Creation of a Real-Time Granular Synthesis Instrument for Live Performance

T. T. Opie

8th October 2003

Creation of a Real-Time Granular Synthesis Instrument for Live Performance

Timothy Tristram Opie

BA (Hons)

Creative Industries: Music

Queensland University of Technology

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Arts

7th of March 2003

Keywords:

Granular synthesis, music, instruments, real-time, performance.

Abstract

This thesis explores how granular synthesis can be used in live performances.

The early explorations of granular synthesis are first investigated, leading up to modern trends of electronic performance involving granular synthesis. Using this background it sets about to create a granular synthesis instrument that can be used for live performances in a range of different settings, from a computer quartet, to a flute duet. The instrument, an electronic fish called the poseidon, is documented from the creation and preparation stages right through to performance.

Contents

Abstract 3

Acknowledgements 13

Preface 15

1 Introduction 19

2 Background 21

2.1 The *Corpuscular* Theory Of Sound 21

2.2 The *Particle* Theory of Sound 26

2.3 The Gabor Grain 28

2.3.1 Signal analysis 28

2.3.2 Hearing Analysis 33

2.3.3 Deception 37

2.4 Granular Composition 43

2.4.1 Tape Splicing 43

2.4.2 Music V on a Burroughs B6700 44

2.4.3 Music 11 on a DEC PDP-11/50 46

2.4.4 Many programs on many computers 47

2.4.5 Real-time 49

2.4.6 The next generation 53

2.4.7 Particle pops 57

3 Live Performance 61

3.1 Australian and New Zealand Artists in performance 61

3.2 The Author's performance experiences with granular synthesis 66

4 Development **71**

4.1 Ready, Aim...... 72

4.1.1 Acoustic Granular Synthesis Machine 74

4.1.2 Electronics 74

4.1.3 Electronic and Computer Fusion 75

4.2 Building Hardware 78

4.2.1 Fritz Lang 78

4.2.2 Poseidon 79

4.2.3 Control Styles 85

4.3 Software Hacking 86

4.3.1 The *AudioMulch* and *Max/MSP* Factor 86

4.3.2 *Csound* 87

4.3.3 *jMusic* 88

4.3.4 Granular Instrument Test Prototype Code 89

4.4 Software Algorithms 89

4.4.1 Grain Length 91

4.4.2 Grain Envelope 91

4.4.3 Grain Density 93

4.4.4 Grain distribution 94

4.5 Programming with *jMusic* 94

4.5.1 Spurt 94

4.5.2 Granulator - The Audio Object 97

4.5.3 GranularInstRT 104

4.5.4 RTGrainLine 110

4.5.5 RTGrainMaker 114

4.6 *jMusic* Summary 117

5 Performance Outcomes **119**

5.1 What About The Children? 120

5.2 Adult Reaction 120

5.3 Russian Film Improvisation 121

5.4 *Elucidation*, a Poseidon and Flute Duet 123

5.5 Personal Reflection on the Results of the Poseidon 125

6 Conclusion **127**

Appendices **129**

A Glossary **131**

B Csound code for Elevated Music prototype **133**

C jMusic code for Spurt **139**

D jMusic code for Granulator - The Audio Object **141**

E jMusic code for the GranularInstRT **149**

F jMusic code for the RTGrainLine **155**

G jMusic code for the RTGrainMaker **161**

Bibliography **165**

List of Figures

2.1 Representation of signal by a matrix of complex amplitudes [Gabor, 1946, p. 435]. 30

2.2 Representation of signal by logons [Gabor, 1946, p. 435]. 32

2.3 Threshold areas of the ear at 500 Hz [Gabor, 1947, p. 593]. 34

2.4 16mm sound-film projector converted into an experimental frequency convertor [Gabor, 1946, p. 452]. 36

2.5 Frequency convertor with sound film [Roads, 1991, p. 174]. 38

2.6 The contribution of individual slits and the resulting light output [Gabor, 1946, p. 447]. 40

2.7 Frequency convertor with magnetic tape [Gabor, 1946, p. 453]. 42

2.8 Curtis Roads sitting at the Creatovox [Roads, 2001, p. 196]. 48

3.1 Donna Hewitt in performance. Photo Mr Snow 62

3.2 Close up of the poseidon control interface. 68

4.1 Circuit board of Dumb Controller designed by Angelo Fraietta. 76

4.2 The electronic parts that went inside the poseidon. 80

4.3 The poseidon drying off. 82

4.4 The completed poseidon connected to a computer. 84

4.5 Half a sine wave. 92

5.1 Timothy Opie holding the poseidon. 122

5.2 An example of the method I used to notate for the poseidon. 124

Statement of Authorship

The work contained in this thesis has not been previously submitted for a degree or diploma at any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Document: Creation of a Real-Time Granular Synthesis Instrument for Live Performance

Name: Timothy T. Opie

Signature: ....

Date: / /

Acknowledgments

First and foremost, Andrew Brown for supervising. Steve Dillon, for providing excellent assistance in concepts and methodology. Craig Fisher, Steve Langton, Aaron Veryard, and Steve Adam, for providing good background material in helping me design and test my instrument. Stuart Favilla, for helping me organise the initial structure of the thesis. Andy Arthurs, Linsey Pollak, and Zane Trow, for organising the REV festival where my fish instrument was showcased. Andrew Sorensen, Rene Wooller, Adam Kirby, and Paul Cohen, for helping me with Java, and providing other insights throughout the year. Rafe Sholer and Joel Joslin, for performing with the fish, whilst I needed a break, especially when I was suffering from travel sickness and needed to take long breaks. Gabriel Maldonado, for his Csound advice. Angelo Fraietta, for designing the Dumb CV to MIDI Controller and giving advice. Nicole Stock, for improvising on my music and performing a flute/poseidon duet with me. Julian Knowles, Donna Hewitt, Julius Ambrosine, Ross Healy, and Warren Burt, for their correspondence with me on their own performances using granular synthesis. Paul Doornbusch, for proofreading and encouragement. Matthew Parker and Ben Deutsch, for being so active on the granular synthesis discussion group and finding useful URLs and programs. The many creators and contributors to LyX and FreeBSD who volunteered uncounted hours writing and documenting these programs and also answering my questions in mailing lists that made my task so much easier.

Preface

I have been involved with music ever since I can remember. From a very early age I was singing in choirs and playing the piano. As a child I would write bizarre pieces of music and get my dad to play them on the piano because I had purposefully made them too complex to play myself. As I progressed with piano lessons my piano teacher was more impressed with my singing ability and she would arrange for me to sing to her piano accompaniment at RSL gatherings and so forth.

During High school I kept up the piano and singing, plus I had already developed a very keen interest in computers. I started computer programming on a Commodore 64. I also had some experience programming with the Apple 2E, and the Spectrum. I mainly used these computers for playing games. My father brought home a Macintosh Classic and I started working with computer images, and also writing small musical scores with Deluxe Music. When the Amiga 500 came out I saved up for a year and bought one. It was on this computer that I really started learning how to program. I first started with Amiga BASIC, which was similar to Commodore BASIC, and equally as unrewarding. There were a number of exceptional music programs for the Amiga, such as a much newer version of Deluxe Music, and there was SoundTracker. There were a number of other clones based this called NoiseTracker, ProTracker and MED. These programs all created MODs. MODs are a kind of MIDI style, but they use sound samples instead of GM. Using ProTracker as my favourite MOD creator I learnt all the hex codes to create many different effects, and started transcribing popular tunes as well as writing my own songs. A program entitled AMOS was released which was a hybrid program which integrated C into a BASIC structure, but which also had an Assembly base which could be switched to very easily if the speed was needed. It also had special features for integrating graphics, sound and music, in fact MOD files could be directly imported. I used this program

for a few years mainly developing computer games with my friend Andrew Heap. Together we created a couple of top quality games. Well, top quality for a couple of high school students, and good enough that an English programming company offered to buy one of our games, *Hairpin*, they had seen floating around on Public Domain. The software market being perhaps more volatile then than even now meant that the company went bankrupt before any deal was made, so we released the game under the Deja Vu royalty scheme. I am still waiting for the first royalty cheque. The primary thing AMOS taught me was how to produce nasty high speed code.

When I was accepted to La Trobe University I chose to study computer science and my music was put on hold. The main thing I wanted to learn was how to program computer games, with super-fast graphics, better musical control and more scope than AMOS allowed. At university I started learning Pascal and C, amongst other languages. I had to try and unlearn all the bad programming habits I had spent the last few years teaching myself. I confess though that I still prefer fast code over aesthetically pleasing code. Now that computers are getting extremely fast the argument for high speed code over aesthetic code is levelling out. So I am learning the art of writing efficient code within a structure that is maintainable by anyone. After a year of studying computer science, computer systems and electronics I still had not even touched music or sound programming on a computer, it was all sorting algorithms and command parsing. In the second year I decided to enroll in music as an extra to give me something to look forward to each week whilst having to put up with computer science. Until that time I had no idea that the music department at La Trobe University specialized in music technology, so at the end of my second year I changed to an Arts degree and pursued a music major.

My initial journey into granular synthesis began in 1997 when I wrote a program in the C programming language called *Interleaver*. This was a simple program which took two sound files and wove them into one sound file. The method for doing this was to break the samples up into byte size chunks and then interleave the bytes in an alternating fashion. If one file was smaller than the other then it would compensate by inserting silence. This process would create a sound file that was double its original length, so the program would then double the sample rate so that when the sound file played back it would still be at the original pitch, but you would hear

the two samples simultaneously despite the fact that the samples were really just alternating from one to the other at a very rapid rate. I developed this idea further, making different changes for slightly different effects. In 1999 I was invited to do an honours thesis and instinctively chose to keep working with my interleaver project, but after a few weeks I realised that a lot of what I wanted to accomplish was already in use in the synthesis method known as granular synthesis. I then started research into granular synthesis. It was a slightly different way to view what I was working with, but it was a lot more elegant. After my honours thesis, *Granular Synthesis: Sound In A Nutshell*, I began a Masters Degree, this time though the initial focus was to create a musical instrument. Again I instinctively chose to keep on the path I had been following for the previous couple of years and started designing a musical instrument that would incorporate the principles of granular synthesis.

This thesis is the detailed story of this most recent journey.

Chapter 1

Introduction

This thesis primarily involves a study of real-time granular synthesis as used in live performance. In addressing this issue I will be looking at the origins of granular synthesis, and how it has built up in popularity to become a widely used tool of the electronic musician. I will also describe a case study whereby I created and performed with a granular synthesis instrument. My instrument construction and performance will be compared to some other recent granular synthesis performances conducted in Australia.

This thesis will by no means attempt to catalogue every granular synthesis piece ever performed in Australia. Such a project would take more than the year I had available to complete a masters degree. It will only provide a sample of different performances. The sample chosen comes from ongoing communication and consultation I had with fellow granular synthesis musicians. As I came in contact with electronic musicians I asked them about their work with granular synthesis and their performances. I also asked them of other colleagues they knew who had been working with granular synthesis, so that I could branch out and ask them similar questions. I consider this a grapevine sample of musicians.

This thesis will also not attempt to test and catalogue every piece of software written for granular synthesis usage. There has been a vast increase in the number of computer programs available in recent years, and to attempt such a task would take a number of years. It would also become obsolete very quickly as new programs emerge, old programs become updated, and some become obsolete and discarded. Such a project is more suited to an Internet bulletin board medium which can be

constantly updated and changed as quickly as the software.

Whilst this thesis will examine some issues concerning electronic performances, it will not try to give any solutions to improving how electronic music can be performed. It will only describe the way in which I dealt with the issue in this circumstance.

Chapter 2

Background

Granular synthesis is a recent development in sound synthesis, but underneath this development lies a long and detailed history of the very nature of sound, a nature which is emulated with and through granular synthesis. It is an exploitation of the limits of our perception of sound. It is only since the advent of computer and sound recording technology that it has been practical to exploit this idea and use it in a musical context. Granular synthesis is based on a certain notion of what sound is, in particular, the human perception or cognition of sound. Audio analysis has been the basis of research for many centuries and in understanding granular synthesis it helps to understand the origin of this research.

2.1 The *Corpuscular* Theory Of Sound

Music research and the studies of the physical and mathematical properties of sound can be traced back many centuries, back to the Greek Empire in 600BC [Moore, 1990]. Many of the concepts of sound are things we now take for granted as common knowledge, concepts such as how the length of a string or pipe will dictate the frequencies of sound it can deliver. Some other early concepts are perhaps not so common knowledge, such as *Musica Speculativa*, the mathematical relationships between musical intervals, but their application is so well known through their prolific use that to do or hear otherwise would seem to contradict good musical judgment. Common musical tools such as resolution through the tritone and the fifth to the tonic were grounded into the musical culture of western society by pioneers of classi-

cal music to the point that music did not complete until the tonic had been reached. Scientific principles concerning sound, such as echo and acoustics were studied and explained with mathematics and physics. Again the science behind these concepts may not be common knowledge but their application in daily life is an intuitive part of living. Echo and acoustics are used intuitively to keep balance and to recognise surroundings from an aural perspective. Many other common-knowledge discoveries were made during this early period in time. Probably the most significant was the atomic theory. Some Greek philosophers believed that all matter on the earth could be reduced to clusters of atoms. They saw atoms as the building blocks of the universe, the building blocks of all matter and energy, including sound. Their view of the atom was significantly different to our current understanding, but they were on the right path. All fundamental science in our society today is based on the philosophy and later the proof of the atomic structure of the universe. Unfortunately after the Roman conquest of the Greek empire and subsequent collapse of the Roman Empire, the majority of the Greek sciences and scholarly works were kept hidden for protection and preservation. It was not until the Renaissance, about a millennium later, that these mysteries were again able to see the light of day, albeit with much scrutiny from the Roman Catholic Church.

Leonardo Da Vinci was an exceptional thinker of the early Renaissance. He learnt all of the Greek sciences and mathematics and used these as a strong foundation from which to make many new discoveries. Not only was Da Vinci a mathematician, scientist, and highly renowned artist, he was also a musician and had a great love for music. Unfortunately not much of his musical life is known, but his contemporaries such as Luca Pacioli, Paolo Giovio, Benvenuto Cellini, and others, extolled him as a supreme musician [Winternitz, 1982, p. xxii]. He has never produced any known compositions, and this has been accredited to the fact that he was an improviser, and therefore never actually committed any work to paper. It is known that he was a performer, and a music teacher. He also studied acoustics, created and improved many musical instruments, and also wrote many original thoughts on the philosophy of music [Winternitz, 1982, p. xxi]. In his research of acoustics he conducted many experiments. Some of these were directed primarily at the propagation of sound, and the way the human ear perceives it. The most notable of his experiments in

this particular field was that which speculated how sound actually consists of waves rather than particles. He demonstrated this theory of waves by dropping two stones in a still pond. Both stones produced circular waves which spread out evenly around the stone and continued to spread out getting weaker the further out they dispersed. He stated that the action of the stones in water would be very similar to the action of sound in the air. One very particular note he made about this experiment was that when the waves from opposing stones crossed paths they passed through each other without stopping. He compared this action to common sound occurrences where a listener hears sound from two sources at the same time without distortion. He also realised the direct link between sound and light and also noted that with the two stones a viewer can also see from two or more light sources at the same time without the light from one source blocking out or crashing against the light of the other source. He attributed this wave property of sound and light to the fact that they must be waves, if they were particles then they would surely collide and lose their circular wave formations. He also noted that when the waves were produced it was not the same water particles moving in an outwards motion, they just passed on the energy on and thus the wave spread out. These findings were enough to convince him, and indeed many following scientists that sound must be composed entirely of waves [Winternitz, 1982, chapter 7].

The famous scientist and astronomer Galileo Gallilei followed up on the sound research of Da Vinci in the early 1600s. In 1638 Galileo conducted an experiment in which he took a glass of water and rubbed the rim of it to produce a tone. He observed that the tone produced caused tiny rip-lets to occur in the water, all emanating from where he thought the sound was being created. Galileo used this experiment to further speculate and to validate Leonardo Da Vinci's theory that sound travelled in waves. This wave theory of sound was significant in sound research and led the way for many advances in sound theory. Mersenne, Boyle and Newton amongst others following the lead of Galileo studied this theory intensely during the 1600s [Jeans, 1947]. Their findings became known as the wave theory. The wave theory can be seen visually by placing a struck tuning fork, into water and examining the waves that are formed. The waves travel through the air in the same fashion.

Contemporary to Galileo was a Dutch scientists called Isaac Beeckman. In some

writings on granular synthesis he is referred to as Isaac Beekman, but he spelt his name as Isaac Beeckman. Beeckman lived from 1588 until 1637. He kept a journal of his scientific experiments for 30 years from which his brother published a small part after his death. It was not until 1905 that the archivist De Waard discovered and published his entire journal in a four volume set. Beeckman was entirely self taught in science and medicine, which resulted in him discovering many things that were already known. This process may have been more laborious but it meant that he had a fresh outlook on the the principles and discoveries he made [Cohen, 1984]. For example Beeckman knew none of the research on sound that Da Vinci and Galileo carried out. Even the writings of the Greek philosophers such as Aristotle were unknown to him until much later in life when he started reading the works of others. Beeckman was very much like Leonardo Da Vinci in many ways. He too liked to work with a broad scope of different media, all the sciences, music, medicine, logic, mechanics, and so on, and he loved to experiment. Like Da Vinci he was equally poor, perhaps worse, at actually finishing a project. He would get excited and move onto something else. Beeckman was however not extolled as a supreme musician, in fact he admits himself he was the worst pupil his singing master ever had and he had trouble discerning dissonance [Cohen, 1984, p. 119].

Some of Beeckman's findings are common occurrences that are now taken for granted, but which at that time needed to be stated formally. For instance, through many experiments Beeckman discovered that sound is created by vibrating objects. In extension to this discovery he started researching how sound actually travels through the air after it has been created. In his journal he stated that the vibrations caused corpuscles to be cut into the air, which would then disperse and spread out causing the sound to spread. Interestingly Beeckman conducted an experiment that was identical to one performed by Galileo in 1638, one year after Beeckman's death. Beeckman showed that rubbing the rim of a glass filled with water caused sound globules to form in the water and the air in the area where the glass was being rubbed. Unlike Galileo he had a totally different explanation for this phenomenon. Beeckman observed that the rubbing on the glass had formed sound corpuscles, and using the water as a medium he was able to see the corpuscles forming in the water. Beeckman rejected any possibility of a wave theory. Whilst rejecting this idea

Beeckman did believe in sound being transmitted through a process of expanding and contracting of particle density through the air. This can be explained as part of the wave theory, but not in the context to which Beeckman places it. Beeckman was a friend of René Descartes who was a firm believer in the atomic theory as described by the Greeks, and this would have been a major influence in his research into the particle nature of sound [Cohen, 1984].

Beeckman firmly believed that the corpuscles[1] of sound were pure particles of sound, and they would group together in globules[2] to assist in dispersion. Another important observation made by Beeckman was that the globules were not necessarily continuous. During certain phases the density would vary, and fewer globules would be dispersed. Beeckman also concluded that the size of the globule determined the pitch. The larger the globule, the lower the pitch. This can be seen by plucking a low string on a cello, the string makes larger movements meaning that the corpuscles that are cut within the air, and hence globules, will be larger. He also extended his theory of string instruments to incorporate wind instruments by showing that a larger pipe requires more pressure to make a tone and hence the globules are compressed, so when they escape the pipe they are larger in portion. The explanation of different sized globules was not originally intended as an explanation of any relationships between corpuscle size and globules, it was Beeckman's explanation as to why high pitched tones sounded so terrible [Cohen, 1984]. Beeckman had many more sound globule theories in his journal, but the main points regarding granular synthesis have been made, primarily the notion that sound is made up of particles and that the particles can group together in globules.

Unfortunately, due to Beeckman's remoteness and the contradictory sound research of Galileo, Da Vinci and many others, Beeckman's ideas were never investigated further. Even though Beeckman made no impact in the wave versus particle debate he did make some very interesting observations which can give us solid insights into the nature of sound and how it can be related to granular synthesis. The most important observation is how the globules are composed and how they function. Essentially the globules could represent a grain of sonic data, the corpuscles being the encapsulated data, but windowed within the globule. The globules themselves

[1]See glossary in Appendix A on page 131.
[2]See glossary in Appendix A on page 131.

can be generated in a continuous or non-continuous fashion depending on the texture that is being created.

2.2 The *Particle* Theory of Sound

Due to the findings of Galileo and his followers the wave theory of sound gained popularity and acceptance. Whilst the wave theory was becoming more popular the atomic theory of matter was gaining ground. The popular opinion was that quite simply that if something is too small to see then it cannot possibly exist. The invention of the microscope by Antonie van Leeuwenhoek led to a great revision in scientific thinking, and to the larger adoption of the atomic theory by the general public. Leeuwenhoek developed a high powered lens which could magnify an object up to 200 times without distorting the image. Direct observation through the microscope showed that "a drop of pond water was a teeming mass of life suggestive of a crowded city." [Wiener, 1964, p. 540]

This invention opened the minds of the world to a newer, smaller world. It had people thinking beyond what they can see with the naked eye, and also fuelled many new philosophical questions of life. Questions such as how small can an object be? Are these smaller worlds similar to ours? Are we just smaller creatures through the magnifying glass of an even larger species. These ideas led the author Jon Swift to write the novel "Gulliver's Travels" which explored these ideas. Swift also wrote a jingle which captured the essence of what the microscope could mean:

> So, naturalists observe, a flea
> Hath smaller fleas that on him prey;
> And these have smaller still to bite 'em;
> And so proceed *ad infinitum*. [Wiener, 1964, p. 540]

The short poem is suggestive of the notion that if a more powerful microscope could be created, then there were infinite layers of worlds to be discovered. This idea was developed to great length by Leibniz and was named the continuistic theory. This train of thought goes against the atomistic theory in which there is a nondivisible element, the atom in this case, the smallest object which can no longer be divided into smaller parts. It was not long before scientists realised that atoms were actually

made up of more discrete elements. Despite the atom being reduced to smaller parts the atomic theory still held the largest support. Despite the atomic theory of matter remaining the dominant theory, the wave theory was still dominating research of light and sound. In fact the scientist Clark Maxwell used the continuistic theory to advance the wave theory by theorising that "light and electricity are transmitted as oscillations of a continuous medium known as luminiferous ether." [Wiener, 1964, p. 542]

He was never able to substantiate this theory, and in 1900 Max Planck proved him wrong. Planck also argued that light is not continuous. It has a granular texture which can be broken down into small atomic quantities, which were labeled quanta. This theory of quanta was the beginning of quantum physics. Planck discovered an essential number 6.6260755x10^{-34} which is referred to as Planck's constant (h). Planck's constant determines the relationship of frequency and energy to radiation according to the equation:

> $frequency = energy \div h$ [Berry, 1988]
>
> This equation was the basis for further analysis regarding quantum physics and also sound analysis.

Albert Einstein made his first major contribution to quantum physics in 1905 when he found that Planck's constant could be used to determine the values of many effects involved with quantum physics, such as light emission. This was proof that light, and with it sound, must be made up of discrete particles [Wiener, 1964]. In 1907 Einstein constructed the concept of phonons in the atomical structure of sound, based on the concept of photonic light particles [Roads, 2001, p. 34].

With the beginning of quantum physics came a new way of analysing sound. Until this point the analysis of sound was undertaken using the wave theory. The wave theory is two dimensional, utilising frequency and intensity. This was a perfect medium for the Fourier Transform (FT) because the FT presupposes that the wave has an infinite time span. The FT is used to de-construct complex wave forms into a combination of simple sine waves [Jeans, 1947, pp. 78 - 82]. It became apparent that this kind of analysis, despite being extremely useful, had many short comings. Using quantum physics and the particle theory to analyse sound, a third dimension is

added to the equation: Time.

2.3 The Gabor Grain

Dennis Gabor was a physicist who won the Nobel Prize in 1968 for the invention of holography. Gabor was a person who was concerned about the limitations of the Fourier Transform. He devised a windowing system which he used to analyse and reproduce sound [Risset, 1991, p. 32]. In 1946 and 1947 he published a number of papers dealing with issues of human communication and transmission. His research started with an investigation of the Fourier Transform and other forms of sound analysis in which he noted the associated problems. He introduced a quantitive approach to analysing sound signals. He took this research to more practical measures and began research into how the ear hears sounds, and how the sounds can be analysed. He defined the quantum of sound as being the smallest indivisible amount of sound information needed for auditory discernment [De Poli et al., 1997, pp. 156 - 158]. Using the information about sound interpretation Gabor proposed a quantitive approach to sound analysis that can also be used to reproduce sounds through a narrower wave band, based on his theory of elementary signals.

2.3.1 Signal analysis

Gabor's research started as a means for data reduction of communication transmissions. In the 1930s communication engineers realised that the wave band size required to transmit data is directly related to the amount of information per time unit. The need to reduce bandwidth was becoming an issue since 1924, as Nyquist and Küpfmüller both discovered that the number of telegraph signals that can be transmitted over a single line is directly proportional to the bandwidth.

The theory of communication was built on the basis of Fourier's sound analysis scheme. The problem with this is the Fourier Transform represents any signal as a sum of sinusoidal curves of constant frequency. Although this is mathematically correct, it fits badly with our normal perception of sound. Gabor made the following statement regarding the Fourier Transform: "Though mathematically this theorem is beyond reproach, even experts could not at times conceal an uneasy feeling when

it came to the physical interpretation of results obtained by the Fourier method." [Gabor, 1946, p. 431]

Carson, a contemporary of Gabor who also researched sound analysis made this statement regarding the Fourier Transform: "The foregoing solutions, though unquestionably mathematically correct are somewhat difficult to reconcile with our physical intuitions, and our physical concepts of such *variable-frequency* mechanisms as, for example, the siren." [Gabor, 1946, p. 431]

In the strict mathematical sense the term frequency refers to an infinite unchanging wave. Therefore a variable-frequency is a contradiction in terms. This is where the Fourier Transform becomes less useful as it assumes a constant and continuous frequency [Risset, 1991, p. 32].

