An introduction to procedural audio and its application in computer games.

Andy Farnell

23rd September 2007

1 Synopsis

Here we'll attempt to define the term "procedural audio" as it applies to computer generated sound effects and music. This has applications in interactive audio systems, particularly video games. First we must distinguish the term from some others that are sometimes used synonymously. Then to put it in context we will consider its history and relation to other procedural media before offering a summary of advantages, disadvantages and possible implementations.

2 What is procedural audio?

Procedural audio is sound qua process, as opposed to sound qua product. Behind this statement lies a veritable adventure into semiotics, mathematics, computer science, signal processing and music. Let us reformulate our definition in verbose but more precise modern terms before moving on. Procedural audio is non-linear, often synthetic sound, created in real time according to a set of programmatic rules and live input. To further define it let's explain it in terms of linear, recorded, interactive, adaptive, sequenced, synthetic, generative and AI audio. Let's also analyse some other phrases like algorithmic composition and stochastic music. By doing so, we may reach an understanding of why procedural audio is so powerful and important.

2.1 Recorded sound

Traditional audio technology has its foundations in recording. Real world sound signals are captured by a microphone, mixed and processed, then mastered into a finished form. A piece of music or sound effects track created this way is fixed. Each time it is replayed the form remains the same. In contrast, procedural audio may be different each time it is played. Recorded audio consists of data where the values are a time sequence of amplitudes, typically about 44,000 per second. The samples are replayed from start to finish in the same order and at the same rate as that in which they were recorded. We call this sampling, a technology that has been common for several decades now. With recording we have always had a distinction between the data and the program or device that replays it. We think of MP3s as the songs and the MP3 player as an application device that reproduces the recorded sound by turning the data back into sound.

Very differently, procedural sound may not need to store any data at all! In fact procedural audio can be though of as just the program, which is another way of saying that the process entirely captures the intended result. The program *is* the music or sound, it implicitly contains the data and the means to create audio. For an introduction see Pohlmann[23].

2.2 Interactive, non-linear and adaptive sound

Because a program it can accept input it's possible to modify the sound output during playback. This is interactive audio. When we play a CD or DVD no further action is needed until the disc has finished. The order of the data is fixed and the procession from one value to the next is automatic. Something that starts at one point and moves at a constant rate to another is deemed **linear**. Alternatively, **interactive**[27] audio uses further input to start new sounds or change existing ones, and the value of the input affects the quality of sound produced. A MIDI piano is a fine example of simple interactive audio. To hear a note you must press a key and the loudness and timbre of the note is determined by the key velocity, how hard you hit it. Clearly a piano consists of more than one sound, taking pitch as a qualifier there are up to 88 different sounds, one per key. Something that can jump between a number of values in any order (discontinuous) or move at different rates or in different directions is **non-linear**. A turntable for playing vinyl is a potentially non-linear device when we do some funky scratching on it. Non-linear devices or programs are capable of being **performed**, they are instruments of a kind. At present all game sounds can be considered as interactive audio (except the music tracks in some cases which are linear). Each sound you hear depends upon a player action or world event. The order and timing of these sounds is not pre-determined as with recorded audio. Interactive applications like games can involve elaborate relationships between user input and audio output, but the common principle which makes the audio interactive is the need for user input. In a video game, certain situations arise which we call **states**. In an attempt to pull the mood of the player along with these states, perhaps representing emotional qualities such as fear in the presence of a monster or triumph at completing a level, music or sound effects are changed. We call this **adaptive** audio. It is a form of interactive sound where a complex function or state machine lies between the players actions and the audible response. Unlike a piano we may not get an immediate response for an input, the result may take some time or further input to yield effects. Normally audio in games is only given as a response to action based on visual play elements, but interactive audio only games for handheld consoles have been demonstrated[4]

2.3 Sequenced sound

Between interactive and recorded sound is what we usually call sequenced sound. This is the method of most music production in use today for genres like hip-hop, rock and pop. The sounds are short recorded clips of individual instruments or vocal lines which are layered together according to instructions stored in a **sequencer**, often as MIDI data. The MIDI sequencer acts as a tool for a composer to re-arrange the recorded parts and then play them back in a fixed

order. Sequenced sound has been used in games for a long time in the guise of "trackers", but this has fallen into disuse for reasons we will examine later.

2.4 Synthetic sound

Synthetic sounds are produced by electronic hardware or digital hardware simulations of oscillators and filters. Sound is created entirely from nothing using equations which express some functions of time and there need not be any other data. Synthesisers produce audio waveforms with dynamic shape, spectrum and amplitude characteristics. They may be generated for sounds corresponding to real instruments like brass or piano, or for completely imaginary ones. The combination of sequencers and synthesisers is mainly responsible for the genre of techno and dance music but can produce ambient backgrounds too. Synthesisers also play a non-musical role for creating sound effects like rain, wind, thunder or just about anything you care to imagine. The power of synthetic sound is that it is unlimited in potential, just as long as you know how to figure out the equations needed for a certain sound.

2.5 Generative sound

This is an abstract term which includes many others like algorithmic, procedural and AI sound. These overlap and are often used as ways of talking about mostly the same thing. For a condensed overview see Jarvelainen [17]. In other words all these things are **generative**, simply because some process *generates* the sound as opposed to a human composing it. The definition is often given that a generative piece requires no input, or the input is given only as initial conditions prior to execution. Analysing many practical implementations this definition breaks down, but if we stick to it strictly then generative sound is *not interactive*.

Generative compositions differ from sequenced ones in that a sequenced composition is laid out in advance and does not change. A generative composition happens as the program runs. The philosophically inclined will wonder whether generative compositions are deterministic. In fact if run on a computer they always are, because computers are deterministic. This is not the same question as whether they are predictable. Something can be deterministic but unpredictable. The answer to this very interesting question depends on whether the algorithm is seeded by truly random numbers. Computers cannot generate truly random numbers, but it is fair to say that a generative sequence can be so complex as to be unpredictable by human perception or reason.

It may include non-computational methods, for example Ron Pellegrino's 1972 processes for "Metabiosis" was an installation with lenses sensing air flow changes using light and photoresistors. Strictly it is not algorithmic, not because it doesn't run on a computer, but because the process does not follow the definition of an algorithm [3]. Whether or not you believe installations that harness environmental unpredictability like Pellegrino's are deterministic depends on whether you believe reality is deterministic.

2.6 Stochastic sound

Some ways of generating sound, often referred to as **stochastic systems**[18] use random or chaotic data. They may filter random data to obtain order through the mathematical rules of statistics and distribution. Alternatively they may use **algorithmic**[26] methods (see below) to generate quasi-random data with a high degree of complexity but well defined distribution. In most stochastic systems the generative part is purely random noise and the distribution is determined by how it is filtered. This follows the **subtractive** model from synthesis, where we start with a lot of data and selectively throw some of it away to get bands of activity where we want. In **generate and test** systems a filter can have a very complex bunch rules to see whether a chunk of randomly generated data should be passed or discarded. Distributions of data are often named according to recognised patterns from statistical theory such as uniform, linear, exponential and Gaussian. Each has particular applications in composition, such as determining note lengths and melodic density [26],[34], or in synthesis for approximating rainfall, city traffic noise and textures for footsteps[10]. Stochastic sound may be generative or interactive since user input can be applied to parameters of the generating equation or to subsequent filters that operate on generated data.

2.7 Algorithmic sound

Algorithmic composition often refers to a process that evolves according to a simple set of easy to understand rules. An algorithm is defined as a set of rules for solving a problem in a finite number of steps. One class of generative sounds are known as the **mathematical methods** which concern finding sequences with useful musical properties by **iterative methods**. This is actually a serious subversion of the common understanding of the word algorithm. With this kind of algorithmic music the "answer" to the "problem" is the steps through which the algorithm goes. We are interested in the partial solutions that happen along the way, not the final result. A better way of describing this pursuit is to consider it as musical applications of sequence. See the online encyclopedia of integer sequences maintained by `AT&T` to get an insight into the extent of this well documented branch of mathematics [32]. In normal computing we want an algorithm to terminate as quickly as possible, to go through the least number of steps and return a value. In iterative algorithmic sound we usually want the opposite effect, to keep the algorithm running through its steps for as long as possible.

Depending on the language used and the program interpreter a few lines of code or even just a few characters can define many hours of evolving sound. Sequences that can be defined by induction and computed with recursive algorithms may only need a pair of numbers to specify their entire range. Like synthesis an algorithmic sequencer uses equations that produce functions of time, but unlike waveforms they are seldom periodic. Instead they are designed to follow patterns that are musically pleasing melodies or harmonies. Algorithms with musical uses leverage complexity (chaos), self similarity and quasi-periodicity to produce patterns that have a degree of order. Fractal equations such as those generating the Mandelbrot or Julia sets, or recursive procedures which generate number sequences like the Fibonacci sequence are common devices.

An important thing distinguishing *synthesis* from *algorithmic sound* is that synthesis is usually about sounds produced at the sample and waveform level under careful control, while algorithmic sound tends to refer to the data, like that from a sequencer, which is used to actually control these waveforms. It refers to music composition where we are interested in the abstract model, the emotional forms given by the rules of harmony, melody and rhythm, rather than in the final nuance of how the sound is rendered to the listener. The same algorithmic source might be hooked up to a string section or brass band with similar effects. See Jacob96[14] for an overview. For extensive resources see AlgoNet[31].

2.7.1 An example of algorithmic sequence

As an example of how a very simple production rule can lead to a complex sequence with a particular behaviour let's consider Collatz "hailstone numbers", also known as the 3n+1 trajectory. Take any number, preferably a quite large one, and if it is even divide by two, if it is odd multiply by three and add one. Now repeat with the new number. Formally:

$$a_n = \begin{cases} a_{n-1}/2 & \text{if } a_{n-1} \text{ is even} \\ 3a_{n-1}+1 & \text{if } a_{n-1} \text{ is odd} \end{cases} \tag{1}$$

If coerced to a range and scale this sequence produces a pair of interesting melodies that converge. Shown in 1 is an implementation in Puredata to generate the next Collatz number in a sequence, the sequence stops at a lower bound of 1 [1].

