration nature of the project, the allegro functionality can be removed to further advance the efficiency of the software.

4.2 PRODUCT EVALUATION

4.2.1 *Original Requirments*

Blender Compatibiltiy

The resulting .obj is fully compatible with the latest Blender software. Notably however, the 2 dimensional preview images do not exactly reflect the nature of the final object. Specifically, the smooth scaling value appears to be currently limited to a small range in order to preserve the 'landscape esque' nature of the object. Though this has the potential to be altered later.

.wav Input Files

5 Files have been tested locally all with consistently similar outputs, unique to themselves, while a second individual has input several 'synthesised' .wav sine / triangle / sawtooth functions, all of which have been handled well within the application.

Ubuntu Target Platform

The application compiles and runs on the Ubuntu OS, however the dynamic runtime path had to be manually specified in the compilation / linking, as /usr/local/ is not by default on the path within Ubuntu, which is where Allegro was installed from Git, as the unstable branch was not available within the repositories.

Affordability

The software is currently releasable under an open source platform. If an alternatively licensed FFT library can be used then the software may be distributed close source as well.

Efficiency

section A.1 shows that the average speed per cycle is approximately 11.5 seconds. However, within the allegro functionality are two calls that pause the program for 5 seconds so the user can view the output. With a total execution time therefore of roughly one and a half seconds, future scalability potential is high.

Time values were obtained by running the application through GNU's time program and outputting to a file.

4.2.2 *User Evaluation*

User Evaluation was done on a voluntary basis with one participant, due to the demo nature of the program and the priority that the program runs weighted above the desire for feedback of the actual functionality.

Participant A ranked their own computer literacy and background using a quantitative weighted response survey, as well as qualitative natural language feedback over the user experience of the software.

Participant A, ranked them self highly computer literate but a none expert in the domain of 3D terrain generation. They weres able to quickly and easily install the library dependencies and build the program from source without difficulty. See section A.3.

Their natural language feedback identified a few issues, particularly with the quiet nature of the application when handling incorrect input, and that the program indicated successful completion while printing out to stderr.

They did not understand the nature of the output, or the functionality of the program, and as such was unable to interpret or evaluate the success of their output, pointing out a lack of documentation describing what the program was doing.

5 CONCLUSION

In regards to meeting my requirements the project was a success, Figure 21 and Figure 22 show that my resulting output is convincing tileable object that retains characteristics that originate from the originating sound file.

Because SILO meets these criteria, it successfully demonstrates the link between visual and audio phenomena, and presents in such a way that a subject that might be considered highly academic may be considered by individuals of a more artistic or visual nature, rather than being put off by the daunting formulae involved.

In addition, the program that constructs these objects is well structured and suited for further development, handles its memory allocations appropriately and runs quickly and efficiently (although how quickly is processor dependant). It is cross platform, working on Ubuntu and OS X, assuming the system it is being compiled on has up to date development tools. Furthermore, with the exception of the FFTW library, the entire project is suitable for release under the LGPL or GPL as a none free application.

There also lies the potential for future projects that students (particularly those in the ilk of 'Creative Computing' or 'Music Technology') may find appealing.

Figure 23 shows some outputs that are typical of those being produced by SILO.

5.0.3 *Further Work*

FFTW replacement

A maths student may find it interesting to work on a project to produce a library that rivals/ stands in for FFTW. This would allow marketing of the product software without the needs to pay MIT for a non free license. This would greatly aid in meeting the affordability requirement, and pose a good challenge for any student; particularly if they are aiming to meet the benchmarks set by FFTW.

It's worth pointing out that this is one of those 'How far do you want to go?' ideas. Osgood has run entire courses at Harvard University focused on the Fourier Transforms applications, in preparation for this

