A Scheme for Generalized Control of Particle Synthesis

David Thall
Media Arts and Technology Program
University of California, Santa Barbara
Summer 2004

Abstract

A new particle-based sound signal generator, processor, and control system is presented, developed, and discussed. The unifying principles that exist among past implementations are first examined. The result of this investigation is a systematic design that, through reclassification and generalization, combines and extends these early models. Sets of signal processing sub-routines are then identified, collected and combined into a general-purpose particle generator. A novel user interface is also presented along with various high-level control regimes. This assists the user in navigating the multi-dimensional parameter space inherent in particle transformations. The combination of fast and efficient processing with scalable and customized controllers results in a system designed exclusively for advanced particle synthesis and processing.

0. Introduction

Particle-based sound synthesis and sampling is a family of powerful, time domain techniques for sound design and music composition. They allow the composer to stream or scatter acoustic particles in multiple dimensions, either in real-time or by script. Using various particle synthesis and processing models, a set of parametric control data is generated and mapped to an underlying particle- scheduling algorithm. In this context, a 'sound particle' can be considered a finite time segment of an arbitrary waveform modulated (shaped) by an amplitude envelope.

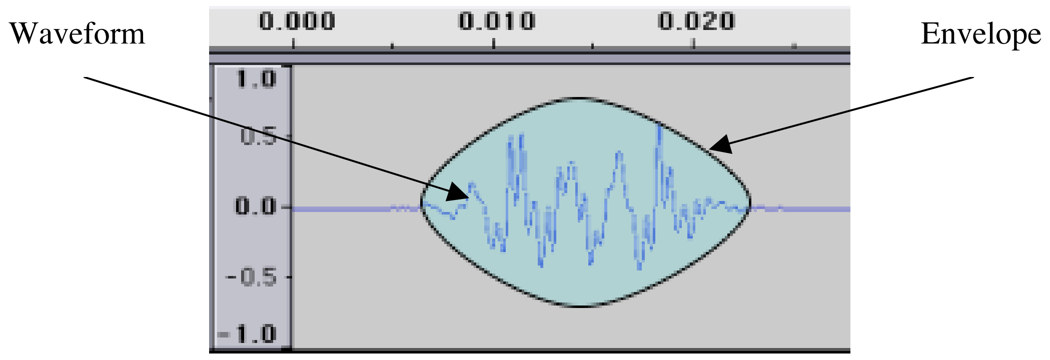


Figure 1: Anatomy of a Particle

Historically, particle synthesis has been used to perform such acoustic transformations as pitch shifting and time-scaling, and has been used for spectrum analysis [Otis, Gabor]. In its recent incarnations, digital audio programmers have utilized the technique to widen the range of possible sound enhancements and transformations. The ability to work with sound at the level of individual sound particles opens up the possibility to deconstruct and reassemble sounds anew, into innovative shapes and textures.

Traditionally, sound synthesis has been limited to the domain of continuous waveforms (mechanical and analog tone generators), or their discrete-time reconstructions (digital sound synthesis). In these models, the process of tone generation is continuous. An analog oscillator never stops once it is powered up. In the digital domain, wavetable synthesis creates sound by cyclically sampling a waveform stored in memory. The output of this function produces a continuous sound that can then be fed into a filter, amplifier, or other device. Through direct intervention, the composer can apply constraints to the length of the sound by modulating it with a function that closes (gates) the event. In order to create polyphony, multiple copies of the generator can be mixed together to varying degrees. While this model is versatile, it is quite different from the particle-based computational model.

Particle synthesis, on the other hand, is based on the concept of the ‘sound object’ [Schaeffer]. In this model, each sound object can be considered a compound collection (mass) of hundreds or even thousands of individual events in a relatively short period of time. Each particle in the mass can be subject to completely different processing, and can thus be considered a unique event. Due to the nature of the rapid rate of flow of individual events, scheduling and controlling these streams and clouds requires new algorithms that are designed to support this model [Roads, de Campo]. The trivial scheduling built into traditional sequencer-type synthesis and sampling routines is inadequate.

Since the introduction of general-purpose programming languages, many developers have attempted to design and implement synthesis and sampling routines that take these differences into consideration [Xenakis, Roads, Truax]. However, the methods that currently exist are limited in that they do not take into account the myriad complexities and variations of the signal-processing model. Just as additive synthesis requires a huge set of low-level control data, particle synthesis demands a similar explosion in high-level data mappings. Early software prototypes have been successful at automating the production of interesting tonal variations and multi-faceted textures; however, they fail to offer the user a comprehensive control regime.

Here we describe the design of a new technique for controlling various microsonic algorithms, the results of which are presented in a new software prototype built by the author. This system is designed as an experimental and pedagogical toolkit, which will, it is hoped, evolve into a complete particle synthesis hardware and software system. It offers many interactive modes of operation, and can synthesize and granulate multiple streams in real-time.

This paper begins by briefly presenting the similarities, differences, and unifying principles that exist between various particle synthesis routines and control regimes that have been implemented, from the past through the present. We then move on to discuss the design and implementation of 'Cloud', a general-purpose, multi-tiered particle synthesis and transformation plug-in written for the Supercollider 3 synthesis server [McCartney]. This section is followed by a deeper look into 'Emission Control', the graphical user interface and high-level control scheme designed exclusively for advanced particle synthesis. Applications of the prototype are then discussed. The paper concludes with a section on possible code optimizations and the implications of building a complete particle synthesis system.

1. Toward a Unification of Particle Synthesis Models

1.1 Historical Particle Synthesis Systems

In 1946, the Nobel prize-winning physicist Denis Gabor published a landmark paper on the theory of hearing and subjective acoustics. He developed a new theory in psycho-acoustics that attempted to describe the human hearing mechanism from the perspective of quantum physics [Gabor]. He felt that Fourier analysis, while mathematically correct, could not represent the cochlear resolution that humans have in the inner ear. Fourier analysis assumes that all sound exists infinitely in time (all sound is composed of sine waves of infinite duration), and there is always a trade-off between the time and frequency resolution of the analysis. Humans, on the other hand, perceive time and frequency as discrete patterns that are converted into electrical pulses in the brain (the ear has finite limits in its resolution). This dual perception can be broken down into an elementary acoustic description, or acoustic quanta, and used both for the analysis and synthesis of sound.

As a result of his initial experiments with sound, Gabor created a frequency converter using a type of 'Sound Film' (see Figure 2). Described as a type of Hilbert Transform (i.e., frequency shifter), the system would repeatedly window a continuous sound playing on a moving strip of film into individual sections. The results were collected using a photocell and summed with previous segments with some degree of overlap. The resulting light output constituted the mixture of the elementary stream in time, producing the complex signal. Depending on the speed of the film, Gabor was able to calculate and perform a shift in pitch, while maintaining the overall duration of the original signal. He had, in fact, performed the first particle-based sound transformation by opto-mechanical means.

