Real-time music visualisation using 3D fractals

Paulius Bieksa -
40216180

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
BEng (Hons) Games Development

School of Computing

April 3, 2019

Authorship Declaration

I, Paulius Bieksa, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained informed consent from all people I have involved in the work in this dissertation following the School's ethical guidelines.

Signed:

Date:

Matriculation no:

General Data Protection Regulation Declaration

Under the General Data Protection Regulation (GDPR) (EU) 2016/679, the University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below one of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Abstract

Music visualisation is used from music player software to live music events. Visualising software that reacts to live music input is less common. This paper details the planning, research and development of a proof of concept for a live music visualisation application, that uses 3D fractals as the visual output.

The application captures live audio input, then runs several analysis algorithms to extract sound attributes, which are later used to dictate the shape of the resulting 3D fractal.

The 3D fractal is rendered using a raymarching algorithm, which samples a signed distance field computed by am iterated function system (IFS). The iterated function system is used to define the shape of the fractal. The transform function of the IFS uses a rotation matrix as the seed for the resulting fractal shape. This matrix is constructed by using values extracted by the sound analysis algorithms.

Performance of the algorithms used was tested. The results showed that the application can run very fast on modern hardware, producing an average of 200 frames per second in the worst configuration.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my supervisor Grégory Leplâtre for inspiring me to undertake this project, his easy-going attitude in the supervisory meetings and his constant support over the course of this project.

I would also like to thank Zoe Wall and Sam Serrels for helping me multiple times when I was struggling with CMake, git and Visual Studio. Lastly I would like to extend my gratitude towards Timothy Stuart Sanderson for sacrificing his own time to help me in figuring out difficult algorithms, when I was stuck.

1 Introduction

This project explores the viability and usefulness of music visualisation in real-time using 3D fractals. The viability refers to the ability of contemporary computers to perform sound analysis and render the resulting message in acceptably short amount of time. The usefulness, on the other hand, refers to whether the fractal images can show the qualities of the sound in a way that could be easily interpreted by a user of the software developed in this project.

1.1 Motivation

Due to the close relation music and mathematics has, there is a large range of approaches for analysing music. One niche area of music analysis, that is not widely explored, is real-time analysis to inform playing of an instrument. While raw statistics can easily become overwhelming, when they have to be observed in real-time, visual information can be interpreted much faster. Implementing a solution for visualising some key attributes of music in real-time could prove a valuable practice tool, and using somewhat more abstract type of visualisation can make it an effective performance tool.

Fractals are an excellent way of providing an abstract visual that stems from mathematics, and therefore can be manipulated based on attributes extracted from music to inform the musician. The parallel architecture of a GPU (graphics processing unit) allows the fractal to be rendered as an implicit shape acceptable time cost.

1.2 Aim and objectives

The aim of this project is to develop and evaluate an application for visualising music in real-time using 3D fractals.The fallowing objectives will need to be met in order to achieve the aim:

- Research work that has already been carried out in the area.
- Identify possible algorithms for rendering 3D fractals.
- Find or develop a framework for displaying 3D fractals in real-time.
- Find libraries, compatible with the graphics framework used, that can get real-time sound data and analyse it in an acceptable amount of time.
- Implement an algorithm that renders fractals based on sound analysis data.

- Test the real-time performance of the application on modern hardware.
- Test the effectiveness of displaying information using 3D fractals.

1.3 Scope

1.3.1 Deliverables

The items to be delivered as a result of this project are: an application for rendering 3D fractals that have their form in some way informed by sound analysis in real-time, performance tests on more than one hardware configuration, evaluation of the software's usefulness as a practice tool.

1.3.2 Boundaries

The software delivered is a proof of concept, it will not require a user friendly UI (user interface). The application will not require to accommodate a wide range of instruments for input. Any input, besides direct waveform input, is considered outside of the scope for this project. Only one type of fractal will be used for the final deliverable of the application.

1.4 Constraints

There are 2 main factors which could considerably hinder the project. The first is the lack of understanding of audio engineering and DSP (digital signal processing) could hamper the search for appropriate libraries for audio analysis, and generally reduce the effectiveness of the resulting application as an informative tool. The second is the unfamiliarity with fractal theory can cause long periods of time that can be used elsewhere to be consumed to understand how to manipulate the fractal, which could also reduce the usefulness of the application for conveying information.

1.5 Chapter outlines

- Literature review - outlines the research into sound visualisation and rendering of fractals.
- Methodology - discusses the methods and approaches used over the course of this project
- Evaluation - presents the results of the test undertaken.
- Conclusions and future work - Discusses the test results, objective fulfillment, self-evaluation, and work to be carried out in the future.

2 Literature review

2.1 Fractals and music

Fractals can be used to describe a plethora of natural phenomena. Lesmoir-Gordon and Rood (2014) write that can be characterized by a fractal distribution called 1/f noise (a.k.a. pink noise, fractal noise). This frequency distribution is found in many places in nature; such as bird songs and ocean sounds. Lesmoir-Gordon et al. suggest that deviating from from this distribution makes music sound less compelling by either making it sound too random, or too monotonous. This suggests that there could be some merit in representing information of music in fractal form.

A case for importance of imagery accompanying music is made by Ghosh et al. (2018). They quote performers of Hindustani raga music insisting that while performing or composing a musical piece, musicians have visual imagery of that particular piece in their mind, which helps them improvise. While this particular statement does not have a citation, Ghosh et al. have conducted a case study using Electroencephalography (EEG), where they found that, while performing a musical piece, the the frontal and occipital lobes undergo increased activity in musicians. While the case study uses a sample of only two subjects, the results encourage further exploration of the link between music and accompanying visuals.

It has been shown, that visual information can be processed in the brain using fractal cognitive strategies Martins et al. (2014). In their research Martins et al. have found in their research that fractal rules lead to more accurate judgment about hierarchies than non-fractal rules. Their experiment focused on image recognition, which suggests a link between fractals and information extraction from an image.

2.2 Visualising sound

There is an overwhelming number of examples of music visualisation in various media players and digital audio workstations. Both artistic and functional implementations can be found. The most relevant to this project are the ones that attempt to combine the two and get something that is both visually appealing and conveys information about the sound. One approach that has been attempted is using symatics to represent sound. Cymatics are physical impressions of sound created in mediums such as water. In his approach McGowan et al. (2017) identifies some visual features that can meaningfully represent specific qualities of sound. Some of the relations are

loudness and size, pitch and vertical position, pitch and brightness, sound dissonance and visual repetitiveness.

2.3 Implicit surfaces

There are two main approaches when it comes to rendering implicit surfaces. The most common is polygonisation. This method, however, has some major disadvantages. Polygonisation is a slow process and any dynamic implicit surface has to be recalculated after any change in topology. The other approach is using ray-tracing in a distance field. The major advantage of using this method is that it can be done directly on the GPU, taking advantage of it's SIMD (Single Instruction Multiple Data) structure, which improves the speed of the computation significantly. The basic idea is to find the intersection with the boundary of a mathematically described surface by incrementally sampling a distance field in a set direction in each fragment. Singh and Narayanan (2010) propose a version of this algorithm that uses what they call adaptive marching points. This algorithm uses a dynamic step size for the ray to find the intersection with the surface in fewer iterations. This method uses the algebraic distance to the surface and proximity to a local silhouette to determine the step size for the ray.

Ray based approaches, however are only designed to render the implicit sufaces. When it comes to storing the distance field information, different techniques are employed. One such technique is regular sampling of a distance field. A distance field would be sampled over a predefined area or volume at set intervals. The distance field equation would be solved at those sampling locations to determine the closest points to the surface. The resulting information could be used to reconstruct the distance field. This could be useful when a distance field is created in a visual environment i.e. digital sculpting. The problem with an approach like this is that to get a high level of detail a very large amount of sampling points has to be used. Even when only a small part of the represented shape requires a higher level of detail, immense volumes are required to accurately represent the shape in question. Frisken et al. (2000) proposes a method of sampling a distance field that avoids this pitfall by sampling the distance field at irregular intervals. Frisken et al. (2000) call this technique an Adaptive distance field, or ADF. ADFs use adaptive, detail-directed sampling, which is more dense in regions where the level of detail is high and less dense in regions where it is low. This technique allows for arbitrary accuracy in the reconstructed field, while maintaining a small memory footprint. To allow fast localization ADFs store the sampled data in a spacial hierarchy together with a method for reconstructing the underlying distance field. An example of a spacial hierarchy used by Frisken

et al. (2000) is an octree.

2.4 Calculating normals for implicit surfaces

After finding the surface points of the fractal, it can be rendered. However, in most applications, some visual fidelity is required. The first problem encountered at that point is that a fractal set by itself can not be solved for surface normal information, which is required for lighting computations. When these 3D fractals are rendered in a scene, or lit in any other way, a solution is required. One simple way of solving, or possibly sidestepping, this problem is to substitute the normal vector of some primitive geometry, like a cube or a sphere, at the surface points. Martyn (2010) has developed an interesting approach to solve this problem. In his approach the fractals are are stored as the smallest subset of a self-similar set (based on defined level of detail). A convex hull is calculated for that subset, and then used for the normal substitution when reconstructing the fractal using GPU instancing. As the convex hull closely approximates the surface of the fractal, it can be used for fairly convincing looking lighting calculations. To render the fractals, Martyn (2010) uses screen oriented circular shapes positioned at the fractals surface. This is a stand-in for a point, in point cloud based rendering techniques. This method for rendering fractals can be used in real-time applications, on modern systems.