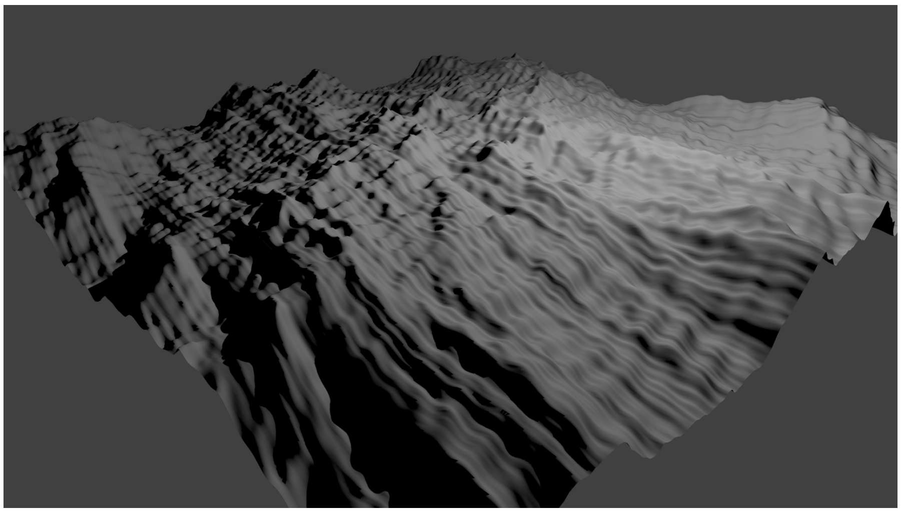


Figure 21: Smoothing factor 2.1, input track was an electric guitar high pitched solo

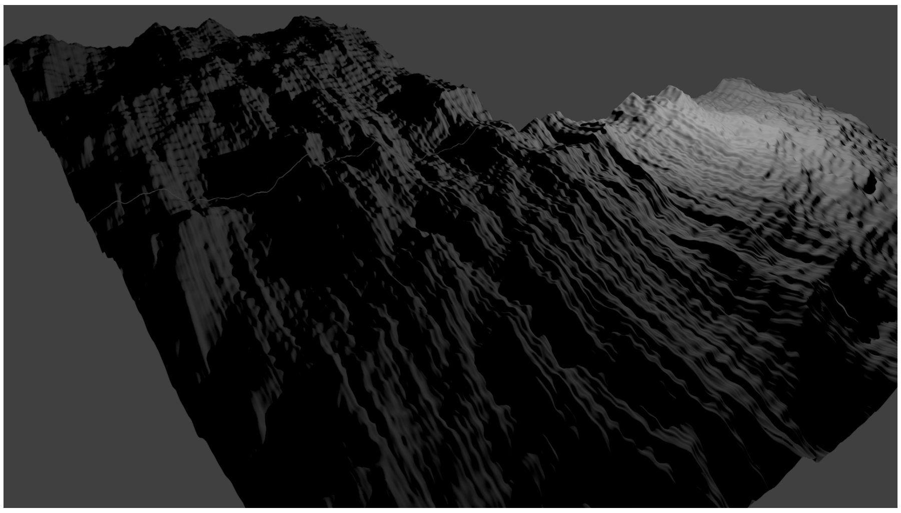


Figure 22: Demonstrating the tile-able nature of the outputs

project the first 5 lectures were watched, and it took multiple repetitions of each to understand exactly what was being taught.

Between that series of lectures, Smith (1997)'s content, and other wells of information regarding the Maths this project could be in the realm of post graduate territory.

Manager

Others may find it more interesting to implement a 'Manager' class and assorted assets. The purpose would be to track a directory of library structure and track and produce 3D outputs based on the files. The class could track exactly how many times a particular file has been sampled, and select alternatives appropriately. It would also maintain a database of the outputs organised by terrain type, and which files generated which types of terrain.

This would be doubly effective with an accompanying configuration application/ text file; a user could specify the maximum number of objects produce a minute in order to save on processing space, or designate a maximum amount of storage to be allocated towards storing the objects.

Further still, branching into Artificial Intelligence and having the manager learn the ability to recognise outputs that inadequate and removing them to manage resources would be ideal.

GUI

As the resulting objects are tile-able, an individual might take it upon themselves to develop an interface to clone, snap and manipulate tiles. They could also take it a step further and integrate this interface into a Blender plugin... or develop a full blown 3D creative application.

Features could include the ability to create composite tiles out of synthesised tiles, freezing the heigh of vertices while normalising others to create distinct peaks introduction of a flooding level to add water features. The ability to save tiles as a unit to use as a brush later on would be a useful feature as well.

Going down the route of artificial intelligence, it would be possible to create a class to automate the above features into vast sprawling landscapes with different terrain types.