2.7.2 Markov chains

A Markov chain (Andrey Markov 1856 - 1922) is the name given to a discrete statistical state machine in which the next state is determined by a probability matrix of weights or likelihoods of moving between states in a sequence. The order of a state machine is how many previous state transitions are taken into account when deciding the next transition, in a Markov machine the next state is often independent of previous states. In 2 there are 3 states and a transition may be to either of the other 2 states, or to itself (the state stays the same). If the states represent notes and durations then a probability matrix can create melodies and rhythms. Statistical data can be extracted from existing music in order to easily compose music in a given style.

For example, a Markov machine can quickly be trained to understand 12 bar blues and notice that more often than not a F7 follows a C and that a where a chord has a major 3rd the melody will use a flattened one. This is interesting when data from more than one composer is interpolated to see what Handel, Jean Michel Jarre and Jimi Hendrix might have written together. When encoding more complex musical data such as phrasing, swing, harmony, accidentals and so on, a high dimensional probability matrix grows very quickly and contains a lot of redundancy. They are best deployed in a specific role as just one

[1] Despite the simplicity of this sequence nobody can prove exactly why this happens. Variously known as the Syracuse, Thwaites or Ulam's problem there is still a £1000 reward for a proof that any positive starting value converges on 1.

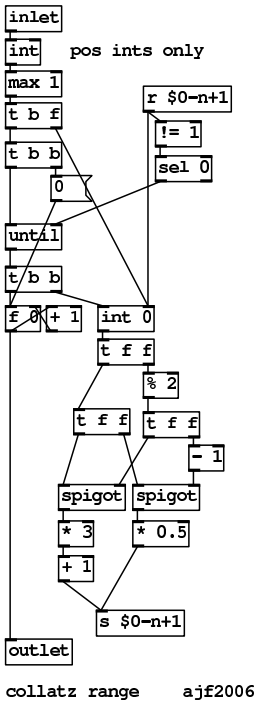

Figure 1: Collatz hailstone sequence in Pd

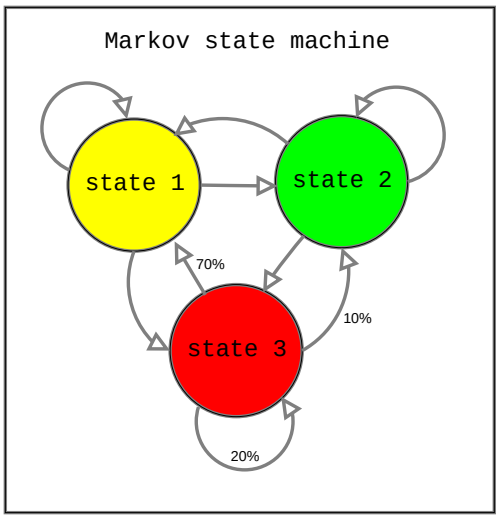

Figure 2: States in a Markov machine

possible tool in conjunction with other methods. The Puredata implementation shown as 3 is the three state example given above.

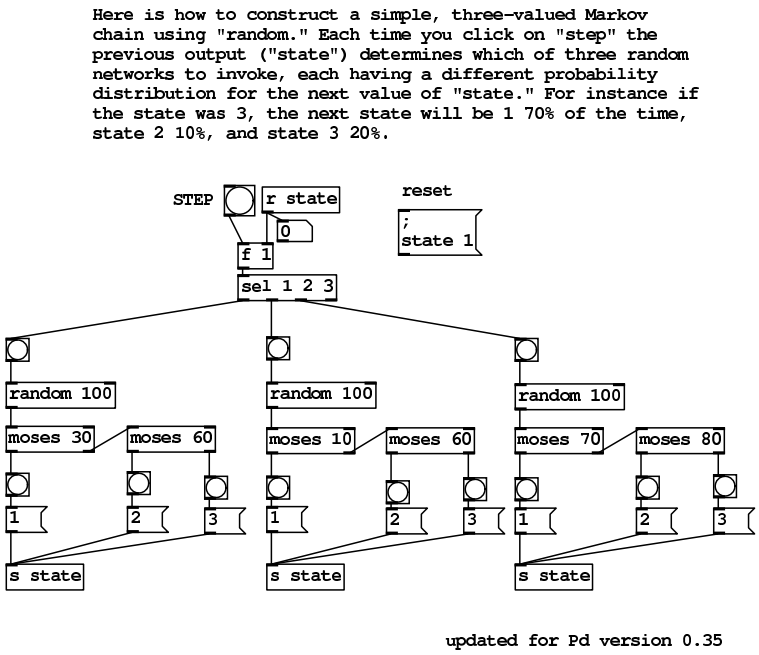

Figure 3: 3 state Markov machine in Pd

2.8 AI sound

A class of algorithmic methods, more complex than the mathematical sequences, are called AI, **artificial intelligence** (something of a misnomer). All algorithms have some kind of memory, to store intermediate variables such as the last two or three values computed, but these are usually discarded immediately to save memory. An AI algorithm is much more persistent, it maintains state and input data throughout its execution as well as constantly evaluating new input. When a generative sequencer is given additional data that corresponds to knowledge, and the ability to make selections based on new input we call this AI composition. The input data is processed by filters and pattern matchers to invoke actions within the system which result in output of sounds appropriate for the input states. Having extra knowledge data might seem to break the definition of procedural sound being purely a program. The question is, to what extent is the data part of the program structure or a separate store of information? Some kinds of AI clearly separate knowledge from process, in others the process is adapted and itself becomes the encoded knowledge.

AI can be broken down into several categories. Adaptive and emergent systems start with a simple initial state or no data at all. Expert systems and Markov chains rely on large amounts of initial knowledge, but Markov chains are not classed as AI because they don't act on input. There are other programs which fall into a grey area between knowledge based AI, cybernetic and stochastic systems such as variable state machines, cellular autonoma, symbolic grammars and genetic algorithms.

2.8.1 Expert systems

Expert systems have a book of rules which they can look at to make decisions. But this knowledge is not fixed. What defines the system as AI is that it may revise its knowledge. Musical knowledge about harmony, modulation, cadence, melody, counterpoint and so forth is combined with input data and state information in an adaptive AI composition tool. This works best for the kind of data that can be expressed as "facts" or "rules", when A happens then B should happen etc. What the expert system is doing is solving a problem, or question. We are asking it to output music that best fits the given input requirements. The knowledge may be very general or it can conform to specific genres such as jazz or rock music. Processing may be freeform so that the AI program develops its own styles and tricks, or very tightly constrained so that it only produces cannons in the style of Bach. What makes an expert system interesting is the explicit way it models human problem solving. It is goal driven and has a solid understanding of where it is trying to get to, including sub-problems and sub-sub-problems. The core is called an inference engine, it takes a scenario and looks at schemas that fit known solutions, maybe trying a few of the ones it thinks look best first. Sometimes these lead to other problems that must be solved. In the course of traversing this knowledge tree it sometimes finds contradictory or incorrect schemas that it can update. The resultant behaviour is potentially very complex. Proper AI music systems are generally built on existing AI frameworks. Expert systems shells are available off the shelf ready to be filled with musical knowledge, or any other kind. They are general purpose software components with well understood behaviours, equally adaptable to solving engineering and biology problems as music composition. For detailed discussion of musical applications of ES see Cope92[8], Cope87[7].

2.8.2 Neural networks

Other kinds of AI systems, much more like the human brain, are neural networks. In a neural network knowledge is stored within the program structure as weights or numbers that determine how likely a given output is for a given input. Initially the network is blank, like a baby with no understanding of the world. Over time it is trained by providing it with examples. An example consists of an input pattern and and an expected output. Feedback systems within the network "reward" or "punish" it according to how well it gets the answer right. Eventually the network converges on an optimal set of weights so that it gives a correct answer each time. Examples of use might be training it to produce counterpoint for a melody, or complete a chord cadence. Neural networks are best at solving very narrow problems in a fuzzy way. A neural network, once trained in picking harmonies will be quite useless at another task. They are single minded and do not multitask well. Their fuzziness is their amazing strength, from which "true AI" seems to emerge. Those who play with neural networks will tell you that their output can seem quite spooky or uncanny at times. In contrast to expert systems which are quite brittle and tend to freak out when given unusual or unexpected input they are able to interpolate and extrapolate examples. For instance, trained in some simple examples of melody and then given input which has never been seen before, a neural system can produce what it thinks is the best fit, often a musically correct or quite creative production.