2.5 3D Fractal generation

Generation of three-dimensional fractals is not a widely explored field. The main area of focus in this field seems to be the expansion of Julia and Mandelbrot sets to three-dimensional space. A common approach, to achieve this goal, is generating the fractals using quaternions and using a projection of the result in 3D space. These fractals would usually be rendered using a ray-tracing algorithm. A different approach for generating three-dimensional fractals was proposed by Cheng and Tan (2007). They propose a method of generating fractals using ternary algebra. This method, according to Cheng and Tan (2007) is superior to quad-algebra-based approaches, because it is more intuitive, less time consuming, and can control the geometric structure of the resulting sets. According to Cheng and Tan (2007) the resulting fractals are more similar to their two-dimensional counterparts, than when using quaternion based approaches. Another advantage is that their approach does not require any projection operation, which insures there is no information loss.

2.6 Fractal rendering in real-time

The time taken to render fractal images has to be short for real-time visualisation, therefore it is prudent to investigate other attempts at real-time rendering of 3D fractals in the context of performance.There has been an attempt to generate 3D fractals in the virtual world - Second Life. Bourke (2009) has used the internal scripting language of second life to manipulate in-game objects into 3D fractal shapes, however he notes that the environment of Second Life is not well suited for this task. Other attempts have been made to generate 3D fractals in real-time. McGraw (2015) uses kaleidoscopic iterated function systems to create fractals on hardware available from the year 2013. McGraw (2015) states frame-rates of 8 to 40 frames per second. While the lower range of that frame-rate is not adequate, computer hardware has advanced in recent years, where generating the fractals should not cause a bottleneck in a real-time system.

2.7 Iterated function systems

There are several ways of calculating fractal geometry. This project uses Iterated Function Systems (IFS), due to availability of examples and the fact that fractals generated this way tend to have a pleasing symmetrical look. An IFS is a topological space X together with a finite set of continuous functions $f_n : X \to X, n = 1, 2..., N$ Barnsley and Vince (2013). These fractals can be generated using fairly simple algorithms. As the name implies, an IFS fractal uses iteration of a transformation function to generate a fractal shape.

2.8 L-systems

A different way of generating fractals is using L-systems. An L-system or Lindenmayer system is an iterated system based on rewriting strings of symbols that describe the topology of the resulting fractal. It is most often used for generating shapes that resemble plants. Hamon et al. (2012) use L-systems to produce virtual plants. They have further modified their algorithms for an L-system to make it interactive. The algorithm modifies the resulting fractal shape based on a collision detection algorithm.

2.9 Common implementations

Some examples of 3D fractal rendering can be found online. There are also informative articles written on by people who create large number of fractals, to teach others on the subject. The fractal algorithm developed for this project is heavily influenced by these articles and examples Quilez (2017); Malin (2017).

3 Methodology

This section discusses the decisions made and approaches used in the development of the real-time music visualiser application. The first part of this process is feature overview. This will provide the context for the choices made in the development process over the course of this project. The feature overview is also a useful tool for project management.

3.1 Project management

This project uses an Agile development approach. The explorative nature of this project makes the traditional waterfall model unsuitable, while Agile development is designed with prototyping in mind. Iterative design was used as a specific Agile methodology, because of its focus on prototyping. Iterative design approach is based on a cyclical process of prototyping, testing, analyzing, and refining the software over the course of development, the testing of the previous cycle influencing the next prototype. This approach is particularly useful for this project because, background research informing the development process is continually conducted during said development process.

Several tools were used to help with project management. Early in the project, features for the application developed were identified and then prioritized used a MoSCoW analysis. Moscow analysis is discussed further in the MoSCoW analysis section. Additionally a project timeline was established in the form of a Gantt chart. A Gantt chart is an essential tool for managing a project; it displays the the tasks required to complete the project and the predicted time requirements for those tasks. A Gantt chart is usually updated over the course of the project to reflect the progress of the project and adjust the plan for the remaining time budget.

In terms of code management, a version control system was used. Version control allows for versions of the source code to be saved and archived. This is extremely useful when making experimental changes to the code. Version control also introduces the option of freely switching between different versions of the source code. It can also be used to store a copy of all the versions saved on other computers as a central repository when working in teams or on multiple machines, or as a backup in case of loosing the local copy. The Version control system used in this project is Git, and the web service used to store the repository online is Github. Git is the most widely used version control system in the world [2016b], which makes it easy find resources on how to use and solve issues with git. While it is encouraged to commit

(save a version of) the work done frequently when using version control, this practice was not rigidly followed in this project. This could have cost some development time, however no significant effect on the project was observed was observed. Git submodules were used to include external libraries as part of the project without polluting the main repository with the contents of those libraries.

Google Drive was used to store various project management materials (project timeline Gantt chart, reviewed literature list, etc.). Google Drive is a cloud storage service that has access to several digital office tools. It was chosen because it is free, and because of the office tools.

3.2 Programming environment

The main programming language used in this project is C++. This language was chosen because of two main reasons. First, C++ is a compiled language, as opposed to interpreted. Compiled languages reduce the code to a set of low level instructions, before being save to an executable, while interpreted languages need to translate the code into those low level instructions at run-time. This, in most cases, means that greater performance can be achieved using a compiled language over an interpreted one. The second reason for using C++ is that it is a popular language. Because using libraries for certain parts of the developed application was planned from the very beginning, a language with an established codebase was needed to have a greater variety of choice when selecting libraries to incorporate into the project.

Some of the code had to be written in a shading language. GLSL was chosen for this purpose, because it part of the OpenGL graphics library, which is the most popular open source graphics library at the time of writing, which makes finding resources for it easier. Additionally OpenGL is considered faster than its competition in some areas [Green (2006)]. Prior exposure to OpenGL an GLSL also influenced the decision to use this graphics package over others.

An integrated development environment (IDE) was used to help facilitate code writing. Visual Studio was the IDE of choice for this project. It incorporates many features that made it a preferred choice over other IDEs. Firstly Visual Studio has an inbuilt compiler and debugger; both of which are instrumental for smooth development. Additionally it uses a powerful code-completion aid called IntelliSense. IntelliSense was considered a very attractive feature when considering the choice for an IDE, due to familiarity, and personal experience with several popular code-completion aids.

To avoid executables, solution files, and IDE configuration files polluting the repository, a build automation software called CMake was used. CMake is very useful for setting up configurations for project builds. In this project CMake is used for the additional purpose of configuring and structuring the project classes and libraries used.

3.3 Features

The software developed over the course of this project, requires some must-have features that heavily influence the choices made in the development of said software. The main must-have feature being real-time audio analysis. This and other features will be discussed individually in the Feature overview section.

The main technique used to prioritize features is the MoSCoW method. This method is flexible and allows for changes during the development process [Cline (2015)]. The technique is very simple and the therm MoSCoW itself is the acronym for the rules used when employing this method - Must have, Should have, Could have, Won't have. All features of a project are categorized into one of these four categories. The categories themselves represent the priority of the feature. These categories are described as:

- Must have - must be included for project success.
- Should have - critical to have but may be worked around in a worst case.
- Could have - desired but of lower priority.
- Won't have - explicitly excluded or deferred feature that will not be in the current project scope.

3.3.1 MoSCoW analysis

Following the research done in the literature review a list of features can be outlined. Each feature is assigned a MoSCoW rating.

3.3.2 Feature overview

After formulating and prioritizing the features to be implemented in the developed application, design choices can be made. A pattern can be noticed here of using external libraries to speed up development time, and to abstract

Feature	**MoSCoW rating**
Live sound input	Must have
Real-time audio processing	Must have
Rendering of a fractal in real-time	Must have
Input through external devices	Should have
Fractal value range normalising based on input range	Could have
Output device independent on input device	Could have
Comprehensive analysis of input audio	Should have
Fractal output directly influenced by audio analysis	Must have
Real-time extraction of spectral features of the audio	Should have
Capability for multiple input streams	Won't have
Multiple fractal functions	Won't have
Surface lighting of the fractal	Could have
Mappings for direct control of fractal feature/topology	Should have
Quality of life features (Multiple resolutions, GUI interface, etc.)	Could have

Table 1: MoSCow analysis if the major features planned for the Visualiser application

as much of the audio-related programming as possible. The deliberate attempt to use libraries that provide abstracted controls for the audio-related side of the project was made due to having no previous education in the field of audio engineering.

The following paragraphs describe the choices made in consideration for each feature.

Live sound input. This feature requires sound to be captured directly from a microphone or another audio input device in real-time. This is usually a low level process, which involves directly dealing with raw audio samples. It was decided that a library should be found to handle this feature. There are many libraries that can be used to handle this task. After some consideration, it was decided that PortAudio - a free, open source input/output library would be used. The library being open source was a one of the factors that influenced the decision to use it. Additionally its simple design and adequate documentation were, in a large part, the deciding factors to use the library, as well as its ability to utilize external audio devices.

Real-time audio processing. The analysis of audio is used as a tool for this project, and is not the focus of the project itself, therefore this feature

had to be abstracted as much as possible with a help of a library. The real-time nature of the project required a library that can extract useful sound features in a relatively short amount of time. After some searching it was found that most choices for audio analysis libraries are not designed with live analysis in mind. While this does not mean that those libraries are not capable of that, one library was found that, among other things, was specifically designed to be fast. Gist [Stark (2014)] was chosen in par because of this. Another thing that strongly influenced this choice was the fact that the library had a very simple structure. All of the analysis functions could be used directly by supplying a buffer of audio samples to process. Figures describing the qualities of sound can be obtained without any understanding of audio engineering or digital signal processing.

Rendering of a fractal in real-time. After some research it was clear that it is more advantageous to render the fractal directly on the graphics processor (GPU) than the central processor (CPU). Most research suggested various ray marching approaches done in parallel on the GPU. It was decided to use a screen space marching algorithm coupled with a mapping function to implicitly define the surface of the fractal, as this was shown to produce 3D fractals in an acceptable amount of time. To achieve this a C++ graphics rendering framework [2016a], developed by Edinburgh Napier University, was used. This framework was chosen because of the previous exposure to the framework, and the fact that it is freely available under the terms of the GNU General Public License. The framework uses OpenGL for handling graphical elements, and GLSL as the shading language.