VST

A music technology student might enjoy developing a VST plug in for an Audio suite such as logic, where they can output other tools' advanced sound synthesis features into a 3D object: as part of these new outputs, it may be interesting to investigate whether their game play ergonomic value correspond to the original audio tracks. Does an eerie spooky audio track translate well into an object that lends itself to a horror game?

Or how about a plugin that goes in the other direction? Takes game object files such as model weaponry or textures and turns them into an acoustic event or series as means of bulk sound generation?

Or perhaps a plugin that allows you to define a 3D environment based on the impulse response of a sound file?

.mp3 Decoder

Libraries that play, read or write mp3 files are subject to warranties, and as a result are very expensive. It's for this reason that libsndfile does not support it as a format.

Despite this however, .mp3 files are very common, so it might be worth looking into ways to legally be able to read this format without infringing the legal rights of its creators.

A reverse engineering challenge perhaps?

As common as they are now though, .mp3 files are slowly on the way out in favour of friendlier formats. Notable, Apples .m4a extension has become a near de facto standard as a result of their music monopoly with iTunes. While audiophiles are trending towards lossless formats such as FLAC.

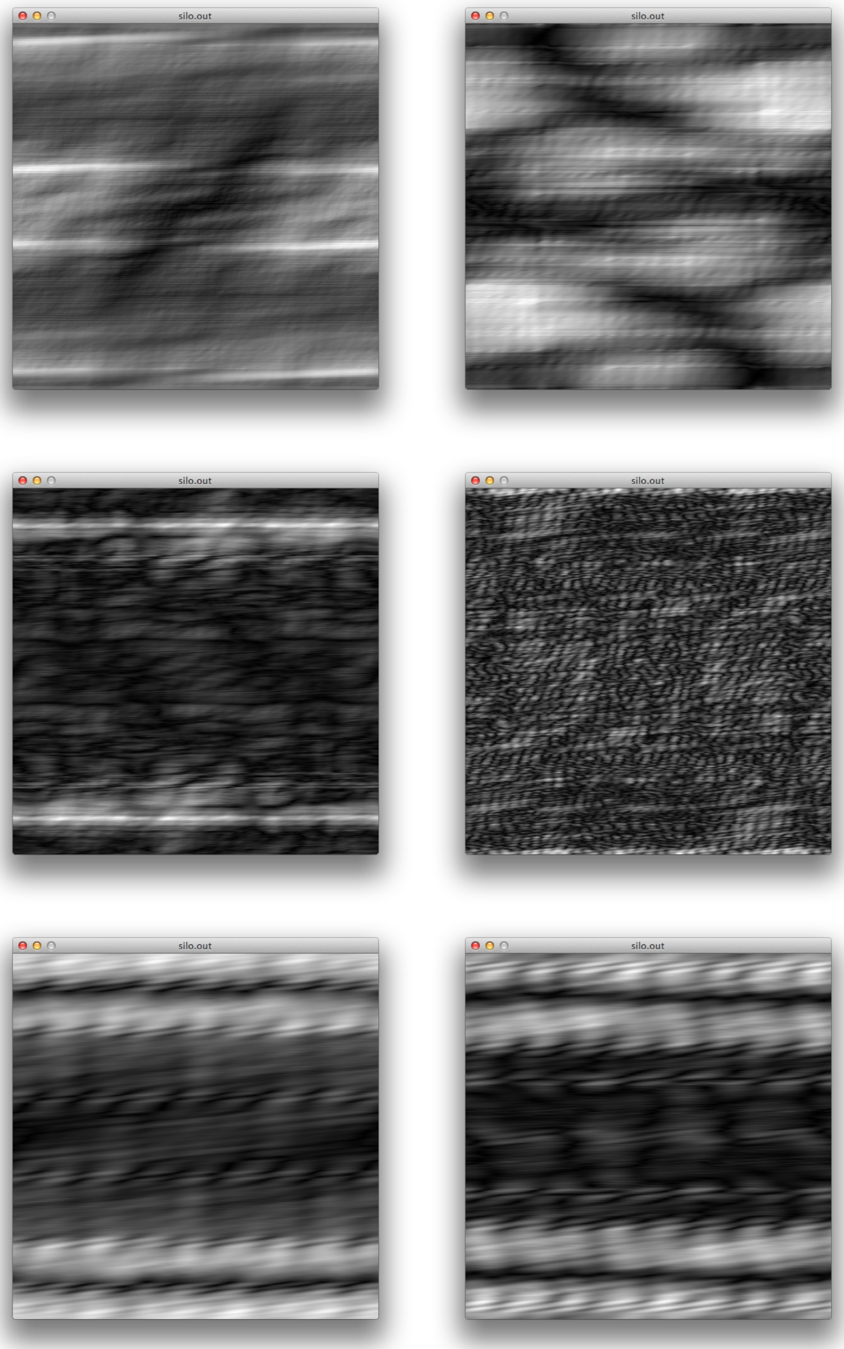


Figure 23: These files actually have a very low smoothing value and don't translate well as 3D objects, but it demonstrates the detail of the terrains gathered from the input audio

Part III

APPENDIX

A APPENDIX TEST

A.1 SPEED TEST RESULTS

```
1.32user 0.08system 0:11.55elapsed 12%CPU (0avgtext+0avgdata 171900928maxresident)k
7inputs+18outputs (0major+12077minor)pagefaults 0swaps
1.29user 0.07system 0:11.53elapsed 11%CPU (0avgtext+0avgdata 171737088maxresident)k
1inputs+1outputs (0major+12100minor)pagefaults 0swaps
1.31user 0.07system 0:11.64elapsed 11%CPU (0avgtext+0avgdata 171638784maxresident)k
0inputs+0outputs (13major+12049minor)pagefaults 0swaps
1.34user 0.08system 0:11.61elapsed 12%CPU (0avgtext+0avgdata 174555136maxresident)k
40inputs+3outputs (0major+12323minor)pagefaults 0swaps
1.31user 0.07system 0:11.49elapsed 12%CPU (0avgtext+0avgdata 171835392maxresident)k
1inputs+1outputs (5major+12102minor)pagefaults 0swaps
1.38user 0.08system 0:11.63elapsed 12%CPU (0avgtext+0avgdata 171425792maxresident)k
0inputs+3outputs (0major+12116minor)pagefaults 0swaps
1.33user 0.09system 0:11.68elapsed 12%CPU (0avgtext+0avgdata 174866432maxresident)k