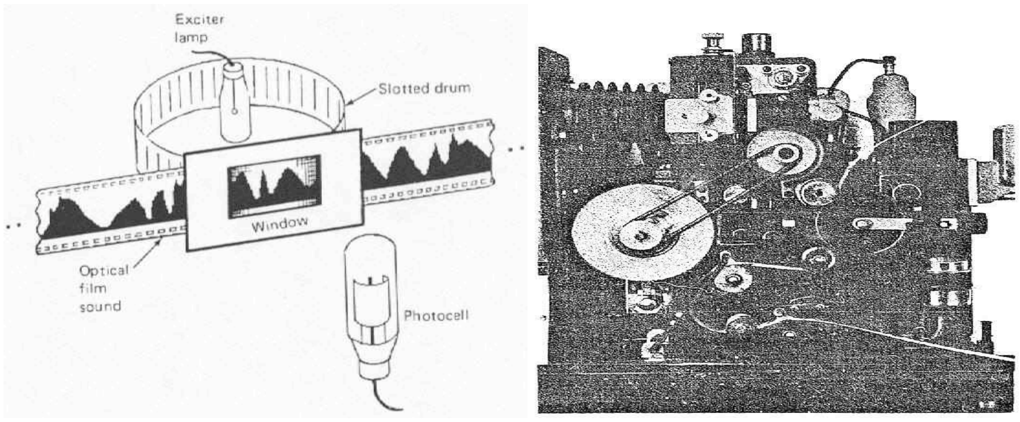


Figure 2: The drawing on the left is a kinematical frequency converter with sound film (Gabor 1946). The photo on the right is a sound film projector converted into an experimental frequency converter (Gabor 1946)

One of the earliest (digital) computer-based particle transformations was carried out on the Illiac II Computer at the University of Illinois Experimental Music Studio in September 1968. Alton B. Otis wrote a sound-processing program to perform ‘rate changing’. In this technique, the rate at which digital sound segments are extracted and overlapped can be independently controlled, without affecting the original pitch of the source material [Otis]. The result was a system that could perform arbitrarily complex time stretching and shrinking, where the resulting length of the original sound could be compressed or expanded according to the input function. Otis’s system was plagued by audio artifacts, however.

1.2 Breakdown of Modern Implementations

Throughout the past 20 years, various modern implementations of particle-based transformation systems have been proposed in environments such as Music V [Matthews, Roads], CSound [Vercoe], MAX/MSP [Puckette, Lippe], Supercollider [McCartney], among others. Some recent commercial synthesis and processing software packages have made attempts as well, such as Symbolic Sounds Kyma, Native Instruments Reaktor, and Propellerheads Reason. All of these systems have expanded the parameter space available for sonic transformations, and most of the newer ones generate multichannel sound in real-time.

The commonalities that exist among most of these adaptations lie in their signal-processing models. For example, the POD system developed by Barry Truax and the IRCAM Signal Processing Workstation’s (ISPW) granular sampling models developed by Corte Lippe both run multiple, time-delayed particle streams in parallel [Truax, Lippe]. The user can define the degree of particle overlap (as a ratio or percentage) in real time, as well as the relative delay between running streams. Other systems such as those developed in SuperCollider (Creatovox, constQ) by Roads and de Campo allow the user to specify an overall density function in time, with an inherent synchronous or asynchronous overlap ratio.

Parameter-linking also occurs within many of the algorithms, and is a major contributor to the resulting sonic characteristics of each implementation. For example, a system such as PulsarGenerator uses an algorithm that can be considered a subclass of generalized granular synthesis [Roads, Pope, de Campo]. Pulsar synthesis produces its output by placing constraints on the underlying control model, allowing for independent modulation of the playback rate and emission frequency (i.e., pitch and formant) of synthesized, duration-dependent grains. As long as the emission frequency and particle duration are inversely proportional to one another, the user will be able to modulate the perceived formant spectrum. The disadvantage in this type of parameter-linking model is that one parameter must be hard-coded to calculate its values via another parameter, namely the duration of the particles.

Another major factor that contributes to the overall sound is the general-purpose, statistical computational model hard-coded into the algorithms. CloudGenerator is a good example of a software program designed specifically around this model [Roads, Alexander]. The user specifies an upper and lower bound between which a sound parameter will be statistically distributed, such as the frequency of a grain. This can change in time, creating a morphological transformation similar to the 'Tendency Masks' discussed by Truax. Native Instruments Reaktor includes the Grain Cloud Sampler module that works in basically the same manner, but with a more limited control range[1]. Both of these systems have the ability to dynamically modulate the parameters of the synthesis routine over time, with the exception being that CloudGenerator can specify the total duration of the cloud transformation (which is a very useful technique in higher-level event mixing and ordering).

The main difference between these various particle-processing models is in the data structure used to specify parameters and mappings to higher-level structuring. In systems such as the Reason Malstrom Graintable Synthesizer and PulsarGenerator, the synthesis routine handles all of the calculations. The high-level parameters and graphical user interface presented to the user hide the low-level implementation. In contrast, the Creatovox performs all mapping in a high-level control specification. In other words, there is no 'granulation' algorithm per se; instead, there is a high-level specification that 'models' one. This allows for faster prototyping, but also limits the system both in programmatic scalability and reusability.

1.3 Unification of Particle Synthesis Routines

The similarities that exist in the modern particle-based synthesis routines far outweigh the differences. Thus, a unification of these signal-processing systems can be beneficial, bringing all of the sound transformation properties of each algorithm into a single vast space of sonic possibilities. The resulting system can be designed to handle all basic similarities, and extend itself where needed to handle the differences.

[1] Most commercial particle synthesis implementations still place limits on parameter ranges due to their insistence on directly updating parameters via the MIDI protocol, a standard supported by music instrument manufacturers. A system such as this will never be able to extend the quantization step of a parameter beyond the 127 values supported by MIDI.

Most implementations can be linked, either directly or through one level of indirection, to the domains of granular synthesis and granular sampling, each of which can be considered a subset of abstract Particle Synthesis. Everything else can be designed through specified constraints on this general model.

The most prevalent particle synthesis implementations are related purely through generalization (see Figure 3). Pure granular synthesis is directly related to the Gabor Transform [Gabor]. This model synthesizes pure sine waves at various audible frequencies, windows them by a Gaussian function of time, and then overlaps and adds the results together to produce the output. His formula for sound quanta was: $g(t) = e^{-a^2(t-t_0)^2} \times e^{j2\pi f_o}$ where $\Delta t = \pi^{1/2}/a$ and $\Delta f = a/\pi^{1/2}$ [Gabor, Roads]. Wavetable granulation, or granular sampling, can be considered a generalization of pure granular synthesis, replacing the sine wave with arbitrarily complex samples from a live or disk-based source. All other systems directly inherit from either one of these basic functions. For example, Pulsar synthesis produces its output by placing constraints on the underlying granular synthesis model, as was discussed in the previous section. The pulsar model can be broken down still further to produce pulsar sampling (a new technique), which is derived by combining the sampling power of wavetable granular sampling with pure pulsar synthesis. Grainlet synthesis and sampling features parameter-linking, allowing for variable constraints among parameters [Roads].