Carson spoke about our physical intuitions. He noted that all speech and music has a *time pattern* as well as a *frequency pattern*. In music it is possible, even common, to change one of these patterns without affecting the other. For example, a song can retain the same time signature and tempo, but be played at a different base *frequency*, in a different key. The same song can also retain its key but be played at a different tempo. Both the key and the time could vary throughout the song, and it could still be recognised as the same piece of music. Both parameters complement each other and work together to create what is intuitively perceived as speech and music, they are not separable entities [Gabor, 1946, p. 431].

Using quantum mechanics, Gabor redefined the Fourier Transform and created an equation to express a signal in terms of both frequency *and* time. Based on Planck's equation $frequency = energy \div h$, it was just necessary to determine how time related to energy and replace the associated value. With this substitution he was no longer dealing with a frequency of an infinite duration. The frequency was now defined as a complex analogue of the sine function [Eargle, 1995, pp. 1 - 11]. Now that the frequency had a direct time relationship it meant he was then able to mathematically examine specific sections more closely by windowing[3] them using calculus.

Gabor then moved away from quantum mechanics and made some extra alter-

[3]To window a frequency or indeed any mathematical equation means to define an area between two given points. This is usually done across the x axis. This method allows the equation to be resolved to a specific answer, but the answer is only valid between the two windowed points.

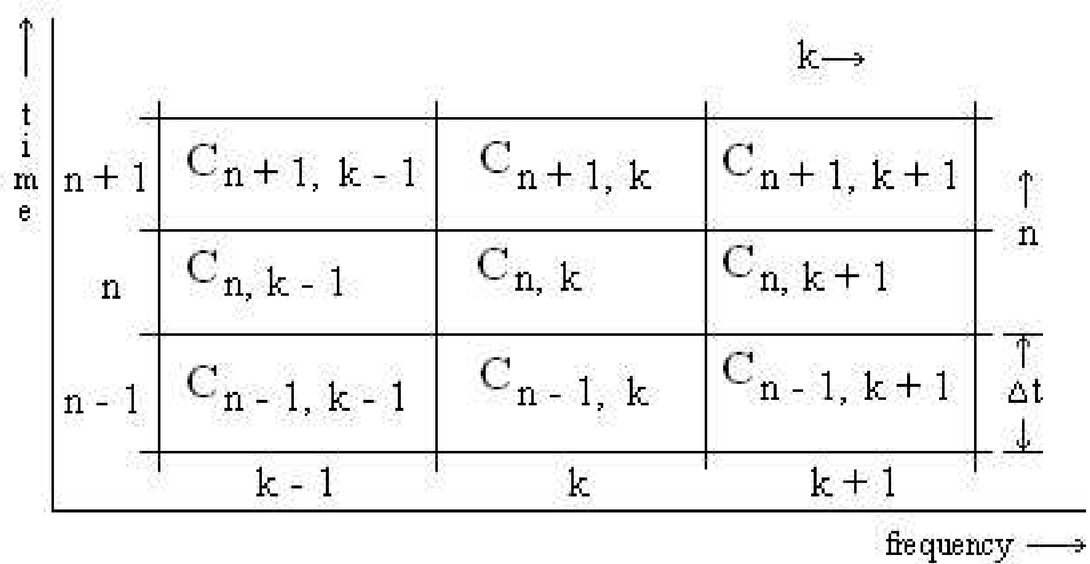

Figure 2.1: Representation of signal by a matrix of complex amplitudes [Gabor, 1946, p. 435].

ations to the equation by devising a grid system. Rather than just using a rectangular view of a signal within a bound time, Gabor chose to envelope the signal with a probability curve. What he was trying to do was capture each elementary signal [De Poli et al., 1997, pp. 156 - 158]. His final equation states:

$s(t) = \exp^{-\alpha^2(t-t_0)^2} . \exp^{i2\pi f_0 t}$ where α is a real constant. [Gabor, 1947, p. 591]

The first section of the equation, $\exp^{-\alpha^2(t-t_0)^2}$, defines the probability curve, whilst the second section, $\exp^{i2\pi f_0 t}$, by which it is multiplied, defines the complex analogue of the sine function within the grain, with frequency f_0.

The components Δt and Δf are determined by the equations:

$\Delta t = \frac{\sqrt{\pi}}{\alpha}$ and $\Delta f = \frac{\alpha}{\sqrt{\pi}}$ [Gabor, 1947, p. 592]

Gabor recognized that the time and frequency domains could be formed into a rectangular cell using the equation:

$\frac{\Delta t}{\Delta f} = \frac{\pi}{\alpha^2}$ [Gabor, 1947, p. 592]

By substituting different constants of α Gabor could build an entire matrix of elementary signals. A basic example of such can be seen in figure 2.1 on the facing page.

Gabor was aware of Heisenberg's uncertainty principle concerning particles at the sub atomic level, in which Heisenberg demonstrated that the more precisely you determine the position of an electron, the less you know about its velocity and vice versa. This is because they are measuring change against each other making them reciprocal to one another [Truax, 1990]. Gabor realised that time and frequency were reciprocal in nature and that depending on which of the two variables he wanted to focus on the other would become less precisely defined [Arfib, 1991, p. 87]. The time blurring is manifest as time domain aliasing, which is experienced in sound as a reverberation or echo. Frequency blurring is manifest as frequency domain aliasing, which is experienced in sound as a noisy frequency. In order to get an accurate time he would need to analyse a larger frequency domain and conversely if he wanted an accurate frequency he would need to analyse a larger time domain. Based on prior research performed by W. Pauli, Gabor determined that the domain size should be determined by this equation:

$\Delta t \Delta f \simeq 1$ [Gabor, 1946, p. 432]

although he later modified this slightly to:

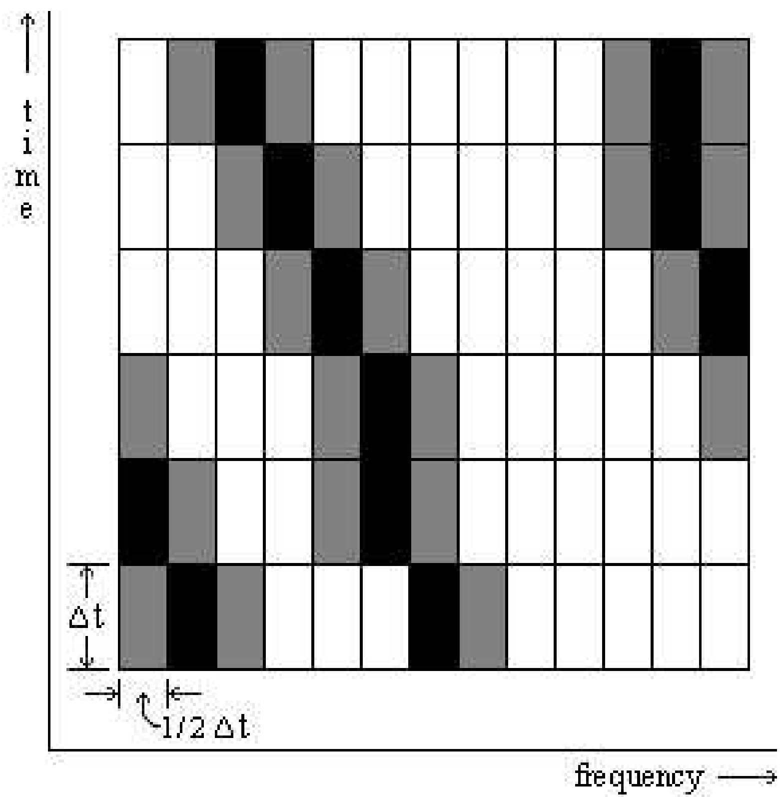

Figure 2.2: Representation of signal by logons [Gabor, 1946, p. 435].

$\Delta t \Delta f \geq 1$ [Gabor, 1947, p. 591]

This means quite simply that if sample Δf needs to be at a bandwidth of 100Hz then Δt must be at least 10ms because $100hz \times 0.01sec = 1$.

So the elementary signal may be represented as a rectangle with a width and height of Δt and Δf, and each rectangle may be placed in a grid to cover the entire time-frequency spectrum. As seen in figure 2.2 on the preceding page, the amplitude of the associated elementary signal is entered into the appropriate rectangle on the grid. Gabor proposed that each rectangle with its associated datum be called a *logon* [Evangelista, 1991, p. 119]. Each logon has a constant frequency, as would be expected in an elementary signal, but in a matrix of logons, all frequencies are accounted for. Figure 2.2 contains an example of a logon matrix.

The Dutch scientist and lecturer Martin Bastiaans has been studying and expanding this analysis for many years. He originally wrote a paper in 1980 in which he verifies and expands on Gabor's elementary signal equation. [Bastiaans, 1980]. He later expanded this significantly to change the rectangular grid to a quincunx (or hexagonal) lattice using the Zak Transform. Bastiaans' works demonstrate improved functions for isolating elementary signals [Bastiaans, 1985, Bastiaans & Van Leest, 1998].

2.3.2 Hearing Analysis

Having defined a logon, Gabor set himself the task of discovering how many logons are required to reproduce speech or music which the ear cannot discern from the original. Bürck, Kotowski, and Lichte had recently performed a number of very accurate experiments to determine the minimum requirements for discernment of sound. They made the following observation: "At 500 and 1000 c/s the minimum duration after which the pitch could be correctly ascertained was about 10 millisec for the best observers." [Gabor, 1946, p. 442]

They then continued this test by adding in a second datum of information. As well as pitch they doubled the intensity of the tone. They made the following observation: "For a frequency of 500 c/s. After 10 millisec the signal was just recognizable as a tone. But unless it lasted for at least 21 millisec, the ear was not ready to register the second datum." [Gabor, 1946, p. 442]

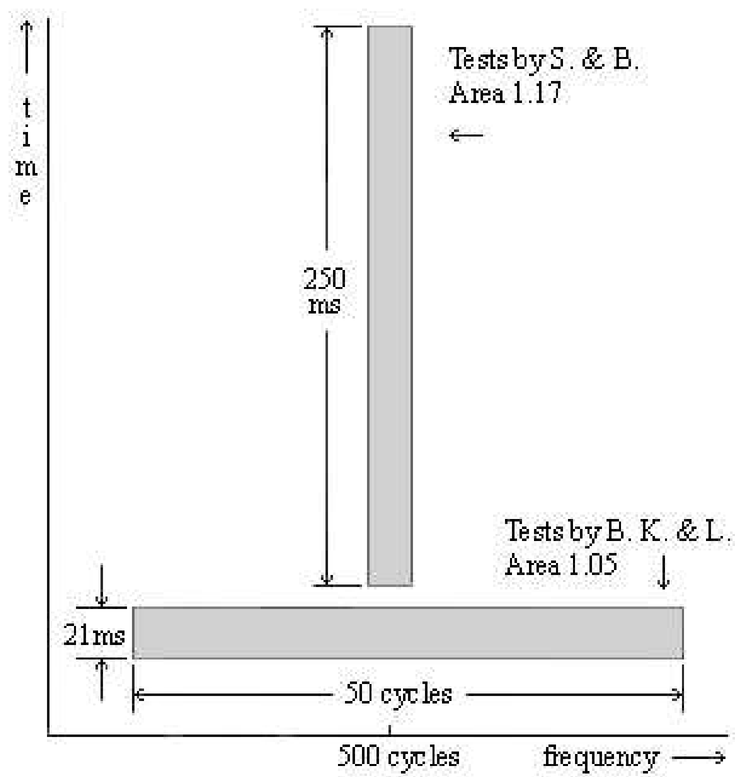

Figure 2.3: Threshold areas of the ear at 500 Hz [Gabor, 1947, p. 593].

At 1000 Hz the tone needed to last for 30ms before it could register the second datum.

Gabor compared these findings to his own research. He compared a single datum to a logon, as each logon contained just an elementary signal with a single datum. Gabor viewed these findings in terms of area, and stated the ratio of the duration for discernment of a second datum was 2.1:1 at 500 Hz, and 3:1 at 1000 Hz [Gabor, 1946, p. 442].

Shower and Biddulph conducted further observations on the discernment of sound. They conducted tests to evaluate how quickly the human ear can discern a change in frequency. They did this by modulating the frequency within a small bandwidth. They measured how long it took for the listener to recognize the pure tone had become a trill. Discerning the slight frequency change under these precise circumstances took 250ms.

Gabor measured the areas of the grids from the experiments and found the areas to be about 1, which was what he originally evaluated the area of each logon to be. The areas have been represented graphically in figure 2.3 on the facing page.

Despite all of these experiments showing that Gabor's reduction method could produce a single acoustical quantum he was still quick to point out that he could claim no such accuracy. The point of accuracy will always depend on the exact definition and interpretation given. He just states that it is near the point of unity, that is it is near the *acoustical quantum limit*. He verifies this statement by analysing the validity of the actual outcomes of the experiments. The experiments merely required a yes/no answer to a simple question. A measuring instrument on the other hand will register the same information as a complex numerical datum which can be deciphered in a number of ways. The human is only trying to discriminate, whereas a measuring instrument is registering a range of values. The main purpose of this investigation into hearing analysis was to find ways to deceive the human ear. The simple conclusion to these experiments is that the best ear in the optimum frequency range can nearly discriminate one acoustical quantum, but it cannot quite, which opens up possibilities for deception. In best conditions the ear discerns one in two, and for frequency discernment this is reduced to only one in four [Gabor, 1947].

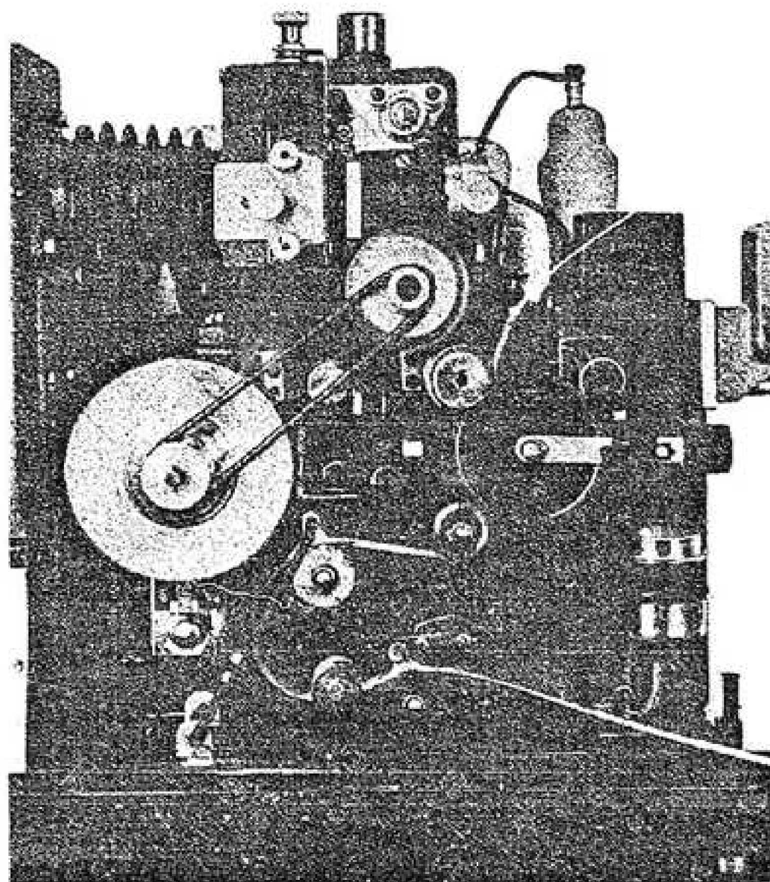

Figure 2.4: 16mm sound-film projector converted into an experimental frequency convertor [Gabor, 1946, p. 452].

2.3.3 Deception

> The ear has the remarkable property that it can submit the material presented to it not only to one test, but, as it were, to several. Ultimately only direct tests can decide whether any such scheme will work satisfactorily. [Gabor, 1946, p. 445]

Gabor decided to put his analytical conclusions to the test. He did this by constructing a number of machines. The main function of the machines was to explore data reduction and time/pitch shifting. He proposed two types of machines, one was of mechanical design with moving parts which he named the kinematical[4] frequency converter. The other machine used electrical circuits. Gabor never created the electrical frequency converter, but he suggested the mathematics behind the idea, and explained how it would function. Fundamentally it was the same as the kinematical system, except the process was purely electrical, with no moving parts. Its primary purpose would be to reduce audio information to acoustical quanta which could be transmitted instantaneously through a small bandwidth communication channel.

Kinematical frequency converters

The first kinematical design was based on a film projector using optical film sound as the input source. The way the optical film projector in figure 2.4 on the preceding page, originally worked was a lamp would shine through a slit, then through the film, and onto a photocell. The photocell collected the sum of light transmitted. The film would move at a roughly constant velocity past the slit, and thus the film would be converted to sound. As seen in figure 2.5 on the following page, Gabor replaced the slot with a drum which had slots placed all around it. The drum would rotate between the lamp and the film allowing light only to pass through the slits as they were moving past. He also added a window between the the sound film and the photocell. Gabor realised that the window would add a loud crack at the start of every cycle, so he graded the sides of the window to allow the signal to fade in and out, just like the probability curve he used to envelope his analysis data. The slits

[4]Kinematics is the branch of physics that deals with pure motion, that is, without reference to mass or the causes of motion.

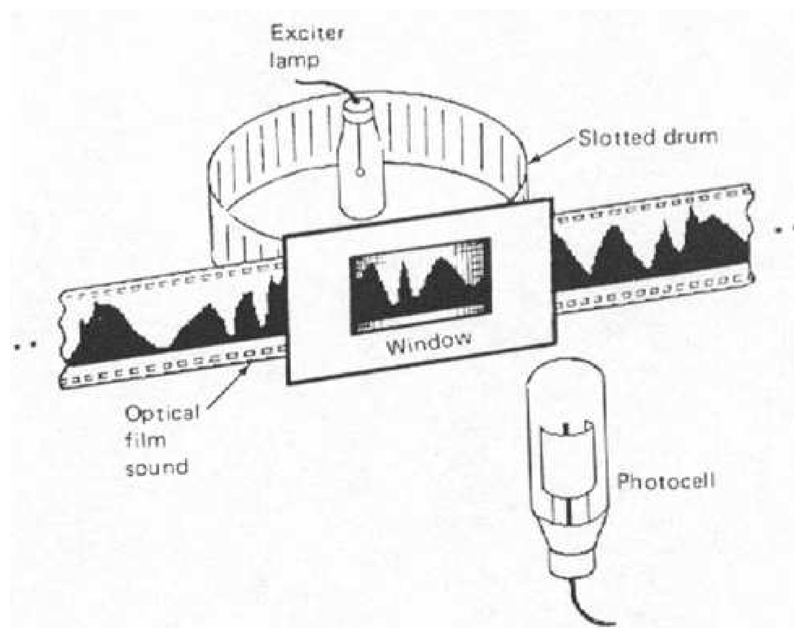

Figure 2.5: Frequency convertor with sound film [Roads, 1991, p. 174].

were distanced in a manner that one slit would start going past the window as the previous slit was nearly across the other side of the window [Gabor, 1946, p. 446].

Gabor experimented with this machine by adjusting the speed by which the optical film travelled through the machine. By varying the speed he could change the frequency. Each time a slit passed over the film a small segment of sound was created. The segments were all reproduced and added together in a sequential fashion. Gabor noted that at some frequencies he got a nearly faithful reproduction, yet at other frequencies there were strong beats [Sundberg, 1991]. This beating pattern is very clear from a graphical analysis as seen in figure 2.6 on the next page [Gabor, 1946, p. 447].

Gabor noted a number of things which would improve the quality of the sound reproduction. These include velocity control of the spinning drum and different slot positions. After some experimentation Gabor noted that the variable that made the most difference to the sound reproduction was the number of slits. If there were too many slits the reproduction would be of lesser quality, and would include more noise. If there were too few slots then the reproduction would be very good in some places, yet nearly non-existent in other places. Gabor noted that he got the best results when one slit started moving past the window when the previous slit was half way across the window. [Gabor, 1946, pp. 448-449]

Based very closely to the first machine, the second kinematic frequency converter built by Gabor was adapted from a 16mm camera. The purpose behind this machine was to be able to test how the velocity of the spinning drum affected the sound reproduction. This procedure produced varied lengths of acoustical quanta. Gabor determined that lengths of 20-50ms were most acceptable. [Gabor, 1946, p. 452]

The third kinematic frequency converter, and by far the most practical was designed by modifying a tape recorder. It followed a similar design to the two previous examples, except that it contained different parts. This time instead of needing a lamp and photocell, the tape recorder used a pick-up coil which was placed inside a rotating drum. The rotating drum consisted of a non-magnetic material with iron rods embedded within it. The pick-up coil could only read the tape when an iron rod was between the magnetic tape and the pick-up coil. The drum was also covered in an oil film, but only as a protection against scraping caused by the friction of the drum

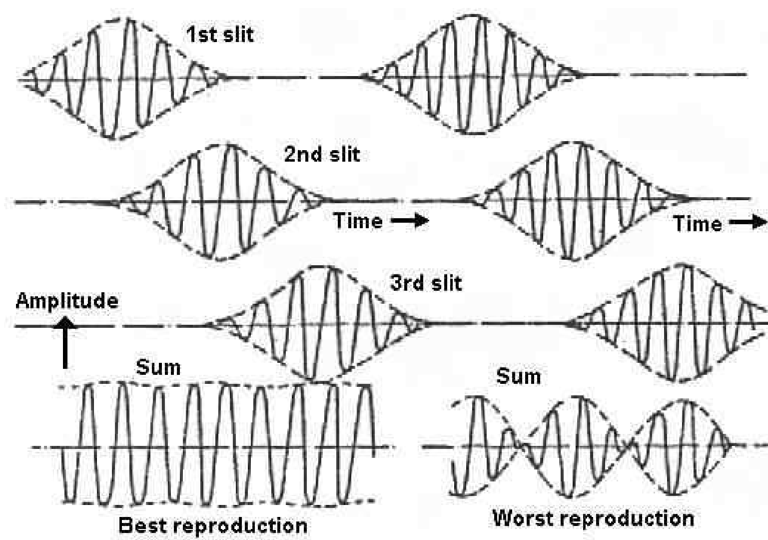

Figure 2.6: The contribution of individual slits and the resulting light output [Gabor, 1946, p. 447].

constantly rubbing against the tape. Gabor set up the tape so that it created a large loop, which meant that it could play infinitely. He also set up a wiper and a recording head so that he could constantly record new material, and manipulate it in the same process [Gabor, 1946, p. 453].

This machine in figure 2.7 on the following page, allowed Gabor to change the pitch of a sound without affecting the duration, and vice versa.

In the early 1950s Pierre Schaeffer and Jacques Poullin created another similar machine with a spinning head device. They labeled their machine a Phonogène. The German company Springer also used this idea to create a machine called a Tempophon. This machine had several spinning playback heads. The composer Herbert Eimert used the Tempophon in his 1963 composition entitled *Epitaph für Aikichi Kuboyama* [Eimert, 1963] [Roads, 2001, p. 61].

Granular synthesis at this point in history consisted only of theory which described the reduction of sound data, and sound analysis, and also a small number of machines which were used to crudely change pitch without changing the time-base of a pre-recorded sound, and vice versa. It still required more research to make it a useful tool for musical applications.

Whilst musicians such as Schaeffer and Poullin were becoming interested in the use of the spinning playback headed machines for concrete music, there was another musician who was paying keen interest to the developments of Gabor. Iannis Xenakis was interested in scientific and mathematic discoveries and the ways they could be used in musical composition. He strived to keep up to date and to understand the latest research, and strongly advised that all composers should do the same. He read the papers written by Gabor, and formalised a compositional theory for *grains of sound*, a term he coined in 1960 [Roads, 2001, p. 65]. The grains referred to by Xenakis also as *Gabor grains* consisted of logons, the elementary acoustical quanta defined by Gabor. Xenakis expanded on the theory developed by Gabor and put more emphasis on the musical properties of the process. Xenakis proposed:

> All sound, even all continuous sonic variation, is conceived as an assemblage of a large number of elementary grains adequately disposed in time..........In the attack, body, and decline of a complex sound, thousands of pure sounds appear in a more or less short interval of time Δt.........A complex sound

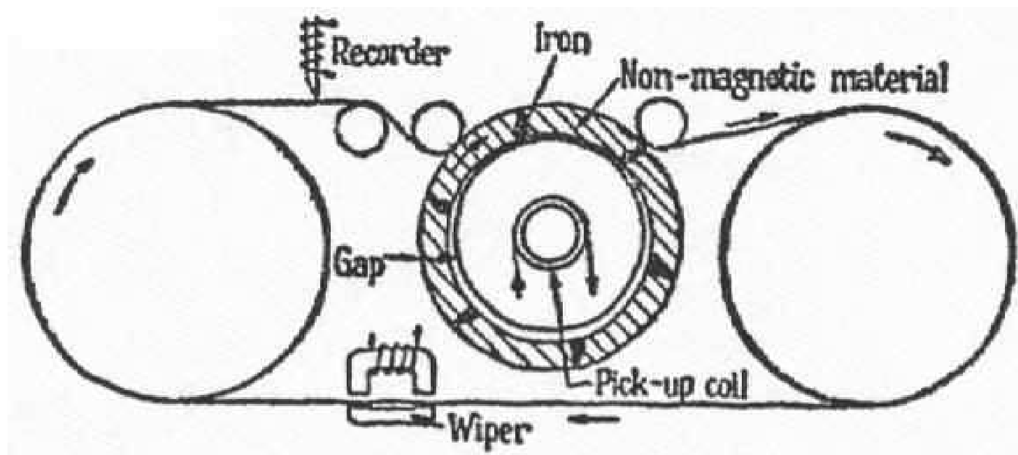

Figure 2.7: Frequency convertor with magnetic tape [Gabor, 1946, p. 453].