0inputs+0outputs (14major+12325minor)pagefaults 0swaps
1.32user 0.07system 0:11.50elapsed 12%CPU (0avgtext+0avgdata 175374336maxresident)k
0inputs+0outputs (3major+12354minor)pagefaults 0swaps
1.31user 0.08system 0:11.51elapsed 12%CPU (0avgtext+0avgdata 172408832maxresident)k
3inputs+3outputs (0major+12145minor)pagefaults 0swaps
```

```
1.32user 0.07system 0:11.47elapsed 12%CPU (0avgtext
   +0avgdata 171589632maxresident)k
4inputs+1outputs (0major+12120minor)pagefaults 0
   swaps
1.34user 0.07system 0:11.56elapsed 12%CPU (0avgtext
   +0avgdata 172032000maxresident)k
7inputs+1outputs (0major+12099minor)pagefaults 0
   swaps
1.30user 0.07system 0:11.45elapsed 11%CPU (0avgtext
   +0avgdata 171327488maxresident)k
0inputs+2outputs (0major+12075minor)pagefaults 0
   swaps
1.32user 0.08system 0:11.57elapsed 12%CPU (0avgtext
   +0avgdata 171524096maxresident)k
12inputs+3outputs (0major+12082minor)pagefaults 0
   swaps
1.32user 0.07system 0:11.50elapsed 12%CPU (0avgtext
   +0avgdata 171573248maxresident)k
0inputs+2outputs (0major+12109minor)pagefaults 0
   swaps
1.30user 0.08system 0:11.70elapsed 11%CPU (0avgtext
   +0avgdata 171311104maxresident)k
12inputs+1outputs (13major+12033minor)pagefaults 0
   swaps
1.30user 0.07system 0:11.52elapsed 12%CPU (0avgtext
   +0avgdata 174669824maxresident)k
0inputs+0outputs (0major+12249minor)pagefaults 0
   swaps
1.35user 0.07system 0:11.59elapsed 12%CPU (0avgtext
   +0avgdata 177618944maxresident)k
1inputs+5outputs (0major+12436minor)pagefaults 0
   swaps
1.29user 0.07system 0:11.47elapsed 11%CPU (0avgtext
   +0avgdata 171671552maxresident)k
0inputs+0outputs (0major+12085minor)pagefaults 0
   swaps
1.42user 0.08system 0:11.83elapsed 12%CPU (0avgtext
   +0avgdata 172294144maxresident)k
3inputs+4outputs (5major+12131minor)pagefaults 0
   swaps
1.33user 0.08system 0:11.72elapsed 12%CPU (0avgtext
   +0avgdata 171851776maxresident)k
2inputs+9outputs (13major+12088minor)pagefaults 0
   swaps
1.33user 0.08system 0:11.61elapsed 12%CPU (0avgtext
   +0avgdata 172376064maxresident)k
```

```
0inputs+2outputs (0major+12118minor)pagefaults 0
   swaps
1.33user 0.08system 0:11.64elapsed 12%CPU (0avgtext
   +0avgdata 179814400maxresident)k
42inputs+5outputs (0major+12586minor)pagefaults 0
   swaps
1.32user 0.08system 0:11.49elapsed 12%CPU (0avgtext
   +0avgdata 171769856maxresident)k
0inputs+0outputs (7major+12100minor)pagefaults 0
   swaps
1.39user 0.08system 0:11.90elapsed 12%CPU (0avgtext
   +0avgdata 171507712maxresident)k
0inputs+5outputs (6major+12150minor)pagefaults 0
   swaps
1.38user 0.09system 0:12.28elapsed 11%CPU (0avgtext
   +0avgdata 172916736maxresident)k
1inputs+6outputs (12major+12205minor)pagefaults 0
   swaps
```