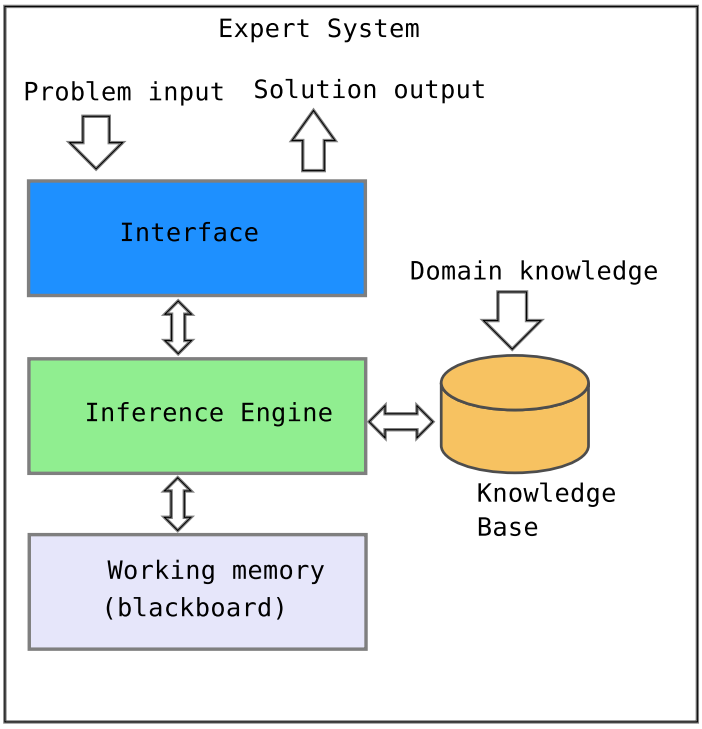

Figure 4: Expert System

They are able to find hidden data which is implicit in a set of examples. Work on neural models of sound is extensive, see Griffiths bibliography [11]. For a comprehensive FAQ sheet on NN technology see Sarle et al [28].

2.8.3 Cellular autonoma

Automata are a collection of very simple self contained programs that live in a bounded environment and interact. Each is a state machine that depends on a matrix of rules, similar to the Markov state machine, but also on input from the environment. The environment can consist of input data from other nearby automata. The emergent intelligence comes from how many of them work together as their respective outputs affect the inputs of others. The original *"game of life"* by J.H.Conway is rooted in finite set algebra [6]. Say that we set a measure of happiness or sadness to each individual in a group of cells that can reproduce. It will depend on how close they are to other cells. Then we specify that too many cells close together will annoy each other and commit suicide while completely isolated cells will die of loneliness. Now something interesting happens. The cells start to move about in patterns trying to build an optimal social arrangement. Hooking these cells up to musical notes can produce interesting compositions[24].

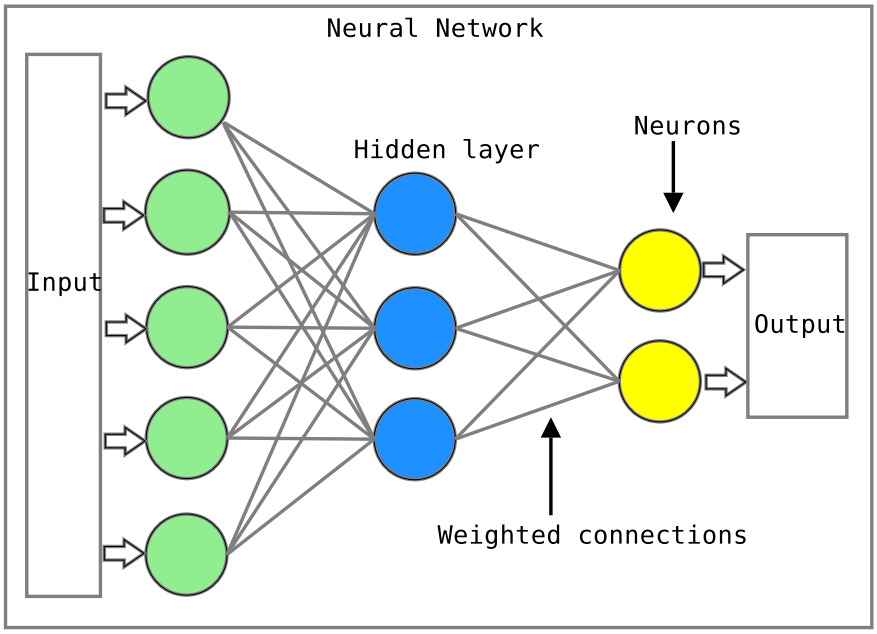

Figure 5: Neural Networks

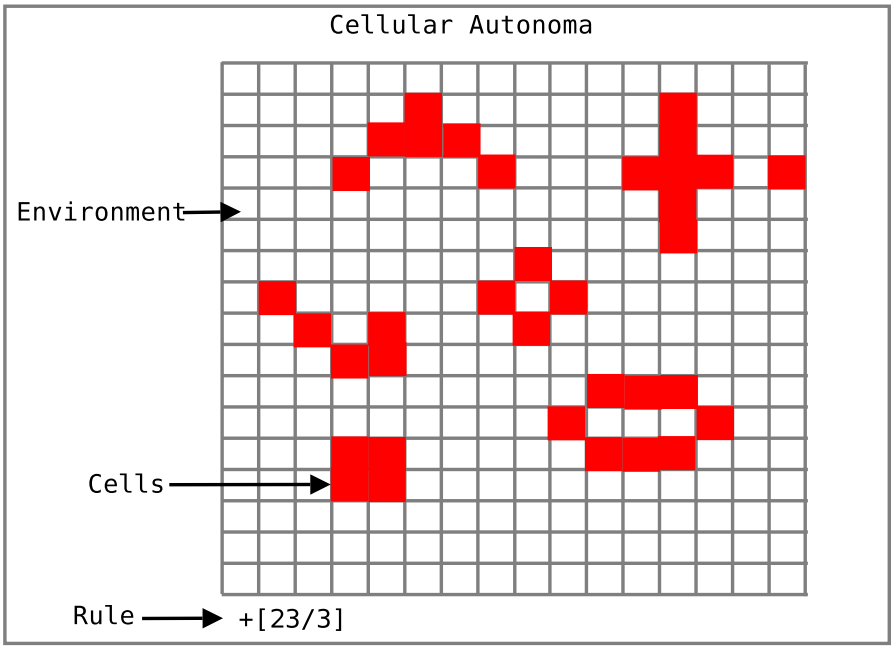

Figure 6: Cellular Autonoma

2.8.4 Genetic algorithms

Genetic algorithms are a way, often a very slow way, of generating and selecting things. In this case we are selecting other algorithms. The idea is similar to cellular autonoma except the cells are far more complex and share much in common with artificial lifeforms. Like neural networks they start with little or no knowledge other than an initial state and some target goals, then they adapt and evolve towards those goals by improving themselves to best cope with an environment which represents the input state. Usually the implicit goal is just to survive, because we will make our selection based on the ones remaining after a period of time. These algorithms, patterns, mini programs, or whatever we wish to call them have certain characteristics like lifeforms. They reproduce and pass on genetic characteristics to their offspring, which can occasionally mutate at random. They die before a maximum lifespan expires. They are affected by their environment. If they were plants with leaves and roots in an environment that was randomly sunny, rainy and windy we might see that when it's too sunny the ones with big leaves prosper for a while because they can photosynthesise, but then die quicker because their leaves have a big surface area. When its windy the ones with big roots survive longer and so on. Eventually plants emerge that are best adapted to the environmental input conditions. If the plants represent musical phrases and the environment represents states in the game that correspond to emotive themes then we would hope to get phrases best adapted to the input. The trick is to choose the appropriate initial rules of life, the right environmental conditions and leave the system running for a long time. The we "harvest" the good ones to use generatively in later simulations[15].

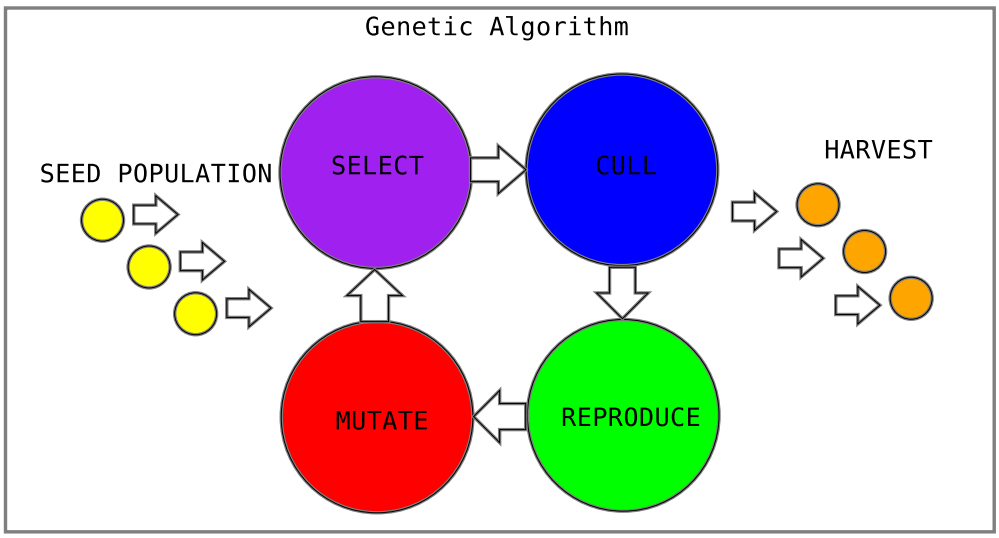

Figure 7: Genetic Algorithm

3 Defining procedural audio

Let's take stock of what we know so far. One definition of "Procedural" is:
"Relating to or comprising memory or knowledge concerned with how to manipulate symbols, concepts, and rules to accomplish a task or solve a problem"
The "problem" is producing sound that fits one or more constraints. Memory can exist in the form of stored knowledge in an expert system, weights in a neural network or Markov machine, genetic code in an adaptive system or just the last value of a register in an unstable recursive filter. Symbols and concepts are all about semantics, the meaning we give to the input, internal states and output of the system. The system doesn't really care about these, they are for our human benefit and may as well represent hatstands, badgers and windmills as far as the program cares. The important parts are really the input and output and that the mapping makes sense in terms that are understandable and useful. Input can represent a physical quantity like velocity or a game state like the approach of a level 4 boss. Output is either directly audio signals or control signals for an audio source (which may itself be procedural).