> may be imagined as a multi-colored firework in which each point of light appears and instantaneously disappears against a black sky. But in this firework there would be such a quantity of points of light organized in such a way that their rapid and teeming succession would create forms and spirals, slowly unfolding, or conversely, brief explosions setting the whole sky aflame. A line of light would be created by a sufficiently large multitude of point appearing and disappearing instantaneously. [Xenakis, 1971, p. 43]

Xenakis then set out to design ways he could arrange grains of sound to create complex and beautiful textures. He was particularly interested in spatialisation, in three dimensional soundscapes composed of sonic clouds all with 3D trajectories. He integrated his vast mathematical knowledge to associate this elementary grain idea with stochastic methods for arrangement [Xenakis, 1971, Xenakis, 1997].

2.4 Granular Composition

Since the concept of granular synthesis was first envisioned there has been a slowly increasing number of composers using it. I will not try and list every single piece ever written, especially as there are hundreds of composers using granular synthesis presently. Instead I will note key composers and some of their works to show how the concept of granular synthesis developed and become easier to use over time. It is impossible to talk about compositions with granular granular synthesis without talking about the software and hardware used to create them because these factors play a highly significant role in the types of sound produced and the limitations. In fact the history of granular synthesis composition is as much about the programs created to create those compositions. As computers become exceedingly fast these issues will be of little concern, but even in 2002 the kind of hardware and software a composer uses when utilising granular synthesis will have an impact.

2.4.1 Tape Splicing

The first recognised demonstration of granulation in musical composition was written by Xenakis in 1958 entitled *Concret PH*. The PH was for the H*yperbolic* P*arabloids* which characterised the Philips Pavilion in which it was first presented, not the grain

envelope shape. This piece uses a loose granular synthesis concept, but has not been created from elementary audio signals in the strict terms as first set out by Gabor, but it has a granular texture. Gabor stated that each grain should be between 20-50ms, whereas *Concret PH* uses audio segments of at least 1 second each, so it is more of an arrangement of small pieces of sound, rather than grains of sound. The piece was constructed by splicing an analogue tape into segments and then arranging the pieces in an artistic fashion, varying the density throughout the piece, by mixing sections together. The granular texture in *Concret PH* is achieved more by the contents of the audio segments rather than the length of the audio segments as it contains crackling sounds from burning charcoal, which have been layered to achieve the granular crackling sound. Unlike many of Xenakis' other works this does not follow any kind of mathematical formula [Harley, 1997, Roads, 2001, pp. 64-65].

The second piece using granulation, also written by Xenakis in 1959, is entitled *Analogique B*. This piece was determined stochastically using Markovian indeterminism, following a detailed method outlined in his book formalized music, but basically he designates the grains using a number of different array matrices of parameters and setting them to create *screens*. A screen is a grid representing the audio and time spectrum. Each cell of the grid is given parameters associated to the grain structure in that audio-time region. Xenakis created books of screens to represent complex audio sounds. The sound source he actually used for this composition was created by recording sine tones generated by tone generators onto an analogue tape. The tape was cut into fragments and then meticulously pieced together. *Analogique B* was created to complement *Analogique A*, which was derived using the same stochastic methods, but written for two violins, two violoncellos, and two contrabasses [Xenakis, 1997, Roads, 2001, pp. 65-67].

2.4.2 Music V on a Burroughs B6700

The computer has had a phenomenal effect on music in general. It has allowed many composers to experiment with algorithms and numbers in ways that would have been too arduous previous to this period. It has also allowed composers to experiment with microtonal music. Xenakis' efforts to compose with granular synthesis by tape splicing are to be highly commended, but in order to make granular synthesis a more

widespread tool for composition it needed to have a digital medium with which to manipulate the grains. The computer made it possible to incorporate many mathematical algorithms directly into the structuring process of the granular texture. In the early days of computing however this was still a very laborious task.

Motivated by Xenakis, the composer Curtis Roads became extremely interested in granular synthesis [Risset, 1991, p. 33]. Roads first heard about granular synthesis when he attended a workshop conducted by Iannis Xenakis in 1972. Since first learning about granular synthesis Roads has spent much time researching and writing about granular synthesis [Roads, 2001, p. 108].

Roads first started working with computer sound synthesis on an advanced mainframe, the dual-processor Burroughs B6700. This mainframe was exceptionally good for its time, but using it was still very time consuming. The Burroughs machine was running the program *Music V,* written in 1996 by Max Mathews, and had to be programmed using the language *Algol* [Phillips, 2000, p. 176]. All input was done through the use of punched paper cards. Due to storage limits on the mainframe, Roads was limited to one minute of monaural sound at a sampling rate of 20kHz. It would take several days to process a minute of sound. The mainframe, as was typical, was also unable to produce audio. Roads would take the punch cards to the Burroughs machine, once it had finished processing the data would be written to a (large) digital tape. The digital tape would then have to be transferred to a disk cartridge which involved setting up an appointment at the Scripps Institute of Oceanography. Finally, the cartridge could be converted to Audio on a DEC PDP-11/20 at the Centre for Music Experiment [Manning, 1995, Roads, 2001, pp. 109-110].

Using this process Curtis Roads composed the first piece of computer generated music using granular synthesis. It was just a technical experiment entitled *Klang-1* and was written in 1974. This piece of music had a duration of 30 seconds. It was an experiment testing three parameters. Grain Envelope, grain duration and density. *Klang-1* featured 766 grains of sound. Each grain needed to be programmed onto a separate punch card. Roads treated each grain as if it were an individual note, just with a very short duration, specifying the frequency, start time and duration of each note. Roads fixed the duration of each grain at 40ms, which was identical to the grain duration of Xenakis' *Analogique B*. The densities ranged from 0 to 25 grains

per second. The grain envelope was a simple Gaussian envelope. The grains were all generated as sine waves with frequencies from a 1000-tone scale ranging from 16.11Hz to 9937.84Hz. At the completion of this experiment of granular synthesis on a computer Roads made the following statement: "I did not intend *Klang-1* to be anything more than a technical experiment. Yet I vividly recall the magical impression that the granular sound had on me as it poured forth from the computer for the first time." [Roads, 2001, p. 302]

A recording of *Klang-1* can be found on the CD accompanying Roads' book *Microsound*. Unfortunately the recording has become distorted due to technical factors.

The second piece of music using granular synthesis realised on a computer is entitled *Prototype* and was written in 1975. This is an actual composition rather than an experiment. To compose this piece Roads wrote a program called PLFKLANG in Algol which could generate thousands of grain specifications based on seven high-level *cloud* parameters. A cloud is a cluster of sound grains. Clouds are significant because they can define an entire texture of grains, whereas the grain is discrete as an individual particle. He also used a different shaped envelope, a quasi-Gaussian curve which had a longer sustain section. He reduced the grain duration to 20ms and increased the grain density [Roads, 2001, p. 303].

Prototype goes for a duration of eight minutes and contains 123 clouds. Due to the monaural one minute limit restriction Roads had to divide each layer into one minute subsections. When he finally had all parts recorded in an analogue format he went to a recording studio where he mixed all of the parts together, as well as adding some reverberation and spatial panning [Roads, 2001, p. 303].

Later versions of the computer program *Music V* maintained and developed by Daniel Arfib contained probably the first code written to perform analysis-synthesis in terms of the Gabor grain. Risset used the granular analysis-synthesis tools of *Music V* on an IBM-PC to perform time stretching of recorded samples in his piece entitled *Attracteurs Etranges* [Risset, 1991, p. 33].

2.4.3 Music 11 on a DEC PDP-11/50

One of Roads' next granular synthesis compositions, entitled *nscor,* was initially released in 1980, although it was released a number of times in different versions, the

final version coming out in 1987. He began composing this piece in 1975. Due to the extensive period over which it was created there is no full score available for *nscor*. In 1980 Roads was given a research position at the Experimental Music Studio at the Massachusetts Institute of Technology where he had access to a DEC PDP-11/50 minicomputer running the UNIX operating system which allowed him to experiment more freely. The computer was not as powerful as the Burroughs mainframe, but he could do all of the work on a single computer, and it even had a keyboard interface for input. Being UNIX based meant that it could be programmed using the C programming language. It also ran a program called *Music 11* written by Barry Vercoe, and based on *Music V* [Phillips, 2000, p. 176]. Roads created two granular synthesis data generating programs in C that produced data which he could use directly in *Music 11*. Using material collated over the previous five years and the new work he created on the PDP-11/50, Roads constructed *nscor*. The final version had a duration of 9 minutes. Even though the sounds were all created on a computer it was still impossible at this time to construct the entire piece on a computer, and so for his 1980 release Roads spent months splicing and mixing segments of realisations of the piece which had been transferred onto analogue tape. All of the mixing was done on two-track and four-track tape recorders. It premiered at the 1980 International Computer Music Conference in New York. *Nscor* was updated a number of times leading to 1987 as advances in computer hardware and software were made, it utilizes a number of different synthesis techniques, granular synthesis being just one of them [Roads, 1985, pp. 142-148] [Roads, 1987] [Roads, 2001, pp. 306-307].

In 1981 Roads released another piece entitled *Field* which also included granular synthesis. This piece like the previous was created digitally in sections, and was also mixed using analogue equipment, although he received a special grant allowing him to mix it in a 24 track studio. Field contains just 6 seconds of granular synthesis which act as a climax dividing the two main sections [Roads, 2001, pp. 306-307].

2.4.4 Many programs on many computers

Since writing these first four pieces, Roads has written a few other pieces of music utilising granular synthesis. In 1988 Roads started using an Apple Macintosh II in his home studio, and started writing Macintosh programs to create granu-

Figure 2.8: Curtis Roads sitting at the Creatovox [Roads, 2001, p. 196].

lar synthesis data for *Music 4C*, and later in 1992 on a Macintosh Quadra 700 for *Csound*, *Csound* being derived from the *Music 11* program, and also written by Vercoe [Phillips, 2000, p. 176]. In 1995 Roads wrote a stand-alone program for the Macintosh entitled *Cloud Generator* which allows the user to set different parameters to create a single cloud of granular particles of sound [Roads & Alexander, 1995]. This is a free program based on his previous data producing programs, although it does not require any other programs to process and synthesize the data to create the audio. This program is extremely useful as a teaching aid, although it can only create single streams of grains, with a single ramp for each variable, meaning that for a composition the composer would need to create each small section as a separate audio file and mix them together afterwards. Roads currently writes programs for *SuperCollider* which is his current favourite choice for sound synthesis [Roads, 2001, p. 111].

2.4.5 Real-time

Roads is currently involved with the *Creatovox* project at the Centre for Research in Electronic Art Technology (CREATE) at UCSB. The Creatovox, shown in figure 2.8 on the preceding page, is a granular synthesizer. The goal of this instrument was "to invent a prototype instrument optimized specifically for expressive performance of particle synthesis and granulation." [Roads, 2001, p. 194]

The *Creatovox* is comprised of two components, the synthesis engine, and the musical performance interface. The synthesis engine controls the following parameters: particle scheduling, overlap limits, amplitude balancing, and graceful degradation in the event of imminent computational overload. It also allows the user to choose from a variety of particle synthesis methods such as pure granular synthesis, grainlet and glisson synthesis, pulsar synthesis and sound file granulation[5]. The prototype for this engine was written in *SuperCollider 2*. The musical performance interface is comprised of a MIDI keyboard, a pedal board, expression pedal, sustain and sostenuto foot switches, MIDI joystick and a fader box [Roads, 2001, pp. 193-197].

Creatovox was first used on the 13th of July, 1999. The first composer to use the *Creatovox* was Bebe Barron who wrote a piece called *Mixed Emotions* in the sum-

[5] See A on page 131 for a glossary of these terms.

mer of 2000 [Roads, 2001, p. 195]. The *Creatovox* is still in prototype stage and is constantly being updated.

Barry Truax started learning about granular synthesis after reading an article published in the Computer Music Journal by Roads in 1978 entitled *Automated granular synthesis of sound*. Truax's approach to granular synthesis was very different to that by Roads. He made the following statement:

> ....Curtis Roads had done it in non-real time, heroically, hundreds of hours of calculation time on mainframes, just to get a few seconds of sound. He had done that, but it remained a textbook case. As soon as I started working with it in real time and heard the sound, it was rich, it was appealing to the ear, immediately, even with just sine waves as the grains. Suddenly they came to life. They had a sense of what I now call volume, as opposed to loudness. They had a sense of magnitude, of size, of weight, just like sounds in the environment do. And it's not I think coincidental that the first piece I did in 1986 called *Riverrun* which was the first piece realized entirely with real-time granular synthesis.... [Iwatake, 1991].

Truax was the leader in real-time granular synthesis, pioneering the way right from the onset of his research. In 1986 he wrote a real-time application of granular synthesis implemented using his previously developed *POD & PODX* system. At the time Truax implemented the real-time granular synthesis algorithms in *PODX* he was using a DMX-1000 Digital Signal Processor controlled by a DEC LSI-11/23 microcomputer. Using both the DMX-1000 and the DEC LSI-11/23 meant that the sound generation could be handled by a processor optimized for audio, whilst the microcomputer would only have to be concerned with number crunching and outputting a stream of numbers [Truax, 1986, Truax, 1987, Truax, 1999b, Truax, 1999a]. The computer was set up so that the granular synthesis parameters could be changed in real-time with a single key stroke. Truax was also able to store a range of presets and assign them to any unused key on the keyboard. The parameters could also be set using a *tendency mask*, with is a graphical representation of a range over time. The graphical tendencies masks would not be processed as graphics, the graphical view was for the benefit of the composer, and were translated to a combination of presets

and ramps [Truax, 1988]. Using this system, Truax composed the piece *Riverrun* in 1986. The duration of the piece is just under 20 minutes. It is the first example of granular synthesis in real-time. The composition was recorded and mixed onto 32 tracks. This composition pushed the DMX-1000 to its limits. These limits included a grain density between 1 and 2375 gains per second, and a maximum of 19 overlapping grains. The contents of the grains was limited to just simple sine waves, or basic FM waves, although in the sections where FM synthesis was used the DMX-1000 could only handle a maximum of 1000 grains per second, with a maximum overlap of 8 grains. The frequency range synthesized is from 20Hz-22kHz. All of the envelopes, and all of the parameters used are simple linear equations. Using simple equations allows for much faster processing, which is crucial for a real-time application [Truax, 1990, Manning, 1995, Roads, 2001].

In 1987 Truax pushed his real-time work forward even more by composing the first composition making use of digitally granulated sound samples. The piece was entitled *The Wings of Nike* and had a duration of about 17 minutes, consisting of 4 movements. The first, second and forth movements were derived from sound samples, the third movement being derived from synthetic grains. The sound samples consisted of just two phonemes, each about 170ms in duration. The short length of the sound sample was due to the 4 k-word memory capacity of the DMX-1000. Truax regarded this work with small sound samples as a *magnification* of the original sound sample. It allows the composer to magnify its microlevel characteristics. Microscopic timbral changes that usually go unnoticed become evident and formant regions are especially pronounced. The work was mixed to stereo from 4 stereo pairs, and has therefore nearly 80 overlapping grains at certain points in time, with grain densities of up to 8000 grains per second. This masterpiece in sound granulation also has accompanying computer images of the statue, *the Winged Victory or Nike of Samothrace* [Truax, 1990, Truax, 1994].

Truax's third piece using granular synthesis came out in 1988 entitled *Tongues Of Angels*. It was developed in the same manner as his previous piece. This piece is described by the composer as a struggle of the performer to match the virtuosity of the tape work and to transcend human limitations. Both the tape section and real-time section were created using the same samples [Truax, 1990].

In writing his fourth piece *Beauty and the Beast* in 1989, Truax composed the first piece that employs real-time granulation of a continuously sampled sound. As inspiration for this piece he went back to Gabor's theory of quantum sound so as to extrapolate key ideas and elements he could exploit. Truax was interested in the notion that frequency and time are reciprocal of one another, not two totally independent variables, noting Heisenberg's uncertainty principle. He also noted that time is reversible at the quantum level stating:

> The quantum equations are symmetrical with respect to the direction in time, and similarly, the quantum grain of sound is reversible with no change in perceptual quality. If a granular synthesis texture is played backwards it will sound the same, just as if the direction of the individual grain is reversed (even if derived from natural sound). That is acoustic time has no preferred direction at the micro level. All of the characteristics that establish the directional experience of time occur at the macro level. Granular synthesis is in a unique position to mediate between the micro and macro time worlds.....Paradoxically, linking frequency and time at the level of the grain, one makes them independently controlled variables at the macro level. [Truax, 1990, p. 130]

Truax used this philosophy as the basis for investigation, calling it *variable-rate granulation*. *Beauty and the beast* explores time invariance. Sound derived from the real world can be slowed down, right to a point of stasis where a brief moment in time can occupy any duration, and then it can even start moving backwards through time.

Beauty and the Beast was written for children and adults, it includes computer images, a solo musician playing oboe and English horn, and a recorded narrator. The dialogue was used as the sound source for the variable-rate granulation.

In 1990 Truax extended the capacity for sound samples by allowing the samples to be stored on hard disk. *Pacific*, composed in 1990, was the first example of this, using natural sounds. In 1991 Truax took yet another leap forward, when with a group of engineers, he created the prototype Quintessence Box for real-time granular synthesis. This box was demonstrated at the 1991 International Computer Music

Conference. It featured a Motorola DSP 56001 chip to handle the signal processing. This box was installed at Simon Fraser University in 1993, where Truax lectures. The Quintessence Box was a major breakthrough for real-time granular synthesis. It allowed sound to be recorded and instantly granulated, providing both real-time input and real-time output. Truax has written numerous other pieces of music using real-time granular synthesis [Truax, 1999b, Roads, 2001].

2.4.6 The next generation

After the ground breaking work completed by Xenakis, Roads, and Truax there have been an ever increasing number of composers working with granular synthesis.

Horacio Vaggione is perhaps not quite in this group of newer composers as he has been working with textures of microsonic events since the 1970s, in a area of research known as *micromontage*. Micromontage has many similarities to granular synthesis except that it is executed in a totally different manner. It is more of a sculptured process in which the composer will take a sound file and carefully cut segments of sound from the source and arrange them in a sequence, bearing in mind splicing, dissolving, and other editing techniques. There is no emphasis on grain size or granulation, so the duration of the segments can vary widely from just a few milliseconds to however long that sound is required; 1 second, 5 seconds, or more, without any granulation. Programs such as *Music N* and *Csound* had audio input functions added to them such as *soundin* which allowed the composer to load in a sound sample, specify a segment from the sample and play it. Many standard multi-track audio editors such as Pro Tools use a graphical micromontage friendly layout as their standard interface in which the composer may cut and paste sound segments. These two methods, granular synthesis and micromontage, share some distinct attributes, but are executed in totally different ways with different goals in mind. Vaggione used micromontage to create many works such as *La Maquina de Cantar* in which he used an IBM PC to control a Moog synthesizer generating up to 40 sounds per second, and *Thema*, created at IRCAM, using *Music N*. Despite his long history with microsonic techniques he did not create any music using the granular synthesis technique until 1995. This work entitled *Schall,* was described by Roads as “an outstanding example of the use of creative granulation” [Roads, 2001, p. 313]. *Schall* was created from sampled pi-

ano notes. It is an interplay between switching time scales, which is something often explored in micromontage, as well as an exploration of pitch and noise, created by varying the grain durations. In 1997 and 1998 he released *Nodal* and *Agon* respectively. An interesting aspect of Vaggione's work is the level of detail to which he goes to in his pieces. There always appears to be a number of levels of activity, especially in the background [Roads, 2001].

Another composer who pioneered micromontage was Paul Lansky, who created granulated works such as *Idle Chatter* (1985), *just_more_idle-chatter* (1987), and *Notjustmoreidlechatter* (1988). These pieces are all generally created from 80ms phoneme fragments and explore articulation of meter and tonal harmonic progressions. Lansky has also worked a little with granular synthesis. [Roads, 2001, Dodge & Jerse, 1997].

As the rise in popularity of granular synthesis has increased, especially over the past 10 years, there have been many patches, plug-ins and add-ons created for the larger synthesis programs such as *Cmix, Csound* and *Max/MSP*. The following paragraphs contain just a few brief notes of some of the programs I have used, or heard others speak highly of.

During the 1990s James McCartney wrote a couple of computer programs that can be used for granular synthesis compositions. The first was a program called *Synth-O-Matic* which was used for creating envelopes to control synthesis parameters. The other was the *SuperCollider* series for Macintosh computers which has become widely used by composers such as Roads. *SuperCollider* is an object oriented graphically based programming language that had been created specifically for working with audio and MIDI [Roads, 2001, pp. 113, 115]. Early versions are free to download from the Internet. McCartney is continuing the *SuperCollider* project and the latest available version is *SuperCollider 3* [Opie, 2000].

Jean Piché has given some excellent performances using granular synthesis, although he is currently most noted for co-creating the program *Cecilia* with Alexandre Burton. Cecilia is a complete music composition environment and front end for *Csound*. It contains many useful graphical objects such as interactive parameter graphs and wave displays. This program is freely available for Macintosh and UNIX based Operating Systems. It has recently become available for the Windows Operat-

ing System [Opie, 2000, Phillips, 2000].

Xenakis conceived a hardware synthesis system that was first built in Paris at the Centre d'Etudes de Mathematique et Automatique Musicales (CEMAMu) in 1977 called *UPIC* (Unite Polyagogique Informatique de CEMAMu). This system has been updated a number of times as technology has improved, achieving real-time in 1988. It provides a graphical interface via a computer and the composer engages the mouse or graphical tablet to draw sonic arcs that represent segments of sound. The system can achieve 64 simultaneous arcs, and 4000 arcs per page. The duration of a single page can be set from anywhere between 6ms and 2 hours. The composer Brigitte Robindoré used this system in 1995 to write a granular synthesis piece of music entitled *Comme étrangers et voyageurs sur la terre [Roads, 2001, pp. 158-160, 320].*

Other synthesis programs that can be used for granular synthesis include: *Nyquist*, an interactive LISP based scripting program written by Roger Dannenberg [Dannenberg, 1997a, Dannenberg, 1997b]. *Max/MSP*, a commercial graphically oriented programming language with highly optimized audio routines written by Miller Puckette at IRCAM. Patches such as *GiST* (Granular Synthesis Toolkit) and other *Max/MSP* granulation extensions were created by a group at IRCAM including Corte Lippe [Phillips, 2000, Roads, 2001]. *KYMA,* a commercial graphically oriented synthesis package developed by the Symbolic Sound Company that has been used by composers such as Agostino Di Scipio. [Roads, 2001]. *AudioMulch*, is a graphically oriented synthesis program written by Ross Bencina that is gaining popularity especially amongst younger composers looking for an inexpensive high quality product with an easy to use interface [Opie, 2000]. *jMusic*, which I will refer to in much greater detail later in this paper, written by Andrew Brown and Andrew Sorensen [Brown & Sorensen, 2000a].

Some granular synthesis application programs currently available include: *MacPOD*, which is based on the *GSAMX* functionality of *PODX* and written by Damián Keller and Chris Rolfe for Macintosh [Keller & Rolfe, 1998, Rolfe, 1998]. *AL & ERWIN,* an algorithmic compositional environment that can create granular clouds based on quantum mechanic algorithms [Fischman, 2003]. *Granulab*, a real-time wave file granulation program which allows the composer to set many parameters pertaining to granular synthesis, such as grain densities, grain duration, random-

ness, and more. Written by Rasmus Ekman for the Windows OS, it also works very well on UNIX systems under WINE[6] [Phillips, 2000, pp. 355-357]. *Granulator*, another wave granulation program recently written by Nicolas Fournel [Opie, 2000]. There are many other programs that could be included, but this covers the basic range without getting into trivialities.

Alongside the compositional element of granular synthesis there has also been concern regarding the reconstruction and organisation of grains and textures. Some of these compositional constructs have already been mentioned in passing, such as stochastic distribution of grains using Markovian indeterminism, randomness, tendency masks and sculpting. Other methods such as using genetic algorithms for grain distribution have been tested [Rowe, 2001]. David Chapman, Michael Clarke, Martin Smith, and Paul Archbold researched and even created a computer language called *AMY* to help them study fractal based granular synthesis [Chapman et al., 1996]. Ecologically-based granular synthesis has been experimented with by Keller and Truax. What is interesting in their approach to sound organisation is their view of what they are trying to achieve. They pointed out a statement made by Dannenberg to the effect that there is a lack of research done in sound organisation for time scales ranging from 10ms to a several seconds. The micro-scale synthesis issues have been well researched, such as synthesis and analysis of a single acoustical quanta. There has also been a lot of research regarding the macro-scale, programs like *UPIC* and many of the algorithms previously mentioned to create large textures. What Keller and Truax began researching were short natural sounds that are recognisable. Sounds like bouncing, scraping, breaking and filling. They created algorithms to create these types of sounds in *Csound* and *Cmask* using granular synthesis [Keller & Truax, 1998]. The examples and code were released on CD under the title *Touch 'n' Go* through Earsay records in 1999 [Roads, 2001]. At MIT, Dan Overholt designed a real-time control system called *The Matrix* (Multi Array of Tactile Rods for Interactive eXpression). It is designed so that the user can control many parameters at once with the unique control device [Overholt, 2000].

[6]WINE: Recursive acronym for Wine Is Not an Emulator. It is primarily a Windows library that allows some Windows programs to run natively on UNIX based systems.

2.4.7 Particle pops

Besides all of the compositions that were designed and composed with granular synthesis as part of the process, there are many popular artists who have also made use of this synthesis technique whilst perhaps having never heard of the term "granular synthesis". They were not looking for synthesis methods, nor were they exploring ways to deceive people's aural functions, they were simply looking for sounds and effects to enhance their music. An early example of this is David Bowie's *The Laughing Gnome* released in 1967 [Bowie, 1967]. This song features Bowie singing a duet with himself, but his other self has a transformed voice. Although undocumented it sounds like Bowie has used some kind of machine, possibly a Tempophon, or something similar. He has shifted the pitch of his voice up an octave whilst keeping the time base about the same. There are a number of songs that use this pitch shifting effect and are all generally used in a comical sense similar to the way Bowie used it. This effect does not seem to be used very often anymore, not for comical effects anyway, it has become too passé, but it can be heard in more subtle contexts.