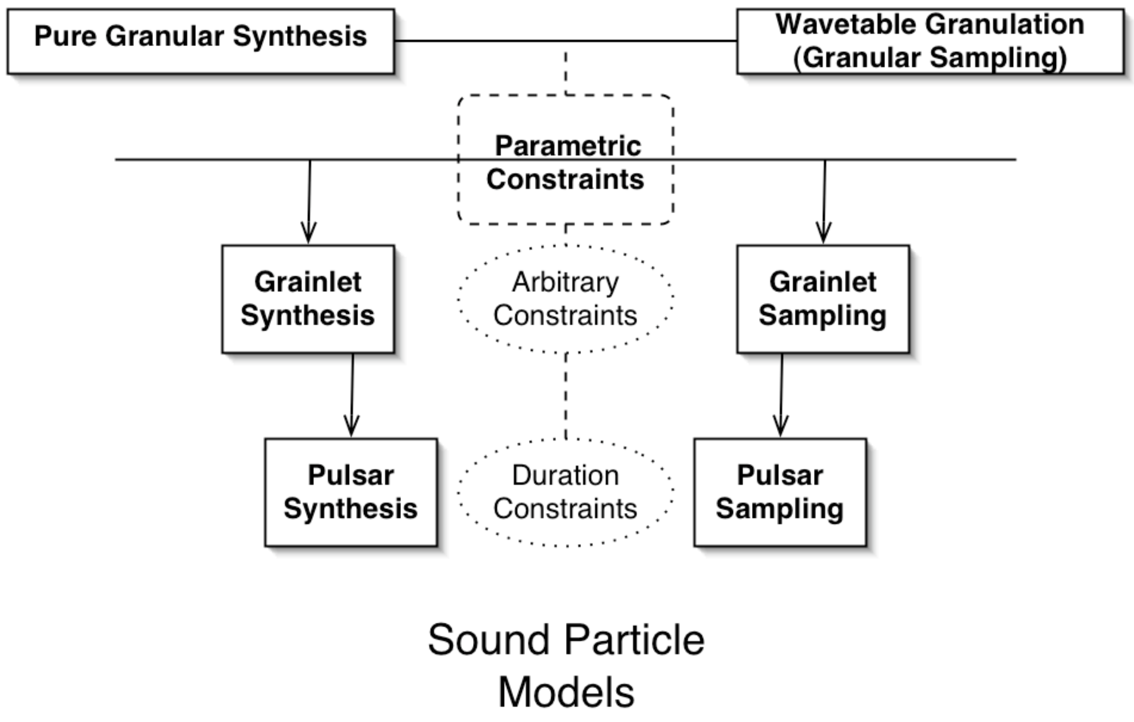


Figure 3: Breakdown of Sound Particle Models

Within these basic models lies a low-level synthesis routine that can be extracted and generalized. A particle scheduler is used to control the emission properties of the system. Pure granular synthesis and wavetable granular sampling use synchronous and asynchronous timers, while pulsar synthesis uses a combination of these as well as more advanced pattern-derived sequence chains (see section 2.2 Time Model) and burst-masking [Roads]. At the moment that a particle is to be emitted, a set of calculations is performed to generate the control data needed to synthesize and/or sample the grain. Some implementations explicitly link parameters together in the algorithm, in order to create a desired result in the output stream [Reason, PulsarGenerator]. Others attempt to decouple as many parameters as possible, in order to experiment more with the ranges and dimensions of the parameter spaces [Reaktor, Creatovox]. All of them modulate the synthesis parameters with a statistical equation, or a variant of one, in order to induce the desired result.

Merging these design patterns together into a unified particle synthesis routine allows for more extensibility. Newer and more interesting implementations may then present themselves, not as completely new particle synthesizers, but as extensions to an already well-defined system of particle-processing units.

The software described in the rest of this paper attempts to merge the different design patterns discussed above into a unified model. Furthermore, I will propose a generalized control regime that can harness the most effective features of each of these algorithms through a set of novel user interfaces.

2. Developing a New Algorithm for Particle Processing

2.1 Requirements

A generalized software algorithm to schedule, synthesize, sample, and process acoustic particles must be designed to handle many types of particle models, from pure granular synthesis to wavetable granular sampling. The synthesis engine must be able to withstand extreme settings, controlling all voice and memory allocation, overlapping multiple streams in real-time, aligning particles to a sample-accurate clock, and (most importantly) producing sound of the highest quality.

'Cloud' is the name of the general-purpose, multi-tiered particle synthesis plug-in written for the Supercollider 3 synthesis server [McCartney]. It is actually a compound organization of many individual, encapsulated functions written in C, the combination of which outputs a controlled stream of acoustic particles. This modular design to the structure of the plug-in has been a contributing factor to its versatility [Bencina].

When the design of the plug-in began, I decided that certain basic requirements were needed before the first line of code could be written. First, the algorithm needed to be robust. One does not want to overload the CPU! One of the major issues that plagues real-time implementations is the lack of 'flow control' during processor-intensive overlaps. As the particle density (overlap ratio) increases (independent of particle duration), different rules must be specified that can throttle the system. This is especially important, since at any point in time the number of overlapping particles can shift from one to tens or even hundreds, causing unwanted CPU spikes. De Campo breaks down the rules into a set of four, though many more can be proposed [de Campo]. We can limit the grain durations when the overlap ratio reaches a threshold. This approach will shorten the peak CPU spikes, but there is still the possibility of overshooting the average overall CPU usage. Limiting the density to a threshold is useful for controlling both peak and average CPU usage, but places explicit limits on the usability and scalability of the system (discussed below). A third option is to limit both duration and density to a threshold, which results in a combination of the advantages and disadvantages from the above two rules. The fourth rule drops grains from the scheduler if the CPU usage goes above a specified threshold. This is a very useful throttling technique, but imposes effects at the audible level, since grains that should have been output are replaced with silence! One other solution is to provide an 'adaptive threshold' for both density and duration. If one or the other parameter peaks the CPU, or causes an average overall CPU increase that lasts longer than a specified amount of time, the algorithm will slowly throttle the range. This is the technique used in 'Cloud', which is charted in figure 4.