3.1 Combining definitions

So "procedural audio" is a term that combines all these last five definitions in some degree to mean a system that has a complex internal states with memory and goals, designed to produce a range of sound outputs for a range of inputs. This is about as general as it gets, it's the description of a general purpose computer or cybernetic system. Perhaps this is why the term is often abused. We really may as well say "computer sound". In fact it's probably better to describe procedural sound by what it is not. It is not pre-sequenced, pre-recorded sound and music.

It's worth noting that we usually reserve the term for systems that work in real time. Many musical control and synthesis programs take a very long time to produce any output and can only render audio in offline mode. Offline systems can use any of the methods considered above, but we are interested in real-time ones that can provide responses in seconds for the case of composition, or microseconds for the case of synthesis. Procedural audio behaves like an undefined recording, you start the program running, usually with some parameters to determine the kind of output you want, and it either immediately begins to produce sound and music that changes continuously over time, or it responds to input by creating sound and music. Obviously, whether a procedure will run in real-time or not depends on the hardware platform so it's nonsense to talk about a program being a real-time procedure without considering where it will run.

4 A brief history of procedural media

Procedural media has been a part of computer games since the very beginning. The early days of computer games were a time of great ingenuity and in order to squeeze large amounts of level data into tiny memory footprints[21] the pioneers of the demoscene exploited generative fractal geometry, reducing seemingly endless landscapes and mazes into a handful of equations written in

skillfully optimised C code[12]. This can produce seemingly infinite universes [20]. One team of pioneers, Bell and Braben produced Elite [2] employing procedural universe generation for a space game. Early examples can be seen on Robert James site [16]. Landscape generation an interesting subject. When I built levels for the Unreal game back in the 1990s it quickly became apparent that random meshes, even when filtered nicely were not at all natural for terrain, so generative methods using predictable and properly contained functions became part of the toolkit to create meshes for UnrealEd, using its Java like UScript capabilities for geometry builders. Today no games world editor would be complete without a natural terrain generator that can produce fantastic but believable natural landscapes modelling erosion and displacement by rivers and glaciers, and applying constraints to ensure that passable routes always exist between specified points[29]. See the Gamasutra article by O'Neil[22] for a practical summary. Plants[9] and even animals[33] have long had procedural generators, one well known algorithm is L-system branching for creating trees. Textures such as fire, water and brickwork are also produced procedurally.

The problem for procedural sound is that while fractal methods lead to meaningful geometry in 3D space they don't necessarily map onto meaningful sound, which is either a very high dimensional space, or the same folded into a 1 dimensional time function depending on how you look at it. Therefore there aren't number sets which just happen to contain the sound of a car engine or blackbird song, or if there are we currently have no way of searching the space of generated sets to find them. Producing sound requires more traditional mathematical approaches based on wave equations, calculus, linear algebra and operator theory, it is essentially the mathematics of old school engineering and physics.

It's important to point out another difference between procedural audio and synthetic audio, which is that synthetic audio is not necessarily compact. A method such as additive synthesis requires a huge number of parameters to approximate a sound, in the limiting case it requires exactly the same amount of data as if the sound were sampled, so that the data has simply been moved from the time domain to the frequency domain. That doesn't mean that efficient and compact representations don't exist, simply that they are more difficult to obtain. Procedural synthetic sound attempts a high degree of efficiency. It attempts to reduce the large amounts of data into a few rules. It can do this using similar techniques to fractal geometry, or by using cellular autonoma such as the Chaosynth built by Eduardo Miranda. It can also achieve efficiency by simplification, employing heuristic physical approximations that hit on just the important psychoacoustic cues as in my own work with simplified synthetic sound effects. But in all cases it relies on a careful partition between the synthesis model and the synthesis method. The former is the mapping that makes sense of the parametric data derived from input, and the latter is the DSP system which produces the waveform data. As a 3D graphics analogy, one is the matrix that represents a mesh and the other is is the rendering program that turns this into a meaningful image for the human visual senses.

5 A brief history of game audio

The history of game audio has completed one full cycle. In the beginning game sound effects and music used special sound effects chips such as the AY-38610 or SID[5] to produce waveforms, ADSR envelopes and noise bursts. The logic for these "synthesiser on a chip" was hybrid analog/digital, having a rather unique sound of its own, albeit very grainy. Compositions were often written very compactly as permutations on scales or modulus/retrograde systems like in 12 tone music. Key changes and parallel melodies are a strong feature of early game compositions because of their efficient representation. Famous pieces of music and backgrounds like Pac-Man and Space Invaders will be familiar to older gamers. Those were sequenced using time-parameter lists that predate MIDI. At this time soundcards were not ubiquitous and so the sound was often output though onboard computer loudspeakers or encoded into the video signal for use with a normal TV set.

5.1 Infancy of digital game audio

Throughout the evolution of the PC during the 1980s soundcards and sampling technology were introduced and the dedicated sound chips were replaced by digital audio handled by the main CPU. It was sampled at incredibly poor bit depths and rates like 8 bit, 11KHz. To begin with only one or two channels of sound were available, but as computing power grew software mixers capable of delivering 4, then 8, 16 and 32 channels of sound emerged. Through the late 1980s and early 1990s computer game music and sound effects mirrored the developments in pro studio production, but lagging behind state of the art samplers and synthesisers by a few years because of the difficulty in running audio software on an ordinary CPU while also delivering graphics. Commercial synthesisers and samplers employed dedicated DSP units and were designed with direct memory access and data pipelines unlike a general purpose computer. Finding it's own optimal form, computer game music became sequenced samples using quite sophisticated tracker formats, a file that contains sample data and sequencing information. In some ways this represents a high point in game music because of the flexible real-time control the game logic had over the music sequences.

5.2 The Middle Ages

From the late 1990s game music moved to recorded tracks on CD. But memory based audio systems were not capable of reproducing full length high quality music. At this time soundtracks were stored on the CD media and played back in exactly the same way as a regular CD during play. This is the time when so called "audio engines" began to appear. An audio engine, as the term is used today, is really an elaborate buffering and memory management system. Early audio engines did not cater for streaming music so the playback of game music relied on a direct connection between the CD-ROM and the soundcard. Now, a powerful game console or PC can load several minutes of audio into memory in a second, replay hundreds of channels simultaneously from RAM or stream and mix many audio channels from secondary storage using DMA. This is partly due to the advantages of compressed audio like FLAC, MP3 and Vorbis and

hardware decompression, but primarily its a simple function of improvements in system power. Presently, a game sound system has the capabilities of a professional sampler from the turn of the century. It can play back hundreds of simultaneous voices, apply envelopes, filtering, panning and surround sound localisation, modulation and reverb to the stored samples. But game sound is still about 5 or 6 years behind the state of the art in DSP. Although technologies like Puredata and Csound have been around for a long time this level of DSP capability has not yet been adopted as standard in game sound. Tools like FMOD [30] and Wwise [1] are presently audio delivery frameworks for pre-recorded data rather than real "sound engines" capable of computing sources from coherent models. This is starting to change now, and it brings the history of game audio full cycle.

5.3 The Renaissance

With powerful native DSP capabilities a renaissance in procedural and synthetic audio will revolutionise game play and allow the lost magic of synthetic sound effects and tracker style interactive/adaptive music to come back. Sequencing and sound effects generation will move out of the studio and back onto the platform. Intelligence can be applied to music scoring in real-time. Already the game Spore has commissioned Brian Eno, a long time champion of procedural composition, to write a soundtrack. The wheel has turned. It is interesting for me as a programmer to witness one full cycle of the so called "wheel of life" that we were taught in computer science classes at university. The forces driving development tend to migrate specialised functions into dedicated hardware which are then slowly subsumed back into the native CPU over time. It has taken 20 years for the capabilities that were available in the SID soundchip to be available again to the game audio programmer, only now they are a vastly more powerful. For the programmer and content designer this has massive advantages. In the absence of specialised proprietary devices with unusual instruction sets, code for synthesisers and effects written in ordinary C or C++ such as Perry Cooks STK or Puredata externals can be made portable. This leads to a common exchange form for synthesists and sound designers. New standards are being discussed for a common XML based exchange format for dynamic audio which builds on the apparently abandoned MPEG4-SA (structured audio) protocol based on Csound, which was probably too far ahead of its time.

6 Implementing procedural game audio

What does this mean for composers, programmers and sound designers as procedural audio comes back into focus? What are the benefits and disadvantages of procedural audio? And what are the technical, political and institutional challenges it faces in the next decade of interactive sound? To answer these questions we need to consider the state of the art in game audio production and examine what problems will be solved or created by introducing high power real-time audio DSP capabilities to the gaming platform.

6.1 The state of the art

If we are to coin a term to highlight the difference between the established model and procedural model then let's call the current way of doing things the "data model". It is data intensive. Massive amounts of recorded sound are collected by sound designers, trimmed, normalised, compressed, and arranged into sequence and parameter tables in the same way that multi-sample libraries were produced for music samplers.8 They are matched to code hooks in the game logic using an event-buffer manager like the FMOD or Wwise audio system and some real-time localisation or reverb is applied. The data model sits nicely with the old school philosophy from the film and music business which is principally concerned with assets and ownership. The methodology is simple, collect as much data as possible in finished form and assemble it like a jigsaw puzzle, using a hammer where necessary, into a product. At each stage a particular recording can be ascribed ownership by a game object or event.

There are many drawbacks to this. One is that it forces aesthetic decisions to be made early in the production chain and makes revisions very expensive. Another that we have already touched upon is the colossal amounts of data that must be managed. Therefore existing game audio systems are as much asset management tools as they are data delivery systems. One very important point is that this approach is incoherent. Sound has long been treated as an afterthought, added at the end of game development, so much that this attitude is reflected in the tools and workflows that have evolved in game development industries. In fact for most cases other than music delivery it is completely unnatural to treat sound as separate from vision. Unlike film where location sound must be replaced piece by piece with "Foley" and "wild effects" games are inherently coherent. They are based on object oriented programming where it is natural to treat the sonic and visual properties of an object as attributes of the same thing. The present way of doing things forces a partition between the design of objects visual and sonic properties and in turn leads to more work tweaking and experimenting with sound alignment or when revisions are made to assets.