A.2 LANDSCAPE CLASS DEFINITION

```
class landscape
{
  int width;
  int height;
  int size;
  string input;
  string output;
  float smooth;

  SNDFILE *infile;
  SF_INFO inInfo;
  long maxSeek;
  long seek;
  float *soundSamples;

  fftw_complex *image;
  fftw_complex *image2;
  fftw_plan p, p2;

  vector<vector<double>> object;

  public:
  landscape(struct global *global, int width, int
     height);
```

```
    ~landscape();

    int setWidth(int w);
    int setHeight(int h);
    int setInput(char *i);
    int setOutput(char *o);
    int setSmooth(float s);
    int getWidth();
    int getHeight();
    string getInput();
    string getOutput();
    float getSmooth();

    int setInfile(SNDFILE *i);
    int setInInfo(SF_INFO);
    int setMaxSeek(long s);
    int setSeek(long s);
    int setSoundSamplesBuffer(long frames);
    int setImageBuffer(int samples);
    int setImageBuffer2(int samples);
    int setPlan();
    int setPlan2();

    SNDFILE* getInfile();
    SF_INFO* getInInfo();
    long getMaxSeek();
    long getSeek();
    float* getSoundSamplesBuffer();
    fftw_complex* getImageBuffer();
    fftw_complex* getImageBuffer2();
    fftw_plan* getPlan();
    fftw_plan* getPlan2();

    int setupVerts();
    int saveLandscape(global *global);
    int setHeights(global *global);
};
```