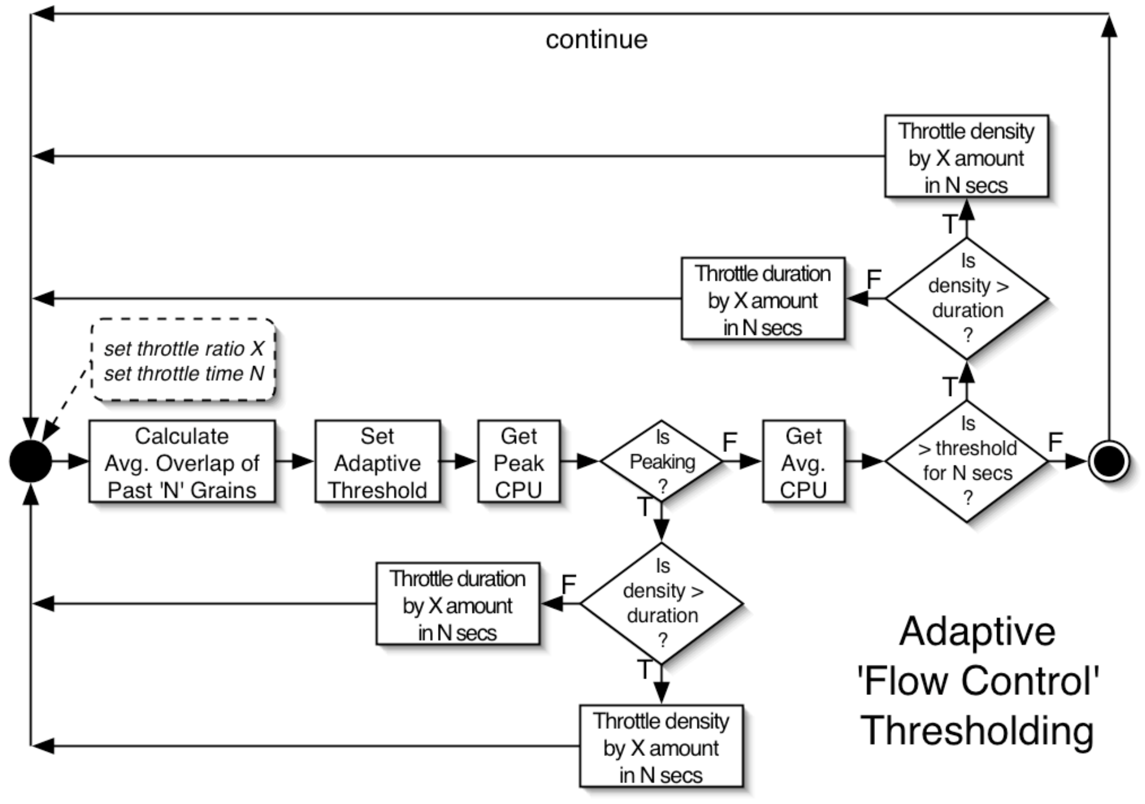

continue
Throttle density by X amount in N secs
Throttle duration by X amount in N secs
Is density > duration ?
set throttle ratio X
set throttle time N
Calculate Avg. Overlap of Past 'N' Grains
Set Adaptive Threshold
Get Peak CPU
Is Peaking ?
Get Avg. CPU
Is > threshold for N secs ?
T
F
Throttle duration by X amount in N secs
Is density > duration ?
Throttle density by X amount in N secs
Adaptive 'Flow Control' Thresholding

Figure 4: An adaptive threshold algorithm for particle 'flow control'

The algorithm also needs to be scalable. It's not just enough to keep the system running under extreme conditions. There must be an independent solution for handling extreme ranges of particle density and durations. This feature has been coded directly into the algorithm through the construction of a data structure that can schedule as well as allocate and de-allocate memory quickly and efficiently (see figure 5). A data structure is constructed at runtime in memory, which is then able to maintain an explicitly set total number of particles, and optimize itself accordingly. If the user of the system runs on a faster machine, they only need to update a single preference, and the system will update itself dynamically. Furthermore, very long duration particles can be constructed from sources much shorter in origin through looping, and this can be controlled dynamically.

For example, if a long duration grain is created, and then the system CPU usage begins to peak, the grain can be prematurely shortened in duration.

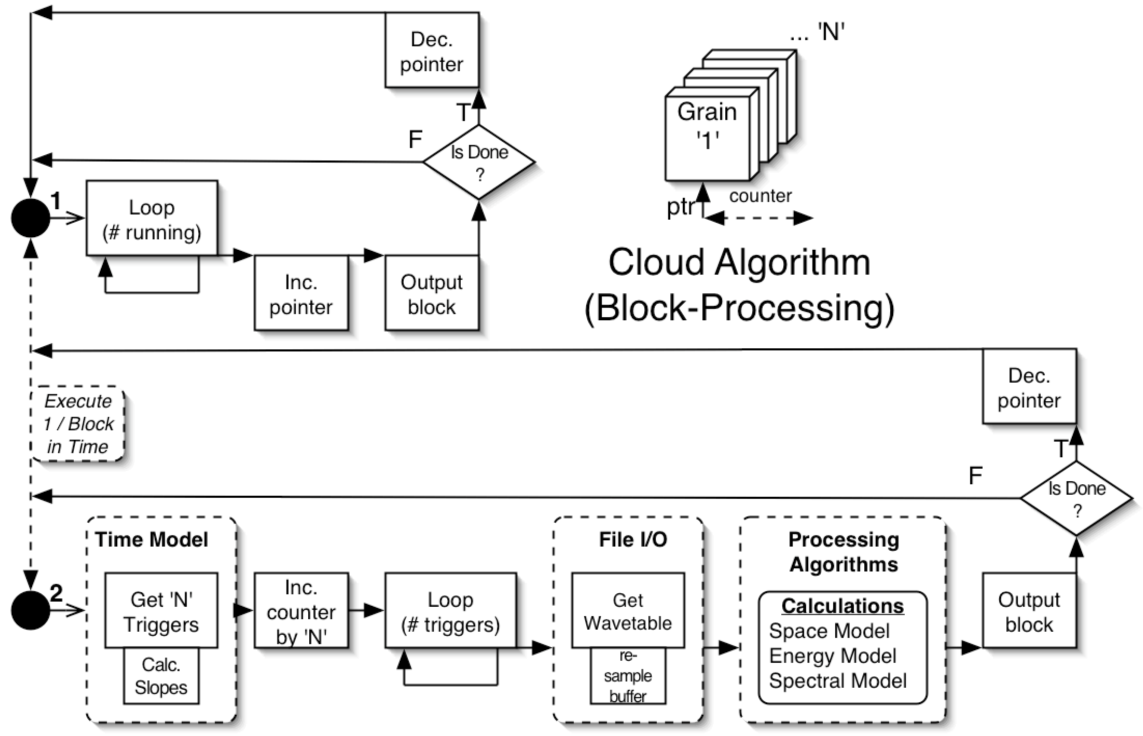


Figure 5: The 'Cloud' algorithm. Two loops run in parallel. The first outputs running grains. The second generates new grains.

The algorithm has been designed to be fast, efficient, and most importantly, have sample accurate scheduling for any number of parallel streams. One addition that makes this possible is the ‘selective interpolation quality’ on individual grains. If a grain is very short in duration (and thus has a high frequency envelope), or is played back at a much higher sampling frequency, the internal interpolation algorithm will switch itself to accommodate. The interpolation algorithms are high quality, but not band-limited. This may be updated in a future release. In order to be fast, many repetitive calculations have been replaced with memory accesses to stored function tables. An example of this is the algorithm used for equal-power gain compensation, which will be discussed in the next sections.

The last requirement is specific to this prototype, but could also be implemented in a stand-alone system. Transparency with the host application has been embedded to allow for generally complex multi-soundfile processing. On-the-fly access to a bank of wavetables loaded dynamically into memory is handled in the runtime environment. The algorithm furthers this architecture-specific property by preprocessing each soundfile to the current sample rate and bit depth of the host.

The following sections break down ‘Cloud’ into its constituent parts, describing the implementations and their development. In order to abstract the particle model into a form that could describe a majority of the model variations and implementations, the architecture has been left open. The system model can be easily reconfigured and expanded.

2.2 Time Model

The particle-scheduling (time-sequencing) algorithm in ‘Cloud’ is the heart of the synthesizer. The algorithm is simple in concept, but allows for arbitrarily complex control sources, and is extremely scalable. Any control signal that is fed into the input is sampled at the current audio-rate of the system. If at any point in time the next sample goes positive (i.e., the slope of the signal crosses the zero-crossing boundary in the positive-going direction), the internal generator loop will calculate, generate, and release (output) a new particle. Thus, any type of signal source can be used, including sources from an external input, a file stored on disk, or a trigger source synthesized in the language. For purposes of high-level modeling, four default types of control sources have been predefined and placed into the system. The four types of controls are synchronous, asynchronous, sequenced, and derived (see Figure 6). They have been carefully chosen to integrate all of the known combinations of emission algorithms in previous particle-processing systems.