Input through external devices. The inbuilt audio devices in most consumer computer hardware only have the capability to capture sound through a microphone. This means that in a lot of situations noise reduces the overall quality of a signal. In most real-world scenarios, when working with music, specialist equipment like a mixer or audio interface hardware is used in addition to a computer. Capturing a signal from that specialist equipment increases the usability of the application being developed in this project. Another problem with using only internal computer hardware is latency. Latency is notoriously high when using non-specialized hardware. These issues were considered when selecting the audio input library and, as mentioned in the Live sound input paragraph, PortAudio was chosen, partly because it can support external devices.

Fractal value range normalising based on input range. This feature was considered as a low priority quality of life feature. The idea being that the software would adapt its output mapping range to the detected minimal and maximal values detected in the audio input. This would mean that an instrument could utilize the entire visual output range no matter of the limitations of the instrument. This feature was not included in the final product as the development time required to achieve this was deemed too great a cost for the benefits it brings. Instead, the program was manually tuned to use the range of a standard tuned guitar for the pitch mapping. The values outside of that range are still accepted, but are mapped to the output in a cyclical fashion. This was deemed enough for the proof of concept nature of the application developed.

Output device independent on input device. This is another quality of life feature that has been cut from the project. This was found to be a problem requiring a very complicated solution and ultimately not worth the large amount of development time it would require.

Comprehensive analysis of input audio. This feature refers to extracting sound features which can be useful as immediate feedback to a musician. Due to this feature occupying the domain of audio engineering, there was an attempt made to use an audio analysis library to get the relevant sound attributes. The Gist library, mentioned when discussing the Real-time audio processing feature. The Gist library can extract multiple sound attributes, however after some experimentation most of those features were shown to be close to constant or otherwise not useful for the purposes of a visualization. The attributes extracted by performing audio analysis are sufficient for a proof of concept, but could not be called comprehensive.

Fractal output directly influenced by audio analysis. This feature represents the very core of the application developed in this project. To implement this, variables that directly influence different aspects of the fractal have been identified in the code of the fractal function. Additionally useful values describing the sound have been found and mapped to said variables. This lead to the rendered fractal being noticeably influenced by the audio input.

Real-time extraction of spectral features of the audio. Spectral features can be used to describe the timbre of the sound. This feature is separate from the Comprehensive analysis of input audio because it requires a Fourier Transform operation to be performed. At the early stages of the project, when these features were identified, it was unclear whether this can be done in acceptable time-frames. After a small amount of research it was found that there is a category of Fast Fourier Transform (FFT) algorithms, which are fast enough to be used in real-time for reasonable buffer sizes, and most audio processing libraries, including the library used in this project - Gist, use FFT algorithms.

Capability for multiple input streams. This could be a useful feature for analyzing direct input from multiple instruments simultaneously. This would be a good addition if the program was extended for actual real-world use, however it is not needed for a proof of concept.

Multiple fractal functions. This feature could be used to give the user of the application choice on the type of fractal visual displayed as well for as displaying multiple fractals at the same time. The latter could be especially useful when paired with capability for multiple input streams. This feature, however is also not useful for a proof of concept.

Surface lighting of the fractal. Lighting could be used to better show the shape of the fractal and increase visual fidelity. This feature was considered low priority and was ultimately cut, due to time constraints.

Mappings for direct control of fractal feature/topology. Direct control of the fractal appearance is preferred to indirect. The mappings in the application developed can be seen as having a direct effect, however it is difficult to understand the correlation between the discreet states in the fractal and their position in a continuous scale. The fractal function used in this project works by iteratively scaling, translating and rotating points in space, which makes it difficult to control the resulting shape in a more direct manner.

Quality of life features (Multiple resolutions, GUI interface, etc.). Small quality of life features are lumped into this category. They are considered low priority and were planned to be implemented as time allows. They

do not influence the design of the application and are here only to improve usability if the project time allows.

3.4 Design and implementation

This section discusses in detail the design and implementation of the Visualiser application.

3.4.1 Application architecture

The Visualiser application can be broken down into three main components:

- audio handler
- real-time engine
- fractal rendering

The first is less emphasized and the former two are considered the focus of the project. The real-time engine houses the entry point for the application and controls the execution of other components. The engine operates in cycles, in each of which, the audio handler is invoked first, then fractal rendering afterwards. The audio handler itself is made up of two parts - audio capture and analysis, both of which are handled separately by external libraries. PortAudio is used to capture sound, and Gist is used to analyze it. Fractal rendering is done entirely on the GPU. Instead of taking the traditional approach of sending 3D models to the GPU, the shape of the fractal is calculated in the final stage of the graphics pipeline - the fragment shader.

This architecture model allows for independent development on different aspects of the application with minimal interdependence between system components. All components are designed to work in a context of a temporal frame. Each component takes care of its own logic and either passes or receives the end result of the relevant computations.

The application starts by initializing the audio and video components. From then on, up until the program is closed, the application iterates through sound capture, sample analysis, sound attribute filtering, value passing to the GPU, and fractal drawing stages. Each iteration is considered a temporal frame and the time between frames needs to be small enough so each individual frame would not be perceived as an individual image.

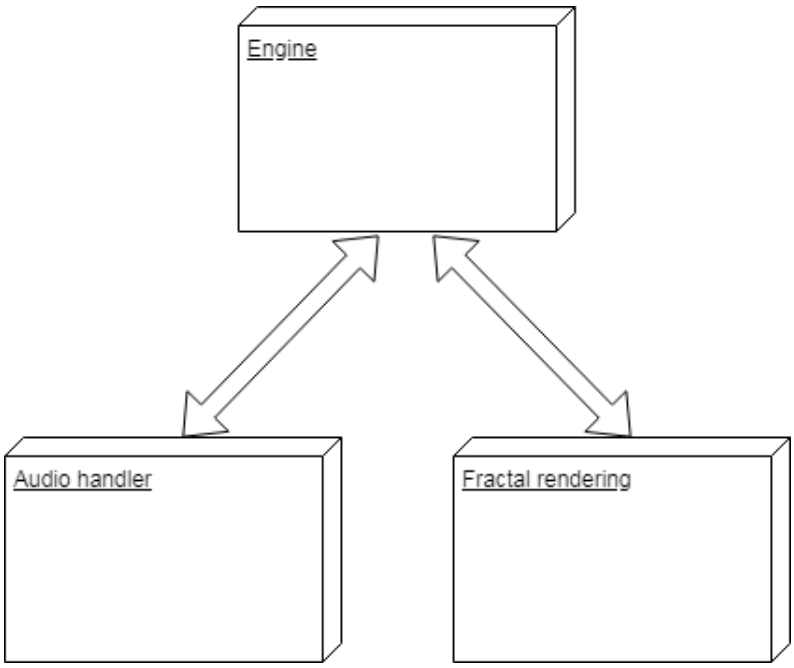


Figure 1: Software architecture

3.4.2 Audio capture

The Visualiser application is designed to capture live audio, which implies capturing the sound through input devices. This is a low level process that requires some interaction with the operating system. Luckily there are many libraries that can can do this, and let the programmer focus only on designing the parts of the software that are relevant to them. This project uses one such library, namely - PortAudio. The library was chosen because of a few reasons. Firstly, PortAudio is open-source and cross-platform. The application developed in this project is itself open-source. Additionally open-source software tends to be of better quality [Wheeler (2015)], when it is reasonably popular. Secondly, after reviewing the documentation for PortAudio, it was determined that including this library would incur a relatively small time-cost on the project, as the user-facing architecture for the software was well designed and simple, and there were multiple example programs, that explain the operation of the library efficiently. Lastly PortAudio comes with a CMake configuration file, which configures the library for inclusion to projects using CMake.

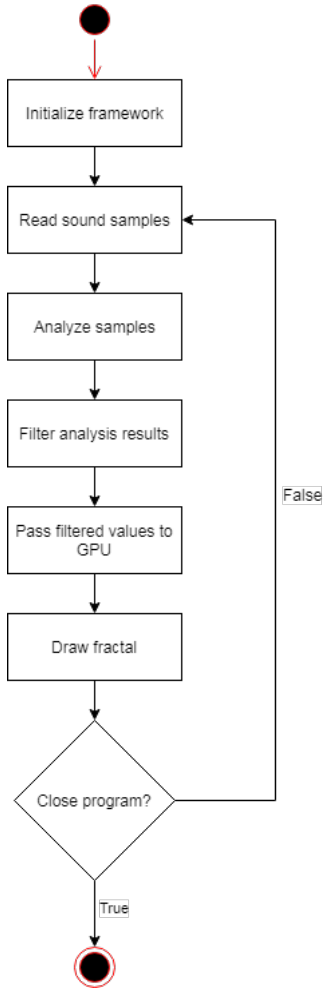


Figure 2: Activity diagram

PortAudio integration Integrating PortAudio into the Visualiser application required some design techniques that require some understanding of operating software architecture. Specifically, PortAudio requires the user to define a callback function, which is invoked by the operating system through interupt handlers. This function has to be defines as static and follow a strict definition. It is very highly recommended that no functions that take an unbounded amount of time would be used in the body of the callback function. The function gets audio samples from/to the sound drivers of the operating system.

PortAudio uses the concept of streams to manage live input/output of audio. Each stream has a callback function associated with it. The stream is also

defined by parameters such as - input/output device, number of channels used for input and output, sample data type, input/output latency, sampling rate, size of audio frame in samples and host API (driver used). The stream, once started, associates the callback function with the specified audio device.

The implementation of these control mechanisms for PortAudio in the context of the Visualiser application is designed to output the input signal without modifying it, and to retrieve the latest audio frame on request. These things have to be performed within a small enough amount of time to not be noticeable by the user of the application.

3.4.3 Audio analysis

The second significant component of the audio handler module in the Visualiser application is audio analysis. This part in particular requires knowledge of audio engineering, therefore it had to be implemented with an external library. Having to write code to to analyze the audio would have required a significant time investment in researching the broad topic of audio engineering.