A.3 PARTICPANT A FEEDBACK

1. Do not agree
2. Somewhat disagree.
3. Neither Agree or disagree
4. Somewhat Agree

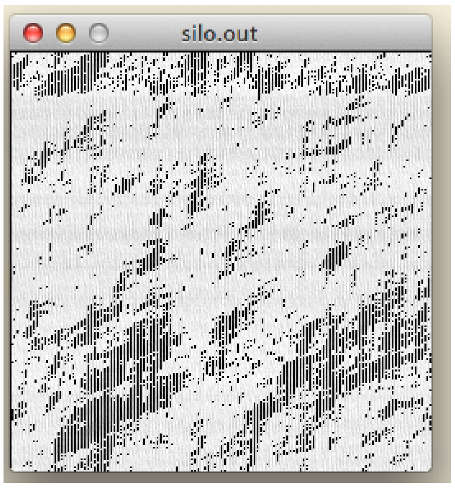
5. Agree

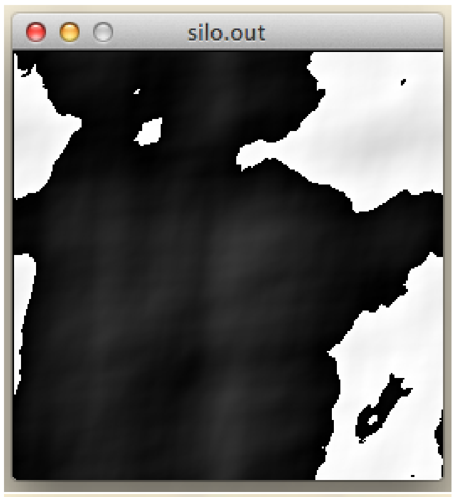
I am experienced in using 3D objects in computer environments: 1
I am computer literate: 5
I have built 3D terrains using software before: 1
I have a vested interest in computer generated landsapes: 1
I am aware of the industries of which subjects such as computer generated landscapes are used: 4
I am a consumer of the prouduts produced by these industries: 4
I have worked with sound digitally to a degree of competency: 1
I have practiced Fourier Analysis of an audio spectrum before: 1
I understand what the software is doing: 1
I understand the resulting output from the software: 1

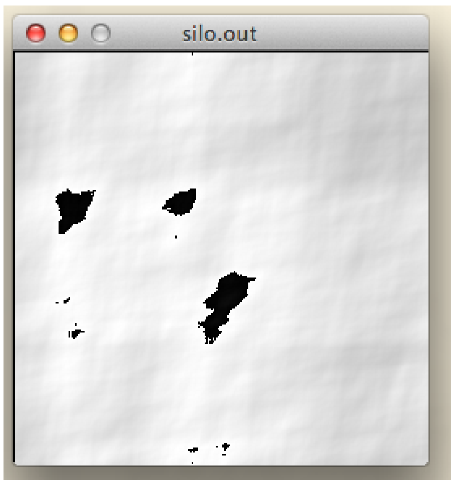
Literally no idea what the program is doing. I generated a few different sounds using an online wav generator. Sine wave, triangle wave, sawtooth wave. All of the outputs looked rather similar, none of them really looked like landscapes.

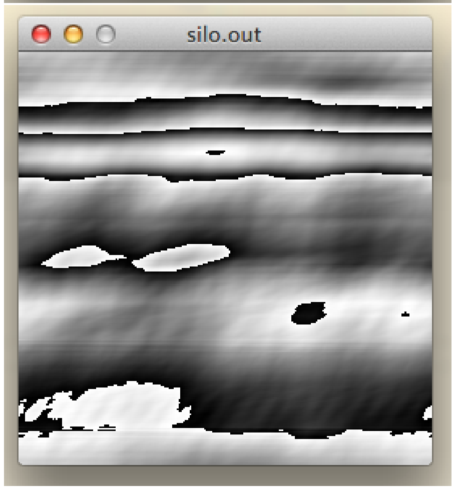
Software didn't crash at any point, and I threw a lot of interesting values at it. It didn't like me giving a -s value of 0, just exited quietly without doing anything. It compiled first time, too, and the dependencies were easy to set up on OSX. Whenever invalid input was given, a helpful message was printed to stderr but the program exits with a successful exit code. Personal pet peeve :)