In the mid to late 1980s a very popular effect in dance music was to repeat fragments of words repetitively to create rhythms. One popular example of this is Mel & Kim's *Respectable* released in 1987 [Mel & Kim, 1987]. The First word in the song is Take, but this gets extended to say 'Ta, Ta, Ta, Ta, T, T, T, T, T, Ta, Ta, Take or leave us'. An effect like this has been simply produced with the use of a sampler, but they are extremely short samples. The effect is used to reduce a word to just fragments of sound, and then use them in a compositional and musical context rather than a verbal context. In the case of *Respectable*, Mel & Kim also incorporate pitch shifting to the small sound segments. Kylie Minogue also first used this effect in 1987 with her cover version of *Locomotion* [Minogue, 1987], although not as effectively as Mel & Kim. This effect is still used occasionally, but not to the prolific extent of the 80s dance tunes.

Another effect that became very popular in the late 1980s and which still continues strongly in the 2000s is the use of record scratching as a rhythmic and audio effect. Rap, Gangsta Rap, and Hip Hop, are styles of music in which pre-recorded vinyl records are used as the base of a new song, although some groups use samplers instead of vinyl. Sometimes only a short passage of the base is used, such as the chorus

line, or sometimes just a short rhythmic patter, but it is repeated continuously. Using a drum machine a strong drum and bass line are usually added to the base. Over the top of the base the composer adds in other sounds, these sounds are generally also created with vinyl records. By controlling the spin rate of the turntable, the direction, and using the turntable mixer the composer is able to generate small enveloped audio segments that can be very different from the original sound source. Using this method it is possible to create a single channel of continuous granular rhythmic patterns. A good DJ can very quickly create long and very detailed rhythmic patterns by effectively scratching the turntable. Some groups that use this technique prolificly include Eric B & Rakim, De La Soul, House of Pain, Public Enemy, NWA, and Run DMC.

A popular artist who uses granular synthesis, in a perhaps more enlightened context is Norman Cook, also known as Fatboy Slim. Cook exploits the persistence of digital audio sample replication in a collage of samples and drum and bass. The samples used consist of long melodic fragments and vocal lines right through to tiny granule sized samples, no larger than a single acoustic quantum. The piece entitled *The Rockafeller Skank* is a particularly good example of the use of granular synthesis in popular music. Approximately two thirds of the way through the piece, as the sample is playing, it is stretched out to the point where it granularises and reaches a point of near stasis. Once it reaches this near stasis, the granules begin to gradually decrease in duration whilst the density slowly increases. When they have become so small that they are barely long enough to hear individually, the density is thick enough that it becomes a buzzing noise and the grains get pitch shifted up in a sliding motion, in a lead up for the final section of the piece [Fatboy Slim, 1998].

A vocal technique called the *vocal fry* [Sun, 2000] could be seen as an example of organically created granular synthesis. The vocal fry is created when the voice is creating a sound which is not strong enough to be properly vocalised, and it comes out in a granulated form. The sound quite literally breaks into a stream of small granularised pieces which can be controlled. A singer with strong vocal control can indeed control many aspects of the sound such as the pitch, grain duration, density, synchronicity, and intensity all to varying degrees. This technique has been used in many forms of music, from popular bands such as Nirvana to experimental vocal

music such as composed by Cindy John. The vocal fry is very limited as it is difficult to control with precision, and it can only produce a single stream or layer of grains. To produce a multi-layered granular texture would require a choir.

Chapter 3

Live Performance

As the technology used to render granular synthesis increases in capacity and performance it is natural that its usage is becoming more prolific. Composers and musicians now have the opportunity to carry their electronic equipment into any venue and perform using granular synthesis in a real-time environment almost as if it was pre-recorded. In the following section I will sample a few artists, their performances, and discuss what kind of tools they prefer to use for real-time granular synthesis. I have specifically made this section mostly australocentric, as the location and local climate pertain more closely to latter parts of this paper.

3.1 Australian and New Zealand Artists in performance

Paul Doornbusch has performed live with granular a number of times. He has used a number of different systems and software programs including the *ISPW, SuperCollider*, and *Max/MSP*, in conjunction with *GIST*, which he highly recommends. These have all worked well for him in performances. His performances involving granular synthesis all took place in Europe in the 1990s, although the Australian Broadcasting Company (ABC) has played his recordings locally. [Doornbusch, 2002]

Donna Hewitt has done some live performances using *AudioMulch* controlled by a joystick as pictured in figure 3.1 on the next page.

In a recent performance she used a live vocal input as her sound source for granulation in real-time. She was particularly interested in sustaining or 'freezing' particular sounds she came across whilst granulating. Hewitt found the *AudioMulch*

Figure 3.1: Donna Hewitt in performance. Photo Mr Snow

interface easy to use and work with. Using the joystick was a fun and fairly effective way to control the sound.

She has also used *MacPOD*, *Granulab*, *Pegger* (*AudioMulch* plug-in), and other programs for granular synthesis.

In correspondence with Hewitt about how the joystick worked she said the following:

> Basically I was controlling the length of the grains on one axis and I think I had transpose/pitch shift on the other axis. I did change what the joystick was controlling throughout the piece though. I'd have to check the patch to remember exactly. I do find some things frustrating about the XY axis thing, one in particular is the fact that it springs back to the centre if you do not hold it. It would be nice to be able to lock it off when you hit a section that is nice....freeze is good for that in the mulch granulator. I guess one could always take the spring out of the joystick.
>
> It was not easy getting the joystick working. I had trouble trying to get a joystick to midi converter that would work reliably with Windows XP. I eventually had to learn to use PD, which enabled me to convert the joystick into midi. The next problem was then getting the MIDI into *AudioMulch* which is the software I was using for the performance. Because I was running Windows XP I could not use Hubis MIDI loop-back and so I tried MIDI Yoke which caused so many problems that I eventually just looped out of the computer using my USB MIDI interface. Once I got that working it was quite easy to use the joystick to control the *AudioMulch* parameters. I did find the joystick a little unreliable as far as trying to obtain fine increments on the parameters. It tended to jump around a little which I believe is called jitter. Apparently this 'jitter' is quite common and there are ways to minimise it using programs such as PD and Max. [Hewitt, 2002]

Julian Knowles has been working with granular synthesis for a long time now, and it comes up frequently in many of his performances. He has used *MacPOD*, *Max/MSP* (with and without *GIST*), *Cloud Generator, Thonk, Metasynth*, the *granular like* pro-

cesses in *GRM tools, Granulab, AudioMulch*, and more. His favourite granular synthesis program is *MacPOD*. He is impressed by the extremely smooth textures that can be created with it, and its simple interface. *MacPOD* does not have some of the *features* that other granular synthesis programs include, and Knowles speaks on this:

> I am not so into features in a sound processor, I am into the sound that can be produced from it. The usefulness or otherwise of a sound processor, for me, rests almost entirely on the algorithm which resides under the bonnet and the sound that emerges as a result. In the case of *MacPOD*, it is an extremely good granulation algorithm. The bottom line is that if a piece of software is very flexible but sounds bad, then it is not of much use to me. It is very clear from the outset that not all granulators sound the same. The same could be said for reverb processors - ie massive differences in sound between various implementations/algorithms.
>
> Even when using a fully featured granulator, I almost never use the pitch randomisation, or have it madly skipping across the source file. I much prefer more subtle, stable, stretched textures. *MacPOD* gives you exactly what you need to do this kind of thing and leaves out all the stuff I would not really use anyway. Its the right sound for my music.
>
> The other great thing about *MacPOD* is that it can be copied as many times as you like and you can run many streams of it. This is due to the fact that it operates under sound manager (and I usually run other stuff via ASIO at the same time). The only thing that bothers me about this is that sound manager lacks resolution. If sound manager could handle 24 bit, I would be very happy. [Knowles, 2002]

Ross Healy has been working with electronic music for about 15 years, although he has only been using granular synthesis for the past four years. For his performances he uses a PC Notebook upon which he produces real-time granular synthesis, amongst other things. As he has been working with electronic music for so long, he has seen technology improve over time, and is happy with how smoothly and quickly a small Notebook computer can number-crunch high amounts of data and produce a smooth sound.

Healy's favourite program is *AudioMulch*, the reason being that it allows him (and anyone) to create their own music processing system. He likes using many different techniques to keep everything interesting, which *AudioMulch* is capable of. He says it is really quite a simple program with lots of potential, it can be whatever you want it to be. Healy also uses *GRM tools* and occasionally *Granulab*. He releases CDs of his work under the name CRAY. [Healy, 2002]

Julius Ambrosine has performed with real-time granular synthesis a number of times. As well as using granular synthesis packages Ambrosine has investigated using other devices in a granular context. A CD player is one example. He stated:

> A raw 5min sample of a CD that has been drawn on and/or scratched so it skips, through a granulator is great. Even better if it is not a sample rather a real time skipping CD giving a feeling of impermanence to the usually permanent medium of CD. I feel the resulting granulated sound also has a feeling of very much a random impermanent aesthetic. The CD player can also be played by hitting it or skipping (forward, rewind) through the already skipping disc. [Ambrosine, 2002]

As well as playing scratched up CDs, Ambrosine also uses *Granulab* quite a lot, and occasionally *MacPOD, AudioMulch, CrusherX*, and certain *VST* plug-ins. He likes to use many different kinds of variations of granulated sounds, rather than just the popular "wash" of granulated sound that has become distinctive as granular synthesis. Ambrosine is especially interested in very low density textures that create bubbly random beats, and also likes using granulation for very subtle filter effects.

Ambrosine feels that the Granulab program is intuitive, and gives good feedback to the person using it. It has been often noted that Granulab has performance issues, in which the program will stall if the user tries to do something that the computer cannot cope with. Ambrosine speaks on this issue, and on Granulab:

> Yes it glitches (which can be great), but you simply have to press [Shift][Q] and turn something down, often the *frequency*, *amplitude* or *output amp*. I feel this is just part of the program, a limitation if you like. Limitations can be good since you are forced to try new approaches.

> Good real-time sounds can be quickly achieved. However it is the way it is approached that is highly relevant to the aesthetic outcome. I've heard a number of people use it and do the obvious, you can predict what some people will do with Granulab. Because it is intuitive does not mean it is by any means easy to achieve good outcomes. I like the general timbre created by Granulab. I also like the fact you can open it a number of times simultaneously. And v.1 is free! [Ambrosine, 2002]

One thing Ambrosine noted about granular synthesis performances is that many people in the audience have no idea what you are doing, but they like the sound.

Susan Frykberg, a New Zealander who studied with Barry Truax at Simon Fraser University in Canada, has used granular synthesis for a number of different projects. In 1991 Frykberg presented a performance, entitled *Woman and House* as part of her Masters Degree. In this performance art Frykberg fuses together a number of different elements of electronic music, vocal work, theatre, storytelling and ritual. For the electronic section Frykberg used the *PODX* system with the DMX 1000 to create her granulated musical works. The music is used to represent different objects in the performances. For example, the coffee machine song uses sampled gurgling and spluttering sounds which are granulated to become more dynamic and percussive granular synthesis textures. The television song uses samples of a television advertisement that have been granulated into four channels and mixed to get a garbled granular synthesis advertisement. The house also had some of its own songs. The electricity for example was created from a 60Hz frequency, the frequency of electricity in Canada, and its harmonic series which is granulated to play in fragments. All of these sounds/songs, and others, were granulated live during the performance [Frykberg, 1991]. In her unique theatrical style Frykberg has created other vocal work and granular synthesis pieces of music that explore feminism and birth [Frykberg, 1998].

3.2 The Author's performance experiences with granular synthesis

I have written three granular synthesis compositions that have been recorded and publicly performed. The first was a piece entitled *Thesisone*, which was composed

using *Csound* and a spreadsheet program. It has a duration of exactly 10 minutes. This piece was created using a 4 second sound sample which features part of a guitar riff from a piece named *Bijou*, played by Brian May from Queen. The piece was composed by extracting 400 grains from the sound sample, and begins by playing every grain individually in a sequential order from beginning to end. Every 4 seconds the sequence repeats, but 3 of the grains are moved to different positions. This process takes place 150 times. By the end of the piece every grain has been rearranged and the original sample is totally unrecognizable. The piece contains a total of 60,000 grains, each one was defined in a spreadsheet, first by defining the original 400 grains with an amplitude, envelope shape, duration and panning, copying those 400 grains, but with three alterations to the order, and also some slightly random changes to the duration, amplitude and panning of each grain, and then reiterating the process. This piece was played at an honours research seminar at La Trobe University in mid 1999, and it explores the slow degeneration of sound through minimal manipulations made to the order of sound grains.

The second piece, entitled *Where the wind meets the sea,* was written in late 1999, and has a duration of approximately 5 minutes. It was inspired by the works of Barry Truax. It was composed in *Csound* making exclusive use of the grain command. The sound source for this piece consists of 3 sounds, a basic sine wave, a sine wave with inharmonic partials, and white noise. It also includes a number of different envelope shapes ranging from a triangle to differently shaped Gaussian like curves. It was composed by firstly drawing the sound in a tendency mask style, then calculating the point, and slopes, and ramps. Because this was to be rendered in non-real time I used exponential curves as well as linear ramps. This piece was submitted as part of my honours thesis. It has also been played at a couple of seminars and has been available on the Internet for two years. It explores the diversity and interaction of multiple streams of audio. I see it as the meeting of very elemental energies [Opie, 2000].

The third piece, entitled *Elucidation,* was written in early 2002, and has a duration of approximately 4 minutes. It was composed using a real-time granular synthesis instrument called a poseidon, as shown in figure 3.2 on the following page. This instrument will be explained in great detail in later chapters. The piece is a duet

Figure 3.2: Close up of the poseidon control interface.

written for flute and poseidon, originally performed by Nicole Stock and myself. This piece starts with the poseidon singing away when the flute begins playing. At first the poseidon ignores the flute, but then becomes enchanted by the flute and tries to imitate it. The poseidon is not very successful and in the end tries to find a way to complement the flute. This piece is a reflection on the difference between acoustic and electronic instruments and ways that they can be integrated in live performance. It also tries to show that the poseidon is an instrument just like a flute, but it just needs to be better understood. This piece was played at the 2002 Australasian Computer Music Conference. It is also available on the Internet [Opie, 2002].

I have also used the poseidon for a number of real-time live improvised performances since its creation, some of which will be discussed in more detail in later chapters.

Chapter 4

Development

The Queensland University of Technology in association with the Brisbane Powerhouse formed a partnership to create a festival known as REV. An acronym for Real, Electronic, and Virtual, REV was a festival about creating new sounds with new instruments. Lindsay Pollack stated:

> Music, and the technology for music production are inextricably linked and always have been, from when the placement of finger holes in the first flute influenced the development of the tuning and temperament of that particular local musical instrument. People have always experimented with sound and the instruments that make sound, but we rarely get to hear and see many of those experiments. [Brown, 2002, p. 2]

REV was about experimenting with sound, producing experimental instruments to do so, and then inviting the public to come in and experiment with the instruments created. All of the new instruments exhibited were explored, tested, and played with by adults and children.

In preparation for the festival it was essential to define what kind of instrument I wanted to create, and then develop a strategy for design and construction of the instrument. I decided at the beginning of this project that I would build a musical instrument based on the principles of granular synthesis. In this chapter I will discuss the ideas and design processes involved in creating an instrument that could be used for real-time granular synthesis in live performance.

4.1 Ready, Aim......

The first step in creating the instrument involved setting some boundaries to work within. The following is a list of aims I wanted it to achieve.

- ❦ Use only granular synthesis techniques
- ❦ Open source software
- ❦ Versatile
- ❦ Be more than a computer program
- ❦ Inexpensive
- ❦ Mobile
- ❦ Visually stimulating

The use of purely granular synthesis techniques was an important part of the aural aesthetic I wanted to achieve. Not content to use it as just an effects process, amongst other sounds and effects, I wanted to create the entire audio output from granular synthesis.

Open source software is not a new concept. In the early days of computing before the emergence of the personal computer it was common practice to distribute software as source code [Olson, 2000, p. 2]. Open source software allows many people to collaborate on the same program, take it home, make their own changes, redistribute it and then wait for someone else to improve it further. Because there are no cost factors involved in working this way it means that the most popular variant of the program will always be the one that is most efficient and useful. After setting up the initial granulator program I wanted to be able to let the public modify it to their own needs and improve it, unconstrained by legal issues and costs.

The versatility of the instrument is an aim of the instrument itself, it has implications for the way it is used. Therefore it is important that it have an easily operated interface which could be used by many people. The ease of operation of any instrument involves many factors such as size, weight, intuitiveness, layout, etc. Another aspect of versatility is how well it combines and interacts with many other different

types of instruments. To make it more versatile in this aspect would require it to have many different control functions, but not so many as to make it unusable.

The computer alone can produce an almost unlimited number of different kinds of sounds, but as its design is primarily for data entry and access, it does not make a very intuitive interface for an inexperienced computer musician, or even an experienced one. As there were going to be many people wanting to use the instrument I decided it would be important to use some different kind of medium other than a computer interface.

The control interface, and indeed the entire system needed to be inexpensive for a number of reasons. Firstly because I wanted to show that a real-time granular synthesis instrument could be developed on a small budget, as opposed to the expensive systems that have been created by Roads, Truax, Manning and others. Secondly it needed to be inexpensive so that I, or others who also wanted to try a similar interface would also be able to do so with a small budget.

Mobility was initially not an aim of this instrument, but after devising a couple of different kinds of instruments I decided that it was a very important factor if I wanted to be able to use the instrument at different locations. It did not have to be mobile to the extent that I could hold the entire instrument and walk around the streets, but it did have to be compact enough that I could easily relocate all equipment quickly and on my own.

The final of my original aims was that the instrument be visually stimulating. It needed to be something that I could openly interact with, and something that the public could see me interacting with. Something with which I would be able to create an entire performance that was visual as well as musical. Something approaching corporeality [Burt, 2002,Partch, 1974]. I believe that in order to engage the audience during an electronic performance, I needed to create something that they could connect with. Having been to many electronic and computer performances that consist of someone sitting in front of a laptop without even glancing up at the audience, I felt that I needed something visual that I could work with, that the audience could see and identify with the sound that was being created.

These next sections will outline some of the ideas and concepts I dealt with in the instrument building stage, and how they address the aims.

4.1.1 Acoustic Granular Synthesis Machine

My initial design for a real-time granular synthesis instrument was actually acoustic. Granular synthesis has been primarily a theoretical and computer exercise, but I wanted to take all the parameters and elements of granular synthesis and produce something acoustical which could create varying granular synthesis textures. It would be an acoustic instrument that emulated a computer synthesis technique, instead of the other way around.

The instrument was to be hand operated. The musician turned a handle to play. Using some lever/pulley system they were able to control the *hammers*, which were essentially tiny weights on fishing line or perhaps thin wire that were spun in a vertical circular motion. The hammers bounced against a tuned sound board during the circular rotation to create a *grain* of sound. The musician was able to spread the hammers right across the tuned board or confine them all to a much smaller space to control the pitch. Some hammers could also be retracted to control density. The velocity of the spinning motion of the hammers would also control the duration of the grains, as well as the density. I spent a lot of time thinking about the practicality of an acoustical representation of granular synthesis and played around with lead weights and wooden objects, but I came to conclusion that it was not a practical idea. It would be very hard to build, it would give only average musical results, and it would become an instrument prone to getting tangled and easily broken. The time needed to build and then maintain outweighed any musical output it would give. Such a project, whilst being visually interesting, would not be practical for public interaction.

4.1.2 Electronics

After moving away from the acoustic granular synthesis machine, I started looking at other possibilities for a real-time granular synthesis instrument. One option I considered was using a programmable, or preprogrammed chip with granular synthesis capabilities which would allow me to experiment with granular synthesis without having to use a computer. I searched the Internet for prices on the Texas Instrument DSP chips and other such similar preprogrammed chips. Quite simply the cost needed to buy one was much more than I could spend on the project. So instead

I started investigating Programmable Integrated Circuit Chips (PIC Chips). There were a few to choose from, and they were inexpensive. There were 2 main problems with PIC chips. I would have needed to learn the appropriate language in order to program the chip, and I would also have to find one that was fast enough to cope with granular synthesis. I realised that if the chip was dedicated to just granular synthesis then it would not need to go anywhere near as fast as any recent computer CPU, but the fastest chips that were available went to a maximum of 20MHz. There was a 33MHz PIC Chip coming soon, but I decided to avoid this avenue as I just could not envision very good results coming from such a slow chip.

4.1.3 Electronic and Computer Fusion

As already stated in the aim, granular synthesis can be achieved entirely on a computer, but a visually stimulating aesthetic is very hard to create with just a computer. I decided I would create the granular synthesis engine on a computer, but I would use an entirely different interface to control it with. The task was then to find a suitable way to create a control interface. There were already a number of commercial control devices available. The two most notable were the I-Cube and the EZIO. The I-Cube has a range of sensors and can generate MIDI code from interaction with the sensors, whenever they are triggered. The main problem with the I-Cube was that it is very expensive, so this was not an option. The EZIO also has sensors, but it generates data which must be sent to the computer via a serial cable. Rather than sending a signal every time it is triggered, the EZIO runs on a polling mechanism, and only sends data at certain intervals. The main problem with the EZIO was that it was not very useful, primarily because of the slow response time it achieved. I instead took another look at the PIC chip. I knew that it could be programmed to act as a control device, and it was certainly fast enough to act reliably. There were a few Internet web sites which described ways to program the PIC chip. This would still require me to learn some PIC programming, but it could still work as an inexpensive, yet efficient control device.

At the time I was investigating different PIC chip projects, I came across some work being conducted by Angelo Fraietta who was building a Smart Controller. He described it thus:

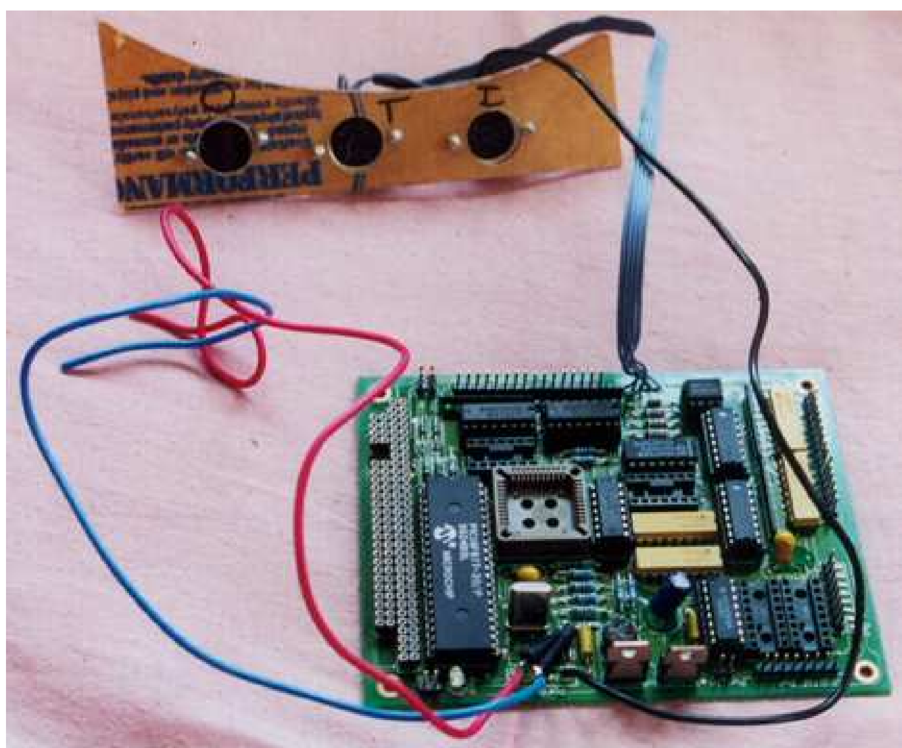

Figure 4.1: Circuit board of Dumb Controller designed by Angelo Fraietta.

> The Smart Controller is a portable hardware device that allows performers to create music using Programmable Logic Control. The device can be programmed remotely through the use of a patch editor or workbench which is an independent computer application that simulates and communicates with the hardware. The Smart Controller responds to input control voltage and MIDI messages, producing output control voltage and MIDI messages (depending upon the patch). The Smart Controller is a stand alone device – a powerful, reliable, and compact instrument – capable of reducing the number of electronic modules required, particularly the requirement of a laptop computer, in a live performance. [Fraietta, 2002]

The idea of the Smart Controller is really good. It hides most of the electronic equipment in one compact box, leaving the performer more open to focus their attention on the audience. I spoke to Fraietta about this project, but he told me the processor running the on the Smart controller was only a 486, and would not handle DSP very well, plus he had not programmed any DSP support into it yet, just MIDI. He also explained that the Smart Controller would not be ready until the end of 2002, which was much too late for my needs. In the building process of the Smart Controller though he developed a Dumb Controller. This consisted of just the input and output protocols. It could not perform any sub routines, or make changes to any data, it could just convert control data to MIDI messages, using a programmed PIC chip, in exactly the same manner as the I-Cube. Fraietta created an entire batch of the Dumb Controllers, and sold them at a very low rate.

The controller pictured in figure 4.1 on the preceding page, consisted of 16 analogue inputs, 8 digital inputs, 8 analogue outputs and 8 digital outputs. It has a MIDI in, MIDI thru, and 2 MIDI outs. It converts the inputs to MIDI data ranging from 0-127. The controller configuration can be easily programmed through MIDI SYSEX, which makes it very easy to customize.

After buying the Dumb Controller the first thing I did was to start to assess how efficiently and quickly it responded. I created a number of patches in *Max/MSP* to evaluate the performance and was very impressed with the results. Even when I was making constant changes on every input node simultaneously, it was still able to keep up with any noticeable latency.

Using MIDI as the control protocol allowed the interface to be more versatile. It meant that it could be used by many different kinds of equipment, and it could be understood by many existing pieces of musical software.

4.2 Building Hardware

Having decided the granular synthesis engine would run on a computer, and having also selected a controller mechanism to communicate with the computer, the next step involved creating some other kind of medium with which to perform using real-time granular synthesis. I needed to create a visually stimulating physical interface. I already had a couple of ideas in mind.