6.1.1 Game audio today

Let's take a quick look at some aspects of current technology and methods.

Switching When an object comes into play, either because it comes within range of the player or acquires relevance, it must be activated. This may involve a prefetch phase where a soundbank is loaded from secondary storage. Although modern game sound systems have hundreds or even thousands of channels it is still necessary to manage voice playback in a sensible way. Like the polyphony assignment for traditional samplers a game audio system prioritises sounds. Those that fall below an amplitude threshold where they are masked by others are dropped and the object instance containing the table replay code is destroyed. Activation may be by triggers or events within the world.

Sequence and randomisation Composite or concatenated sounds may be constructed by ordering or randomly selecting segments. Examples are footsteps or weapons sounds that comprise many small clicks or discrete partials in

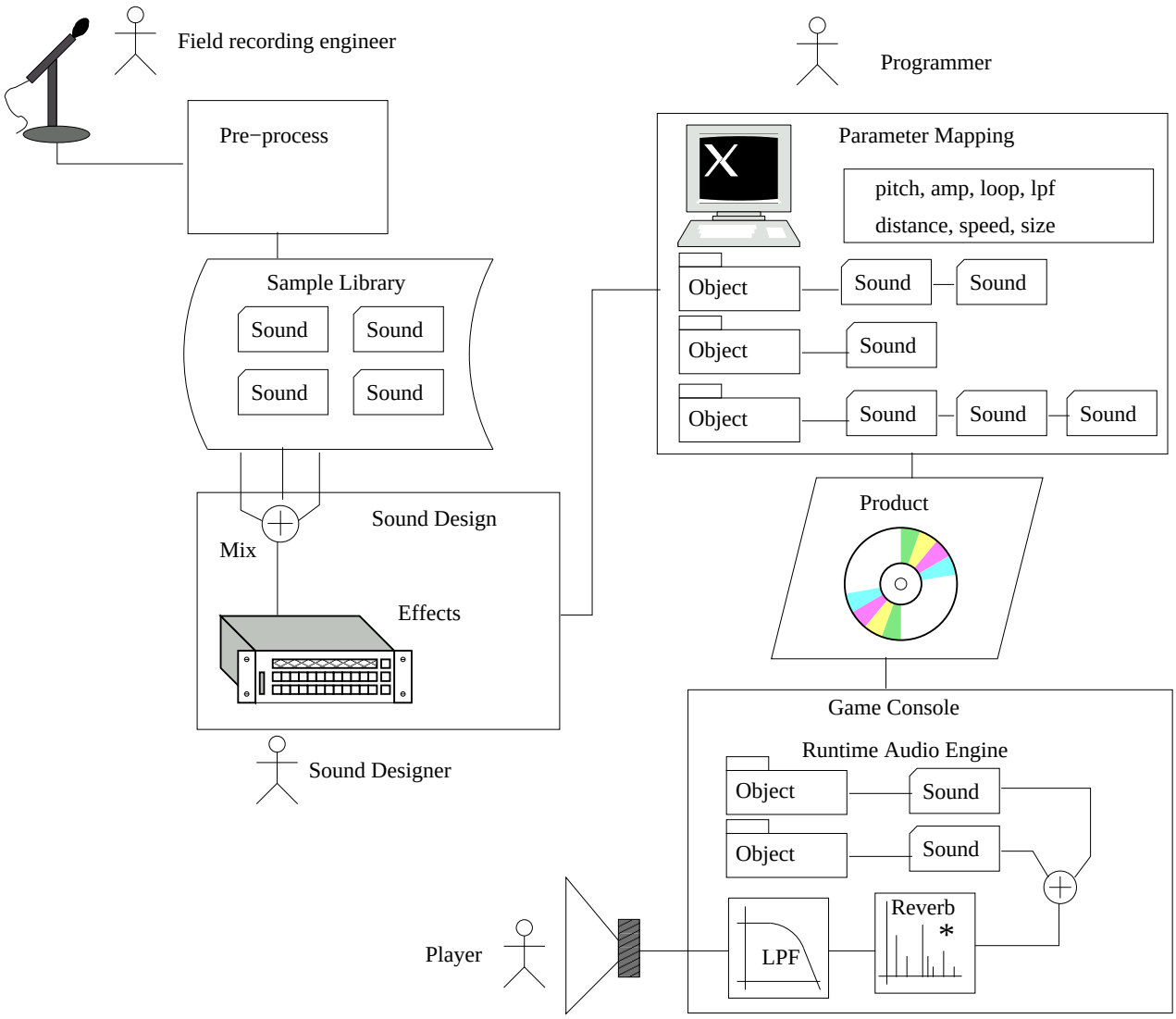

Figure 8: Use of recorded audio in todays games

combination.

Blending Crossfading and mixing of sample data is very much like a normal sampler. Velocity crossfades for impact intensity are really no different from a multi-sampled piano.

Grouping and buses Most game audio systems incorporate a mixer much like a traditional large frame multi-bus desk with groups, auxilliary sends, inserts and busses. The difference between a digital desk used for music and one used in game audio is more to do with how it is used. In traditional music the

configuration of the desk stays largely the same throughout the mix of a piece of media, but in a game the entire structure can be quickly and radically changed in a very dynamic way. Reconfiguring the routing of the entire mix system at the millisecond or sample accurate level without clicking or dropouts is the strength of game audio mixers.

Real-time controllers Continuous real-time parameterisation from arbitrary qualities can be applied to a sound source. Object speed, distance, age, rotation or even temperature are possible. Presently these are usually routed to filter cutoff or pitch controls, the range of dynamic real-time control for non-synthetic sounds is quite poor.

Localisation Simple panning or inter-aural phase shift according to head transfer response is applied to the sound in order to place it perceptually for the player actor. Relative actor speed, orientation and the propagation medium (air, fog, water etc) all contribute to how the sound is received. This is closely connected to "ambiance" below.

Ambience This is an extension of localisation which creates much more realistic sound by contextualising it. Reverb, delay, Doppler shift and filtering are applied to place point sources or extents within the environment. Echos can be taken from the proximity of nearby large objects or world geometry so that sound sources obtain natural ambience as the player moves from outdoors, through a forest, into a cave and then into a corridor or room.

Attenuation and damping This is directly linked to distance but may also apply filters to affect fogging (absorbtion), or material damping caused by intervening objects that occlude sound. Localisation, ambience and attenuation are all really aspects of the same process, placing dry discrete sources or extents into a natural sounding mix.

Replication and alignment If we ignore Einstein for a moment and assume the existence of a synchronous global timeframe then networked clients in a multi-player game would all march like an army in lockstep. In reality clients do not follow this behaviour, they are more like a loose crowd following along asynchronously because of network latency. The server maintains an authoritative "world view" which is broadcast to all clients. This data may include new objects and their sounds as well as time tagged packets that indicate the relative rather than absolute timing between events. It is necessary to reschedule some sound events pushing them forwards (if possible) or backwards a few milliseconds to make them correspond to visual elements. Without this, network jitter would scramble the sequence and timing of events so variable delays are used to align sounds back to correct object positions or states which are interpolated on the client.

Music dialogue and menus These are often given special treatment having their own groups or subsystems. Dialogue is often available in several languages which can contain sentences of differing length or even an entirely different semantic structure. Where music is dynamic or interactive this is currently

achieved by mixing multitrack sections according to a composition matrix that reflects emotive game states. Short musical effects or "stings" can be overlaid for punctuation and atmospheres can be slowly blended together to affect shifting moods. Menu sounds require a separate code environment because they exist outside the game and may continue to be used even when all world objects, or the level itself has been destroyed.

6.1.2 Procedural alternatives

How do new procedural methods fit into or modify this methodology? The difference between the foregoing and what we are presently considering is that procedural sound is now determined by the game objects themselves. Sounds heard by the player actor very closely reflect what is actually happening in the game world, being more tightly bound to the physics of the situation. Procedural audio relies much more on real-time parameters than discrete events. Instead of selecting from pre-recorded pieces we generate the audio at the point of use according to runtime parameters from the context and object behaviours. A fairly direct analogy can be drawn between the rendering and lighting of 3D scenes compared to static texture photographs, and the synthesis of procedural audio compared to using recorded samples. In each case the former is more flexible but more CPU intensive. Another difference is the granularity of construction. Instead of large audio files we tend to use either extremely small wavetables or none at all where sources are synthesised.

component construction Take for examples a gun and a car engine. Traditional methods would require at least two sounds, sampled, looped and processed, tagged and mapped to game events. In fact these are very similar physical entities. For the gun we model a small explosive detonation of the shell and apply it to the formant and radiance characteristics for the stock, magazine and barrel. These each impart particular resonances. Similarly an internal combustion engine is a repeated explosion within a massive body connected to a tubular exhaust. Standing waves and resonances within the engine are determined by the engine revs, the length and construction of the exhaust and the formants imparted to the sound by the vehicle body. A synthetic sound designer can create the sounds for most objects assembling efficient physical models from reusable components 9.

dynamic LOD Instead of simply applying filters to attenuate recorded sources we are able to rather cleverly tailor a procedural synthetic sound to use less resources as it fades into the distance. Think of a helicopter sound. When in the distance the only sound audible is the "chop chop" of the rotor blades. But as it approaches we hear the tail rotor and engine. Similarly the sound of running water is very detailed pattern of sinewaves when close, but as it fades into the distance the detail can be replaced by cheaply executable noise approximations. In fact psychoacoustic models of perception and Gabors granular theory of sound suggest this is the correct way to do level of detail, making sounds with less focus actually consume less resources is merely a bonus from a computational point of view. This can lead to perceptually sparser, cleaner mixes, without the "grey goo" phenomena that comes from the superposition of an overwhelming number of channels of sampled audio.