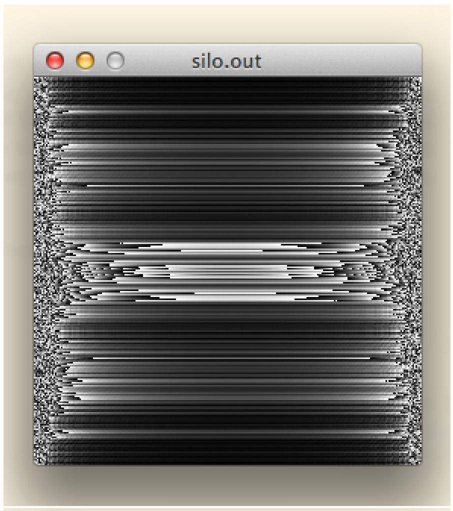
B
TEST IMAGES FROM DEVELOPMENT

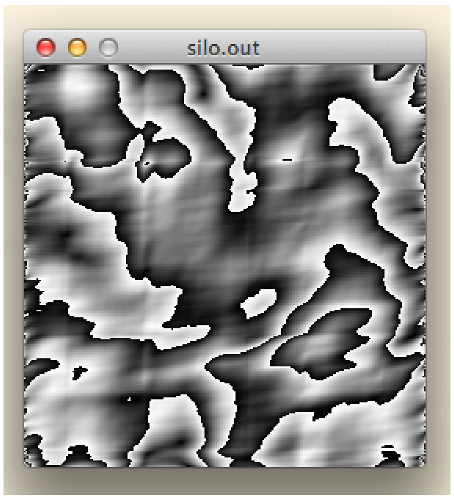


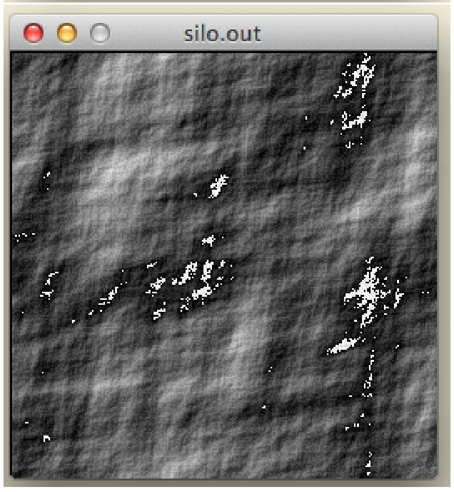







silo.out


silo.out


silo.out

BIBLIOGRAPHY

Analog.com (2014), ‘Mixed Signal Seminar’. Accessed: 2014-06-01.
URL: *http://www.analog.com/static/imported-files/seminars-webcasts/MixedSignal_Sect5.pfg*

Autodesk (2014*a*), ‘3DS Max Store’. Accessed: 2014-05-04.
URL: *http://www.autodesk.co.uk/products/autodesk-3ds-max/buy*

Autodesk (2014*b*), ‘Maya Store’. Accessed: 2014-05-04.
URL: *http://www.autodesk.co.uk/products/autodesk-maya/buy*

Belle, T. E. and Thayer, T. A. (1976), ‘Software Requirements: Are they really a Problem’, pp. 61–68.

Blender.org (2014), ‘Blender File Formats’. Accessed: 2014-05-04.
URL: *http://www.blender.org/features/*

Boulanger, R. C. and Lazzarini, V. (2013), *The Audio Programming Book*, pp. 189–193.

Bourke, P. (2014), ‘Frequency Synthesis of Landscapes (and Clouds)’. Accessed: 2013-11-01.
URL: *http://paulbourke.net/fractals/noise*

Chapman, J. (2013), ‘A Universe of Triangles’. Accessed: 2013-11-29.
URL: *http://www.youtube.com/watch?v=KdyvizaygyY*

Codeproject.com (2014), ‘Wav File Structure’. Accessed: 2014-05-01.
URL: *http://www.codeproject.com/KB/audio-video/CWave/WAVE_file_sound_data.png*

easpouse, H. E. (2004), ‘EA: The Human Story’. Accessed: 2013-11-27.
URL: *http://ea-spouse.livejournal.com/274.html*

FFTW (2014*a*), 'FFTW Benchmarks'. Accessed: 2013-11-01.
URL: *http://www.fftw.org/benchfft/*

FFTW (2014*b*), 'FFTW FAQ'. Accessed: 2013-11-01.
URL: *http://www.fftw.org/faq/section1.html*

FFTW (2014*c*), 'FFTW FAQ'. Accessed: 2013-11-01.
URL: *http://www.fftw.org/faq/section3.html#whyscaled*

Frade, M., Vega, F. and Cotta, C. (2008), Modelling video games' landscapes by means of genetic terrain programming - a new approach for improving users' experience, *in* M. Giacobini, A. Brabazon, S. Cagnoni, G. Caro, R. Drechsler, A. EkÃ¡rt, A. Esparcia-AlcÃ¡zar, M. Farooq, A. Fink, J. McCormack, M. O'Neill, J. Romero, F. Rothlauf, G. Squillero, A. Uyar and S. Yang, eds, 'Applications of Evolutionary Computing', Vol. 4974 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 485–490.
URL: *http://dx.doi.org/10.1007/978-3-540-78761-7_52*

Freestockmusic.com (2014), 'Audio Formats'. Accessed: 2014-05-04.
URL: *http://www.freestockmusic.com/audio-formats/*

Hill, M. (2013), 'Grand Theft Auto V: meet Dan Houser, architect of a gaming phenomenon'. Accessed: 2013-11-27.
URL: *http://www.theguardian.com/technology/2013/sep/07/grand-theft-auto-dan-houser*