4.2.1 Fritz Lang

The first idea was that I would create a gigantic interface. Whilst walking around the Brisbane Powerhouse for the first time I noticed a massive old generator of some description which had been left in a walkway. Upon seeing this I started thinking about the 1920s movie *Metropolis*, and the large mechanical devices the workers had to operate. I imagined mounting large control mechanisms on the machine, big clunky levers to move up and down, ratchety winding controllers, and large buttons. Each control mechanism would be capable of generating a control voltage which would be sent to the Dumb Controller hiding behind the machine, where it would be converted to MIDI and sent to the computer also hiding behind the machine. The computer would process the sound based on the MIDI values and the settings of the the granular synthesis program, and would create a stream of audio which would be played throughout the room, resonating from the machine. I also thought of attaching some small surface microphones to the machine, so that I could create granulated sounds collected from the clanking of the machine itself. I still find this idea really fascinating, but I decided that as the instrument was also to be used as a research tool, it would be impractical to build a large immovable instrument, especially as it would need to be entirely assembled and later disassembled all within 10 days. With this in mind I set the new goal of mobility. I wanted to create an interface that could be used after the REV festival.

4.2.2 Poseidon

The second idea was to create a handheld interface about the size of an alto saxophone. If I could mount the Dumb Controller inside the interface it would mean I would only need to worry about two components, the computer and the interface, which would be attached via a long MIDI cord. I starting drawing designs of instrument shapes within which the Dumb Controller could fit inside. Knowing that there would be many children wanting to play with the instrument I decided that some kind of animal shape would be more appealing, rather than trying to design something totally artistic. Based on the shape required to accommodate the Dumb Controller it became apparent very quickly that a long thin creature shape would fit the best, so I decided to make it a sea creature, complete with googly eyes. I named the instrument the poseidon after the Greek god of the sea.

The design of poseidon so far was a sea creature with a Dumb Controller mounted inside its belly. Choosing the way in which the musician interfaced with the poseidon was the next step. I wanted to choose a more tactile kind of control mechanism, so that I would not have to look at it every time I made a change. There were a few different options available:

- Wheels
- Pressure sensors
- Sliders

Wheels are small, and easy to install, but I chose against them because they require the performer to constantly take their finger away in order to keep turning the wheel in a single direction. This would make it very easy to forget what location each parameter is set. The pressure sensors were abandoned for two reasons, firstly it is nearly impossible to keep the same pressure on an object, so it would produce constant fluctuating results, secondly they were very expensive. Sliders were not quite as easy to install, and they take up more room, but they are very cheap and the performer can keep their finger resting on each slider at all times, so they always know the location of the slider parameter.

To power the poseidon I had the choice of a battery or a power supply. Putting a

Figure 4.2: The electronic parts that went inside the poseidon.

battery inside the poseidon would make it too heavy, so I decided to add a tiny power supply inlet. The cable would run parallel with the MIDI cable. I also decided I was going to give the poseidon a microphone mouth, so I decided to add an internal microphone attached to a 2.5mm jack located next to the MIDI out and power supply inlet plugs. When I wanted to use microphone input as the sound source it would require adding another thin cable in parallel to the MIDI cable. I figured that if I already had one cable, it would not make much difference if I had a few cables bound together.

Now the design was complete it was time to build the poseidon.

Building poseidon based on the design was quite time consuming. I will go through the process in steps:

1. The Dumb Controller was just a circuit board, so first I needed to solder all the sliders and power source to the circuit board, pictured in figure 4.2 on the facing page.

2. Rigorous testing of the soldering and sliders, using *Max/MSP* ensured that nothing would break.

3. I hammered a 1KG Quik tin into shape. It needed to have flat sides, so that the sliders could be placed there. This was easily done with a block of wood and a hammer.

4. I then cut slots in the tin through which the sliders would protrude.

5. I gave the tin an undercoat of metal priming white paint. This would reduce the chance of rust in the future

6. Initially I was going to leave it with the slots cut out of the metal, but the edges felt too sharp, especially for little children with small hands, so I made the holes larger and then made smaller holes of the required size in a piece of ply wood which I sanded very smoothly and then glued onto the tin.

7. I screwed a hunk of rough wood on the bottom of the tin. This would act as an anchor later when the tail was attached.

Figure 4.3: The poseidon drying off.

8. The next step was to put the Dumb Controller inside the tin and screw all of the sliders into place. The fit was tight, but everything fitted in just perfectly.

9. Using a Corn Flakes box and sticky tape, I created a mould for the tail which I attached directly to the tin.

10. The mold had a small hole in it, into which I sprayed Bostik Foaming Gap Filler. The foam expanded, totally filling out the mold and surrounding the anchor I had attached to the tin.

11. Within 24 hours the foam had set hard, so I removed the mold from the outside and sanded the tail to make sure it was smooth everywhere.

12. I glued half a polystyrene ball to the lid of the tin, for a head.

13. I gave the interface a second coat of white undercoat, making sure to mask the sliders, as shown in figure 4.3 on the preceding page.

14. I painted the instrument a marble blue/green colour.

15. Lastly, I needed finger holds for each slider so I shaped hard setting plasticine to suit each finger, and glued them into place. The finished poseidon is pictured in figure 4.4 on the following page.

The other hardware component upon which the instrument relied was a computer. The type of computer I used was solely dictated by the software I chose. I had already decided that in keeping the performance visually stimulating I would keep the computer hidden away, so that the audience would not start thinking about the other pieces of hardware, they would just be interested in the control interface, and the sound that was being produced. Despite the computer being kept hidden away, it still would have been impractical to use a large powerful computer, or network of computers. Itagaki, Manning and Purvis suggested a live real-time granular synthesis process using 53-networked computers that could control nine voices at once [Itagaki et al., 1996]. Whilst this opens up a fantastic range of options, it poses many problems that would negate some primary objectives of this instrument. These include primarily cost, and computer visibility. The instrument was designed with only one voice, so in order to have multiple voices I would need more performers, each playing

Figure 4.4: The completed poseidon connected to a computer.

an instrument in a granular synthesis orchestra. The program ran on just a single computer of reasonable speed, which reduced costs a lot.

4.2.3 Control Styles

Two key elements in working with granular synthesis are the method chosen to extract the sound grains and the method by which to layer them. Hamman, Bowcott and many others have researched various mathematical and scientific algorithms that have been used for the layering process [Bowcott, 1998, Hamman, 1991]. I decided these methods were not useful for this instrument design, as they are highly automated, and do not require much in the way of real-time control. Initially, I examined and produced ways of incorporating mathematical algorithms in order to reduce the number of controllers needed, but eventually decided to incorporate a direct parameter control to the instrument interface. This was done to assist in the research element of the instrument design. I wanted to work from basics and then build from that. I have previously stated that sliders would be used to control the interface, but I had to decide how many to use. I decided to use 10 sliders and map these to the 10 parameters that that I thought were most integral to granular synthesis. These parameters were:

- Grain duration & random offset
- Grain density & random offset
- Grain frequency & random offset
- Grain amplitude & random offset
- Grain panning & random offset

This has now evolved slightly, and I am working on grain envelope shape control, sound source selection, and a more refined grain frequency control with a brass instrument based fingering style, which will be added to the interface as a digital button system, as well as the sliders.

4.3 Software Hacking

Whilst working on the hardware component, I was also busy working with the software. Originally when I decided I would use a computer as part of the instrument I chose to program in C. This seemed like the sensible choice. I know the language, and have already written a number of small audio applications such as *interleaver*, and some MIDI applications such as a text based sequencer. The reason I eventually decided not to do this was there are a wide range of programs already written and tested. I could make my own program more fine-tuned and specific towards my needs, but felt no need to reinvent the wheel for a single project. I instead looked at the software available, specifically looking for a piece of software which had MIDI mapping support, allowed me to create specified sound grains, and gave good sonic results. After quickly trialling a range of different granular synthesis programs, I limited the experimentation down to four programs. *Max/MSP*, *AudioMulch*, *Csound* and *jMusic*. All programs allow the user to define all essential parameters which I required, they all had MIDI integrated and the sound quality was a matter of testing.

4.3.1 The *AudioMulch* and *Max/MSP* Factor

I was very interested in using *AudioMulch* as I went to the same university as the creator, Ross Bencina. *AudioMulch* is a very simple program to use. It has a graphical interface and the musician just plugs components together. I knew this program had already been used successfully for real-time granular synthesis performance, so it was a good option. *Max/MSP* is also an easy program to use. It also uses a graphical interface for programming, and joining components together. It has many more functions than *AudioMulch* and is also much more flexible. Using *Max/MSP* I created a granular object which was mouse controlled and processed sounds which came from the microphone. The result was very good. In fact to this date I have not been able to get such good microphone results from any other granular synthesis program I have used. *Max/MSP* has some excellent functions making curve generation a breeze. I could design a curve and control all the variables for it in real-time, so I could use it as a constantly changing envelope shape if I wished. I also experimented with tools such as delay, which would come in very useful in a real-time environment.

I found that a Macintosh G3 with just 128MB RAM could cope with single delay line with a duration just over 10 minutes. The MIDI features where also excellent. As I mentioned earlier I used *Max/MSP* to test the MIDI output of the MIDI controller device I bought from Angelo Fraietta. *Max/MSP* also has a large group of granular synthesis tools already built by other musicians such as GIST. Both *AudioMulch* and *Max/MSP* had the features I was interested in working with, especially *Max/MSP*.

There are two reasons why I chose against using both of these programs. The first reason was I am used to a text programming environment, not a graphical environment. When using graphics I feel like I have much less control over the program. The second and more important reason was that neither *AudioMulch* nor *Max/MSP* are open source. *AudioMulch* is quite a cheap program, but it is still not open source.

4.3.2 *Csound*

The next program I started experimenting with was *Csound*. *Csound* is based on the *Music V* program, and is scripted using the *Csound* language. *Csound* itself is written in C. There are a number of different versions of *Csound* to choose from. I tried all the versions I could find and found that when it came to real-time sound rendering, *CsoundAV* by Gabriel Maldonado was the best and easiest version to use. The granular synthesis audio functions in *Csound* work quite well. I tested the different variations and decided to use the basic grain command. It allowed me to control the grain density, the pitch, the synchronicity, and the duration of each grain in real-time. Unfortunately, I could not change the grain contents or the envelope shape in real-time. I would have to stop the instrument, change some parameters and then restart the instrument again. I found no way around this. These options were both things that would be very useful in experimenting with granular synthesis. Another problem I came across was that when rendering granular synthesis in real-time I got noisy sonic results when I worked with sound samples or microphone input. Even when I was working with a very fast computer it made no major difference. I tried changing the buffer sizes to make it more efficient, but I could not avoid it. I would have to work with pre-generated sine waves and other similar kinds of audio signals. So *Csound* was not going to give me the kind of functionality I was looking for.

4.3.3 *jMusic*

jMusic is a Java based sound synthesis and compositional tool written by Andrew Brown and Andrew Sorensen [Brown & Sorensen, 2000a]. Being Java based it would allow me to develop and test the instrument on a number of different computer platforms. This kind of flexibility in a programming language does have trade-offs with performance and gives some problems with certain hardware, MIDI being of major concern, but nothing I could not create some work around for. The entire source code for *jMusic* is freely available from `http://jmusic.ci.qut.edu.au` hosted at sourceforge.

If I wanted to use *jMusic*, the first stage would be to learn Java, as I had never before used Java. I found a tutorial on the Internet which explained Java from a C programming point of view, and I worked from there until I had worked out how Java is structured. The next thing I did was start going through the *jMusic* tutorials written by Andrew Brown. The tutorials are designed to teach the user how to use Java whilst they are learning how to use *jMusic*.

jMusic had one drawback which the others did not, it was, and still is, heavily in the development stages. If I wanted granular synthesis it would be up to me to program the audio object in Java and make it a part of *jMusic*. Programming a granulating audio object would allow me to fine-tune it for my specific needs, but without having to worry about audio buffering, real-time functionality, and many other things as these are already part of *jMusic*. *jMusic* is also open source, covered by the GNU[1] General Public License (GPL) which my instrument also inherited making it open source. This allows others to develop it further and modify it as they like, which is something I wanted to encourage.

Because *Csound* already contained most of what I needed, I decided to keep working with *Csound* until I got the *jMusic* version running. I decided to make the *Csound* version as a prototype of the basic functions I wanted to achieve with *jMusic*. I would be able to use the prototype version for quality control and comparisons, plus it would allow me to start working on the hardware controller whilst I was still programming the *jMusic* audio object.

[1]GNU is a recursive acronym *GNU is Not Unix*, and is pronounced gah-new

4.3.4 Granular Instrument Test Prototype Code

The code for the prototype can be found in Appendix B on page 133. It contains a range of different options which can be turned on and off and changed around to test different effects and environments. I specifically included a lot of commenting in the code so that it would be self explanatory.

The prototype contains two sections. The orchestra and the score. The orchestra follows this basic algorithm:

1. Initialize *Csound*
2. Map MIDI functions to appropriate parameters.
3. Modify parameters to suite the required format
4. Pass parameters to the *Csound* grain function
5. Pan Audio
6. Send audio out a stereo channel.

The score section defines the other parameters such as the wave shapes, envelope shapes, and control shapes. The very last command tells the computer to start playing for a period of over 8 hours.

This prototype proved to be very useful in setting a benchmark to work towards in *jMusic*.

4.4 Software Algorithms

Having chosen to create a real-time granular synthesis audio object for *jMusic*, I had to create an algorithm to define how it would be programmed.

1. A sound source of some description is required. This could be a microphone input, sound file, or could be generated by some other synthesis method. Both the microphone and synthesized sound require processor power to create the original sound.

2. Segments of the sound source would be used to create the grains. If the synthesized sound is going to remain static then the synthesis process can be turned off after the first segment is captured. For microphone input, non-static synthesized sounds, and sound files this segmentation process must repeat itself many times, depending on the density of the texture that needs to be produced. Over 8000 segments may be required per second.

3. Each grain must be enveloped in order to minimise clicks and artifacts. The equation required to produce the envelope may involve random numbers or follow some set mathematical function in order to create each individual grain envelope. The envelope shape is a vital ingredient to the timbre of the grain.

4. Any final grain transformation process that needs performing should be done here.

5. The grains need to be mixed and added together so that they can be formed into a granular texture. There are many methods for organising how the grains will be added, from genetic and ecological algorithms, to fractal functions and user defined functions.

For a perfect rendition the output must come out at the very instant of input, but this is an impossibility, and so a suitable latency for the operation must be decided upon. If the operation time exceeds the latency period, then changes need to be made to make the code more efficient, or the parameters such as density, or the complexity of equations may need to be reduced or bounded. The other change that can be made is to reduce the amount of audio data that is being processed by using a smaller buffer. I found it most unsatisfactory when the latency period exceeded one quarter of a second.

The next sections describe simple algorithms used to produce the basic features of granular synthesis, using both synchronous and asynchronous granular synthesis with a random base. Once these basic features are working, then many other features can be experimented with to add variety and change to the sound, such as adding a glissando, or trill to each grain. There are of course many other ways to accomplish the same thing, for example, if creating sound textures from ecologically based equations, the grain density and distribution functions would be totally different.

4.4.1 Grain Length

Grain Length using *synchronous* granular synthesis:

1. Get the length of the grain from the user in milliseconds.
2. Convert the millisecond value to a number of samples by multiplying the time by the sample rate and then dividing by 1000.

This process only needs to be done once, as the same value is repeated. It should be stored for continual retrieval.

Grain Length using *asynchronous* granular synthesis:

1. Get the length of the grain from the user in milliseconds.
2. Get random offset of the grain length from the user in milliseconds.
3. Convert both millisecond values to number of samples by multiplying the time by the sample rate and then dividing by 1000.
4. Subtract half of the random offset from the grain length.
5. Create a random number between 0 and the random offset value, and add the new value to the grain length.

This process is repeated for every single grain.

In both synchronous and asynchronous granular synthesis the number of samples can now be applied directly to the sound file. The number of samples is the number of samples that will be copied from the sound file into the temporary buffer where the grain will be created. It is also the length of the envelope that will be placed over the grain.

4.4.2 Grain Envelope

1. Set up a loop with iterations to the final value of the current grain duration.
2. Divide the iteration number by the grain duration to get a value between 0 and 1.

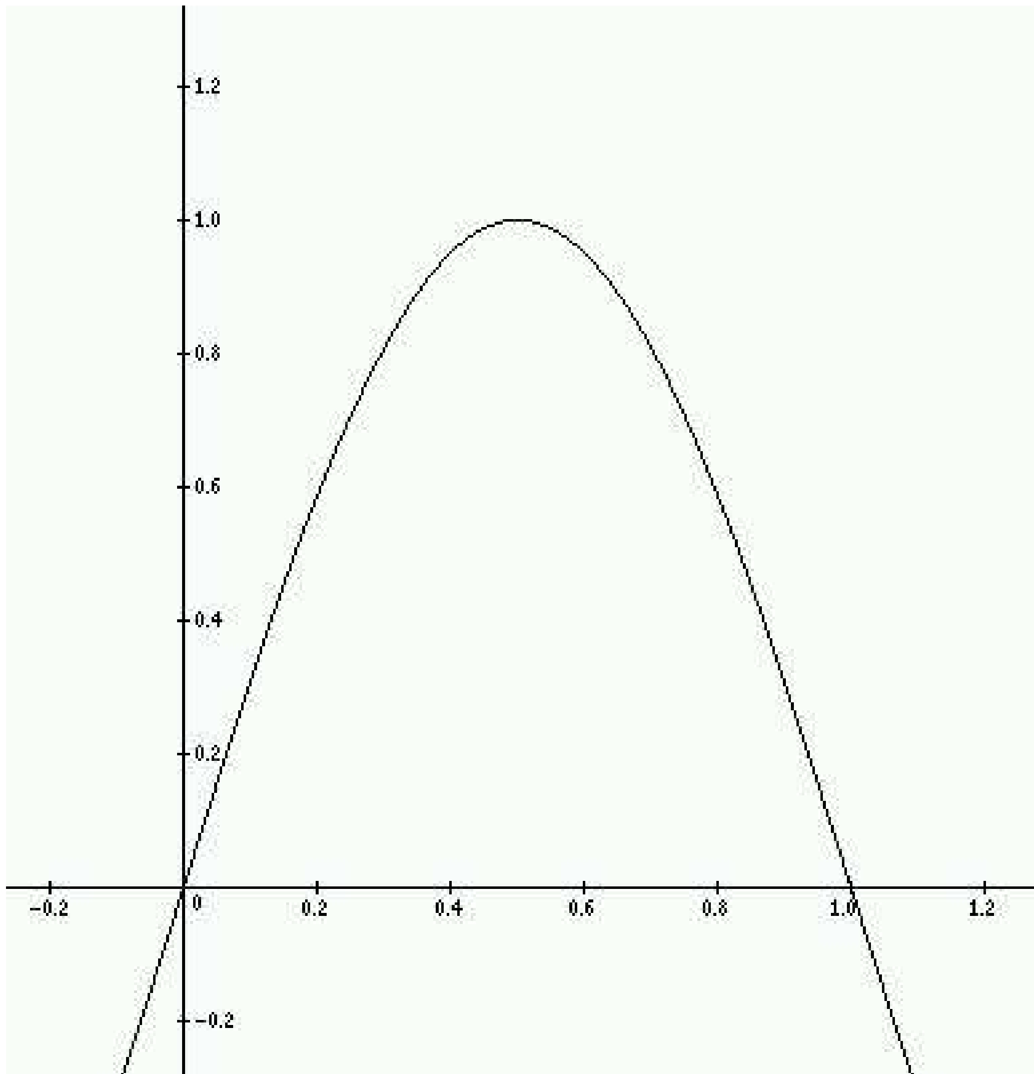

Figure 4.5: Half a sine wave.

3. Multiply this number by a mathematical function that defines the desired envelope. For example, a simple half sine shaped envelope can be obtained by multiplying the value by π, and then getting the sine of the value. In brief: $\sin(\pi \times \frac{currentiteration}{grainduration})$.

4. Multiply the resulting number by the value of the corresponding sound sample.

The equation will create an envelope as depicted in figure 4.5 on the facing page, with a curved shape passing through these points: f(0)=0, f(0.5)=1, and f(1)=0. So between the x factor of 0 and 1, a half sine wave starting at 0, rising to 1, and falling symmetrically back to 0 again is created.

4.4.3 Grain Density

Grain density using synchronous granular synthesis:

1. Get the grain density value from the user. This will be a number to indicate the number of grains to play per second.

2. Divide the sample rate by the grain density.

This process only needs to be performed once, with the value being stored for continual retrieval.

Grain density using asynchronous granular synthesis:

1. Get the grain density value from the user. This will be a number to indicate the number of grains to play per second.

2. Get the random offset value of the grain density from the user.

3. Subtract half of the random grain density offset value from the grain density.

4. Create a random number between 0 and the random offset value, and add the new value to the grain length.

This process needs to be repeated for every single grain.

In both cases the process will yield a value that indicates how many samples must pass through before the successive grain is created and added to the sonic texture.

4.4.4 Grain distribution

1. Set up buffer with a pointer that is used to nominate which sample is being processed.
2. Set up loop with iterations to the value of the current grain duration.
3. Go through loop adding the value of each sample of the enveloped grain to the buffer.
4. Move the buffer pointer forward from the starting point of the recent grain, by adding the current grain density value.
5. Repeat steps 2 to 4 until all grains are added, or buffer is full, in which case pass the buffer out an audio channel and go back to step 1.

4.5 Programming with *jMusic*

These next sections will go through the code used to create the real-time granular synthesis instrument.

4.5.1 Spurt

One very good example I came across in the tutorials was a program called Spray [Brown & Sorensen, 2000a, Brown & Sorensen, 2000b]. This tutorial shows how an audio file can be segmented, and the small segments distributed across a stereo soundstage. The way this program works is that it uses the musical data statement in *jMusic* to define a musical score. The main difference though is is that rather than assigning a crotchet, or some other musical note, each note is just defined by a small duration, the number in this case being 0.150 (150ms). So theoretically this note is far too long to be used in granular synthesis. It still produces a granular texture, and sounds like many small sound samples played one after the other.

The method that this program uses to create the audio is comparable to the way Roads created the score for Klang-1, except that assigning notes in *jMusic* is a very easy process compared to punching over 700 cards [Roads, 2001].

I took this program and started writing variations of it, to try out many different experiments. The following is one of these variations of Spray, called Spurt. It

was designed with just the basic elements needed for granular synthesis. It uses the same concept of building up a score using the musical notation function of *jMusic*, but instead the grain size has been changed to a random duration of between 20ms to 50ms. The sine wave panning feature has been reduced to a simple random number function. The use of a sampled sound has been discarded in favour of synthesising a sound file using VibesInst, which utilizes the *jMusic* oscillator function. This makes the program a lot more portable, as there is no need to worry about having an original sound file. The pitch range was increased so that it covered three octaves from C3 to C6. One final major change was an experiment with overlapping grains. The overlapping function is random so that it produces an asynchronous granular texture. I set it up so that it would overlap up to 7 grains at any one time. It is random between 0 and 7 overlapping grains at any single place in time. The following example shows the main class within spurt.java. The full code can be found in Appendix C on page 139.

```
public Spurt() {
    Score score = new Score();
    Part p = new Part("Spurt", 0, 0);
    Phrase phr = new Phrase();
    // Set up an instrument
    Instrument inst = new VibesInst(44100);
    // iterate through many notes
    for(int i=0; i<5000; i++) {
        // set duration of note
        double dur = 0.02 + (Math.random() * 0.03);
        // create a note pitched between C3 + C6,
        // asyn duration with MAX 7 overlap, random amp.
        Note n = new Note(C3 + (int)(Math.random() * 37),
                  dur*(Math.random() + 0.15),
                  (int)(Math.random() * 80 + 47));
        n.setPan(Math.random());
        n.setDuration(dur);
        // add the note to the phrase
```

```
            phr.addNote(n);
        }
        // pack the phrase into the jMusic score structure
        p.addPhrase(phr);
        score.addPart(p);
        // render the score as an audio file
        Write.au(score, "TestSprurt.au", inst);
    }
```

This program Spurt.java produces granules of sound and layers them in a random fashion. When I first started experimenting with this program I was disappointed in how long it took to produce the sound files. For example I tried to create a 40 second sound file which had a grain density of about 2500 grains per second, so it had to calculate and layer 100,000 grains within the 40 second sound file. It took more than four hours to complete the rendering process. Waiting more than four hours for forty seconds of audio was a depressingly slow result. There was no way this particular program could be converted for real-time granular synthesis. After checking the code to see where the speed could be improved I decided to remove the score viewing function. This function is excellent for viewing the grain distribution, but for just creating audio files there is just no need. When the same file was re-rendered it took only about 15 minutes to complete, so I realised that graphics were the real bottleneck in java, not audio. I kept experimenting further with this program to try and make it faster. When using a much lower grain density I found I could reduce the file rendering time to about 6 times the length of the sound file. So a one minute audio file would take about 6 minutes to render. I was still unhappy with this speed result, but decided that if I could create my own audio object, and skip the entire note and score generation process that it would save a lot more time. I decided that even though this particular program was very slow I would still use it to test different algorithms and control ideas.

Another variation on Spurt.java used a Q/Spread control algorithm[2]. This was achieved by setting a mathematical equation to give a value for Q. A number of dif-

[2]The Q/Spread control algorithm refers to a mean and standard deviation form of distribution, where the mean, or centre point is referred to as Q, and the standard deviation is referred to as the spread, a symmetrical outer boundary surrounding the centre point.

ferent equations ranging from straight lines through to exponential curves and sine waves were used to control the parameters. I then tried setting mathematical equations to control the spread, or the deviation from Q. These equations were similar to those controlling Q. A random number would be generated within the parameters set by Q/Spread. The Q/Spread control algorithm was used to control all of the parameters in the program, the grain duration, density, amplitude, panning, and pitch.