Early in the development process, an application under the The GNU General Public License was found that analyses audio in real-time an was used to inform a live visual accompaniment to musical performers. The software is called Sound Analyser [sta]. It is designed as a plugin for digital audio workstations. It incorporates JUCE - an open-source application framework. The Sound Analyser application was considered for the project, and attempts were made to incorporate it into the final application. However, it was found that the application has to built through the JUCE tool, which has its own graphical output tools. These tools are not designed for computing the visuals directly on the GPU. Additionally, the Sound Analyser application itself was very poorly documented. After some time and effort it was decided to abandon any continued attempts to incorporate this software and look for other approaches.

Another course of research was conducted and a library called Gist was found. This library satisfies the requirements for a suitable audio analyser, and curiously, was developed by the same person - Adam Stark. The library is designed to be very simple to use and offers access, by means of an explicit function, to multiple sound attributes calculated in a relatively short amount of time. This library was incorporated into the Visualiser application with little difficulty. The Gist library is used to calculate the relevant sound

attributes immediately after retrieving the audio frame. The sound attributes are returned to the real-time engine in a form of a C++ struct.

3.4.4 Bridging audio and video parts

The engine controls the flow of the program. It initiates the execution of both the audio handler frame and the graphics frame. First thing done in an application frame is the execution of an audio frame. The values received from the audio handler are not necessarily suitable to be used as seeds for the fractal generation. They could be scaled to correct ranges in the fragment shader, but that would break the modular approach somewhat. To avoid this sound attributes are scaled in the real-time engine module and then passed to the fragment shader through a global GLSL variable (i.e. uniform).

3.4.5 Fractal rendering

The rendering of a fractal has two main parts: a raymarching algorithm and an iterated function system (IFS).

The raymarching algorithm samples locations along a straight line starting at the virtual camera position and extending towards a direction calculated by modifying the forward axis of the camera by the x and y positions of each pixel being rendered. The raymarching algorithm works in steps, by querying the IFS function at each step to get the distance to the closest surface position of a fractal, then querying the IFS function on the next step further along the ray by that distance. When the IFS function returns zero, the ray has found the surface of a fractal.

The IFS function, as the name implies, uses iteration to construct a fractal. Each iteration, a shape is transformed by a function in each of the iterations and then combined using the union operator with the result of the previous iteration. Over multiple iterations a self-similar shape is constructed. This shape can be defined in arbitrary accuracy, however with a limited amount of screen real-estate there comes a point where no more detail can be rendered. The transformation function applied in this implementation is a rotation, which makes the final shape of the fractal dependent on the axis of rotation and the angle of rotation. Additionally the fractal colour is sampled from a 1D texture, and the sampling location is also used as an input parameter for the shader.

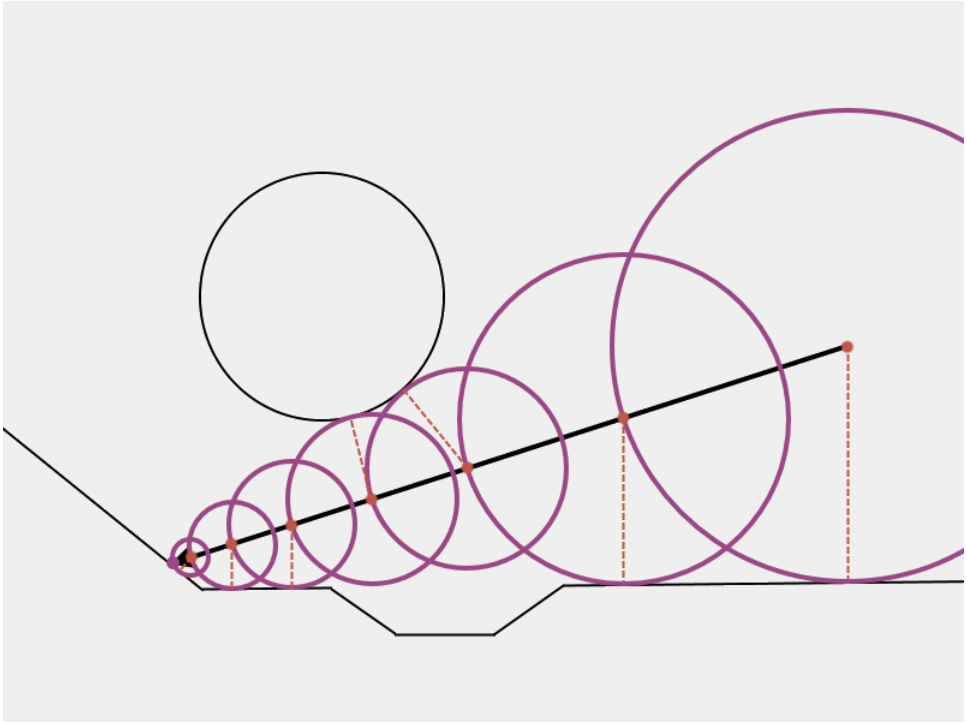


Figure 3: Illustration of a raymarching algorithm

3.4.6 Sound attribute mapping

The Visualiser application maps five distinct parameters extracted with the help of audio analysis to the inputs of a fractal shader. The parameters are: pitch, spectral centroid, spectral crest, spectral rolloff, and zero crossing rate. Pitch is mapped to the rotation angle for the IFS transformation function. The rotation angle affects the resulting fractal shape the most and in the most consistent way, which is why pitch, as the most straightforward of the audio attributes is mapped to it. Spectral centroid is described as having a close correlation to the brightness of the tone, and is mapped to the colour of the fractal. The remaining three attributes were found to be the easiest to use by experimentation, and were combined into the rotation axis for the IFS transformation function. While this mapping does affect the output, the information represented is purely visual and utterly incomprehensible.

4 Evaluation

The aim of this project was to create a proof of concept of a real-time music visualiser application that uses 3D fractals as its visual output. To this end, the application was benchmarked on different hardware configurations, to measure the execution time of the audio capture and analysis, and the fractal rendering components of the application.

An additional evaluation was planned where the program would be presented to musicians, then having the musicians complete a questionnaire on the value of the tool as a practice tool. This evaluation technique was scrapped, after trying to improve the clarity of information presentation surpassed the cutoff date of when the testing would still be feasible time-wise, while any information displayed visually was still functionally unintelligible for the user.

4.1 Performance testing

For each hardware configuration as series of performance tests were run measuring the input latency, time taken to execute a frame for the audio handler component, and the time taken to render one frame of a fractal. The latency was calculated by using a tool provided as part of the PortAudio library. The time taken to compute audio and video frames is measured using the system clock. The application is run for 5000 frames and the time taken in each frame is recorded into a file. This pattern of testing is repeated for different configurations of the software. The changes in software configuration are six different rendering resolutions and five different audio buffer sizes. The resolutions tested are 800×600, 1280×720, 1920×1080, 2560×1440, 3840×2160 and 7680×4320 pixels. The tested buffer sizes are 128, 256, 512, 1024 and 2048 samples. After some testing it was found that the latency data provided by the PortAudio tool was unreliable and is ,therefore not included.

The program was tested on two computers. All other applications were closed and the internet connection was disabled during testing, to reduce the possible inaccuracy of the results. The following are the specifications of the computers used for testing.

	Personal laptop	University PC
CPU	Interl(R) Core(TM) i5-5200U	Interl(R) Core(TM) i7-4790K
CPU clock	2.20 GHz	4.00 GHz
Operating system	Windows 7 Professional N	Windows 10 Enterprise
Operating system architecture	64 bit	64 bit
GPU	Nvidia GeForce 940M	Nvidia GeForce 980
Graphics clock	1071 MHz	1139 MHz
Memory bandwidth	14.4 GB/s	224.32 GB/s

Table 2: System information of testing machines

In a survey of Steam users the most common specifications, among the users of the Steam service, were identified [Vazquez (2018)]. This was used as a very rough metric for defining a modern personal computer. The survey found the average CPU clocks to be between 3 and 3.29 GHz, and the typical graphics card to be Nvidia GeForce GTX 1060. These statistics are inherently biased because Steam is a gaming service, but they still serve as a useful point of reference. This typical machine falls close to the more powerful end of the spectrum, when compared to the machines tested in this project.

4.1.1 Audio processing data

The following is the summary of data audio processing time data gathered. It can be seen that the time taken to process an audio frame rises quite rapidly with increase in buffer size.

Buffer size:	128	256	512	1024	2048
Intel i5 laptop	2.70E-004	0.000254215684	0.0005952712914	0.00155534289	0.003926345296
intel i7 PC	7.82E-005	0.0001390478	0.00044237434	0.00138788644	0.00292167986

Table 3: Average time to compute an audio frame

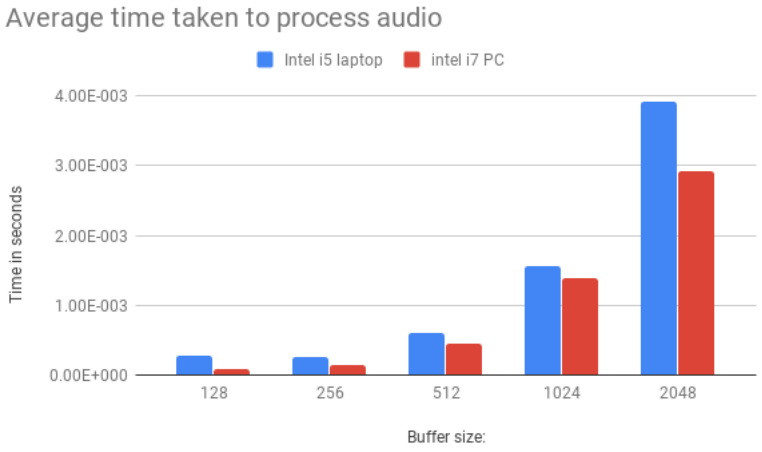


Figure 4: Average time to compute an audio frame

4.1.2 Video processing data

The following is the summary of data video processing time data gathered. Strangely, bigger resolutions take less time to compute. These findings obviously look suspicious, but another round of testing was conducted isolating just the graphics rendering part, and running it for 10000 iterations produced similar results. A lack of increase in time taken, when increasing the resolution could be explained by all calculations occurring in parallel on the GPU, but there has been no satisfactory explanation found for the opposite trend.