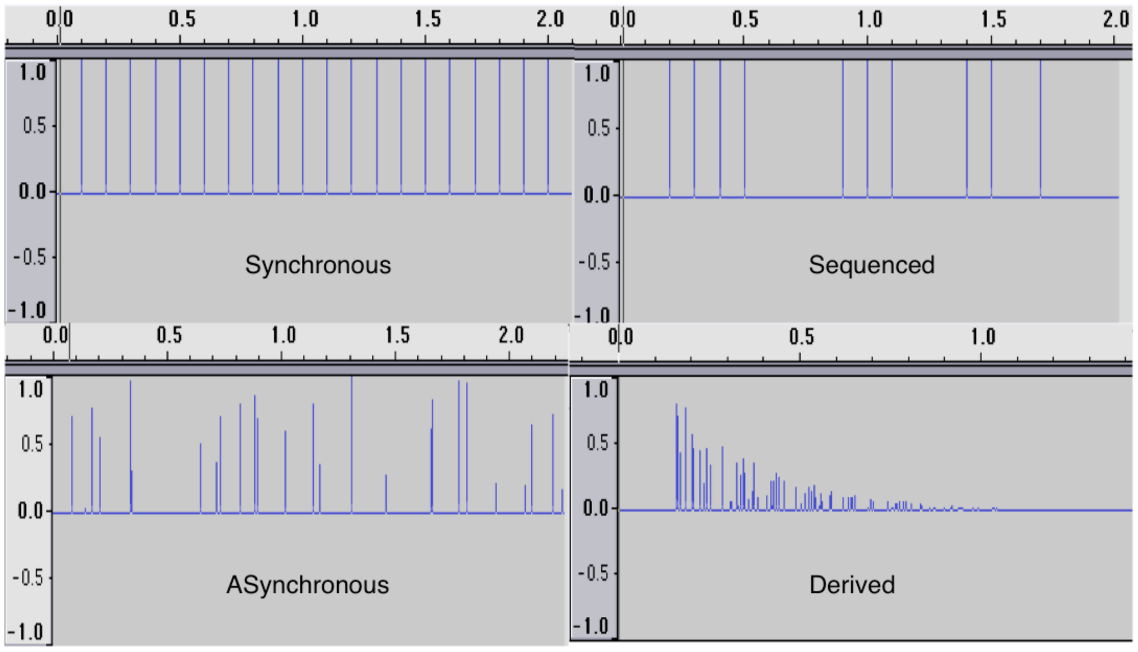


Figure 6: Four types of particle-scheduling trigger sources

A synchronous scheduler produces timed triggers at a regular interval, and is measured in Hertz. This is used to create pitch-synchronous granular synthesis, and is especially useful in creating pitched tones and streams [Cavaliere, Piccialli]. It is also the basic streaming mechanism used in pitch and time scaling algorithms, as well as in basic pulsar models [Roads, Dutilleux]. This is by far the simplest trigger to generate, and can be derived from any periodic, symmetric source.

Asynchronous scheduling is produced by spreading a random trigger distribution over time, and is measured as an average number of triggers per second. This is used to create asynchronous granular synthesis, and is especially useful for creating statistical noises, clouds, and an assortment of special effects like explosions and textures.

A sequenced trigger source is a generalization of the much-used Burst Generators in the electronic music studios of the past [Roads]. Here, chained sequences of trigger patterns are created [Putnam, Thall]. For example, a pattern can be programmed that sends out an arbitrarily complex on-off sequence such as [4, -4, 3, -3, 2, -2, 1, -1], which will result in 4 on's, 4 off's, 3 on's, 3 off's, and so on at the desired frequency measured in Hertz. This technique is useful for emulating bouncing rigid bodies as sounds, and is a more general-purpose implementation of the sub-octave burst generator programmed into PulsarGenerator.

The final trigger source is a derivation from a pre-analyzed source. For example, the trigger can be derived from an envelope follower connected to a live audio source, and controlled with a 'sensitivity' parameter. Other sources may include impulse responses of rooms or hardware music devices, MIDI clocks, or even extracted from soundfiles. An example of this type of trigger source is given in Figure 6, which is a stochastic impulse response generated on-the-fly during synthesis.

This type of open-ended, linear scheduling system is versatile and scalable. A good example of this is in the creation of polyphonic streams. Simply mixing multiple trigger sources together results in parallel streams and events. There is no need to run multiple copies of the algorithm, resulting in a much faster, cleaner performance.

2.3 Space Model

Rendering particles in a multi-dimensional space is a key feature built into 'Cloud', and a key component of the particle synthesis model. Most commercial applications that perform a subset of particle-processing limit the output range to stereo, or have the algorithm feed its summed output into a system-wide panning or spatialization routine. The spatial models built into this algorithm differ in that they 'scatter' individual particles to various points in space, based on the set of control signals fed to the inputs. The resulting sound object is a combinatorial texture that can extend into multiple directions in space over time. This is sometimes referred to as a 'cloud' [Xenakis, Roads].

The current spatial panning mechanism is based on a flat, polygonal model. This algorithm scatters the source to a point around the perimeter of a polygon with equal-power gain compensation, and can also simulate a position from within. As more output channels are added to the calculation, the polygon (curvature) approaches a circle. An example of this multi-channel expansion is given in Figure 7. In the two-channel mode, a basic stereo setup is preserved. As channels are added, the first channel starts at the top vertex of the polygon, with each additional channel placed at the corners in a clockwise manner. In order to deal with varying speaker placement, the implementation allows for rotating the orientation of the polygon, as well as reversing it.

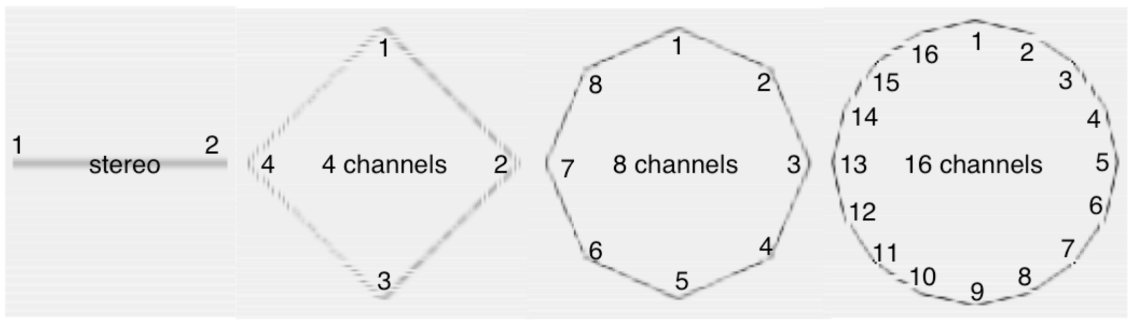


Figure 7: Polygonal Panning Mechanism and multi-channel expansion.

An extension to this azimuth-based panning model is the spherical panning algorithm. Here, the polygonal model is subdivided and rotated ninety degrees about the Z-axis. This is useful in a situation where the user would like to move the center of the source to an arbitrary position in a 3D space, allowing for directional localization (azimuth and elevation) cues [Loomis]. This algorithm has yet to be tested, as there are currently few spaces that can accommodate this feature.