Andrew Brown made his own variant of Spray which he used to create a quasi-granular texture for an installation he created for REV called *InterMelb Sonic City*, which featured a map of the city of Melbourne, and a character who could be guided through the city. The audio had been sampled in Melbourne and was broken into different kinds of sounds. depending on which part of Melbourne the character had been guided to, it triggered different combinations of audio samples to be granulated together into a city texture [Brown, 2002].

4.5.2 Granulator - The Audio Object

I started by looking at different ways of creating grains of sound in *jMusic*. One very simple method was to create notes using the score composition method within *jMusic*. This gave a good audio result, but the computation involved in the actual score creation was too intense and was not suitable for a real-time instrument that required more speed. I then started looking at creating a Java audio object to integrate into *jMusic*. Andrew Sorensen helped in creating a base, which I modified to suit the granular synthesis process and the instrument interface. The granulator is the main granular synthesis processing engine in the entire process, but it is only a part of the whole instrument. The granular synthesis instrument itself is comprised of four separate classes; Granulator.java, GranularInst.java, RTGrainLine, and RTGrainMaker.java. These are all integrated to act as one process. I will go through each of the four parts separately, starting with Granulator.java.

The granulator was designed with a very broad base, so that it could be adapted to many different uses of granular synthesis. The basic algorithm behind the granulator audio object is as follows:

1. Get control variables.

2. Load in a buffer of audio data.
3. Extract chunks of sound according to the grain duration parameters.
4. Make any changes to the chunk of sound that may be defined by the parameters.
5. Envelope the chunk to form a grain.
6. Place the grain in a new buffer according to the parameters given.
7. When the new buffer is full pass the new buffer out.
8. Repeat process indefinitely, or until stopped.

This process is very simple, and it is easy to apply many different variations to it without having to actually change the granulator program itself. For example the granulator does not care about where the buffer of audio comes from, how it was created, or even what size it is. It will just process all buffers as if they are raw audio data, and then it will pass out a buffer of equal size when it has finished processing the buffer it was given. So the buffer could just consist of audio from an audio file stored on the computer hard drive, it could be audio generated through one of the many instruments and oscillators contained within *jMusic*, it could be from a microphone, or it could be audio that has already been manipulated through any other process within *jMusic*, such as re-sampled data, or filtered data. This has been left very open, so that other add-ons can be made more easily without having to change the granulator code itself. It is up to the files passing this information into the granulator to make sure that the audio is in the correct format.

The granulator also accepts a large number of different parameters which can either be passed to it during the initialisation of each buffer, or can be passed to it directly during the buffer processing stage. Having a lot of parameters may cause a problem when trying to write music with the program, or other add-ons to the granulator, but this has been compensated for by having default values for all settings. This means that if the user does not include all of the parameters, or if they forget some of the parameters then the default parameters will be used, although most of the default parameters are actually controlled by GranularInst.

The main past of the granulator code is contained below. The first section shown sets up the variables to be used in granulator. All of the variables have been labeled. To view the full code for granulator.java refer to Appendix D on page 141.

```
public final class Granulator extends AudioObject{
//------------------------------------------------
// Attributes
//------------------------------------------------
   private int grainDuration; // grain duration
   private int cgd; // counter for grain duration
   private float[] grain; // grain buffer
   private float[] newbuf; // output buffer
   private int grainsPerSecond = 50; // grains per second
   private float[] tailBuf; // overlap buffer
   private double freqMod = 1.0; // pitch control
   private float[] inBuffer = null; // temporary buffer
   private boolean inBufActive = false; // states whether the
                                  //temporary buffer is needed
   private boolean ri = false; //random indexing
   private boolean rgd = false; //random grain duration
   private int gdb = 1000; //the smallest rnd grainduration
   private int gdt = 1000; //the highest rnd grain duration + gdb
   private boolean rf = false; //random frequency
   private double rfb = 0.8; //the lowest rnd frequency
   private double rft = 0.5; //the highest rnd frequency + rfb
```

The constructor defines how this class is called from the above classes. It includes some of the default settings. Granulator is called by defining three parameters. The first parameter is the buffer of audio data, the second parameter is the initial value of the grain duration, and the third parameter is the number of grains per second.

```
//------------------------------------------------
// Constructors
//------------------------------------------------
```

```
public Granulator(AudioObject ao, int duration, int gps){
   super(ao, "[Granulator]");
   this.grainDuration = duration;
   this.grainsPerSecond = gps;
   this.cgd = 0;
   this.grain = new float[this.grainDuration];
   tailBuf = new float[0];
}
```

This section is the heart of the granulator. It works out how many grains it needs to create, and how far apart to space each of the grains. It then calls the setGrain method in order to retrieve each grain. It then places the grain within the buffer and returns the buffer which can either be written to an audio file, or played directly out of the loud speaker.

```
/**
 * @param buffer The sample buffer.
 * @return The number of samples processed.
 */
public int work(float[] buffer)throws AOException{
   if(inBuffer == null){
      newbuf = new float[buffer.length];
      this.previous[0].nextWork(newbuf);
   }else{
      newbuf = new float[buffer.length];
      for(int i=0;(i<inBuffer.length)&&(i<newbuf.length);i++){
         newbuf[i] = inBuffer[i];
      }
      inBuffer = null;
   }
   //number of grains to fit in buffer
   int nog = (int)((float)newbuf.length /
            ((float)(sampleRate*channels) /
```

```
                (float)grainsPerSecond));
        //time between grains
        int tbg=(newbuf.length/nog);
        //add any grain tails
        for(int i=0;(i<buffer.length)&&(i<tailBuf.length);i++){
            buffer[i]+=tailBuf[i];
        }
        tailBuf = new float[newbuf.length];
        inBufActive = true;
        //add all new grains
        for(int i=0;i<nog;i++){
            int index = i*tbg;
            setGrain(index);
            for(int j=0;j<grain.length;j++){
                if(index >= buffer.length){
                    tailBuf[index-buffer.length]+=grain[j];
                }else{
                    buffer[index] += grain[j];
                }
                index++;
            }
        }
        inBufActive = false;
        return buffer.length;
    }
```

The next ten methods are all parameters that can be set or changed at anytime by calling these set methods from outside this class. These methods are what make real-time granular synthesis work. You can just turn the granulator on and then control the sound entirely by using the set methods. Just one example of the set method is listed here, as the other nine are all very similar.

```
    // Deviation from the input frequency
```

```
    public void setFreqMod(float fmod){
        this.freqMod = fmod;
    }
```

This next section contains the code used to create each individual grain. It is called from the above work method which is where the texture is created. So this method is called for every single grain within the texture. This method has to work out the exact number of samples required to create a grain of the specified length. It also needs to determine exactly where to retrieve the selection of samples from. Once this has been determined it copies that section to a small buffer, which then gets enveloped and sent to where the method was called from. This method also takes care of pitch shifting if a pitch shift has been designated. The algorithm for pitch shifting was designed with a trade-off of quality in favour of speed of execution. If the sample needs to be raised in pitch, the algorithm simply determines how many less samples will produce a frequency at that pitch and removes them at a constant rate. For lowering of a sample pitch, it determines how many more samples are needed, and some samples are repeated in a similar manner.

```
    //---------------------------------------
    // Private Methods
    //----------------------------------------
    /**
     * Set the grain
     */
    private void setGrain(int index) throws AOException{
        if(ri) index=((int)(Math.random()*(double)newbuf.length));
        float[] buf = newbuf; //reference to the active buffer
        if(rgd)this.cgd = gdb+(int)(Math.random()*gdt);
        else this.cgd = this.grainDuration;
        double cfm = this.freqMod;
        if(rf) cfm = this.rfb+(Math.random()*(this.rft-this.rfb));
        if(inBufActive){
            inBuffer = new float[newbuf.length];
```

```
            int ret = this.previous[0].nextWork(inBuffer);
            inBufActive=false;
        }
        this.grain = new float[cgd];
        int count = 0;
        float tmp = 0.0f;
        //pos. values of skip are the iterations to skip
        //neg. values of skip are the iterations to add between
        double skip = -1.0/((1-cfm)/cfm);
        double remains = 0.0;
        int upSample = 0;
        if(skip<0){skip=-1.0/skip; upSample=1;}
        if(skip==0)upSample=2;
        int ind=0;
        for(int i=index;true;i++){
            if(i==buf.length){i=0;buf=inBuffer;}
            if(upSample==0){//remove samples (up sample);
                if(++ind>=(int)(skip+remains)){
                    remains=(skip+remains)%1.0;
                    ind=0;
                    continue;
                }
                if(count >= cgd)break;
                grain[count++]=buf[i];
            }else if(upSample==1){//add samples (downsample)
                if((skip+remains)>=1.0){
                    float p=(tmp-buf[i])/((int)skip+1);
                    for(int k=0;k<(int)(skip+remains);k++){
                        grain[count++]=p*k+buf[i];
                        if(count==cgd)break;
                    }
                }
```

```
                if(count==cgd)break;
                grain[count++]=buf[i];
                tmp=buf[i];
                remains = (skip+remains)%1.0;
            }else{ //no resample ;)
                grain[count++]=buf[i];
            }
            if(count==cgd)break;
        }
```

This final section of the Granulator is where the envelope is created. This is created by using the positive section of a sine wave as an envelope. Once the envelope has been placed over the grain contents it is ready to be added back to the main buffer where it becomes a part of the granular synthesis texture.

```
        //Envelope our new grain
        for(int i=0;i<cgd;i++){
            this.grain[i] = this.grain[i]*
                                (float)(Math.sin(Math.PI*i/cgd));
        }
    }
```

Despite testing a number of different envelope shapes in the Csound code, I had at this stage just implemented the most basic envelope shape in the jMusic version. This decision was based primarily on efficiency, and to keep the code small whilst I fine tuned other areas. Since adding the above code, I have already implemented three envelope types, which the musician may swap between in real-time by moving a slider, as well as the half sine wave envelope, there is now a full cosine wave shaped envelope, and a pyramid envelope. I still prefer the effect of the half sine wave envelope.

4.5.3 GranularInstRT

Once the granulator audio object was created it was then necessary to create an interface for it, so that I did not have to access the granulator manually. This involved set-

ting the granulator up as an instrument, hence the name GranularInst.java. GranularInst contains all of the parameters that a user would need to access in order to control the granulator. The main point of GranularInst is to set up the data in the format required for the granulator. As was stated in the last section, the granulator requires a buffer of audio data to be passed to it, and does not even check to see what size it is, or if it really is audio data. GranularInst on the other hand checks the audio file or other sound source, and then sets up the buffer ready to send the information to the granulator, which is then sent. GranularInst lets you choose from a range of different sound sources. The user can select to use either an audio file, a microphone input, or a number of different internally generated sound sources such a sine waves, triangle waves, square waves etc.

GranularInst requires just one parameter. It needs to know what sound source is desired. The sound source can be chosen according to the number represented by each sound source type. If the user decides to use an audio file there is a preset filename which the program seeks and loads. If the user wants to use a different audio file, instead of selecting the number corresponding to audio files, they can just enter the name of the audio file. When GranularInst receives a file name it will automatically use the audio file mode. It will seek the defined audio file and load it into the granulator.

After the GranularInst has been set up the user can then use the set methods to make alterations to the parameters listed in GranularInst. In this instance the only change required is the grain duration. The user would make a call to setGrainDuration(gdur), where gdur is an integer represented by milliseconds. So if the user wants to change the grain duration to 33 milliseconds they just send a 33 to setGrainDuration. If they then want to change the duration to 41 milliseconds then they just send a 41 to setGrainDuration. If the grain duration changes follow some pattern then it is very simple to set up an equation to control the grain duration, and just keep updating setGrainDuration.

The algorithm for this class is very simple. It goes as follows:

1. Find out the sound source type.

2. Depending on the sound source type make the appropriate call to the granula-

tor.

3. send any parameters changes to the granulator.

The GranularInst will just wait patiently for any parameter changes. It will run as long as the granulator needs to run.

Following below is the main part of the source code for GranularInstRT.java. The full source code can be found in Appendix E on page 149.

This first section is where all of the variables are initialized. All of the variables are self explanatory.

```
public final class GranularInstRT extends jm.audio.Instrument{
   //-----------------------------------------------
   // Attributes
   //-----------------------------------------------
   // the name of the sample file
   private String fileName;
   // How many channels is the sound file we are using
   private int numOfChannels;
   // the base frequency of the sample file to be read in
   private double baseFreq;
   // should we play the whole
   // file or just what we need for note duration
   private boolean wholeFile;
   // The points to use in the construction of Envelopes
   private EnvPoint[] pointArray = new EnvPoint[10]
   private Granulator grain;
   private Volume vol;
   private StereoPan pan;
   // used to define the audio input type
   private int sounds;
```

This section defines how GranularInst can be called when the user wants to use it. There are two methods for calling GranularInst. The first one involves typing in the

file name of an audio file. GranularInst will detect that the input is a string and will set up the other variables to the default for an audio file. The other method for calling GranularInst is using an integer. The code below outlines what each integer represents. Note that when using this constructor you can still call an audio file, but the audio file used will be the default “song1.au”.

```
//----------------------------------------------
// Constructor
//----------------------------------------------
public GranularInstRT(String fileName){
   // Use constructor when you want to granulate an audio file.
   // Only the name of the audio file is required
   this.fileName = fileName;
   this.numOfChannels = 2;
   this.sounds = 7;
}
public GranularInstRT(int sounds ){
   /**
    * The variable sounds is an integer used to select
    * which sound source type will be used.
    * It will be defined as such:
    * SINE WAVE = 0
    * COSINE WAVE = 1
    * TRIANGLE WAVE = 2
    * SQUARE WAVE = 3
    * SAWTOOTH WAVE = 4
    * SAWDOWN WAVE = 5
    * SABERSAW WAVE = 6
    * AUDIO FILE = 7
    * MICROPHONE = 11
    *
    * Use this constructor when you want to granulate internally
    * produced audio. Note: you can still granulate audio files
```

```
         * if you use this constructor, but it will assume the audio
         * file has the name song1.au.
         */
        this.sounds = sounds;
        this.numOfChannels = 2;
        this.fileName = "song1.au";
    }
```

This next section starts by determining what kind of sound source has been selected. This works by sifting through all of the possibilities. First, it determines whether it is an internally created sound or whether it is an audio file or microphone input source. This is done by just filtering out all of the internal oscillator operations. Then, whatever is left over will be either microphone or an audio file. It has been intentionally left very open. This has been done for a number for reasons. The first is that it will be easier to expand in the future if other internal sound sources are added. Also, it means that the user does not have to be as careful if they want to just randomly call any sound source type. This is particularly useful if the sound source is called with a slider which may not be very accurate. The user could simply set up a range from -10 to 20 so that the top end is for the microphone, the bottom end is for audio files, and there are a number of oscillator sources in the middle.

The oscillator options are filtered out first because they can only be certain numbers. The oscillator is called by sending an integer which refers to the wave type. They correspond directly to the integers given in GranularInst. If someone tries to call the oscillator with a number not in the oscillator range then it will create an error, this way it will only let valid oscillator numbers go through, and as the range of oscillator options increases the range can be increased. All other integer inputs are filtered to either call the microphone or the audio file.

After the sound source has been set up, it is then passed to the granulator. Once passed to the granulator, the GranularInst sets up default panning and volume attributes. These can also be overwritten at any stage. They are just generic *jMusic* functions which have nothing to do with the Granulator.

```
    //---------------------------------------------
```

```
    // Methods
    //---------------------------------------------
    /**
     * Create the Audio Chain for this Instrument
     * and assign the primary Audio Object(s). The
     * primary audio object(s) are the one or more
     * objects which head up the chain(s)
     */
    public void createChain(){
       // define the chain
       if (sounds<0 || sounds>6){
          if (sounds>10){
             // if sounds is > 10 then the microphone
             // is the input source.
             //Default is 11, but this way it doesn't matter
             // if a wrong number gets inputed.
             // (8820 = buffer length of 1/5 of a second)
             RTIn grin = new RTIn(this,44100,2,8820);
             grain = new Granulator(grin,50,100);
             vol = new Volume(grain,0.5f);
             Volume vol2 = new Volume(vol,0.1f);
             pan = new StereoPan(vol2);
          } else {
             // if sounds is < 0 or > 6 and < 11 then it will
             // process an audio file. Default is 7.  Again it is
             // very open ended to accommodate wrong input numbers.
             SampleIn grin = new SampleIn(this, this.fileName);
             grain = new Granulator(grin,50,100);
             vol = new Volume(grain,0.5f);
             Volume vol2 = new Volume(vol,0.1f);
             pan = new StereoPan(vol2);
          }
```

```
        } else {
            // At this stage the only values left are between 0-6
            // These correspond directly to the oscillator input
            // values, so can be added directly.
            Oscillator grin = new Oscillator(this, sounds, 44100,2);
            grain = new Granulator(grin,50,100);
            vol = new Volume(grain,0.5f);
            Volume vol2 = new Volume(vol,0.1f);
            pan = new StereoPan(vol2);
        }
    }
```

This last section contains all of the set functions that can be used to change the Granulator variables at any time. GranularInst is used to set the variables which the Granulator receives and interprets. There ten set methods, but only one is listed here.

```
    public void setGrainsPerSecond(int sp){
        grain.setGrainsPerSecond(sp);
    }
}
```

This class is the class that the users call without ever having to refer to Granulator directly. If any core parts of Granulator are changed then the changes will be reflected in GranularInst. This makes the Granulator much easier to control without having to worry about the exact code of the granulator. Of course understanding the code for the audio object will help in using it, but it is not necessary.

4.5.4 RTGrainLine

GranularInst and Granulator are the core classes needed to create granular synthesis, but they need to be called from another file in order to run. GranularInst and Granulator are both now integrated into *jMusic*, and are part of the default *jMusic* download. These files should not be modified by general musicians if they want to keep *jMusic* in a pure form. They can of course experiment with the core files, and

if they make improvements they are encouraged to submit the additions to the developers so that they can be integrated into the core *jMusic* package. For musicians just wanting to make music with *jMusic*, and not develop the software, the Granulator and GranularInst are made and ready to use. The job of the musician then is to create the class that calls GranularInst, which in turn calls the Granulator.

There are two options to look at when creating a class that uses the GranularInstRT. They come down simply to the function you want to perform with granular synthesis. If the musician wants to compose a piece of music using *jMusic*, and they do not want or do not need the audio to be played in real-time as it is being created then they just need to create one file with a main class, which has step by step instructions telling GranularInst what to do. If the musician wants to work with granular synthesis in real-time, then there is an extra step. They need to set up a class to control the granular synthesis piece in real-time. They also need to create a file with a main class like the non real-time musician, but the main class will be more of an initialisation process rather than a score.

I had already decided long ago to use a MIDI control device to work with granular synthesis, so that meant I had to write two more files in order to start creating music. The two files were called RTGrainLine and RTGrainMaker. RTGrainLine contains the control parameters, and RTGrainMaker contains the main class and initialisation process.

The reason I have named the control class RTGrainLine is because it is used to communicate directly with the class called RTLine, which is the class *jMusic* uses to pass all real-time data through. In this case it is information about grains of sonic data, hence RTGrainLine. The function of this class is to set up all of the real-time control features, and to tell GranularInst when to actually start running, and what to do. RTGrainLine waits for control information to be passed to it. When it receives information it first determines where the information needs to be sent, and it then converts the information into a suitable format for that purpose. For example, I have set up a number of sliders to send control messages which are used to control a number of different granular synthesis parameters. One of the sliders controls the grain duration. If I have started the program running and I then move the slider, RTGrainLine will immediately be told that there is some information coming from

the grain duration slider. It will find the function relating to the grain duration and put it in the format required by GranularInst. In this case it converts the information coming from the slider into integer format, after which it then converts the straight integer value into an integer representing the number of milliseconds. Now that the value has been converted into an integer denoting the number of milliseconds for the grain duration it is then passed to GranularInst where the change can be applied, and the grain duration will now change accordingly.

The algorithm for this class goes like this:

1. Await information from the main class.
2. When information is received go to the corresponding function.
3. Make format changes as dictated by the function.
4. Send the new value to the GranularInst.
5. Go back to step one.

Following below are the main parts of RTGrainLine.java. The full source code can be found in Appendix F on page 155.

The first section contains all the variable definitions. Most of these should be self explanatory. The last three variables are all flags which assign certain characteristics to the Granulator. The first flag, durationRndFlag, determines whether the duration of each grain should be set randomly. The second flag, freqRndFlag, determines whether the frequency of each grain will be determined randomly. The last flag, asyn, determines whether the grain texture is synchronous or asynchronous. All of these flags can be changed at any point.

```
public class RTGrainLine extends RTLine {
   private Note n = new Note(72, 3000.0);
   private int pitch = 72;
   private int dynamic = 100;
   private GranularInst[] inst;
   private boolean durationRndFlag = false;
   private boolean freqRndFlag = false;
   private boolean asyn = false;
```

The constructor defines how this class is called. RTGrainMaker calls this class by providing three pieces of information. The first piece of information is the array of instruments that are to be used. In this case this is actually just one instrument, GranularInst, but I am not going to hard set that in case I decide to add another instrument at some future time. The second and third parameter are both used in conjunction to set how efficiently the real-time process runs. If the musician is experiencing a lot of lag then this can be improved by either increasing the control rate or decreasing the buffer size. If the musician is experiencing no lag then the audio quality can be improved by either reducing the control rate or increasing the buffer size. The trick is to find the perfect balance of buffer size and control rate. Unfortunately, these figures cannot be predetermined because the values tend to vary a lot depending on what kind of computer is being used, and what operating system it is running under.

```
/** Constructor */
public RTGrainLine(Instrument[]
      instArray, double controlRate, int bufferSize) {
   super(instArray, controlRate, bufferSize);
   inst = new GranularInst[instArray.length];
   for (int i=0; i<instArray.length; i++) {
      inst[i] = (GranularInst)instArray[i];
   }
}
```

The rest of this class is devoted entirely to routing the control variables so that the instrument functions in real-time. The operation of these functions have already been explained above, so I will not go through each one. They are used to capture the control changes, convert them to the correct format and then pass them on further where the control changes will be put into effect. Just the first control function has been listed. The other nine are very similar, just referring to different parameters each with corresponding value configurations, most of which were explained in the algorithm section.

```
public synchronized Note getNote(){
```

```
            n.setPitch(pitch);
            n.setDynamic(dynamic);
            n.setDuration(n.getRhythmValue());
            return n;
        }

        // added for control change
        public synchronized void
                externalAction(Object obj, int actionNumber){
            if(actionNumber == 1){ // grain duration
                Integer value = (Integer) obj;
                int duration = (value.intValue()) * (44100/1000);
                for(int i=0;i<inst.length;i++){
                    inst[i].setGrainDuration(duration);
                }
            }
```

Now that the control changes have a mechanism set up to take care of them there is just one last step to take care of. That is the main class.

4.5.5 RTGrainMaker

RTGrainMaker is the class from which everything else hangs. This is the main class, the class which initialises everything and puts the whole process into motion. This class contains a few new parts which have not been looked at yet. The most notable is Midishare, which was developed by the Computer Music Research Laboratory of Grame [Computer Music Research Laboratory of Grame, 2001]. Midishare is a program which is used to send and receive MIDI information through the MIDI port. Midishare can also be configured to route MIDI data through the computer. The benefit of this is that it can be routed to Java using Midishare's own Java classes. I found out very early on when trying to work with MIDI that despite its long standing standard framework it still has implementation problems on the computer, especially when used with Java. I tried using a number of different MIDI applications that were meant to work with Java, but it was a very hard process with little success.

The reason Midishare was used was because it was quite simply the first program I actually got running at a decent standard. The reason for my success can be contributed directly to Rene Wooller who spent a lot of time testing the Midishare Java code, discovering the purpose of all the undocumented Java classes. Rene gave me a test code he produced which showed implementations of many of the Midishare functions. So with some experimentation I got Midishare running, although it still proves a problem as it sometimes locks up after stopping, such that the computer needs to be restarted before using Midishare again. This of course is a very annoying glitch, but I learned to live with it. I am happy to say that recently I finally got to try out the new version of Midishare for Macintosh OSX 10.2 and was very pleased at how much better it is running on that platform now.

This algorithm for this class goes as follows:

1. Set up MIDI
2. Set up instrument
3. Initialise real-time process
4. Start the process running
5. Catch MIDI messages and send them to RTGrainLine to be processed.

This class is very simple, with only a few functions. The full source code for RTGrainMaker.java can be found in Appendix G on page 161.

```
public final class RTGrainMaker extends MidiAppl implements JMC {
   private RTMixer mixer;
   private RTGrainLine grain;
   private int sampleRate = 44100;
   public static void main(String[] args){
      new RTGrainMaker();
   }
   private GranularInst GranInst;
   public RTGrainMaker(){
```

```
    // setting up midishare
    // open the MIDI application
    try { this.Open("myApp");
    } catch(MidiException me) {
       System.out.println("Could not open");
    }
    // System.out.println("Success!");
    Midi.Connect(this.refnum, 0, 1);
    Midi.Connect(0, this.refnum, 1);
    // end midi share

    int channels = 2;
    //NOTE: control rate must be
    //LARGER than grainDuration/1000
    double controlRate = 0.1;
    int bufferSize=(int)((sampleRate*channels)*controlRate);
    Instrument[] instArray = new Instrument[1];
    for(int i=0;i<instArray.length;i++){
       instArray[i] = new GranularInst(7);
    }
    GranInst = (GranularInst) instArray[0];
    grain = new RTGrainLine(instArray,controlRate,bufferSize);
    RTLine[] lineArray = {grain};
    mixer = new RTMixer(lineArray, bufferSize,
                        sampleRate, channels, controlRate);
    mixer.begin();
 }

 public void ReceiveAlarm(int event) {
    //System.out.println("MIDI mesg = " + event);
    switch (Midi.GetType(event)) {
       // MIDI in controller messages
```

```
            case Midi.typeCtrlChange: {
               int controllerType = Midi.GetData0(event);
               int value = Midi.GetData1(event);;
               if (controllerType == 12)
                  mixer.actionLines((new Integer(value)), 1);
               if (controllerType == 13)
                  mixer.actionLines((new Integer(value)), 2);
                 [***Repetitive parts cut out***]
               break;
            }
            default:
            Midi.FreeEv(event); // otherwise dispose the event
            break;
         }
      }
      protected void finalize() {
         Midi.Close(this.refnum);
         this.Close();
         System.out.println("Closed");
         System.exit(1);
      }
   }
```

4.6 *jMusic* Summary

The granulating components of *jMusic* are still being refined to include more features, as mentioned previously, envelope shape control, sound source control, and more refined frequency control integrated with the brass instrument based fingering style. Another feature that is being developed is microphone input. I specifically want to incorporate microphone input because it adds a large array of sound sources and control that can be changed as quickly as the performer can create a vocal sound. In previous experiments, the microphone was found to cause too much latency when the

whole process started working in real-time. This is something I wish to resolve in the near future.