Resolution:	800x600	1280x720	1920x1080	2560x1440	3840x2160
Nvidia GeForce 940M laptop	0.0005646200506	3.31E-05	3.12E-005	3.28E-005	3.15E-005
Nvidia GeForce GTX980 PC	0.0008797672333	0.0006594082692	2.00E-005	1.29E-005	1.20E-005

Table 4: Average time to compute an video frame

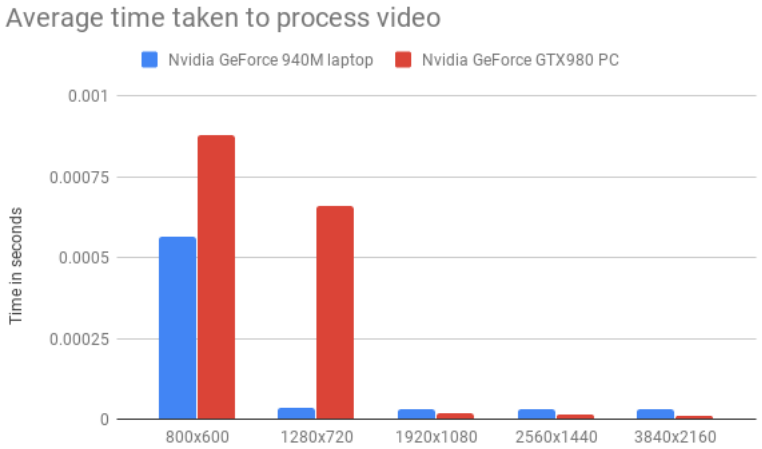


Figure 5: Average time to compute an video frame

4.1.3 Summary

The data shows that the time taken to process audio and video is extremely small. Combining the longest audio and video processing times only produces roughly 5 milliseconds. This translates to a framerate of 200 frames per second. Most real-time games are considered to be running smoothly at the rate of 60 frames per second, and even 30 is widely considered adequate.

5 Conclusions and future work

The aim of this project was to develop and evaluate an application for visualising music in real time using 3D fractals. This chapter compares the outcome of the project to the objectives set out at the start of the project.

5.1 Fulfillment of objectives

The application developed in this project can indeed analyze audio input and display 3D fractals in real-time. The audio analysis tool used in this project is not sophisticated enough to always get accurate results when more complex sounds are analyzed. This leads to a less than smooth animation at times. However, when the sound analysis tool is working well, the output can be quite visually appealing.

The output image looks too abstract to make the application useful as a practice tool. The pitch of the note is clearly related to shape of the fractal, and the colour changes based on the brightness of the tone, but the other sound attributes are show no clear patterns. While different states of a fractal can be distinguished and can be invoked by playing specific sounds, the absence of any implicit or explicit scale makes the information conveyed utterly incomprehensible.

5.1.1 Researching work that has been carried out in the area

The research of other work in the area is discussed in the literature review section of this document. A large part of this research is focused on rendering techniques. The research performed was sufficient for the development of an application that satisfies most of the objectives set out for the Visualiser application specifically.

Research has been undertaken, but more research would have likely benefited the project by reducing development time and possibly solving providing a fractal algorithm, the shape of which can be controlled more intuitively. This objective has been partially achieved.

5.1.2 Identifying possible algorithms for rendering 3D fractals

This objective was pursued during the literature review stage of the project. A group of algorithms was discovered that can be used to construct 3D

fractals. These algorithms are referred to as iterated function systems. Examples of these algorithms being used to produce 3D fractals were examined and compared to find the common approaches and common pitfalls. The algorithm used in the application developed in this project was based on this process.

With the algorithm to render 3D fractals found, this objective has been mostly achieved. Further research may have revealed a better algorithm for this task.

5.1.3 Finding or developing a framework for displaying 3D fractals in real-time

After some research into 3D fractal rendering, it was discovered that the requirements for the rendering engine were very minimal. It was found that the most efficient way of rendering fractals is directly on the GPU. Which in terms of graphics programming means - programming logic written almost exclusively inside shaders. This in turn means that the requirements for a rendering engine were: the ability to write custom shaders, ability to pass data from the engine to the shaders. Additionally, frameworks that are designed to be used as libraries, and frameworks with CMake configurations were preferred.

After a short exploratory phase, a framework that meets all of these requirements was chosen. The framework is the OpenGL Graphics Framework developed and used for teaching in the Edinburgh Napier University. This rendering engine has many more features than the project requires, but the fact that it meets all requirements, and previous work has been undertaken using this engine, made it a clear choice for this project.

With the graphics framework chosen this objective has been achieved.

5.1.4 Finding libraries, compatible with the graphics framework used, which can get real-time sound and analyse it in an acceptable amount of time

Both the capturing of sound and sound analysis int this application had to be done in real-time. Additionally, the sound analysis library had to offer tools that can find sound attributes without requiring extensive knowledge of audio engineering.

A framework called JUCE was considered for a while but after some experimentation it was decided to look for alternatives. The main reason JUCE was not desirable for this project is that the framework requires additional software called the ProJuicer to be used to integrate the framework, and it required changes to be made to the architecture of the application being developed. The audio analysis aspect was also geared more towards programmers who are more familiar with the field of audio engineering. This was not an outright deal breaker, but it was enough cause to look for better alternatives.

PortAudio and Gist were eventually chosen to handle all the audio processing needs of the application developed. While PortAudio required some additional programming to properly integrate it, Gist only required an instance of a analyser structure to be used. The advantages of these libraries are discussed in more detail in the methodology chapter of this document.

Both libraries were successfully integrated into the Visualiser application. No problems have been encountered with the final integration of PortAudio. However, either due to insufficient analysis tools in Gist or more likely inadequate knowledge of audio engineering, it was difficult to get values that are useful in describing sound generated by a musical instrument. This objective is considered mostly achieved.

5.1.5 Implementing an algorithm that renders 3D fractals based on sound analysis data

The algorithm used for rendering 3D fractals has some values that can be considered inputs for the algorithm. Values extracted by analysing the audio were scaled and mapped to these inputs to crate a dynamic fractal that represents the sound input. These mappings are discussed in the methodology chapter of this document.

While the fractal shape is clearly dependent on the audio input, the information conveyed this way is less than useful, therefore this objective is considered partly achieved.

5.1.6 Testing the real-time performance of the developed application on modern hardware

One of the goals of the project was to evaluate the viability of software that can visualise live music in the form of 3D fractals. Some performance

measuring was undertaken, and compared. The resulting measurements show that this sort of visualisation is possible in frame rates that are far above acceptable for the machines tested. The number of machines tested on is admittedly small, but comparing the specifications of these machines with the estimated typical computer hardware suggests that such software is indeed viable.

5.1.7 Testing the effectiveness of displaying information using 3D fractals

Due to the development stage running behind schedule and the fractal algorithm being completely unsuited to conveying data, this objective was scrapped. As this was defined as an objective at the beginning of the project, it is considered unfulfilled.

5.2 Reflection

The application developed over the course of the project can visualise music in the form of 3D fractals, which was the minimum viable product. The application lacks user experience features, and is overall unpolished, but it was considered a proof of concept first and foremost.

Over the course of this project a large amount of information was learned. While I tried to avoid low level audio processing by using external libraries, some time had to be dedicated to properly capture audio input. I gained some knowledge of programming for input devices. I developed a deep understanding of GPU raytracing techniques. Additionally I learned a lot about fractal geometry, implicit suface rendering techniques and distance fields. I have learned some things about project management, mostly through practice, and by learning from mistakes that were made. I have learned to better use tools like git and CMake. I also learned how to look for and incorporate external code.

If the project was repeated, I would make various changes. Project management would be changed the most, as there were many mistakes along the course of this project. Firstly, more planning would be done and the planning would be done earlier. I would aim to have specific goals defined and documented each week. Not documenting progress was a big mistake, and in future projects, strict documenting of work done after each work session would be undertaken. I would define an evaluation plan at the start of the project to better focus the development work. I would start development earlier, while still reviewing literature. This in my opinion would

increase available development time, even if early prototypes would have to be scrapped or changed, due to information fount in literature. The early prototyping would also inform the literature review process, which would hopefully lead to a better literature review.

All in all I have gained a lot of knowledge, both through research and through experimentation. This project has highlighted some of my personal strengths and weaknesses, which paints a clear roadmap for self-improvement.

5.3 Future work

The application developed, over the course of this project, is a proof of concept. This means that the focus was put on the core features of the application. The feature currently implemented could also be improved in some areas, This opens up a very large amount of avenues for improvement. Some areas for improvement are more pronounced than others though.

Finding or implementing a better sound analysis tool would make the visuals smoother. Additionally, with better understanding of audio engineering and digital signal processing more meaningful sound attributes could possibly be extracted. Finally, adding a graphical user interface and expanding the range of options the user has would bring the application closer to the realm of consumer software and out of the "proof" of concept stage.

References

Sound analyser: A plug-in for real-time audio analysis in live performances and installations.

Barnsley, M. and Vince, A. (2013). Developments in fractal geometry. *Bulletin of Mathematical Sciences*, 3(2):299–348.

Bourke, P. (2009). Evaluating second life for the collaborative exploration of 3d fractals. *Computers & Graphics*, 33(1):113–117.

Cheng, J. and Tan, J.-r. (2007). Generalization of 3d mandelbrot and julia sets. *Journal of Zhejiang University-SCIENCE A*, 8(1):134–141.

Cline, A. (2015). *Agile development in the real world*. Springer.