So far the spatial models only account for point-source scattering within a multi-dimensional space. Width, depth, and spatial envelopment are all needed in order to complete the perceived spatial attributes. As particle density increases, there must be a way to control the width of the total spatial event. This is referred to as the 'ensemble width' by Rumsey, and can be controlled by altering the relationships between amplitudes and timings of the signals arriving from the output channels [Rumsey]. A variant of this will be discussed below. Depth and spatial envelopment are currently modeled using an adaptive global reverberation algorithm that can be selectively bypassed from the output stream. Reverberation scales to the number of output channels currently being used.

The trajectory of particle streams toward and away from the listener is modeled using synthesized Doppler-Shift effects [Chowning]. However, it must be noted that the perception of velocity and acceleration are heavily 'input dependent' [Chowning, Lippe]. Synthetic particle streams generally reproduce more accurate perceptual results.
The spatial algorithms used can create an abundance of possible transformations, and are implicitly tied to the particle-based acoustic model. In the future, sound designers and composers will be able to move complex sounds and streams of sound through space with

virtuoso execution, as was dreamed by Varese, and partially realized by Stockhausen [Varese, Stockhausen].

2.4 Energy Model

Specifying the levels of individual particles in an overlapping stream or dense cloud is non-trivial. There are two basic problems that need to be considered. In a real-time system with user-controllable scheduling, there is no way to directly know how many particles may be overlapping at any single point in time. When a particle is output, a coefficient is calculated to create the maximum amplitude for the envelope that will modulate its gain. If the next particle that gets output sums with the previous output to a value greater than 'full scale digital', the output will clip the D/A converter. Another problem has to do more with perception, spatial location, and depth. In order to maintain a consistent perception of loudness while scattering grains in space, we must rescale the amplitudes of these particles without affecting the desired spatial cues.

The most basic solution for controlling peaking levels during overlap is a simple gain compensation function. Two methods of simple gain compensation, equal-power and equal-gain, are currently being used. The simplest formula for equal-power gain compensation is $a(x) = a \times N^{-(1/2)}$, where $a(x)$ is the weighed output amplitude, a is the normalized input amplitude, and N is the current number of overlapping particles. This produces a logarithmic decrease in gain as the particle overlap increases linearly. Equal-gain gain compensation replaces the $N^{-(1/2)}$ with N^{-1}, resulting in an exponential decrease in gain. These are both fast and efficient algorithms for calculating particle amplitude coefficients in a complex time distribution, and have the added bonus of both emulating the logarithmic scaling of human hearing, and keeping the output of the system below 0 dBFS. With equal-power scaling, the output gain rises linearly in response to an exponential increase in particle density. With equal-gain scaling, the output gain decreases exponentially in response to a linear increase in particle density. Equal-power is more useful for shorter duration grains with high density, while equal-gain is more useful for longer duration grains with low density. The algorithm switches between these two functions programmatically with an adaptive threshold.

Originally, both gain compensation functions calculated the maximum amplitude coefficients before particles were generated. The latest version of 'Cloud' uses lookup tables stored in memory for faster operation. The problem with this model is that there is still no way to compensate for 'inter-grain' overloads. For example, if a cloud of very short duration grains with varying overlap are output and summed on top of a very long duration grain, there is no way to change the gain of a grain that may clip the output without lowering the level of the entire mixture as a group. One solution is to place a 'look-ahead' limiter into the signal-processing chain, time delay the entire stream, and then track amplitude variations, making gain adjustments as needed. This approach is used in many modern limiting applications, such as the Waves L1 Limiter [Waves].

The width of a spatial distribution of particles can have an effect on the perceived loudness [Rumsey]. Depending on the spatial density of a cloud, gain compensation is applied to maintain a consistent perception of width. For example, a sparse cloud and a dense cloud occupying the same spatial location should be perceived as having the same spatial width. Unfortunately, this cannot be a complete description due to perceived 'localization blur', which is defined as the smallest change of a specified sound attribute needed to perceive a change in position [Blauert]. According to Blauert, the duration as well as spectral bandwidth of a sound can determine both how wide we perceive its source to be, and its relative location in space. A collection of short duration sounds only adds to the complexity of the model. Thus, a per-particle and per-stream spectral model must be taken into consideration when calculating spatial-density controlled gain compensation.

As the duration of an acoustic particle decreases, high-frequency amplitude modulation side effects from the envelope cause the spectral bandwidth to increase. A filter process is added to the generation of each particle in the output stream to compensate spatially, which is discussed in the next section. While filtering helps resolve spatial information, it also lowers the total output gain of the particle. In order to deal with this dilemma, spectral gain compensation is applied per-particle and per-stream to account for lost energy. Two forms of this function seem to produce the best results. In the first form, $a(x) = a \times \{1 + (1/Q)^{0.707}\}$, the output amplitude is scaled by the reciprocal of the resonance parameter, and then further scaled by 0.1 times itself. In the second form, $a(x) = a \times \{1 + (1/Q)^{0.707}\}$, the output amplitude is scaled by the reciprocal of the resonance parameter raised to the power of 0.707 (the RMS average of a sine wave). Both equations have been produced purely through empirical testing, and are still under investigation.

While not directly related to perceived loudness, the shape of the amplitude envelope used on individual particles has an effect both on the spectral content of the particle, and its perceived level. Many types of amplitude envelopes are available in 'Cloud', the default type being a Gaussian shape. Others include linear and exponential attacks and decays [Fischman], parabolic curves, trapezoidal segments, and raised cosines [Bencina]. These can be used individually, or switched according to rules imposed by the user. For example, the ratio between the attack and decay segment length's can remain constant as the particle duration changes. This can be very useful for controlling the perceived texture of the grains in the stream, especially when applied to long duration particles.

Gain compensation in particle-based sound processing may be the most important measure for output quality control. Furthermore, the perceived quality of the particle stream will depend heavily on the choice and usage of the various algorithms.

2.5 Spectral Model

Micro-filtration, or filtering on a micro-time scale, couples naturally with all forms of particle synthesis and sampling [Roads]. Early implementations restricted filtering to the level of the phrase, which is the model still used in commercial wavetable and subtractive synthesizer modules. By contrast, the first implementation of per-grain filtering was successfully tested and demoed in the Constant-Q Granulator built by Curtis Roads, and later recoded in real-time by Alberto de Campo [Roads, de Campo]. By applying a constant-Q band-pass filter to each particle in a stream, the user is able to transform source material continuously from the raw and unfiltered waveform to a pure sine wave. This can create scintillating spectral variations on the micro-time scale, as tens or even hundreds of filters are overlapped and mixed together in time.

The implementation within 'Cloud' is a 4th order, constant-Q band-pass filter based on the 2nd order recursive (IIR) Butterworth band-pass filter designs of Dodge and Jerse [Dodge, Jerse]. The magnitude response for this filter with variable resonance can be seen in figure 8. This filter function is both fast and efficient[2]. It has a fast transient response to short duration events, and can be modified to allow for the inclusion of soft-clipping amplitude characteristics [Rossum]. With the addition of high-quality cubic interpolation, the filter output quality is excellent.