Working with *jMusic* has been very rewarding. It has allowed me to create a fine-tuned granulation routine which has been seamlessly incorporated into the already high functionality of *jMusic*.

As well as using the hardware interface, I have been asked by a number of people in my granular synthesis discussion group to make a graphical based interface, so that they can experiment with it on their computers without having to build any hardware. This is something I have already started to plan. Using the Java graphical toolkit, Swing, and customized graphics, I intend to represent the poseidon in a 2 dimensional display on a computer, with all of the associated controllers and functionality. This Java graphical interface will not be used for live performances, not by me anyway. It will be designed for quick experimentation, and for computers with no MIDI ports. By making a graphical interface many more people will be able to test and expand and improve the granulating components of *jMusic*.

Chapter 5

Performance Outcomes

After spending months building and programming a real-time granular synthesis engine, the real test came when it had to be used in live performance.

The first public exhibition of this instrument was held at the REV festival in Brisbane in April 2002. For the most part the instrument was located in the elevator, with an accompanying performer. The two QUT music undergraduate students who helped me perform with the poseidon were Rafe Sholer and Joel Joslin. They both picked up the instrument playing techniques very quickly. Within half an hour, they were already picking out certain finger combinations that worked together and gave pleasing aural results. They could both see a lot of potential with the instrument and enjoyed working with it. The location of the instrument had some positive and negative results. On the positive side, it gave the performer and the listeners an intimate space where they could absorb many of the sounds without being distracted by other exhibitions at the festival. On the negative side many people missed out on seeing it because they were not expecting anything to be in the elevator, and most of the displays were on the split ground level, for which an elevator was not required. It was also cramped in the elevator resulting in many people not wanting to take the time to stay and learn more about the instrument. The elevator also broke down on the final day of the exhibition. The worst part was traveling on the elevator so long gave me motion sickness, although the other two performers handled it much better.

5.1 What About The Children?

As expected, the children loved it. They were curious about the shape of the instrument and of what the different controllers did. It was really enjoyable to see young children taking such an interest in the sound they were creating. It is interesting to note that the children were interested in hearing how sound changes. They were not looking for music for relaxation or dancing, they were interested in loud sounds and changing sounds. The children were also not content to just watch someone else play the instrument they were anxious to make the sounds themselves. None of the children had any concerns as to how the sound was being produced, they just wanted to be in control of it. In analysing the response of the children, I think they really appreciated having such a visual interface to work with.

5.2 Adult Reaction

The adults were a lot more restrained. Some were curious as to the technicalities of granular synthesis, some were interested in the design of the interface, some just did not know what to think. Those that did not know what to think were generally caught off guard by finding a performer in the elevator, and then confused as to why a sea creature was being used to create sound. In general these people did not want to try playing the poseidon. Those that were asking questions about the instrument were generally more interested in playing with the instrument.

In reflecting upon the response of the adults, I think that the interface design was too visual and detracted from the audio, although the audio itself was a little disconcerting for some of the audience. They were not sure what to make of the sounds they were hearing. It was not the kind of music one expects to hear in an elevator, instead it was a soundscape of noise and unnatural sounds. If I had sampled *The Girl From Ipanema* and used it as the sound source for the granulation process, I might have had a much different reaction. Unfortunately I did not think about this until afterwards. Another interesting observation I made was that despite the elevator being glass, allowing the passengers to see where they were travelling, if I raised the pitch whilst the elevator was going down, or vice versa, then the passengers in the elevator thought they had gone in the wrong direction, which shows just how influential

sound is in our society.

5.3 Russian Film Improvisation

One highlight of the exhibition was to do a live performance with Benn Woods, Andrew Kettle and Greg Jenkins. We improvised music for a silent experimental Russian film entitled *Man with a movie camera* [Kettle, 2002]. This performance really allowed me to try out the different sounds I could create in a large space and a large sound system. The effect was vastly different from that of the elevator performances. We will be doing some future performances together as a result. The main reason I am now working on ways to modify the frequency control of the instrument is because I found it too restrictive being limited to sliders. I could jump quickly from one frequency to another with very little slide, but it was inaccurate. With a slider and preset frequencies, I could play more accurately. This would enhance performance especially when being played as part of an ensemble. For the generation of soundscapes the slider method proved to be adequate.

In actual performance I found that using a physical interface shaped like a more conventional instrument helped the audience to become more involved with the music, and I was likewise more able to interact with the audience. I could watch the audience at all times and not have to look at the instrument. If I was to perform the exact same music but with the computer sitting on the desk, and myself in front of it playing with a mouse, or even a mixing panel, the audience participation would be decreased, as I would be interacting with the computer rather than with the intended audience. This of course could be applied to all computer-based music. The shape and lightness of the instrument helped with the performances. As the photo in 5.1 on the next page depicts, the performer can hold the instrument quite comfortably and play, without even needing to watch the instrument. It also includes a strap to go around the neck so the performer is freer with their actions and body movements. The shape suited my hand very well. For a small child it was too large, but as I was the main performer, it was more suitable to shape it to my hand size. Joel and Rafe also had similar sized hands, so they also found it easy to hold and maneuver.

Another finding I made was that the audience did not expect such a wide range

Figure 5.1: Timothy Opie holding the poseidon.

of sounds to come from one instrument. I was able to create sounds from growly rumbles that vibrated the floor, right up to bursts of high pitched flutters. I found the granular synthesis real-time instrument to be very useful and very flexible in performance. Even though it was created on a small budget it gives the performer a wide range of sounds to explore, that can be changed instantly.

One last performance issue of exceptional worth was the fact that the program did not crash once in the 3 days that it was running.

5.4 *Elucidation*, a Poseidon and Flute Duet

As briefly introduced earlier in this paper, I wrote a duet for the poseidon and flute entitled *Elucidation*. The piece was written and performed 2 months after the REV Festival. There were a number of motives for writing this piece of music. The most practical was that I wanted a piece of music to showcase at the Australasian Computer Music Association Conference in July 2002. The most important motive was that I wanted to see how well the poseidon worked with other more traditional instruments. I believe that for an instrument to be effective it needs to be able to work with a large range of different instruments within different contexts. The flute was the first non-electronic instrument with which the poseidon had been matched. Compared to playing the flute I found the poseidon much harder to control. There were more things that needed to be dealt with simultaneously. I did not feel I had the same expressiveness achievable from a flute, that said, I had only been playing the poseidon for two months, which is still the rote learning stage for any instrument. I will try playing the piece again in a decade to see how much better my playing technique has become, and how much better the two instrument work together. I intend to write more music for the poseidon in conjunction with acoustic instruments because the poseidon was created to be played on stage like any hand held instrument. This first song was just an initial testing stage in an instrument that can be advanced over time.

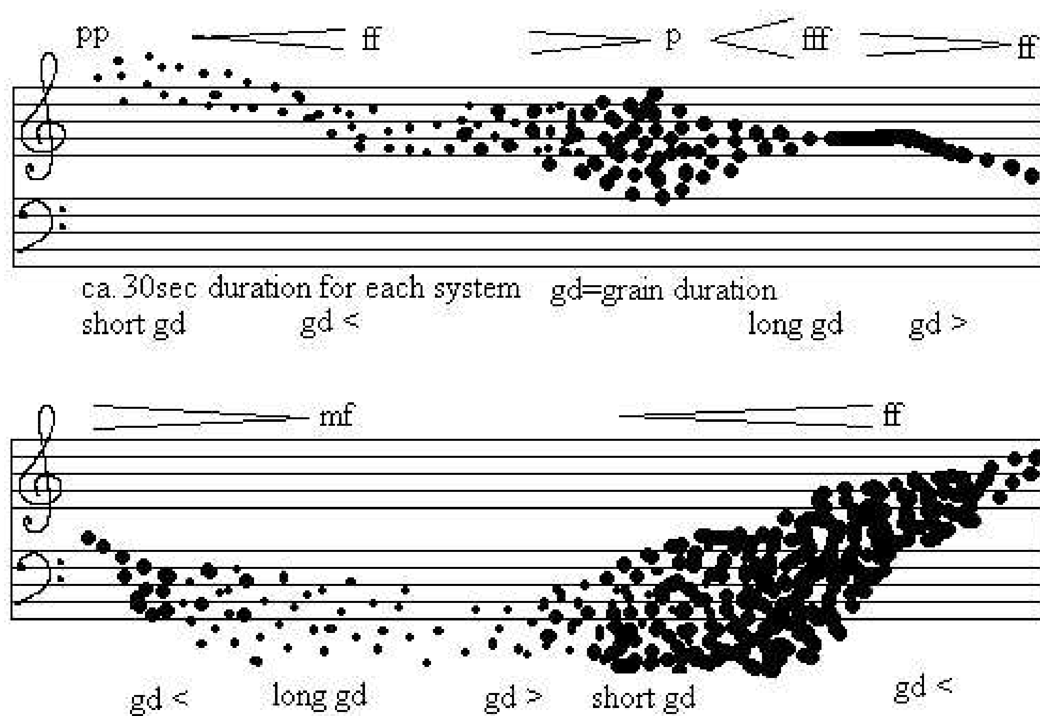

Figure 5.2: An example of the method I used to notate for the poseidon.

5.5 Personal Reflection on the Results of the Poseidon

I am pleased with the results of the instrument. I achieved all the goals I wanted to achieve to a highly acceptable level. This does not mean that the poseidon is complete and perfect, but it has set a solid benchmark from which it can now be refined and improved. The instrument could carry the title poseidon-stable-1.1, if it were just a computer program. Instead, it also consists of hardware, which makes refinement a little more difficult, but this is a small price to pay for a mobile interface that can be used with comparative ease, and which visually enhances the performance.

I have already mentioned the improvements that can be made to the poseidon a number of times through the paper. There are other things not yet mentioned that can be done such as optimising the code more to get better performance. I think such improvements go without saying. The question that really needs to be asked now is how the poseidon and its associated software can be used to benefit and enhance the future of granular synthesis.

I think there are a number of beneficial aspects that have been addressed that will help for the future, and there are new avenues that have been opened up to explore.

The first benefit is that there is now open-source code in Java that can be used by other Java programmers who wish to work with granular synthesis. It is documented and easy to follow. The exploratory fields opened up by the code is that now anyone can expand on the work. Programmers who write efficient and highly optimised code can improve the speed of the code. Programmers who feel there is a feature missing can add the feature as they desire. The more people who work on the project the more useful it can become to the community as a whole. The current code is just a foundation upon which to build and expand. As I learn more about granular synthesis I will be making my own updates to the code. One thing I really want to improve is the grain envelope design. I want to learn more and experiment more with different grain envelope shapes, and I want to set up the instrument so that I can change the envelope specifications instantly during performance. This granular synthesis development has and will continue to improve the *jMusic* project.

The second benefit is that the poseidon has given a way to perform live on a stage which has not been done before with granular synthesis. Non-computer interfaces

have been produced in the past such as joysticks, mixing boards, keyboards, and other objects, but nothing that was designed as a friendly sea creature. In fact nothing that is portable, hand held and resembling a more acoustic instrument has been used before. This design shows a way that granular synthesis can be integrated with more traditional styles of performance. I would like to see more work done in the area of instrument and interface designs.

Having created a granular synthesis instrument, the question now is, would it be beneficial to create many of them and form an orchestra of granular synthesis instruments? The answer is an undeniable yes. Since the inception of this idea I have been thinking about how granular synthesis would sound if there were many different streams being created simultaneously by many different people, all with their own particular intonations and playing styles. Even if they were all playing the same piece of music with an extremely high degree of accuracy it would still add a depth and character to the piece that could never be achieved by a solo instrument. Using an interface that requires a high degree of human input adds an infinite number of new variables to the music, variables that cannot be fully replicated by a machine.

Bearing in mind an orchestra, the next question is how does one notate a granular synthesis score? This is also a question I have been contemplating since the initial inception of the instrument. So far I do not have a very accurate answer. In the score for *Elucidation* I drew pictures on manuscript paper indicating the approximate pitch, pitch range, and time. If the texture was very sparse then I drew dots, whereas if the density increased the dots became lines and large circles. I used dynamic markers to signify intensity, and for any other instructions I just wrote underneath. An example of such as score can be seen in figure 5.2 on page 124. It served me fine because I knew what I wanted, but anyone else reading the same music might interpret it differently. This is also something that can be expanded upon in the future.

Chapter 6

Conclusion

This thesis has shown how real-time granular synthesis can be used in live performance. From the initial concepts of the nature of sound, the theory and practice of granular synthesis has grown immensely to provide a range of new ways to explore sound. It has advanced from a scientific issue into a compositional technique, from which it has become a performance medium, with many associated tools.

The poseidon presents a compact and more versatile means of performance with granular synthesis in a live performance setting, a performance aimed at entertaining the audience both aurally and visually. It also presents an interface which can be exploited to work within the framework of a non-electronic ensemble of instruments. The computer code that operates with the poseidon is open source and is an extension to jMusic. This allows other musicians access to the workings of the program, and the ability to change it to their own desire.

I have demonstrated the use of the poseidon in a number of different musical settings, including the elevator performances, the four piece electronic outfit, and the flute duet. The poseidon still needs some adjusting to help with its integration, especially with acoustic instruments. These features, such as the trumpet pitch control style, the envelope control, and the audio input control have been discussed.

Creating and using the poseidon has been a very exciting project. It is something that I will continue to work and perform with. I look forward to the day when I can conduct an orchestra of finely tuned granular synthesis instruments.

Appendices

Appendix A

Glossary

Corpuscle: Beeckman believed that when he played a note the sound would be manifest as a collection of corpuscles. In a very simple example, if he played a note on a harp he could see the string vibrate. Each vibration would cut into the air, like an arc, and those cut out sections contained sound and were set free to spread out, to propagate the sound.

Globule: This term refers to a group of corpuscles. The intensity of a sound could be determined by how many corpuscles were in the globule, although the pitch was determined by the length of the individual corpuscle as Beeckman knew that the longer strings created lower sounds, therefore a long corpuscle would have a lower pitch.

Gabor Grain: A single quantity of the smallest amount of sound needed so that it would still be recognised as a sound. It was referred to as the Gabor grain by many people because Gabor defined how much information that entails.

Logon: Gabor's name for a Gabor Grain.

Wavelet Transform: The Wavelet Transform is very similar to granular synthesis except that it is more strict in its definition and construction. A Gabor grain can be set at any length arbitrarily, Gabor has just suggested that it is most effective between 20 and 50ms. A wavelet though derives its "grain" length determined by the pitch of the contents. The wavelet is designed to start and end at 0 phase. Wavelet synthesis can be used for better pitch shifting and reproduction than granular synthesis, but it requires so much analysis that it

is much slower to work with in a real-time environment. Gabor grains do a satisfactory job. Daniel Arfib (1991) once asked "Is the wavelet transform a Gabor time-frequency representation?" To which he gave the reply "Perhaps it could be if Gabor had thought about it, but he did not."

Grainlet Synthesis: The term for the wavelet transform when used in a compositional context in a matter like granular synthesis. Also referred to as wavelet synthesis.

Glisson Synthesis: A derivative of granular synthesis whereby the contents of each grain are modified with a glissando. Implementation of this feature to the Granulator class in jMusic has already begun.

Pulsar Synthesis: A form of particle synthesis whereby each grain is created as a pulsar, generated by an impulse generator.

Sound File Granulation: Conversion of a sound file into a stream of audio grains.

Appendix B

Csound code for Elevated Music prototype

```
;**ELEVATED*MUSIC*********************************************
;*******REAL-TIME*GRANULATOR**********************************
;************FOR*EXPERIMENTS*IN*SOUND*************************
;*****************AND*SENSUOUSLY*DELICIOUS*LIVE*PERFORMANCES***
;*********************CREATED*BY*TIMOTHY*OPIE*AT QUT*FOR*REV**
;*                                                           *
;* This real-time MIDI controlled granulator was created as  *
;* part of the REV project at QUT Australia and also as      *
;* a practical experiment in granular synthesis for a        *
;* Masters Degree project at QUT.                            *
;*                                                           *
;* This Csound instrument receives MIDI controller inputs    *
;* and uses this information to create grains of sound which *
;* are sent directly to the audio output and create a        *
;* granular synthesis texture.                               *
;*                                                           *
;* This instrument has 2 main functions. The first is to     *
;* function as an instrument that can be used real-time in a *
;* live performance scenario. During the REV exhibition it   *
```

```
;* will be used as an exhibition piece where the public may   *
;* experiment with it, and it will also be used for a number  *
;* of performances.                                           *
;* The second function is to act as a research tool with      *
;* which I can experiment and capture a range of different    *
;* textures, which will be analysed in my Masters Thesis      *
;*                                                            *
;**************************************************************

<CsoundSynthesizer>
<CsOptions>

-+K -b100 -+S1 -+C1 ;-m0 -+O
; -+S1
; -+C1 mic input
; -+* -+X -+q these may or may not increase performance depending
;             on what is going on
; -m0 -+O omit ALL print statements (big speed increase, very very
;         bad for debugging)

</CsOptions>
<CsInstruments>

sr      = 44100
kr      =  4410
ksmps   =    10
nchnls  =     2

;/////////////////////////////
        instr 1
;///////////////////GRAIN-INST
```

```
;=========MIDI=CONTROL=SETUP========================================
; first we will set up the midi controller using this basic routine:
; the 7-bit value is fine as that is all we need to generate

;k1,...,kN sliderN ichan, ictlnum1, imin1, imax1, init1, ifn1, ...,
;                          ictlnumN, iminN, imaxN, initN, ifnN

k1,k2,k3,k4,k5,k6,k7,k8,k9,k10,k11,k12,k13,k14,k15,k16  \
        slider16    1,  12,0.015,0.06,0.02,20,  \; k1
                        13,0.0,5.0,0,20,        \; k2
                        14,1,4000,3000,20,      \; k3
                        15,0,4000,0,20,         \; k4
                        16,0,1000,1,20,         \; k5
                        17,0,1000,0,20,         \; k6
                        18,0.0,1.0,0.5,20,      \; k7
                        19,0.0,5.0,0,20,        \; k8
                        20,0,800,0,20,          \; k9
                        21,50,4000,440,21,      \; k10
                        22,0,127,5,0,           \; k11 (unused)
                        23,0,127,5,0,           \; k12 (unused)
                        24,0,127,5,0,           \; k13 (unused)
                        25,0,127,5,0,           \; k14 (unused)
                        26,0,127,5,0,           \; k15 (unused)
                        27,0.0,10.0,0,20;       \; k16
;=============
;K definition:
;k1  = grain duration Q from 0.015-0.06 i:0.02
;k2  = grain duration spread (random factor) 0-200% i:0
;
;k3  = Amplitude Q 1-1000 i:80
;k4  = Amplitude spread (random factor) 0-200% i:0
;
```

```
;k5  = Grain density Q 0-1000 1
;k6  = Grain density spread (random factor) 0-200% i:0
;
;k7  = PAN 0.0=left, 1.0=right i:0.5
;k8  = ?
;
;k9  = Frequency Q 0-800 difference 1:0
;k10 = Frequency 40-5000 i:440
;k16 = ?
;====================================================================
; grain generation
; ar    grain   xamp, xcps, xdens, kampoff, kcpsoff, kgdur,
;               igfn, ienvfn, imgdur

;kgdur = k1+(rnd(k2))-(k2*0.5)  ;grain dur with RND factor added
agps  = k5+(rnd(k6))-(k6*0.5)   ;grain density (gps) with RND f. a.

agrain  grain   k3, k10, agps, k4, k9, k1, 30, 1, 0.10

audioleft  = agrain*sqrt(1-k7)    ;equal power panning
audioright = agrain*sqrt(k7)

        outs    audioleft,audioright
        endin
</CsInstruments>
;///////////////////////////////////////////////////////////////////
;///////////////////////////////////////////////////////////////////
;///////////////////////////////////////////////////////////////////
<CsScore>
; Grain Envelopes
;       Hanning shaped envelope
f1      0 8192 20 2 1
```

```
;       Exponential shaped envelope, non-symmetrical with sustain
;f2     0 8192 5 0.01 2192 1 2000 1 4000 0.01
;       Exp. shaped envelope, non-sym with sustain
;f3     0 8192 5 0,01 2192 1 6000 0.01
;       Exp. shaped envelope, symmetrical
;f4     0 8192 5 0.01 4048 1 4048 0.01
;       Bartlett (triangular) shaped envelope
;f5     0 8192 20 3 1
;       Gaussian shaped envelope
;f6     0 8192 20 6 1
;       Blackman-Harris (wider Gaussian) shaped envelope
;f7     0 8192 20 5 1

; MIDI control functions
;       a straight line going from 0 to 127 in 128 steps
f20     0 128 7 0 128 127
;       an exp. curve going from 0 to 127 in 128 steps
f21     0 128 5 0.01 128 127
;       a straight line going from 0 to 127 in 128 steps
;f25    0 4097 7 0 4097 4097

;grain content functions
;       simple sine wave
f30     0 4097 10 1
;       sine wave with inharmonic partials
f31     0 4097 9 1 0 7 0.2 3 0.2 2 4 7
;       uniform white noise
f32     0 4097 21 1 1

;the sounds start here!
; NOTE: 3600=1 hour 28800=8 hours (min needed at powerhouse)
i1 0 30000
```

```
e


</CsScore>
</CsoundSynthesizer>
```

Appendix C

jMusic code for Spurt

```
// File: Spurt.java
// Author: Timothy Opie
// Description: Uses the score creation function of jMusic to
// create a score with granular synthesis properties.
// Based on Spray.java by Andrew Brown.
//=============================================================

import java.io.*;
import jm.JMC;
import jm.music.data.*;
import jm.util.*;
import jm.audio.*;
import jm.audio.io.*;
import jm.audio.synth.*;

public class Spurt implements JMC {
   public static void main(String[] args) {
      new Spurt();
   }
   public Spurt() {
      Score score = new Score();
      Part p = new Part("Spurt", 0, 0);
```

```
        Phrase phr = new Phrase();
        // Set up an instrument
        Instrument inst = new VibesInst(44100);
        // iterate through many notes
        for(int i=0; i<5000; i++) {
           // set duration of note
           double dur = 0.02 + (Math.random() * 0.03);
           // create a note pitched between C3 + C6,
           // asyn. duration with MAX 7 overlap, random amp.
           Note n = new Note(C3 + (int)(Math.random() * 37),
                    dur*(Math.random() + 0.15),
                    (int)(Math.random() * 80 + 47));
           n.setPan(Math.random());
           n.setDuration(dur);
           // add the note to the phrase
           phr.addNote(n);
        }
        // pack the phrase into the jMusic score structure
        p.addPhrase(phr);
        score.addPart(p);
        // display the score on screen
        //View.print(score);
        //View.show(score);
        // render the score as an audio file
        Write.au(score, "TestSprurt.au", inst);
    }
}
```