Frisken, S. F., Perry, R. N., Rockwood, A. P., and Jones, T. R. (2000). Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 249–254. ACM Press/Addison-Wesley Publishing Co.

Ghosh, D., Sengupta, R., Sanyal, S., and Banerjee, A. (2018). *Musicality of human brain through fractal analytics*. Springer.

Green, S. (2006). Nvidia opengl update. `https://www.nvidia.com/object/opengl-nvidia-extensions-gdc-2006.html`.

Hamon, L., Richard, E., Richard, P., Boumaza, R., and Ferrier, J.-L. (2012). Rtil-system: a real-time interactive l-system for 3d interactions with virtual plants. *Virtual Reality*, 16(2):151–160.

Lesmoir-Gordon, N. and Rood, W. (2014). *Introducing fractals: A graphic guide*. Icon Books Ltd.

Malin, P. (2017). larval. `https://www.shadertoy.com/view/ldB3Rz`.

Martins, M., Fischmeister, F. P., Puig-Waldmüller, E., Oh, J., Geißler, A., Robinson, S., Fitch, W. T., and Beisteiner, R. (2014). Fractal image perception provides novel insights into hierarchical cognition. *NeuroImage*, 96:300–308.

Martyn, T. (2010). Realistic rendering 3d ifs fractals in real-time with graphics accelerators. *Computers & Graphics*, 34(2):167–175.

McGowan, J., Leplâtre, G., and McGregor, I. (2017). Cymasense: A real-time 3d cymatics-based sound visualisation tool. In *Proceedings of the*

2017 ACM Conference Companion Publication on Designing Interactive Systems, pages 270–274. ACM.

McGraw, T. (2015). Interactive procedural building generation using kaleidoscopic iterated function systems. In *International Symposium on Visual Computing*, pages 102–111. Springer.

N/A (2016a). Opengl graphics framework. `https://github.com/edinburgh-napier/set08116_framework`.

N/A (2016b). Version control systems popularity in 2016. `https://rhodecode.com/insights/version-control-systems-2016`.

Quilez, I. (2017). Distance functions. `http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm`.

Singh, J. M. and Narayanan, P. (2010). Real-time ray tracing of implicit surfaces on the gpu. *IEEE transactions on visualization and computer graphics*, 16(2):261–272.

Stark, A. (2014). Gist - an audio analysis library. `https://github.com/adamstark/Gist`.

Vazquez, S. (2018). Steam reveals the most popular pc specs for december 2017. `https://www.gameinformer.com/b/news/archive/2018/01/06/valve-reveals-the-most-popular-pc-specs-for-december-2017.aspx`.

Wheeler, D., A. (2015). Why open source software / free software (oss/fs, floss, or foss)? look at the numbers! `https://dwheeler.com/oss_fs_why.html`.

Appendices

A Project Overview

Initial Project Overview

SOC10101 Honours Project (40 Credits)

Title of Project: Real-time audio visualisation using 3d fractals

Overview of Project Content and Milestones

The project aims to develop a stand-alone application for visualising audio input in real time. While similar work has been done before, there is no complete package, that does not require any programming knowledge, for the task. The main focus of project is to deliver a high fidelity visual output. The main milestones are: integrating a real-time audio analysis library/framework, developing a solution for rendering 3d fractals, developing a prototype where the graphical output is informed by the audio analysis, improving responsiveness, improving graphical fidelity.

The Main Deliverable(s):

The minimum viable product is an application that renders 3d fractals based on some form of audio input. The main deliverables are the major parts of the program, which are: audio input and analysis tool integration into the application, the graphics renderer and the code required to connect the two. Stretch goals will include these deliverables: increased visual responsiveness based on the audio input, high fidelity shaders.

The Target Audience for the Deliverable(s):

Musicians and live bands are the main target audience for the tool being developed in this project. The application is going to be mainly used as a performance tool, with a possibility that the graphical output can be used to inform playing of the musician.

The Work to be Undertaken:

A big part of the project will be investigation into rendering techniques, basics of sound analysis theory and existing frameworks and examples of similar work. This work will dominate the first several weeks of the project. As the project moves toward its middle stages more focus will be put on prototyping the application. Moving forward after that the focus of work will be increasingly put on developing a stable, high fidelity version of the prototyped application.

Preliminary timeline:

- 8-11-2018: Literature review completed
- 8-11-2018: Prototype for the application
- 22-11-2018: Evidence of progress for interim meeting prepared

B Second Formal Review Output

SOC10101 Honours Project (40 Credits)

Week 9 Report

Student Name: Paulius Bieksa

Supervisor: Gregory Leplatre

Second Marker: Kevin Chalmers

Date of Meeting: 05/12/2018

Can the student provide evidence of attending supervision meetings by means of project diary sheets or other equivalent mechanism? **No**

No evidence provided, but meetings have been attended.

Please comment on the progress made so far

Progress is satisfactory: some development work has been completed and some literature has been reviewed, but you need to formalise your management approach properly to ensure that you are more productive. More work on the literature review is required.

Is the progress satisfactory? **partly**

Can the student articulate their aims and objectives? **yes**

If yes then please comment on them, otherwise write down your suggestions.

The aim is to develop a system that allows the visualisation of sound using 3D fractals. It is clear why you are interested in undertaking this project, but it is important to be able to articulate the rationale for your project and what is interesting or challenging about your project. For example:

- *Can a perceptually meaningful visualisation of sound be produced?*
- *Which fractal algorithms can be used (and how) for the system to work in real time?*

* Please circle one answer; if **no** is circled then this **must** be amplified in the space provided

C Diary Sheets, Gantt charts and MoSCoW analysis

SOC10101 Honours Project (40 Credits)

Week 9 Report

Student Name: Paulius Bieksa

Supervisor: Gregory Leplatre

Second Marker: Kevin Chalmers

Date of Meeting: 05/12/2018

Can the student provide evidence of attending supervision meetings by means of project diary sheets or other equivalent mechanism? **No**

No evidence provided, but meetings have been attended.

Please comment on the progress made so far

Progress is satisfactory: some development work has been completed and some literature has been reviewed, but you need to formalise your management approach properly to ensure that you are more productive. More work on the literature review is required.

Is the progress satisfactory? **partly**

Can the student articulate their aims and objectives? **yes**

If yes then please comment on them, otherwise write down your suggestions.

The aim is to develop a system that allows the visualisation of sound using 3D fractals. It is clear why you are interested in undertaking this project, but it is important to be able to articulate the rationale for your project and what is interesting or challenging about your project. For example:

- *Can a perceptually meaningful visualisation of sound be produced?*
- *Which fractal algorithms can be used (and how) for the system to work in real time?*

* Please circle one answer; if **no** is circled then this **must** be amplified in the space provided

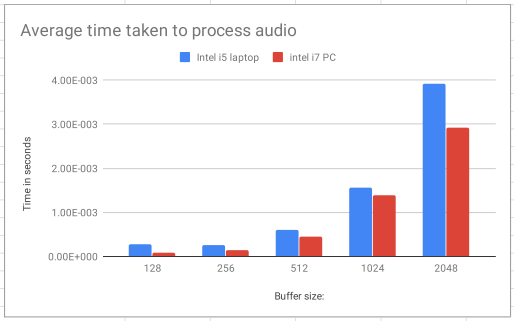
14/01 - 20/01	21/01 - 27/01	28/01 - 03/02	04/02 - 10/02	11/02 - 17/02	18/02 - 24/02	25/02 - 03/03	04/03 - 10/03	11/03 - 17/03	18/03 - 24/03	25/03 - 31/03	01/04 - 03/04
January	January	January	January/Februar	February	February	February	February/March	March	March	March	April
week 1	week 2	week 3	week 4	week 5	week 6	week 7	week 8	week 9	week 10	week 11	week 12
				Sound	Sound	Final touches	Dissertation	Dissertation	Dissertation	Dissertation	Submission
February											
Friday	Saturday	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	Monday	Tuesday
8	9	10	11	12	13	14	15	16	17	18	19
Introduction	Lit review 2	Lit review 2	Lit review 2	Lit review 2	Lit review 2	Lit review 2	Lit review 2	Lit review 2	Lit review 2	Lit review 2	Tech review
Code improveme	Code improveme	Code improveme	Code improvements								
Wednesday	Thursday	Friday	Saturday	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
20	21	22	23	24	25	26	27	28	29	30	31
Tech review	Methodology	Methodology	Methodology	Methodology	Evaluation	Evaluation	Conclusions	Future work	Final touches	Final touches	Final touches

Feature	MoSCoW rating
Live sound input	Must have
Real-time audio processing	Must have
Rendering of a fractal in real-time	Must have
Input through external devices	Should have
Fractal value normalising based on input range	Could have
Output device independent on input device	Could have
Comprehensive analysis of input audio	Should have
Fractal output directly influenced by audio analysis	Must have
Real-time extraction of spectral features of the audio	Should have
Capability for multiple input streams	Won't have
Multiple fractal functions	Won't have
Surface lighting of the fractal	Could have
Mappings for direct control of fractal feature/topology	Should have
Quality of life features (Multiple resolutions, GUI interface, etc.)	Could have

D Testing data

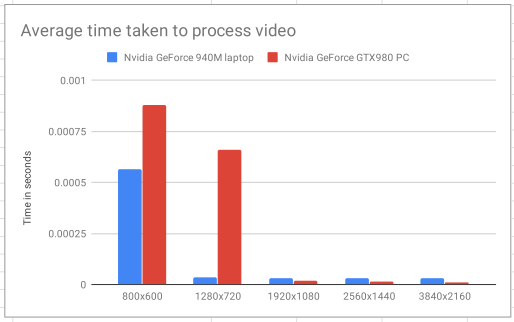
D.A Averaged data

Laptop						
Buffer size:	128	256	512	1024	2048	
Audio processing average:	2.70E-004	0.000254215684	0.000595271291	0.00155534289	0.003926345296	
Resolution:	800x600	1280x720	1920x1080	2560x1440	3840x2160	
Rendering average:	0.000564620050	3.31E-05	3.12E-005	3.28E-005	3.15E-005	
Time taken to process audio						
Buffer size:	128	256	512	1024	2048	
Intel i5 laptop	2.70E-004	0.000254215684	0.000595271291	0.00155534289	0.003926345296	
intel i7 PC	7.82E-005	0.0001390478	0.00044237434	0.00138788644	0.00292167986	