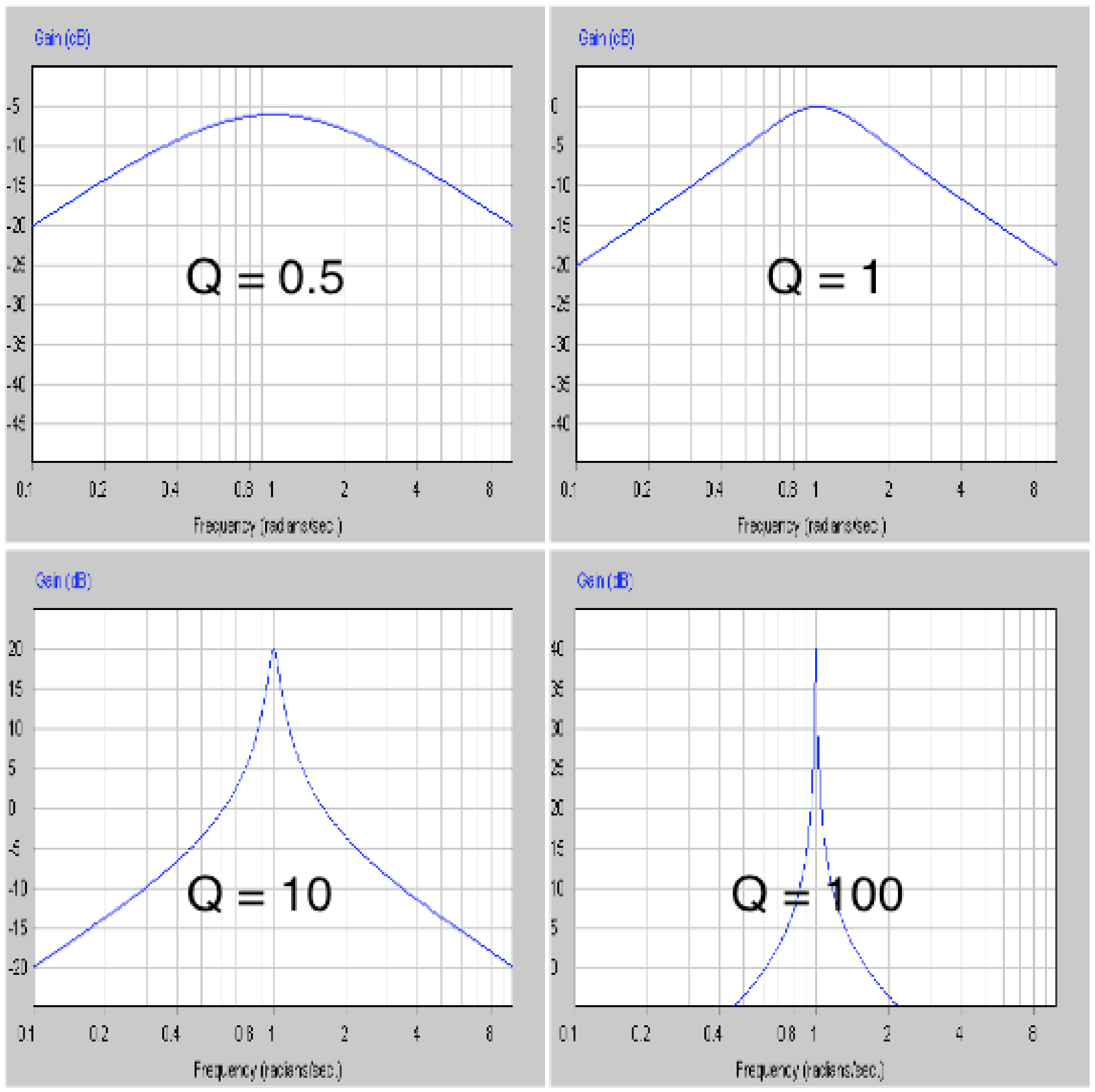


Figure 8: 4th-order, resonant band-pass filter with variable resonance.

[2] On a 1Ghz Apple Macintosh Powerbook G4, over 100 filters have been output in parallel without significantly high CPU load.

2.6 Control Model

The last major factor in the design and implementation on 'Cloud' is the control model. The proposed intent of the design was to have real-time, audio-rate control inputs for as many low level parameters as possible. Thus, various high-level interfaces could be built on top of the unit generator to exploit the variations within. All parameters have normalized, latched (sample and hold) inputs. As soon as the particle scheduler receives a trigger, all of the parameter control inputs are sampled for their current values. These values are then used in the calculation and generation of each successive grain. If a continuous change is needed in a parameter (e.g., frequency glide or filter sweep), another control input is provided to set the range and scaling.

This dynamic model allows for arbitrarily generated and shaped inputs as modulation sources. Furthermore, complex parameter-linking (such as that used in pulsar synthesis) can be set from a higher level control specification, without affecting the decoupled structure of the low-level generator loop.

The first input is the physical number of output channels. This can be updated dynamically, but requires re-computation of the spatial algorithm. The trigger input can take any source, and will output a new particle at every positive-going zero-crossing. The next two inputs are descriptors to sample wavetables and envelope function tables loaded dynamically into memory, which are accessed by number. The selected wavetable can be re-sampled at a different rate, creating pitch-shifting effects. Furthermore, a spectral formant can be specified that can be shifted independently of the playback period. The scan rate and current offset position into the chosen wavetable can be given, allowing for both time modulation effects, as well as sequence-able and statistical reordering of particles sent to the output. The duration of particles can be specified as any value in a very wide range, from a minimum of 4 samples (to account for linear interpolation) to a maximum limited only by system memory. Spatial position, which is defined in degrees, is combined with the maximum amplitude and envelope shape to produce the particle location. Filter center frequency and resonance set the values for the coefficients of the internal band-pass filter. Finally, the type of gain compensation for particle overlap as well as the sample-level interpolation quality can be set on a per-particle basic.

2.7 Assessment

'Cloud' is a fast, efficient implementation of generalized particle synthesis. It has the ability to synthesize, sample, and process thousands of grains in real-time on a modern personal computer. The design is modular, allowing for easy adaptation to many variations of particle-based processes. Furthermore, the algorithms are scalable, and could easily be ported to a stand-alone software package or hardware prototype capable of multichannel, poly-stream output in real-time. In the next section, we will take a closer look at the high-level user interface designed to take advantage of the particle model.

3. An Interface for Advanced Particle Transformations

3.1 Requirements

A generalized control scheme to perform high-level, multi-dimensional particle transformations is needed. The system should be powerful, providing numerous options and specifications. Furthermore, the various control devices should allow for morphological composition planning on multiple time scales, with an emphasis in directly manipulating the advanced particle processing parameters through sets of advanced graphical views.

‘Emission Control’ is the name of a graphical user interface library written in the Supercollider 3 programming language. It has been designed specifically for control of particle-based sound processing. The system is fast, efficient and completely configurable. A set of novel user interface elements have been constructed that work together to control all aspects of the underlying particle processor, from making simple parameter updates to automating complex multi-dimensional parameter morphologies. A tree diagram of the class hierarchy can be viewed in Figure 9.