DSP chain reconfiguration Extending the highly dynamic structure of existing mixers we now use DSP graphs that are arbitary. A fixed architecture in which sources flow through plugins in a linear fashion is too limited. The ability to construct an FM synthesiser or formant filter bank with a variable number of parallel bands requires a very flexible DSP engine, a real *SOUND ENGINE*. This is much more like a sophisticated synthesiser than a sampler. Software architectures that already have this behaviour are Reaktor and Max/MSP/Puredata. One essential feature of these tools is that they give a visual representation of the DSP graph which is easy to program, which allows sound and music designers who are not traditional programmers to speedily construct objects for real-time execution.

musical aspects Instead of delivering mixed tracks or multitrack stems for clientside mixing the composer now returns to the earlier tracker style philosophy (only in a more modern incarnation with XMF, DLS2, EAS type formats). Music is delivered in three parts, MIDI score components that determine the themes, melodies and transitions of a piece, a set of "meta" data or rules for assembling scores according to real-time emotive states, and a set of instruments which are either multi-sample library banks or synthetic patches. The composer is no longer concerned with a definitive form for a piece, but rather with the "shape" and "boundaries" of how the piece will perform during play. Chipset MIDI on some soundcards has improved greatly in recent years reaching par with professional samplers and synthesisers, but for platform independence the most promising direction is with native implementations.

6.2 Forces driving procedural audio

6.2.1 Order of growth

As games worlds get bigger and the number of world objects increases the combinations of interactions requiring a sound explodes. Unlike textures and meshes which have a linear $\mathrm{O(n)}$ relationship to world objects, sound is interactive and grows in accordance with the relationships between objects. This is polynomial growth in practice, but theoretically worst case is closer to factorial. A problem many game audio directors are currently complaining of is that even with a large team of sound designers they are unable to generate sufficient content to populate the worlds. Games worlds are only going to keep growing fast so procedural synthetic audio that allows sounds to be automatically generated for all possible object-object interactions is very attractive. Instead of exhaustively considering every possible drop, scrape and impact noise the games world is automatically populated with default interactions. The sound designer can now concentrate on using their real skills, to tune the aesthetic of critical sounds, or replace them with samples and forget about the drudge work of creating thousands of filler noises. This is very liberating for the sound designer.

6.2.2 Asset control

In terms of project management the data model means that large amounts of data must be taken care of, so part of the audio teams job is to keep track of assets and make sure there are no missing pieces. Asset management is now one

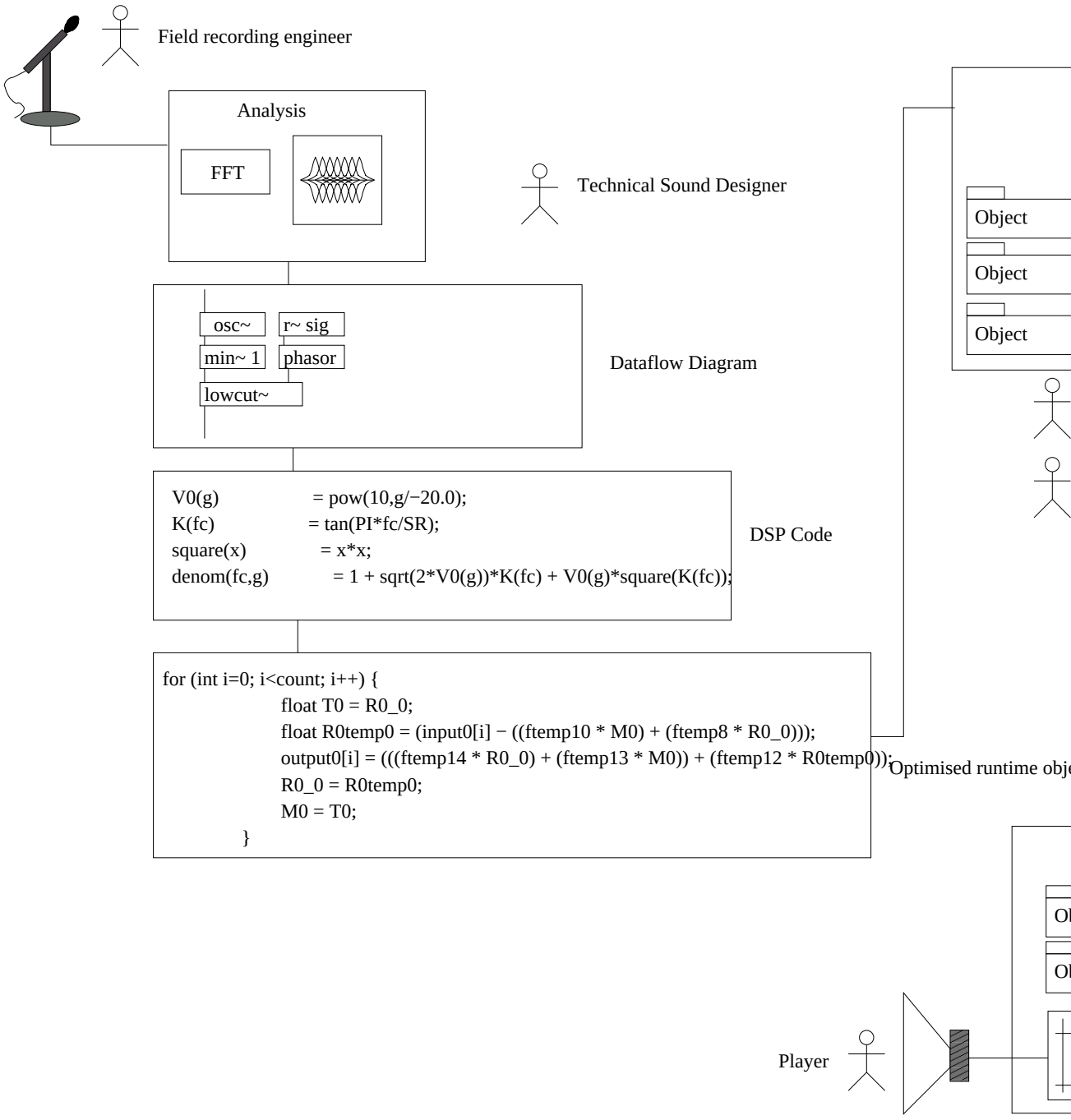

Figure 9: Use of synthesis in procedural sound effects

of the biggest challenges in game audio. Each game event, object pickup, door opening, car engine or whatever must have it's own piece of sound data kept on secondary storage, usually as .wav files, logged and cross-referenced against

event lists. In contrast procedural audio requires management of code, or rather sound objects, which happen to be the world objects. This brings the view of the all media in the project under one roof. These have enormous scope for reuse and arrangement in a class hierarchy that tends to manage itself. More specific objects may be derived from general ones. Instead of gigabytes of raw audio data the project manager must now be able to deal with more compact but more complex systems of parameters that define each sound object.

6.2.3 Data throughput

With the data model, each recorded sample must be brought from secondary storage into main RAM or directly via a data pipeline to the sound core during play. This means that the data model is "data bus intensive". A problem that occurs in current game audio that holds it back is that sound data must compete with graphics data, also a data intensive operation. In contrast, procedural audio is CPU intensive. The instructions necessary to generate procedural audio are very small. Extremely compact programmatic data has almost negligible load on the data bus. Instead of large DMA transfers the CPU works to compute the data in real time. This radical change has important effects on system architecture and program design. One immediate advantage is that it frees up great amounts of data throughput for other uses.

6.2.4 Deferred form

The data model requires that most of the work is done in advance, prior to execution on the platform. Decisions such as sound levels, event-sound mappings and choices of effects are made in advance and cast in stone. Procedural audio on the other hand is highly dynamic and flexible, it defers many decisions until runtime. Data driven audio uses prior assignment of polyphony limits or priorities for masking, but dynamic procedural audio can make more flexible choices at runtime so long as we satisfy the problem of predicting execution cost. This means that critical aesthetic choices can be made later in the process, such as having the sound mixers work with a desk "in-world" during the final phase of production, much like a film is mixed. They can focus on important scenes and remix the music and effects for maximum impact. With runtime dynamic mixing it is possible to "set focus" on an object that the player is looking at, or a significant actor that requires highlighting in context.

6.2.5 Object based

In terms of how the sound designer and programmer work, procedural audio heralds the creation of a new class of audio producer/programmer. Instead of working with collected data and a fairly naive sample replay engine the sound designers focus shifts from specific instances of sound to the behaviour and physics of whole classes of sounds. The designer now works more like a programmer creating and modifying sound objects. A sound object is really a method of an existing world object, thus the false divide between visual and audio mediums is bridged. Simple operations such as scaling can be automatic. For example, if the sound designer works on an object that models a metal cylinder, and if a tin can is scaled up to a steel barrel the sound may scale as easily as the 3D

artist changes the size of an object mesh. If the barrel is now textured with a wood material the sound automatically changes to become a wooden barrel. For a great many objects the designer is concerned with physical and material properties and geometry, the actual sounds are an emergent property of the work of the sound designer and the 3D object designer who work more closely together. In the best case this direction practically eliminates audio middleware tasks that perform explicit mapping, for example instead of manually specifying footstep sounds for each texture the appropriate choices are made automatically from texture material properties, after all the sound is an object attribute of the surface-foot interaction and if the texture is correctly labelled by the texture artist and the speed and weight of the actor are known then footfalls can be correctly synthesised directly (often using granular methods).