Hnaidi, H., GuÃ©rin, E., Akkouche, S., Peytavie, A. and Galin, E. (2010), 'Feature based terrain generation using diffusion equation', *Computer Graphics Forum* **29**(7), 2179–2186.
URL: *http://dx.doi.org/10.1111/j.1467-8659.2010.01806.x*

Karmali, L. (2013), 'GTAV currently holds seven Guiness World Records'. Accessed: 2013-11-27.
URL: *http://uk.ign.com/articles/2013/10/09/gta-5-currently-holds-seven-guinness-world-records*

Katsbits.com (2014), 'Making Better Textures for Games, 'Power of T'wo and Proper Image Dimensions'. Accessed: 2014-05-01.
URL: *http://www.katsbits.com/tutorials/textures/make-better-textures-correct-size-and-power-of-tw*
php#type

Knuth, D. E. (1974), 'Computer Programming as an Art', *Communications of the ACM* **17**(12), 667–673.

Lamsweerde, A. v. (2009), *Requirements Engineering From System Goals to UML Models to Software Specifications*, John Wiley and Sons, Ltd, Glasgow, UK, pp. 13–17.

LeFebvre, D. (2013), '3DS Max File Formats'. Accessed: 2014-05-04.
URL: http://support.digitaltutors.com/entries/22828301-How-can-I-tell-what-file-format-I-need-#3dsMax

Marshall, D. (2001), 'Insufficient Audio Sampling Rate'. Accessed: 2014-05-01.
URL: http://www.cs.cf.ac.uk/Dave/Multimedia/node149.html

Miller, P. and Buckley, M. (2013), 'Game Developer Quality-of-Life Survey'. Accessed: 2013-11-27.
URL: http://gamasutra.com/view/feature/188671/game_developer_qualityoflife_.php

Murray, James, D., Ryper, V. and William (1996), 'OBJ Format Specification'. Accessed: 2014-05-04.
URL: http://www.fileformat.info/format/wavefrontobj/egff.htm

Musgrave, F. K., Kolb, C. E. and Mace, R. S. (1989), 'The synthesis and rendering of eroded fractal terrains', *SIGGRAPH Comput. Graph.* **23**(3), 41–50.
URL: *http://doi.acm.org/10.1145/74334.74337*

Osgood, B. (2008), 'The Fourier Transforms and its Applications'. Accessed: 2013-11-01.
URL: http://www.youtube.com/watch?v=gZNm7L96pfY

Pickover, C. A. (1995), 'Generating extraterrestrial terrain', *Computer Graphics and Applications, IEEE* **15**(2), 18–21.

Pierce, R. (2014*a*), 'Complex Numbers'. Accessed: 2014-05-01.
URL: http://www.mathisfun.com/numbers/complex-numbers.html

Pierce, R. (2014*b*), 'Imaginary Numbers'. Accessed: 2014–5-01.
URL: http://www.mathisfun.com/numbers/imaginary-numbers.html

Pixologic (2014), 'Zbrush Store'. Accessed: 2014-05-04.
URL: http://store.pixologic.com/ZBrush-4R6-Single-User-License

Plunkett, L. (2012), 'Every Game Studio that's Closed Down since 2006'. Accessed: 2013-11-27.
URL: *http://kotaku.com/5876693/every-game-studio-thats-closed-down-since-2006*

Reddy, M. (unknown), 'OBJ Format Specification'. Accessed: 2014-05-04.
URL: *http://www.martinreddy.net/gfx/3d/OBJ.spec*

Sinclair, B. (2008), 'Ubisoft sues over Assassins Creed leak'. Accessed: 2013-11-27.
URL: *http://www.gamespot.com/articles/ubisoft-sues-over-assassins-creed-leek/1100-6195570/*

Smith, S. (1997), *The Scientist & Engineer's Guide to Digital Signal Processing*.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of May 9, 2014 (`classicthesis` version 1.0).

DECLARATION

I confirm that unless otherwise stated the work enclosed and undertaken as part of this project is mine and mine alone. All information enclosed is, to the best of my knowledge and ability, up to date and correct.

Keele, Staffordshire, UK, May 9, 2014

Christopher Philip Ridgers,
May 9, 2014