Latency					
Buffer size:	128	256	512	1024	2048
Intel i5 laptop	2.67E-003	0.005333	0.010667	0.021333	0.042667
intel i7 PC	5.17E-002	0.009833	0.015167	0.025833	0.047167

Games lab					
Buffer size:	128	256	512	1024	2048
Audio processing average:	7.82E-005	0.0001390478	0.00044237434	0.00138788644	0.00292167986
Resolution	800x600	1280x720	1920x1080	2560x1440	3840x2160
Rendering average:	0.000879767233	0.000659408269	2.00E-005	1.29E-005	1.20E-005
Time taken to process video					
Resolution:	800x600	1280x720	1920x1080	2560x1440	3840x2160
Nvidia GeForce 940M laptop	0.000564620050	3.31E-05	3.12E-005	3.28E-005	3.15E-005
Nvidia GeForce GTX980 PC	0.000879767233	0.000659408269	2.00E-005	1.29E-005	1.20E-005



D.B Personal laptop testing data

D.B.1 Buffer size = 128, resolution = 1280 x 720

Audio porcessing	Rendering
9.52E-005	0.00083788
0.000269652	0.00255983
0.000146955	0.00282668
0.00020947	0.0252647
0.000216468	0.00259948
0.000183345	0.00413668
0.000216468	0.00225892
0.00023093	0.00538184
0.000225799	0.00821458
0.000192209	0.0120224
0.000264987	0.0210734
8.26E-005	0.00579425
0.000147422	0.0141633
0.000139957	0.019735
0.000240727	0.0023009
8.49E-005	0.00328527
0.000140891	0.00605923
0.000176347	0.00916163
0.000159085	0.00998691
0.000355493	0.0120909
0.000188943	0.015435
0.000132494	0.0194653
0.000217401	0.0062883
0.000190342	0.0107278
0.000146956	0.0153939
0.000151154	0.0202248
0.000146955	0.00869184
5.27E-005	0.0132395
0.000121763	0.0184623
0.000150687	0.00580265
0.000146955	0.00787169
0.000148356	0.0111588
0.000126895	0.0147007
0.000133893	0.0178488
0.000166083	0.00274924
0.000161418	0.00264707
0.000155819	0.00491578
0.000149754	0.00896149
0.000135293	0.0122295
0.00015582	0.015832
5.32E-005	0.00353346
0.000138092	0.00664892
0.000142291	0.0100462
0.000162351	0.0132162
0.000241194	0.0149307
0.000137625	0.0169629
0.000145556	0.00271565
0.000167482	0.00434195
5.18E-005	0.00793607
5.41E-005	0.0799224
5.78E-005	0.000545835
0.000241193	0.00124282

D.B.2 Buffer size = 256, resolution = 3840 x 2160

Audio porcessing	Rendering
0.000296711	0.000612548
0.000188477	0.00619779
0.000182878	0.0646003
0.000172615	0.0527664
0.000195008	0.00201866
0.000164683	0.00130067
0.000219267	0.0202565
0.000305575	0.0297545
8.12E-005	0.0404118
0.000260321	0.0406955
0.000347095	0.0380825
8.49E-005	0.0244394
0.000372288	0.0270193
0.000204339	0.0258539
0.000193608	0.0409507
0.000194075	0.0387585
0.000197807	0.034452
0.000295311	0.0325061
8.21E-005	0.0306041
0.000180545	0.0427039
0.000188943	0.0387683
0.00018801	0.0372903
0.000201539	0.0338226
0.000239328	0.0360377
0.000310706	0.0147553
0.000283181	0.0319295
0.000245392	0.0438716
0.000211336	0.036897
0.000224399	0.020856
0.000247258	0.0212624
0.000212269	0.0320573
0.000179146	0.0447883
0.000181478	0.027629
0.000309307	0.033455
0.000271051	0.0231756
0.000178679	0.0225262
0.000173547	0.0335077
0.000189876	0.026891
0.000221599	0.0377806
0.000201539	0.0331303
7.98E-005	0.0306638
0.000276649	0.0441963
0.0002202	0.039978
0.000262654	0.035525
0.000282715	0.0303969
0.00026592	0.0423862
0.000294377	0.0394396
0.000314438	0.0339197
0.000283648	0.0300858
0.000306974	0.0397247
0.000303241	0.0273594
0.000348495	0.0381627

D.B.3 Buffer size = 512, resolution = 800 x 600

Audio porcessing	Rendering
0.000295777	0.000479588
0.000362491	0.00275857
0.000316304	0.0139552
0.000290646	0.0153515
0.000337298	0.0019748
0.000424539	0.00399486
0.000212269	0.000557498
0.000204805	8.58E-005
0.000466992	0.000268719
0.000510846	0.00348215
0.000297643	0.00186937
0.000343363	0.00347328
0.000169815	0.000345696
0.000213203	0.00207324
0.000483321	0.002188
0.000316304	0.0142878
0.00016375	7.70E-005
0.000182878	0.00484767
0.000289712	0.00130114
0.00104968	0.000376487
0.000321436	0.00106741
0.000345695	0.00161278
0.000410076	0.00101096
0.000425938	0.00132213
0.000164217	0.000477255
0.000395147	0.000642872
0.000407277	0.000537438
0.00109494	0.000110567
0.000419407	0.000871936
0.000471191	0.000590621
0.000435269	0.000619546
0.00046186	0.000885932
0.000456729	0.000651736
0.000298577	0.000514111
0.000382084	0.000634009
0.000209936	0.00089573
0.00047399	0.000702121
0.000517377	0.000841145
0.000201539	0.00185537
0.000460461	0.00134453
0.000511312	0.000868204
0.00123769	8.30E-005
0.000459528	0.000628877
0.00090226	0.000247259
0.000433869	0.00065127
0.000350361	0.00127595
0.000578025	0.00802191
0.000375553	0.000564029
0.000472124	0.000581758
0.000824351	0.00150968
0.000418007	0.000688592
0.000348494	0.000898062

D.B.4 Buffer size = 1024, resolution = 1280 x 720

Audio porcessing	Rendering
0.000482387	0.000466525
0.000509446	0.00261954
0.000520643	0.00183531
0.000695123	0.0137564
0.000677862	0.0016543
0.000590155	0.00256636
0.000433869	0.000112899
0.000466526	9.80E-005
0.000958243	0.0181031
0.000534172	0.0155031
0.000479121	0.00449264
0.000620946	0.0132983
0.000541636	0.013331
0.000650804	0.0134443
0.000635874	0.0132181
0.000587356	0.0128201
0.000480055	0.0124128
0.000752972	0.0140527
0.00080429	0.014396
0.000890131	0.0134985
0.000989034	0.0136603
0.000877535	0.0136767
0.000571961	0.014424
0.000745974	0.01524
0.000818286	0.0152824
0.000499649	0.00480428
0.000589222	0.00992813
0.000731045	0.0146386
0.00084861	0.0193818
0.000817353	0.00741776
0.000726847	0.0132932
0.000611148	0.0181441
0.00078283	0.00555026
0.000660134	0.00953765
0.000982503	0.0133151
0.000715184	0.0183923
0.000622812	0.00710565
0.000751573	0.0124623
0.000770234	0.0179216
0.000790295	0.00556005
0.000650337	0.0109046
0.000759504	0.0139953
0.000996966	0.0183792
0.00106601	0.00591974
0.00136879	0.0105024
0.000833215	0.0150795
0.000554699	0.00405644
0.000951713	0.00751713
0.000871936	0.0118185
0.000882667	0.0258553
0.000815954	0.00600932
0.00119897	0.0114798

D.B.5 Buffer size = 2048, resolution = 2560 x 1440

Audio porcessing	Rendering
0.00280662	0.000864472
0.0023275	0.00400792
0.00258268	0.0717787
0.00215488	0.0350673
0.00196827	0.000323303
0.00176907	0.000600885
0.00260881	0.0228472
0.00158245	0.0290342
0.00149755	0.0369778
0.00211383	0.0315726
0.00254396	0.0158857
0.00162211	0.031023
0.0017686	0.0292218
0.00176253	0.0351098
0.00177793	0.0228425
0.00178586	0.0250431
0.00177653	0.0332917
0.00185071	0.0293141
0.00175787	0.0452651
0.00178773	0.0432488
0.00212362	0.039796
0.00223512	0.0384678
0.00197667	0.0368079
0.00186703	0.036242
0.00323675	0.0392824
0.00208397	0.0323312
0.0018689	0.025149
0.00211896	0.034019
0.00192255	0.0272026
0.00174527	0.0390986
0.00182691	0.0371466
0.00323116	0.035205
0.00259668	0.0343284
0.00180219	0.0309027
0.00187823	0.0419747
0.00211663	0.0408019
0.00194355	0.0396211
0.00185584	0.0371378
0.00193282	0.0356752
0.00180452	0.0318343
0.00181385	0.0255399
0.00243293	0.0327646
0.00175833	0.024159
0.00182038	0.0350897
0.00202519	0.0286185
0.0017476	0.0394494
0.00189876	0.0312241
0.0016669	0.0246419
0.0017476	0.0344417
0.00180219	0.0273622
0.00178866	0.0374135
0.00177373	0.0307506