6.2.6 Variety

Further advantages of procedural audio are versatility, uniqueness, dynamic level of detail, focus control and localised intelligence. Let's consider the first of these for a moment. As we mentioned above, a recorded sound always plays precisely the same way. Procedural sound may be highly interactive with continuous real-time parameters being applied. Generative music for example can change its motifs, structure and balance to reflect emotional dimensions. The sound of flying bullets or airplane propellers can adapt to velocity in ways that are impossible with current resampling or pitch shifting techniques. Synthesised crowds can burst into applause or shouting, complex weather systems where the wind speed affects the sound of rainfall, rain that sounds different when falling on roofs or into water, realistic footsteps that automatically adapt to player speed, ground texture and incline, the dynamic possibilities are practically endless. We will consider dynamic level of detail shortly because this is closely tied up with computational cost models, but it is also related to dynamic mixing which allows us to force focus in a sound mix according to game variables.

6.2.7 Variable cost

Playing back sample data has a fixed cost. It doesn't matter what the sound is, it always requires the same amount of computing power to do it. Procedural sound has a variable cost, the more complex the sound is the more work it requires. What is not immediately apparent is that the dynamic cost of procedural audio is a great advantage in the limiting condition. With only a few sounds playing sampled methods vastly outperform procedural audio in terms of cost and realism. However as the number of sounds grows past a few dozen the fixed cost of samples starts to work against it. Some procedural sounds are very hard to produce, for example an engine sound, while some are extremely easy and cheap to produce, for example wind or fire sounds. Because of this we reach a point in a typical sound scene where the curves cross and procedural sound starts to outperform sample data. What makes this even more attractive is the concept of dynamic level of detail. A sampled sound always has the same cost so long as it is playing, regardless of how much it is attenuated. In mixing a sound scene LOD is applied to fade out distant or irrelevant sounds, usually by distance or fogging effects that work with a simple radius, or by zoning that attenuates sounds behind walls. Until a sampled sound drops below the hearing

or masking threshold it consumes resources. Research at on dynamic LOD techniques has shown how a synthetic source can gracefully blend in and out of a sound scene producing a variable cost. We can employ physcoacoustic, perceptual methods to constructing only the parts of the sound that are most relevant [13], or cull unneeded frequencies in a spectral model [25]. What this means is that for a complex sound scene the cost of peripheral sounds is reduced beyond that of sampled sources. The magic cutoff point where procedural sound begins to outperform sampled sources is a density of a few hundred sources.

6.3 Factors against procedural audio

Several problems seem to stand against procedural game audio. Dynamically generated and synthetic sound is not a panacea, indeed there are areas where it fails and will never replace recorded sound. It still faces some practical software engineering issues, such as how to manage phasing in procedural methods alongside existing ones. Some of these to consider briefly are realism, cost, training, conflict with established methods and impediments to research and development.

6.3.1 Aesthetic

For sound effects realism is an interesting issue because it seems to fall into the political rather than technical category. Extremely high realism is possible but it cannot be pursued until the political obstacles to further development are removed. Once games did not have super 3D graphics, early titles like Wolfenstein and Quake were basically box walled mazes covered in low resolution textures. Synthetic sound is stuck at an equivalent stage of development, mainly because it has been excluded and neglected for 15 years. However these early games competed in the same market as one of the most visually interesting releases ever made. Myst presented a series of 2D and layered 2D images that were hyper-realistic fantasy landscapes created in Photoshop and other art packages. It was a visually stunning achievement although the images were essentially photographic. Nobody ever said to the developers of 3D games in 1995, "look guys, these 3D graphics you're producing aren't very realistic, why don't you do it like Myst?". Having seen photo-realistic scenery they were not spoiled by it and understood the benefits of real-time interactive content even if it wasn't yet realistic. For sound effects there is a "last mile" problem. In synthetic sound there are many naysayers who complain that it "isn't very realistic". They have been spoiled by sampled sound (photographs) and unable to take the view that these photographs are limited, that absolute realism isn't the point and that if synthetic procedural sound had support, budgets and time to develop it would improve in exactly the same way that 3D graphics have over the years since there are no fundamental obstacles.

When it comes to the aesthetics of procedural music scores the issue is far more cloudy. It seems there are fundamental obstacles, hard AI problems which may never be solved. The procedural approach has no hard and fast metrics by which to judge its output[19]. Composer Pedro Camacho reduces the test to some simple old wisdom, "There are only two types of music, good and bad.", with procedural music falling into the latter. Bruce Jacob, a long time researcher and algorithmic composer offers a critique[15]in which the role of

procedural music is considered as a tool for inspiration, or as a kind of desktop calculator for the composer, but not a substitute. As I understand it the problem is essentially that expert systems, neural networks and Markov machines can codify the knowledge of a real composer but they cannot exhibit what makes a real composer, creativity. Composer and producer Will Roget points out that some of the goals of composition cannot be codified at all, to develop a concept in ways that are "memorable and maybe even clever". We all recognise good music by these characters, even if we are unable to articulate why a piece is memorable or clever.

6.3.2 Variable cost

This was mentioned as an advantage but is a double edged sword, it's also a disadvantage. Like variable interest rates, whether it's a friend or foe depends on what's happening. Because the cost of producing a synthetic sound can be hard to predict prior to execution we don't know how to allocate resources. This problem is common with other dynamic content production methods and it requires that we can either guess the cost of an operation in advance and carefully schedule resources, or produce methods which gracefully degrade as they run out of resources rather than suddenly breaking. Another solution is to restructure programs to include a pre-computation stage, a short wait that replaces what are currently load times with data model media.

6.3.3 Skills and tools

In fact this resistance is probably closely connected with the final point, from where much institutional and personal resistance to procedural audio comes. Even sound designers who are comfortable with progressive technology feel threatened by the need to adapt their skills and learn new tools. Moving from data to procedural models will never be a quick or complete process. Some sounds, most obviously voice artists, will never be replaced by procedural content. However there is an enormous investment of skills and knowledge in using existing tools, and at the time of writing this, late 2007, there is a shortage of sound designers and programmers in the industry even without a disruptive new technology. One of the greatest challenges in moving forwards to procedural audio will be developing tool chains and training new sound designers to use them.

6.3.4 Industrial inertia

Of course there is an enormous industry built upon the data model in the form of sample libraries and procedural audio is extremely disruptive to this status quo. A single procedure, for example a specialised FM or physical model for metal structures, can replace sounds for bells, iron gates, doors, weapon barrels, spanners, and thousands of other sounds with less than 1k of code. While procedural audio seems threatening to the traditional sound designers role it is not. The important place this technology has is in automatic generation of sounds for enormous new projects. Procedural sound can exist alongside sample data, indeed it must do so for a long time. While sample libraries and traditional field recordings may no longer be used for all sounds they will certainly not become extinct.

6.3.5 Software Patents

Software patents are an odious concept that has emerged in the United States during the last few years. Rejected by the rest of the world in recognition of their anti-progressive and stifling effect on technology and economies, shunned by respectable computer scientists, and even the major corporations who understand their damaging effects, they are very unlikely to prosper. But in the current political climate this has not stopped a software patent arms race running completely out of control in the United States. We cannot ignore that the USA is the largest market for games in the world. While software patents on well established methods and fundamental mathematical processes are absurd and unwelcome the consequent legal grey area leaves many developers cautious about pushing forward with progressive technologies for the US market. Until the US legal system is reformed the market for procedural audio, and many other new technologies is sadly limited to non-US countries.

6.3.6 Summary of developmental factors

Summary of pros and cons for Procedural Audio

For	Against
Rapid growth of product size	Shortage of audio programmers
Demand for more interactivity	Established production methods
Increased processing power	Lack of development tool-chains
Deferred aesthetic settings	Outsourced content producers
Huge body of research potential	US market closed by patent trolls
Automatic physical parametrisation	Physics engines designed for graphics
Simpler asset management	No effective open DSP engines

7 Who is developing procedural content?

7.1 Individuals

Ariza. C. http://www.flexatone.net/ - Composer, programmer: Christopher Ariza - Algorithmic composition, web based composition tools, ear training software

Casey, Michael http://www.doc.gold.ac.uk/~mas01mc/ - Research professor. Search, structure and analysis of audio signals. Currently working at Goldsmiths College London. Contributer to the MPEG-7 standard for describing multimedia content.

Collins. K. http://gamessound.com/ - Researcher: Karen Collins. Focusing on emotional analysis and synthesis of music for use in game worlds. Algorithmic test for affective/emotional/semiotic content. Distributed classification systems, dynamic composition of musical themes. Currently at University of Waterloo, Ontario, Canada.

Cook P. http://www.cs.princeton.edu/~prc/ Programmer, researcherPerry Cook. Author of the Synthesis Toolkit and several highly relevant books on real-time synthetic game audio.

Essl. K. http://www.essl.at Composer: Karlheinz Essl. Austrian software developer and composer of generative music. Software: Amazing Maze (granular synthesiser) Lexikon-Sonate (algorithmic music) fLOW (ambient soundscape generator).

Farnell, A.J. http://obiwannabe.co.uk/ - Researcher, programmer: Andy Farnell. Synthetic sound effects, compact procedural audio for games. Physics based modelling. Efficient real-time parametrisation from discrete and continuous control hooks. Based in UK.

Holzmann. G. http://www.intute.ac.uk/artsandhumanities/cgi-bin/fullrecord.pl?handle=200702 - Programmer, musician: George Holzmann. Puredata programmer.

Kry P.G. http://www.research.rutgers.edu/~kry/ - Researcher, programmer: Paul G Kry. Physical simulations from geometry, sound interactivity. Currently at Rutgers.

Lee. C. http://sonicvariable.goto10.org/info.html - Composer, developer: Chun Lee. Music composition with MaxMSP/Puredata, software developer DesireData project. Based in London UK.

Pai. D http://www.cs.rutgers.edu/~dpai/ Research professor: Dinesh Pai. Modelling of real life objects, audio perception cognitive psychology. Currently at Rutgers.