Appendix D

jMusic code for Granulator - The Audio Object

```
/**
 * Granulator.java
 * An Audio Object for granulating input.
 * @author Andrew Sorensen
 * @version 1.0,Sun Feb 25 18:42:48 2001
 * Revised and modified extensively by Timothy Opie
 * Last changed 04/08/2002
 */
package jm.audio.synth;
import jm.music.data.Note;
import jm.audio.AudioObject;
import jm.audio.Instrument;
import jm.audio.AOException;
public final class Granulator extends AudioObject{
//------------------------------------------------
// Attributes
//------------------------------------------------
   private int grainDuration; // grain duration
   private int cgd; // counter for grain duration
   private float[] grain; // grain buffer
```

```
private float[] newbuf; // output buffer
private int grainsPerSecond = 50; // grains per second
private float[] tailBuf; // overlap buffer
private double freqMod = 1.0; // pitch control
private float[] inBuffer = null; // temporary buffer
private boolean inBufActive = false; // states whether the
                              //temporary buffer is needed
private boolean ri = false; //random indexing
private boolean rgd = false; //random grain duration
private int gdb = 1000; //the smallest rnd grainduration
private int gdt = 1000; //the highest rnd grain duration + gdb
private boolean rf = false; //random frequency
private double rfb = 0.8; //the lowest rnd frequency
private double rft = 0.5; //the highest rnd frequency + rfb
//----------------------------------------------
// Constructors
//----------------------------------------------
public Granulator(AudioObject ao, int duration, int gps){
   super(ao, "[Granulator]");
   this.grainDuration = duration;
   this.grainsPerSecond = gps;
   this.cgd = 0;
   this.grain = new float[this.grainDuration];
   tailBuf = new float[0];
}
/**
 * @param buffer The sample buffer.
 * @return The number of samples processed.
 */
public int work(float[] buffer)throws AOException{
   if(inBuffer == null){
      newbuf = new float[buffer.length];
```

```
      this.previous[0].nextWork(newbuf);
   }else{
      newbuf = new float[buffer.length];
      for(int i=0;(i<inBuffer.length)&&
            (i<newbuf.length);i++){
         newbuf[i] = inBuffer[i];
      }
      inBuffer = null;
   }
   //number of grains to fit in buffer
   int nog = (int)((float)newbuf.length /
            ((float)(sampleRate*channels) /
            (float)grainsPerSecond));
   //time between grains
   int tbg=(newbuf.length/nog);
   //add any grain tails
   for(int i=0;(i<buffer.length)&&(i<tailBuf.length);i++){
      buffer[i]+=tailBuf[i];
   }
   tailBuf = new float[newbuf.length];
   inBufActive = true;
   //add all new grains
   for(int i=0;i<nog;i++){
      int index = i*tbg;
      setGrain(index);
      for(int j=0;j<grain.length;j++){
         if(index >= buffer.length){
            tailBuf[index-buffer.length]+=grain[j];
         }else{
            buffer[index] += grain[j];
         }
         index++;
```

```
            }
        }
        inBufActive = false;
        return buffer.length;
    }
    // Deviation from the input frequency
    public void setFreqMod(float fmod){
        this.freqMod = fmod;
    }
    // Set the grain duration
    public void setGrainDuration(int gdur){
        this.grainDuration = gdur;
    }
    // Set the number of grains per second
    public void setGrainsPerSecond(int gps){
        this.grainsPerSecond = gps;
    }
    // Set a random grain duration
    public void setRandomGrainDuration(boolean bool){
        this.rgd = bool;
    }
    // Set the random grain durations bottom value
    public void setRandomGrainBottom(int b){
        this.gdb = b;
    }
    // Set the random grain durations top value
    public void setRandomGrainTop(int t){
        this.gdt = t;
    }
    // Set random index position
    public void setRandomIndex(boolean bool){
        this.ri = bool;
```

```
}
// Set random frequency
public void setRandomFreq(boolean bool){
   this.rf = bool;
}
// Set the random frequency bottom value
public void setRandomFreqBottom(double fl){
   this.rfb = fl;
}
// Set the random frequency top value
public void setRandomFreqTop(double ft){
   this.rft = ft;
}
//---------------------------------------
// Private Methods
//----------------------------------------
/**
 * Set the grain
 */
private void setGrain(int index) throws AOException{
   if(ri) index=((int)(Math.random()*(double)newbuf.length));
   float[] buf = newbuf; //reference to the active buffer
   if(rgd)this.cgd = gdb+(int)(Math.random()*gdt);
   else this.cgd = this.grainDuration;
   double cfm = this.freqMod;
   if(rf) cfm = this.rfb+(Math.random()*(this.rft-this.rfb));
   if(inBufActive){
      inBuffer = new float[newbuf.length];
      int ret = this.previous[0].nextWork(inBuffer);
      inBufActive=false;
   }
   this.grain = new float[cgd];
```

```
int count = 0;
float tmp = 0.0f;
//pos. values of skip are the iterations to skip
//neg. values of skip are the iterations to add between
double skip = -1.0/((1-cfm)/cfm);
double remains = 0.0;
int upSample = 0;
if(skip<0){skip=-1.0/skip; upSample=1;}
if(skip==0)upSample=2;
int ind=0;
for(int i=index;true;i++){
   if(i==buf.length){i=0;buf=inBuffer;}
   if(upSample==0){//remove samples (up sample);
      if(++ind>=(int)(skip+remains)){
         remains=(skip+remains)%1.0;
         ind=0;
         continue;
      }
      if(count >= cgd)break;
      grain[count++]=buf[i];
   }else if(upSample==1){//add samples (downsample)
      if((skip+remains)>=1.0){
         float p=(tmp-buf[i])/((int)skip+1);
         for(int k=0;k<(int)(skip+remains);k++){
            grain[count++]=p*k+buf[i];
            if(count==cgd)break;
         }
      }
      if(count==cgd)break;
      grain[count++]=buf[i];
      tmp=buf[i];
      remains = (skip+remains)%1.0;
```

```
            }else{ //no resample ;)
               grain[count++]=buf[i];
            }
            if(count==cgd)break;
         }
         //Envelope our new grain
         for(int i=0;i<cgd;i++){
            this.grain[i] = this.grain[i] *
                            (float)(Math.sin(Math.PI*i/cgd));
         }
      }
   }
```

Appendix E

jMusic code for the GranularInstRT

```
/**
 * GranularInstRT.java
 * Author: Timothy Opie
 * Last Modified: 04/08/2002
 * Designed to function with jMusic
 * by Andrew Brown and Andrew Sorensen
 */
import jm.audio.io.*;
import jm.audio.Instrument;
import jm.audio.synth.*;
import jm.music.data.Note;
import jm.audio.AudioObject;
import jm.audio.io.RTIn;
public final class GranularInstRT extends jm.audio.Instrument{
   //----------------------------------------------
   // Attributes
   //----------------------------------------------
   // the name of the sample file
   private String fileName;
   // How many channels is the sound file we are using
```

```
private int numOfChannels;
// the base frequency of the sample file to be read in
private double baseFreq;
// should we play the wholeFile or just
// what we need for note duration
private boolean wholeFile;
// The points to use in the construction of Envelopes
private EnvPoint[] pointArray = new EnvPoint[10]
private Granulator grain;
private Volume vol;
private StereoPan pan;
// used to define the audio input type
private int sounds;
//----------------------------------------------
// Constructor
//----------------------------------------------
public GranularInst(String fileName){
   // Use this constructor when
   // you want to granulate an audio file.
   // Only the name of the audio file is required
   this.fileName = fileName;
   this.numOfChannels = 2;
   this.sounds = 7;
}
public GranularInstRT(int sounds ){
   /**
    * The variable sounds is an integer used to select
    * which sound source type will be used.
    * It will be defined as such:
    * SINE WAVE = 0
    * COSINE WAVE = 1
    * TRIANGLE WAVE = 2
```

```
     * SQUARE WAVE = 3
     * SAWTOOTH WAVE = 4
     * SAWDOWN WAVE = 5
     * SABERSAW WAVE = 6
     * AUDIO FILE = 7
     * MICROPHONE = 11
     *
     * Use this constructor when you want to granulate internally
     * produced audio. Note: you can still granulate audio files
     * if you use this constructor, but it will assume the audio
     * file has the name song1.au.
     */
    this.sounds = sounds;
    this.numOfChannels = 2;
    this.fileName = "song1.au";
}
//-----------------------------------------------
// Methods
//-----------------------------------------------
/**
 * Create the Audio Chain for this Instrument
 * and assign the primary Audio Object(s). The
 * primary audio object(s) are the one or more
 * objects which head up the chain(s)
 */
public void createChain(){
    // define the chain
    if (sounds<0 || sounds>6){
        if (sounds>10){
            // if sounds is > 10 then the microphone is the input
            // source. Default is 11, but this way it doesn't
            // matter if a wrong number gets inputed.
```

```
            // (8820 = buffer length of 1/5 of a second)
            RTIn grin = new RTIn(this,44100,2,8820);
            grain = new Granulator(grin,50,100);
            vol = new Volume(grain,0.5f);
            Volume vol2 = new Volume(vol,0.1f);
            pan = new StereoPan(vol2);
        } else {
            // if sounds is < 0 or > 6 and < 11 then it will
            // process an audio file. Default is 7.  Again it is
            // very open ended to accommodate wrong input numbers.
            SampleIn grin = new SampleIn(this, this.fileName);
            grain = new Granulator(grin,50,100);
            vol = new Volume(grain,0.5f);
            Volume vol2 = new Volume(vol,0.1f);
            pan = new StereoPan(vol2);
        }
    } else {
        // At this stage the only values left are between 0-6
        // These correspond directly to the oscillator input
        // values, so can be added directly.
        Oscillator grin = new Oscillator(this, sounds, 44100,2);
        grain = new Granulator(grin,50,100);
        vol = new Volume(grain,0.5f);
        Volume vol2 = new Volume(vol,0.1f);
        pan = new StereoPan(vol2);
    }
}
public void setGrainsPerSecond(int sp){
    grain.setGrainsPerSecond(sp);
}
public void setGrainDuration(int gdur){
    grain.setGrainDuration(gdur);
```

```
    }
    public void setFreqMod(float fmod){
       grain.setFreqMod(fmod);
    }
    public void setRandomIndex(boolean b){
       grain.setRandomIndex(b);
    }
    public void setRandomGrainDuration(boolean b){
       grain.setRandomGrainDuration(b);
    }
    public void setRandomGrainTop(int top){
       grain.setRandomGrainTop(top);
    }
    public void setRandomGrainBottom(int b){
       grain.setRandomGrainBottom(b);
    }
    public void setRandomFreq(boolean bool){
       grain.setRandomFreq(bool);
    }
    public void setRandomFreqBottom(double d){
       grain.setRandomFreqBottom(d);
    }
    public void setRandomFreqTop(double d){
       grain.setRandomFreqTop(d);
    }
    public void setVolume(float d){
       vol.setVolume(d);
    }
    public void setPan(float p){
       pan.setPan(p);
    }
}
```

Appendix F

jMusic code for the RTGrainLine

```
/**
 * Real-time Granulator Line
 * Author: Timothy Opie
 * Latest Update: 04/08/2002
 * Designed to function with jMusic designed by
 * Andrew Sorensen and Andrew Brown
*/
import jm.music.rt.RTLine;
import jm.audio.Instrument;
import jm.music.data.Note;
public class RTGrainLine extends RTLine {
   private Note n = new Note(72, 3000.0);
      //500000 too large for mic :(
   private int pitch = 72;
   private int dynamic = 100;
   private GranularInst[] inst;
   private boolean durationRndFlag = false;
   private boolean freqRndFlag = false;
   private boolean asyn = false;
  /** Constructor */
   public RTGrainLine(Instrument[] instArray, double controlRate,
                        int bufferSize) {
```

```
      super(instArray, controlRate, bufferSize);
      inst = new GranularInst[instArray.length];
      for (int i=0; i<instArray.length; i++) {
         inst[i] = (GranularInst)instArray[i];
      }
   }
   public synchronized Note getNote(){
      n.setPitch(pitch);
      n.setDynamic(dynamic);
      n.setDuration(n.getRhythmValue());
      return n;
   }

   // added for control change
   public synchronized void
         externalAction(Object obj, int actionNumber){
      if(actionNumber == 1){ // grain duration
         Integer value = (Integer) obj;
         int duration = (value.intValue()) * (44100/1000);
         for(int i=0;i<inst.length;i++){
            inst[i].setGrainDuration(duration);
         }
      }
      if(actionNumber == 2){ // random grain duration flag
         Integer value = (Integer) obj;
         if (value.intValue() >= 64)
            durationRndFlag = true;
         else
         durationRndFlag = false;
         for(int i=0;i<inst.length;i++){
            inst[i].setRandomGrainDuration(durationRndFlag);
         }
```

```
}
if(actionNumber == 3){ // random Grain Duration Top
   Integer value = (Integer) obj;
   float vol = (float)value.intValue()/127.0f;
   for(int i=0;i<inst.length;i++){
      inst[i].setVolume(vol);
   }
}
if(actionNumber == 4){ // random Grain Duration Bottom
   Integer value = (Integer) obj;
   int rDurBottom = value.intValue() * (44100/1000);
   for(int i=0;i<inst.length;i++){
      inst[i].setRandomGrainBottom(rDurBottom);
   }
}
if(actionNumber == 5){ // Grain Frequency
   Integer value = (Integer) obj;
   float frequency = ((float)(value.intValue()+1.0)/57.0f);
   if (frequency>2.0) frequency = 2.0f;
   for(int i=0;i<inst.length;i++){
      inst[i].setFreqMod(frequency);
   }
}
if(actionNumber == 6){ // Frequency Random Flag
   Integer value = (Integer) obj;
   if (value.intValue() >= 64)
      freqRndFlag = true;
   else
   freqRndFlag = false;
   for(int i=0;i<inst.length;i++){
      inst[i].setRandomFreq(freqRndFlag);
   }
```

```
        }
        if(actionNumber == 7){ // Random Frequency Top
           Integer value = (Integer) obj;
           float pan = (float)value.intValue()/127.0f;
           for(int i=0;i<inst.length;i++){
              inst[i].setRandomFreqTop(rFreqTop);
           }
        }
        if(actionNumber == 8){ // Random Frequency Bottom
           Integer value = (Integer) obj;
           double rFreqBottom = ((value.intValue()+1.0)/57.0);
           if (rFreqBottom>2.0) rFreqBottom = 2.0f;
           for(int i=0;i<inst.length;i++){
              inst[i].setRandomFreqBottom(rFreqBottom);
           }
        }
        if(actionNumber == 9){ // Grains Per Second
           Integer value = (Integer) obj;
           int gps = (value.intValue()*7)+11;
           for(int i=0;i<inst.length;i++){
              inst[i].setGrainsPerSecond(gps);
           }
        }
        if(actionNumber == 10){ // Basic Panning
           Integer value = (Integer) obj;
           int pan = (value.intValue()/127);
           for(int i=0;i<inst.length;i++){
              inst[i].setPan(pan);
           }
        }

        if(actionNumber == 11){ // Asynchronous/Asynchronous GS
```

```
            Integer value = (Integer) obj;
            if (value.intValue() >= 64)
               asyn = true;
            else
            asyn = false;
            for(int i=0;i<inst.length;i++){
               inst[i].setRandomIndex(asyn);
            }
         }
      }
   }
```

Appendix G

jMusic code for the RTGrainMaker

```
/**
 * Real-time Grain Maker
 * Author: Timothy Opie
 * Latest Update: 04/08/2002
 * Designed to function with jMusic designed by
 * Andrew Sorensen and Andrew Brown
 */
import grame.midishare.*;
import grame.midishare.Midi;
import grame.midishare.MidiAppl;
import grame.midishare.MidiException;
import jm.JMC;
import jm.music.data.*;
import jm.audio.*;
import jm.util.*;
import jm.audio.RTMixer;
import jm.music.rt.RTLine;
public final class RTGrainMaker extends MidiAppl implements JMC {
   private RTMixer mixer;
   private RTGrainLine grain;
```

```
private int sampleRate = 44100;
public static void main(String[] args){
   new RTGrainMaker();
}
private GranularInst GranInst;
public RTGrainMaker(){
   // setting up midishare
   // open the MIDI application
   try { this.Open("myApp");
   } catch(MidiException me) {
      System.out.println("Could not open");
   }
   // System.out.println("Success!");
   Midi.Connect(this.refnum, 0, 1);
   Midi.Connect(0, this.refnum, 1);
   // end midi share

   int channels = 2;
   //NOTE: the control rate must be LARGER than grainDuration
   double controlRate = 0.1;
   int bufferSize = (int)((sampleRate*channels) *controlRate);
   Instrument[] instArray = new Instrument[1];
   for(int i=0;i<instArray.length;i++){
      instArray[i] = new GranularInst(7);
   }
   GranInst = (GranularInst) instArray[0];
   grain = new RTGrainLine(instArray, controlRate, bufferSize);
   RTLine[] lineArray = {grain};
   mixer = new RTMixer(lineArray, bufferSize, sampleRate,
      channels, controlRate);
   mixer.begin();
}
```

```
public void ReceiveAlarm(int event) {
   //System.out.println("MIDI mesg = " + event);
   switch (Midi.GetType(event)) {
      // MIDI in controller messages
      case Midi.typeCtrlChange: {
         int controllerType = Midi.GetData0(event);
         int value = Midi.GetData1(event);
         //System.out.println("controller type " +
         //controllerType + "value = " + value);
         if (controllerType == 12)
            mixer.actionLines((new Integer(value)), 1);
         if (controllerType == 13)
            mixer.actionLines((new Integer(value)), 2);
         if (controllerType == 14)
            mixer.actionLines((new Integer(value)), 3);
         if (controllerType == 15)
            mixer.actionLines((new Integer(value)), 4);
         if (controllerType == 16)
            mixer.actionLines((new Integer(value)), 5);
         if (controllerType == 17)
            mixer.actionLines((new Integer(value)), 6);
         if (controllerType == 18)
            mixer.actionLines((new Integer(value)), 7);
         if (controllerType == 19)
            mixer.actionLines((new Integer(value)), 8);
         if (controllerType == 20)
            mixer.actionLines((new Integer(value)), 9);
         if (controllerType == 21)
            mixer.actionLines((new Integer(value)), 10);
         if (controllerType == 27)
            mixer.actionLines((new Integer(value)), 11);
```

```
            break;
          }
          default:
          Midi.FreeEv(event); // otherwise dispose the event
          break;
        }
    }
    protected void finalize() {
        Midi.Close(this.refnum);
        this.Close();
        System.out.println("Closed");
        System.exit(1);
    }
}
```

Bibliography

Ambrosine, J. (2002). *Correspondence*. Email: 30/9/2002, 8/11/2002, 12/11/2002

Arfib, D. (1991). Analysis, Transformation, and Resynthesis of Musical Sounds with the Help of a Time-Frequency Representation, In G. De Poli, A. Piccialli, & C. Roads (Eds.), *Representations of Musical Signals*, pp. 87–118. Cambridge: MIT Press.

Bastiaans, M. (1980, April). Gabor's Expansion of a Signal into Gaussian Elementary Signals. *Proceedings of the IEEE*, *68*(4), 538–539.

Bastiaans, M. (1985, August). On the Sliding-Window Representation in Digital Signal Processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *ASSP-33*(4), 868–873.

Bastiaans, M., & Van Leest, A. (1998). From the Rectangular to the Quincunx Gabor Lattice via Fractional Fourier Transformation. *IEEE Signal Processing*, *5*(8), 2003–205.

Berry, M. (1988). Planck's Constant. In A. Bullock, O. Stallybrass, & T. Stephen (Eds.), *Modern Thought*. London: Fontana Press.

Bowcott, P. (1998). Cellular Automation As A Means Of High Level Compositional Control of Granular Synthesis. In *ICMC Proceedings 1998*, pp. 55–57. ICMA.

Bowie, D. (1967). *Rock Reflections*. Polydor. The Laughing Gnome.

Brown, A., & Sorensen, A. (2000a). Introducing jMusic. In A. Brown, & R. Wilding (Eds.), *Proceedings Of The Australasian Computer Music Conference 2000*, pp. 68–76, Brisbane. ACMA.

Brown, A., & Sorensen, A. (2000b). *jMusic Homepage*. `http://jmusic.ci.qut.edu.au`.

Brown, M. (2002, April). REV: Real Electronic Virtual. Brisbane Powerhouse, Australia.

Burt, W. (2002). *Correspondence*. Email: 20/10/2002, 30/10/2002, 11/11/2002

Chapman, D., Clarke, M., Smith, M., & Archbold, M. (1996). Self-Similar Grain Distribution: A Fractal Approach to Granular Synthesis. In *ICMC 1996 Proceedings*, pp. 212–213, Hong Kong. ICMA.

Cohen, H. (1984). *Quantifying Music*. Dordrecht: D. Reidel Publishing Company.

Computer Music Research Laboratory of Grame (2001). *MidiShare*. `http://www.grame.fr/MidiShare`.

Dannenberg, R. (1997a). Machine Tongues XIX: Nyquist, a Language for Composition and Sound Synthesis. *Computer Music Journal*, *21*(3), 50–60.

Dannenberg, R. (1997b). The Implementation of Nyquist, a Sound Synthesis Language. *Computer Music Journal*, *21*(3), 71–83.

De Poli, G., Piccialli, A., & Roads, C. (1997). Granular Synthesis. In *Musical Signal Processing*. Lisse: Swets & Zeitlinger.

Dodge, C., & Jerse, T. (1997). *Computer Music*. New York: Schirmer Books.

Doornbusch, P. (2002). *Correspondence*. Email: 26/8/200, 6/11/2002

Eargle, J. (1995). *Music, Sound, & Technology*. London: International Thomson Publishing Inc.

Eimert, H. (1963). *Epitaph fuer Aikichi Kuboyama*. Wergo 60014.

Evangelista, G. (1991). Wavelet Transforms that We Can Play, In G. De Poli, A. Piccialli, & C. Roads (Eds.), *Representations of Musical Signals*, pp. 119–136. Cambridge: MIT Press.

Fatboy Slim (1998). *The Rockafeller Skank*. Brighton: Skint Records.

Fischman, R. (2003). *Al & Erwin*.
http://www.keele.ac.uk/depts/mu/staff/Al/Al_software.htm.

Fraietta, A. (2002, July). Smart Controller - Artist Talk. In *Proceedings of the Australasian Computer Music Conference 2002*, pp. 149–152, Fitzroy. Australia. ACMA.

Frykberg, S. (1991). Woman and House. Master's thesis, Simon Fraser University.

Frykberg, S. (1998). *Astonishing Sense*. New Westminster BC, Canada: Earsay Productions. Audio CD available from http://www.earsay.com

Gabor, D. (1946). Theory Of Communication. *The Journal of the Institution Of Electrical Engineers*, *93(3)*, 429–457.

Gabor, D. (1947). Acoustical Quanta And The Theory Of Hearing. *Nature*, *159(4044)*, 591–594.

Hamman, M. (1991). Mapping Complex Systems Using Granular Synthesis. In *ICMC Proceedings 1991*, pp. 475–478. ICMA.

Harley, J. (1997). Iannis Xenakis: Electronic Music. *Computer Music Journal*, *22*(2), 75–76.

Healy, R. (2002). *Correspondence*. Email: 29/9/2002, 9/11/2002, 10/11/2002

Hewitt, D. (2002). *Correspondence*. Email: 29/8/2002, 19/10/2002, 6/11/2002

Itagaki, T., Purvis, A., & Manning, P. (1996). Real-Time Granular Synthesis On A Distributed Multi-Processor Platform. In *ICMC Proceedings 1996*, pp. 287–288. ICMA.

Iwatake, T. (1991). *Interview with Barry Truax*.
http://www.sunsite.ubc.ca/WorldSoundscapeProject/barry.html.

Jeans, J. (1947). *Science & Music*. Cambridge: University Press.

Keller, D., & Rolfe, C. (1998). The Corner Effect. In *Proceedings of the XII Colloquium on Musical Informatics*, http://www.thirdmonk.com/Articles/CornerEffect/CornerEffect.html.

Keller, D., & Truax, B. (1998). Ecologically-Based Granular Synthesis. In *ICMC Proceedings 1998*.

Kettle, A. (2002). *Homepage*.
http://home.pacific.net.au/~kettle/history/his.htm.

Knowles, J. (2002). *Correspondence*. Email: 25/8/2002, 29/8/2002, 19/10/2002

Manning, P. (1995). *Electronic & Computer Music*. Oxford: Clarendon Press.

Mel & Kim (1987). Respectable, In *Smash Hits '87*. CBS Productions.

Minogue, K. (1987). Locomotion, In *Smash Hits '87*. CBS Productions.

Moore, F. (1990). *Elements of Computer Music*. EngleWood Cliffs: Prentice Hall.

Olson, M. (2000, April). *Open Source Software: What is the Return on Investment when the Price is Zero?* http://http://sern.ucalgary.ca/courses/SENG/693/F99/readings/Olson_Open%SourcePaper.PDF.

Opie, T. (2000). *Granular Synthesis Resource Web Site*.
http://zor.org/synthesis or
http://www.granularsynthesis.live.com.au.

Opie, T. (2002, July). Granular Synthesis: Experiments In Live Performance. In *Proceedings Of The Australasian Computer Music Conference 2002*, pp. 97–102, Fitzroy, Australia. ACMA.
http://www.iii.rmit.edu.au/sonology/ACMC2002/program.html.

Overholt, D. (2000). *The Matrix*.
http://xenia.media.mit.edu/~dano/matrix/.

Partch, H. (1974). *Genesis of a Music*. New York: Da Capo Press.

Phillips, D. (2000). *Linux Music & Sound*. San Francisco: No Starch Press.

Risset, J.-C. (1991). Timbre Analysis By Synthesis: Representations, Imitations, and Variants for Musical Composition, In G. De Poli, A. Piccialli, & C. Roads (Eds.), *Representations of Musical Signals*, pp. 7–43. Cambridge: MIT Press.

Roads, C. (1985). *Composers and the Computer*, Vol. 12(2). Los Altos: William Kaufmann Inc.

Roads, C. (1987). *New Computer Music*. nscor.

Roads, C. (1991). Asynchronous Granular Synthesis, In G. De Poli, A. Piccialli, & C. Roads (Eds.), *Representations of Musical Signals*, pp. 143–185. Cambridge: MIT Press.

Roads, C. (2001). *Microsound*. Cambridge: The MIT Press.

Roads, C., & Alexander, J. (1995). *Cloud Generator Manual*. http://www.create.ucsb.edu. Distributed with the program "Cloud Generator" for Macintosh Computer.

Rolfe, C. (1998). *MacPod. Real-time asynchronous granular synthesis software for the Macintosh PowerPC*. Third Monk Inc. http://www.thirdmonk.com.

Rowe, R. (2001). *Machine Musicianship*. Cambridge: MIT Press.

Sun, X. (2000). Voice Quality Conversion in TD-PSOLA Speech Synthesis. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 953–956, Istanbul, Turkey. IEEE. http://mel.speech.northwestern.edu/sunxj/.

Sundberg, J. (1991). *The Science of Musical Sounds*. San Diego: Academic Press.

Truax, B. (1986). Real-Time Granular Synthesis with the DMX-1000. In *1986 ICMC Proceedings*, pp. 231–235. ICMA.

Truax, B. (1987). Real-Time Granulation of Sampled sounds with the DMX-1000. In *1987 ICMC Proceedings*, pp. 138–145. ICMA.

Truax, B. (1988). Real-Time Granular Synthesis with a Digital Signal Processor. *Computer Music Journal*, *12*(2), 14–26.

Truax, B. (1990). Composing With Real-Time Granular Sound. *Perspectives Of New Music*, *28*(2), 120–134.

Truax, B. (1994). Discovering Inner Complexity: Time Shifting and Transposition with a Real-time Granulation Technique. *Computer Music Journal*, *18*(2), 38–48.

Truax, B. (1999a). *Granulation of Sampled Sound*. http://www.sfu.ca/~truax/gsample.html.

Truax, B. (1999b). *POD & PODX System Chronology*. http://www.sfu.ca/~truax/pod.html.

Wiener, N. (1964). Spatio-Temporal Continuity, Quantum Theory And Music, In *The Concepts Of Space And Time*, pp. 539–546. Dordrecht: D.Reidel Publishing Company.

Winternitz, E. (1982). *Leonardo da Vinci as a Musician*. New Haven: Yale University Press.

Xenakis, I. (1971). *Formalized Music*. Bloomington: Indiana University Press.

Xenakis, I. (1997). *Electronic Music*. Albany NY: Electronic Music Foundation. Concret PH (1958) Track 2