D.B.6 Rendering isolated, all resolutions

800x600
0.0107958
8.40E-005
0.0331903
0.0107319
0.000812217
0.000116631
0.00166829
0.00194494
0.00116911
0.000698852
0.0015176
0.00115371
0.00144949
0.00116118
0.00143223
0.00114998
0.00945736
0.00125541
0.00144622
0.00113645
0.00144156
0.000830412
0.000808485
0.000859803
0.000858869
0.000948442
0.000874731
0.000848606
0.000857004
0.000848139
0.00083741
0.000847673
0.00084674
0.0110851
0.000839275
0.000982032
0.0013072
0.000744572
0.00170094
0.000952641
0.00206717
0.000809885
0.000763699
0.000870999
0.000847674
0.000835544
0.00084674
0.000866334
0.000847206
0.000849539
0.000840675
0.000859803

D.C University PC testing data

D.C.1 Buffer size = 128, resolution = 1280 x 720

Audio porcessing	Rendering
7.66E-005	0.0006296
3.53E-005	7.84E-005
3.53E-005	0.0003679
4.24E-005	0.0002683
3.56E-005	0.000304
3.97E-005	0.0003226
3.31E-005	6.23E-005
7.44E-005	0.0015046
6.51E-005	0.0005759
7.28E-005	0.000977
6.38E-005	0.0005523
6.66E-005	0.0001138
6.30E-005	0.0005297
6.48E-005	0.0005163
6.32E-005	0.0005158
6.69E-005	0.000688
6.84E-005	0.0007665
6.28E-005	0.0005149
7.32E-005	0.0003991
6.78E-005	0.0008222
6.70E-005	0.0006946
7.09E-005	0.0007667
7.21E-005	0.0007113
6.68E-005	0.0019068
6.44E-005	0.0006616
6.89E-005	0.0009849
6.61E-005	0.0005802
6.37E-005	0.0007292
6.53E-005	0.0006386
6.95E-005	0.0007398
6.94E-005	0.0006605
6.44E-005	0.0005259
6.69E-005	0.0009025
6.54E-005	0.0008886
6.28E-005	0.000519
6.53E-005	0.0006632
6.25E-005	0.0005129
6.46E-005	0.000584
6.30E-005	0.0006517
6.58E-005	0.0006399
6.39E-005	0.0005838
6.33E-005	0.0005705
6.57E-005	0.0006338
6.33E-005	0.0005148
6.44E-005	0.0005681
6.27E-005	0.0005224
6.30E-005	0.0005795
6.42E-005	0.0013758
6.18E-005	0.0005678
6.44E-005	0.0005547
6.21E-005	0.0005107
7.22E-005	0.0005674

D.C.2 Buffer size = 256, resolution = 3840 x 2160

Audio porcessing	Rendering
0.0001091	0.0005322
5.91E-005	5.24E-005
5.83E-005	0.0003281
5.52E-005	0.0002137
5.03E-005	0.000196
4.95E-005	5.90E-005
5.15E-005	0.0016997
5.60E-005	0.0019771
5.89E-005	0.0055673
5.29E-005	0.0126425
5.84E-005	0.0116021
5.16E-005	0.0129707
5.35E-005	0.0147006
5.43E-005	0.0158695
5.81E-005	0.0003489
6.64E-005	0.0002114
5.66E-005	0.0024588
5.61E-005	0.0013386
5.78E-005	0.0012879
5.52E-005	0.0002603
5.33E-005	0.0004844
5.39E-005	0.0002985
5.69E-005	0.0002601
6.28E-005	0.0002133
5.77E-005	0.0002733
5.23E-005	0.0002145
5.00E-005	0.0002136
5.22E-005	0.000227
5.33E-005	0.0002282
5.14E-005	0.0002154
6.14E-005	0.000227
4.89E-005	0.0002084
5.11E-005	0.000217
5.23E-005	0.0002241
6.92E-005	0.0002967
5.12E-005	0.0009078
5.56E-005	0.0047983
5.20E-005	0.0088037
5.12E-005	0.0136194
5.23E-005	0.018012
5.31E-005	0.0066595
5.13E-005	0.0086651
5.17E-005	0.0095573
5.23E-005	0.0118894
5.97E-005	0.0148751
5.12E-005	0.0185056
5.50E-005	0.0021491
5.68E-005	0.004355
5.17E-005	0.0078959
5.88E-005	0.0099833
7.52E-005	0.0115288
5.63E-005	0.0115308

D.C.3 Buffer size = 512, resolution = 800 x 600

Audio porcessing	Rendering
0.0001835	0.0006919
0.0001414	6.84E-005
0.0001393	0.0002062
0.0001277	0.0001221
0.0001361	0.0001948
0.0001403	0.0001003
0.0001214	5.80E-005
0.0001374	0.0007787
0.00013	0.0001599
0.0001284	0.0001313
0.0001322	0.0001552
0.0001289	8.85E-005
0.0001538	0.0001487
0.0001296	0.0001406
0.0001596	0.0001332
0.0001238	0.0001357
0.0001282	0.0001585
0.0001244	0.0001151
0.0001384	4.05E-005
0.0001248	0.0001554
0.0001235	0.000139
0.000124	0.0001231
0.0001312	0.0001366
0.0001357	0.0005859
0.0001411	0.0001878
0.000128	0.0001501
0.0001388	0.0001577
0.0001292	0.0001544
0.0001485	0.0001802
0.0001228	0.0001425
0.0001307	0.0001587
0.0001279	0.0003379
0.0001441	0.0002484
0.0001295	0.0002039
0.0001331	0.0001715
0.0001285	0.0002737
0.0001264	0.000174
0.0001246	0.0002894
0.0001391	0.0002001
0.0001247	0.0002489
0.0001304	0.000153
0.000124	0.0001531
0.0001303	0.0001661
0.000147	0.0004453
0.0001257	0.0001487
0.0001268	0.0001563
0.0001305	0.0001589
0.0001679	0.0001767
0.0001249	0.0001508
0.0001248	0.000145
0.0001249	0.0001672
0.0001258	0.0001543

D.C.4 Buffer size = 1024, resolution = 1280 x 720

Audio porcessing	Rendering
0.0004502	0.0005649
0.0004009	6.50E-005
0.0003799	4.70E-005
0.000392	5.11E-005
0.0003935	5.61E-005
0.0004086	6.64E-005
0.0003703	5.78E-005
0.0003947	0.0004638
0.0004044	6.50E-005
0.0004357	5.79E-005
0.000395	5.58E-005
0.0003949	6.15E-005
0.0003971	5.62E-005
0.0003951	5.71E-005
0.0003813	5.49E-005
0.0003885	5.70E-005
0.0003874	5.56E-005
0.0004	6.23E-005
0.0003927	5.90E-005
0.0004937	5.54E-005
0.0003992	5.93E-005
0.0004491	5.31E-005
0.0003927	5.59E-005
0.0004111	0.0003506
0.0003703	5.84E-005
0.0003926	4.64E-005
0.0003885	5.50E-005
0.0004125	5.56E-005
0.0003873	5.64E-005
0.0003898	5.49E-005
0.000394	6.37E-005
0.0003955	5.89E-005
0.0003964	5.59E-005
0.0003938	5.72E-005
0.0003892	5.60E-005
0.0003882	5.47E-005
0.000426	5.26E-005
0.0003912	5.51E-005
0.0004658	8.71E-005
0.0003999	6.52E-005
0.0004847	6.32E-005
0.0003898	5.67E-005
0.0004014	5.55E-005
0.0003957	5.28E-005
0.0003959	5.43E-005
0.0012149	0.0001433
0.0011929	0.0001468
0.0011864	0.0001468
0.0011809	0.0001431
0.0011766	0.0001427
0.0011821	0.0001463
0.0011838	0.0001562

D.C.5 Buffer size = 2048, resolution = 2560 x 1440

Audio porcessing	Rendering
0.0013532	0.0005732
0.0013375	5.57E-005
0.0013089	4.09E-005
0.0013454	4.88E-005
0.0012831	5.51E-005
0.0012672	6.20E-005
0.0013074	5.96E-005
0.0014311	6.37E-005
0.0012978	5.81E-005
0.0014154	5.88E-005
0.001353	5.40E-005
0.0013358	6.48E-005
0.0014377	6.77E-005
0.0013145	5.19E-005
0.0013618	5.69E-005
0.0013519	5.61E-005
0.0013254	5.55E-005
0.0013713	5.78E-005
0.0012741	5.23E-005
0.0013376	6.71E-005
0.0013635	5.52E-005
0.0014976	5.45E-005
0.0014268	0.0001025
0.0013648	8.00E-005
0.0013644	6.08E-005
0.0013663	5.03E-005
0.0013423	5.54E-005
0.0013522	5.11E-005
0.0012683	6.02E-005
0.0013036	5.67E-005
0.0013361	5.28E-005
0.0013332	5.78E-005
0.0013488	5.07E-005
0.0013462	5.37E-005
0.0013082	5.47E-005
0.0013539	5.31E-005
0.0013996	6.42E-005
0.00137	8.34E-005
0.001357	5.63E-005
0.001365	6.04E-005
0.0013011	7.14E-005
0.0013693	6.24E-005
0.0013489	5.59E-005
0.001348	5.07E-005
0.0013037	5.55E-005
0.0013822	5.68E-005
0.0013592	5.93E-005
0.0012648	5.37E-005
0.0013625	5.67E-005
0.00134	5.43E-005
0.0014285	5.65E-005
0.0013229	6.33E-005

D.C.6 Rendering isolated, all resolutions

800x600
0.0006609
4.90E-05
0.0003661
0.0002622
0.0002957
0.0002519
6.18E-05
0.0019056
0.0002817
0.0005487
0.0005863
0.0006373
0.0001082
0.0005491
0.0005381
0.0007128
0.000552
0.0005339
0.0003758
0.0005524
0.0007164
0.0005413
0.0005781
0.0006234
0.0042491
0.0006682
0.0011844
0.0015809
0.0013985
0.0012747
0.0011333
0.0011723
0.0013477
0.001146
0.0012657
0.0011496
0.0013504
0.0014065
0.0011701
0.0011228
0.0011208
0.0011434