Marilungo C. http://www.cesaremarilungo.com/ - Composer, developer. Procedural and algorithmic music composer. Smalltalk/SuperCollider programmer. Based in Italy.

McCormick C. mccormick.cx/ - Programmer, developer: Chris McCormick. Interactive games audio programmer, Puredata programmer.

Miranda. E.R. http://neuromusic.soc.plymouth.ac.uk/ - Researcher: Eduardo Reck Miranda. Expert in parallel and distibuted implemetations for procedural music and sound synthesis. Software: Cha0synth - cellular autonoma sound synthesis. Author: "Computer Sound Design", "Computer Sound Synthesis for the Electronic Musician". Currently at Plymouth UK.

Puckette, M.S. crca.ucsd.edu/~msp/ - Researcher: Miller Puckette. Researcher, programmer, musician: Author: "Theory and techniques of Electronic Music" Software: Pure Data (Pd), a graphical programming language interactive computer music and multimedia works. Currently at CRCA.

Raghuvanshi. N. http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/r/Raghuvanshi:Nik Researcher - Nikunj Raghuvanshi. Development of efficient methods for physical modelling of rigid body excitation.

Smith. J.O. http://ccrma.stanford.edu/~jos/ - Research professor: Julius Orion Smith. Expert in physical modelling, waveguide synthesis and filter DSP. Currently at CCRMA.

Van den Doel. K. http://www.cs.ubc.ca/~kvdoel/ - Researcher, programmer: Kees van den Doel. Expert in parameterisation and modelling.

7.2 Commercial

Cycling74 http://www.cycling74.com/ - Company: The commercial version of Puredata known as Max/MSP. Largely compatible with Pd but a nicer GUI interface and professional patch management facilities aimed at the studio musician.

Earcom http://earcom.net/ - Project leader: Paul Weir. A group working on aesthetically sensitive interactive and generative music for airports, supermarkets and other public spaces. Developing in Max/MSP language. Server based management, generation and delivery of procedural audio content. Public sound installations in spaces such as Glasgow and Heathrow Airports and the chain of Tesco supermarkets in the UK.

Native Instruments http://www.native-instruments.com/ Company: Authors of possibly the best software synthesiser ever produced? *Reaktor*® is certainly the most widely used. Also designers of the *Kontakt*® format, a versatile sample bank specification.

Seer Systems. http://www.seersystems.com/ Company: Still actively researching and developing prodecural music generation and delivery systems after 10 years, many products for the Windows PC including the much applauded *Reality*® software synthesiser.

Have I forgotten you? Sorry. Are you not on this list but actively doing R&D in generative music, synthesis or sound with games or interative applications? - please drop the author an email with a brief summary

8 Document PDF Link

A pdf version of this document may be found here http://obiwannabe.co.uk/html/papers/proc-audio/proc-

References

[1] Audiokinetic. *Wwise middleware game audio delivery system.* audiokinetic.com/. Note: A game sound system with extensive GUI capabilities for binding and managing audio assets to game world events.

[2] I. Braben, D. Bell. *Elite.* Publisher: AcornSoft 1984, http://www.iancgbell.clara.net/elite/, 1984. Note: Elite is one of the oldest and most loved space games has long championed generative methods to create universes from compact algorithms.

[3] K. H. Burns. *Algorithmic Composition Definition.* http://eamusic.dartmouth.edu/~wowem/hardware/algorithmdefinition.html, 1996. Note: Semiotics and history of algorithmic composition.

[4] Conor Cahill. *The interactive audio game project.* MSc. thesis in Multimedia Systems, Trinity College Dublin,, http://www.audiogame.com/, 1997. Note: Conor builds an entire adventure game concept around audio only interaction.

[5] K. Collins. *Music of the Commodore 64.* http://www.icce.rug.nl/~soundscapes/VOLUME08/Loops%5fand%5fbloops.shtml, 2006. Note: Description of early game sound using the SID architecture.

[6] John Horton Conway. *Regular Algebra and Finite Machines.* Chapman and Hall, Ltd. London., 1971. Note: The original work behind most musical applications such as Mirandas Chaosynth, Conways work in discrete mathematics and game theory lays the foundations.

[7] D. Cope. An expert system for computer-assisted composition. *Computer Music Journal*, pages 11(4):30–46, 1987.

[8] D. Cope. Computer modeling of musical intelligence in emi. *Computer Music Journal*, pages 16(2):69–83., 1992.

[9] Timm Dapper. *Practical procedural modelling of plants.* Bremen Univerity, http://www.td-grafik.de/artic/talk20030122/overview.html, 2003. Note: Growing procedural plants, L-system branching, fractal growth, rendering.

[10] A. Farnell. Procedural synthetic footsteps for video games and animation. *Proc. Pdcon2007, Monteal, Quebec, Canada.*, 2007.

[11] Griffith. *A bibliography of Connectionist Models of Music.* http://www.csis.ul.ie/staff/niallgriffith/mnpdpp%5fbib.htm. Note: Extensive bibliography of connectionist musical applications, neural networks. adaptive/cybernetic models of listening and production.

[12] Farbrausch Group. *fr-08: .the .product,.kkrieger,.debris.* Publisher: .product, http://www.theprodukkt.com/. Note: I have been following this group since their early demo scene releases. They now have an entire first person shooter that runs in less than 100k.

[13] J. Hahn H. Fouad, J. Ballas. *Perceptually-based Scheduling Algorithms for Real-time Synthesis of Complex Sonic Environments.* https://www.cmpevents.com/sessions/GD/S3853i2.doc, 1997. Note: In Proc. Int. Conf. Auditory Display. Thoughts on psychoacoustics applied to variable level of spectral detail for occlusion and distance fading.

[14] B. Jacob. *Algorithmic composition as a model of creativity.* http://www.ece.umd.edu/~blj/algorithmic_composition/algorithmicmodel.html. Note: Creativity, knowledge and composition - intelligent philosophical analysis.

[15] B. Jacob. Composing with genetic algorithms. *Proceedings of the 1995 International Computer Music Conference, Banff, Alberta.*, 1995.

[16] Robert James. *A Pictorial History of Real-Time Procedural Planets.* http://baddoggames.com/planet/history.htm, 2003. Note: Examples of early procedural world content to about 2002, good to compare with modern output such as Terragen2.

[17] H. Jarvelainen. Algorithmic musical composition. *Helsinki University of Technology, TiK-111080 Seminar on content creation.*, 2000.

[18] K. Jones. Compositional applications of stochastic processes. *Computer Music Journal*, pages 5(2):45–61, 1981.

[19] Otto Laske. *Algorithmic Composition In the New Century.* http://www.perceptionfactory.com/workshop/Otto.htm. Note: The role of composer in relation to software.

[20] Guy W. Lecky-Thompson. *Infinite Game Universe: Mathematical Techniques.* Charles River Media., http://www.amazon.com/Infinite-Game-Universe-Mathematical-Development/dp/1584500581, 2001. Note: World and universe generation, space themed.

[21] Black Lotus. *Demoscene pioneers.* http://en.wikipedia.org/wiki/The_Black_Lotus. Note: The famous Black Lotus, demoscene pioneers of procedural content in small memory footprints.

[22] Sean O'Neil. *A Real-Time Procedural Universe, Part One: Generating Planetary Bodies.* Publisher: Gamasutra, http://www.gamasutra.com/features/20010302/oneil_01.htm, 2001. Note: Part one of a three part series on procedural textures, world geometry and rendering.

[23] Ken C. Pohlmann. *Principles of Digital Audio (3rd Ed.).* Mcgraw-Hill, http://mcgoodwin.net/digitalaudio/digitalaudio.html, 1995. Note: First edition published in about 1988, this was one of my univerity textbooks, excellent intoduction to digital audio fundamentals.

[24] Miranda E. R. *Evolving Cellular Autonoma Music.* http://galileo.cincom.unical.it/esg/Music/workshop/articoli/miranda.pdf. Note: Mirandas work on concurrent interactive and distributed processes spans composition and signal level synthesis.

[25] Nikunj Raghuvanshi and Ming C. Lin. *Real-time Sound Synthesis for Computer Games.* https://www.cmpevents.com/sessions/GD/S3853i2.doc. Note: Given at GDev conference, Raghuvanshi considers optimisations for simplifying eigenvectors for geometry based finite element physical models of solid body impacts.

[26] C. Roads. The computer music tutorial. *MIT Press.*, pages chapter 18–19, 1996.

[27] R. Rowe. Interactive music systems. *The MIT Press, Cambridge MA.*

[28] Warren S. Sarle. *Neural Networks FAQ.* ftp://ftp.sas.com/pub/neural/FAQ2.html, 1997. Note: Comprehensive Q&A compiled from newsgroup comp.ai.neural-nets.

[29] Planetside Software. *Terragen.* Publisher: Planetside Software., http://www.planetside.co.uk/terragen/, 1998. Note: Advanced procedural landscape generation software.

[30] Firelight Technologies. *FMOD game audio system.* www.fmod.org/. Note: Australian based middleware audio API. Multi platform with more mature dynamic DSP directions but less polished programming IDE.

[31] Various. *AlgoNet.* http://www.flexatone.net/algoNet/. Note: Research resource for computer aided algorithmic music composition.

[32] Various. *Online Encyclopedia of Integer Sequences.* http://www.research.att.com/~njas/sequences/. Note: Compendium of known integer sequences and their basis, Maintained by N. J. A. Sloane at AT&T.

[33] Will Wright. *Spore.* Publisher: Maxis, http://www.spore.com/, 2007. Note: Large scale life simulation game using evolutionary algorithms and fractal generative algorithms to make 3D and sound elements entirely with procedural content.

[34] I. Xenakis. Formalised music. *Indiana University Press*, 1971.