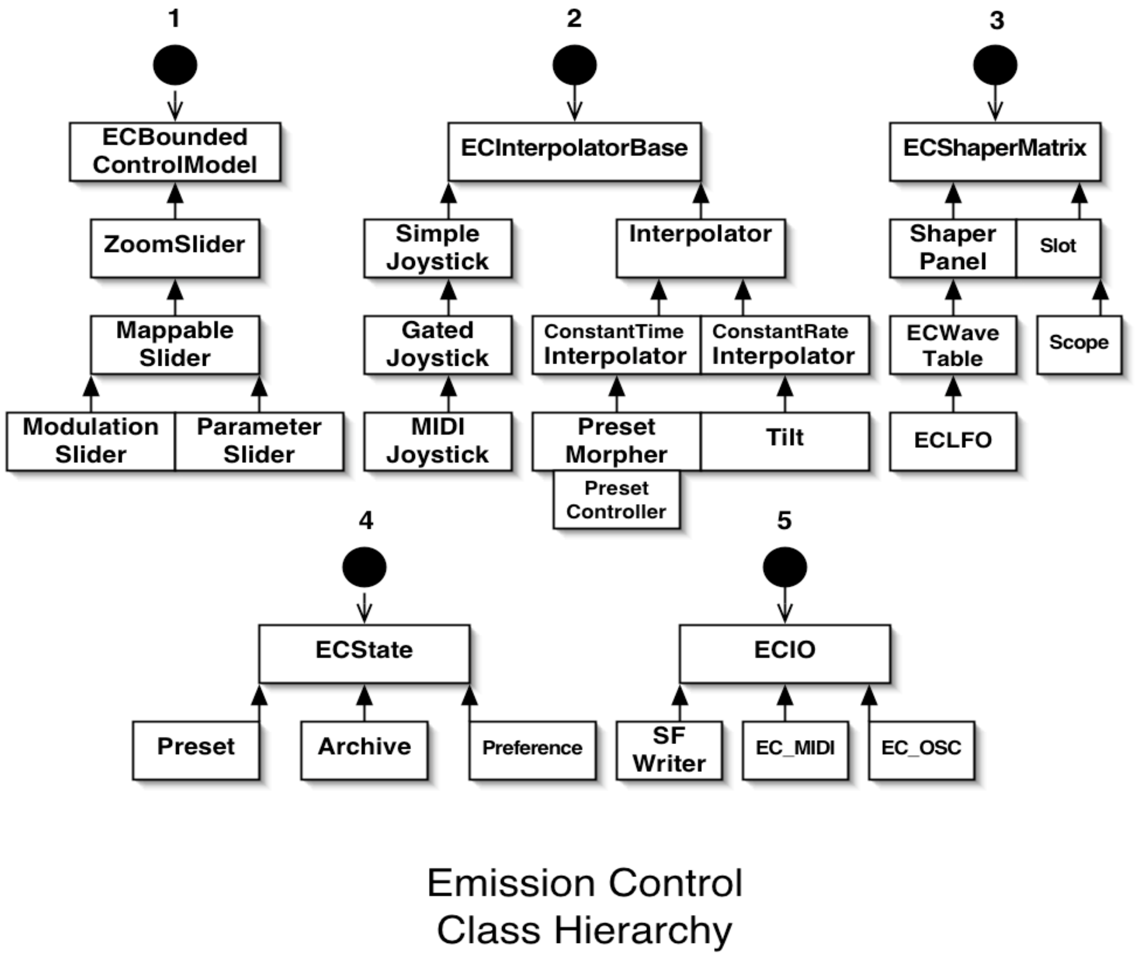


Figure 9: Emission Control Class Hierarchy

The library is organized into a five-part topology. The ECBoundedControlModel superclass defines all one-dimensional parametric controllers that can be mapped to 'Cloud'. Classes within this hierarchy include containers for viewable components such as range-bounded sliders, numerical views, and progress bars. All subclasses inherit the 'Zoomable' interface, which will be described in the next section. ECInterpolatorBase defines all types of multi-dimensional parametric interpolations. This includes two-dimensional software joysticks, constant-time and constant-rate interpolators, and arbitrary parameter-linking facilities. The ECShaperMatrix superclass defines all high-level modulation routing within the synthesis and processing unit generator graph (described in the chapter on DSP), and implements a complex modulation matrix for deterministic and statistical parameter control. ECState and all its subclasses maintain the current and past state of the entire system. This includes preset management, reading and writing archives (banks of presets and modulation routings), and storing user preferences. ECIO delegates all high-level input and output processing to its subclasses. This includes classes that implement the MIDI and OSC protocol for hardware controller mappings, as well as real-time reading and writing of multichannel soundfiles.

3.2 Novelty: Interface advances, problems, & solutions

A key advantage to particle-based processing routines is the ability to perform multiple high-level operations on a per-particle basis. Until recently, particle synthesis control architectures have been limited to pre-defined low-level models, such as wavetable granulation and pulsar synthesis. The user can manipulate the key parameters within the confines of the model, according to parametric range specifications hard-coded by the developer. 'Emission Control' opens up the processing model to allow for user-specified ranges of parametric control and constraint. Furthermore, no assumptions are made about which model to use. The boundaries between the various particle synthesis models are fuzzy and softly defined, allowing the user to continuously morph the system response across multiple models.

Devices that allow this extra degree of parametric freedom are the ZoomSlider and its subclass objects. A ZoomSlider has resolution zooming capabilities. At any point in time, the user of a subclass slider instance may rescale its upper and lower range limits, as well as define the mathematical curvature and direction of its mapping. A MappableSlider extends this capability by providing the user with parameter-linking facilities. An instance of MappableSlider can be seen in Figure 10. While simple in concept, the musical variations that can be achieved are phenomenal. For example, if the playback frequency of the waveform within each particle is inversely linked to the duration of each particle, varying one of the two parameters will cause the other to follow as the reverse function. This will result in a completely configurable form of Grainlet Synthesis [Roads]. This model can be extended to as many parameter-links as desired, and limited in scale, range, direction, and curvature.

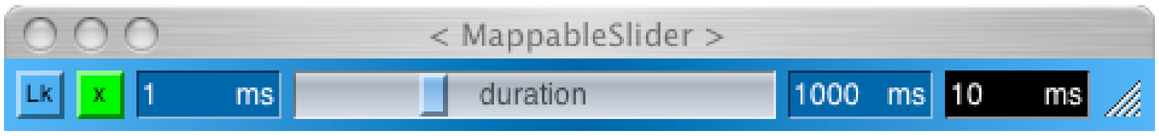


Figure 10: An instance of MappableSlider. The lower and upper bounds are set to 1 ms and 1000 ms respectively, from left to right. The current value is 10 ms. This slider is currently mapped to the X-axis of a joystick.

Combining the ‘Zoomable’ interface with two-dimensional, resizable joysticks allows for complex variations using simple gestures. The Joystick Class has many distinguishing features. Its elegant design allows it to control any number of

ZoomSliders simultaneously, interpolating each slider's output value continuously between the lower and upper bounds set by the user. The user can resize each Joystick instance on the X-axis and Y-axis, effectively compressing and expanding the quantization step size of mapped parameters. Furthermore, controls are provided to constrain the movement to only one axis, or in a perfectly slanted motion. The Joystick also produces a gate signal when clicked and released, to allow for opening and closing an arbitrary control signal routed from the modulation matrix (discussed below). This could be used for gating streams of particles with an amplitude envelope. An example of Joystick resizing is given in Figure 11.

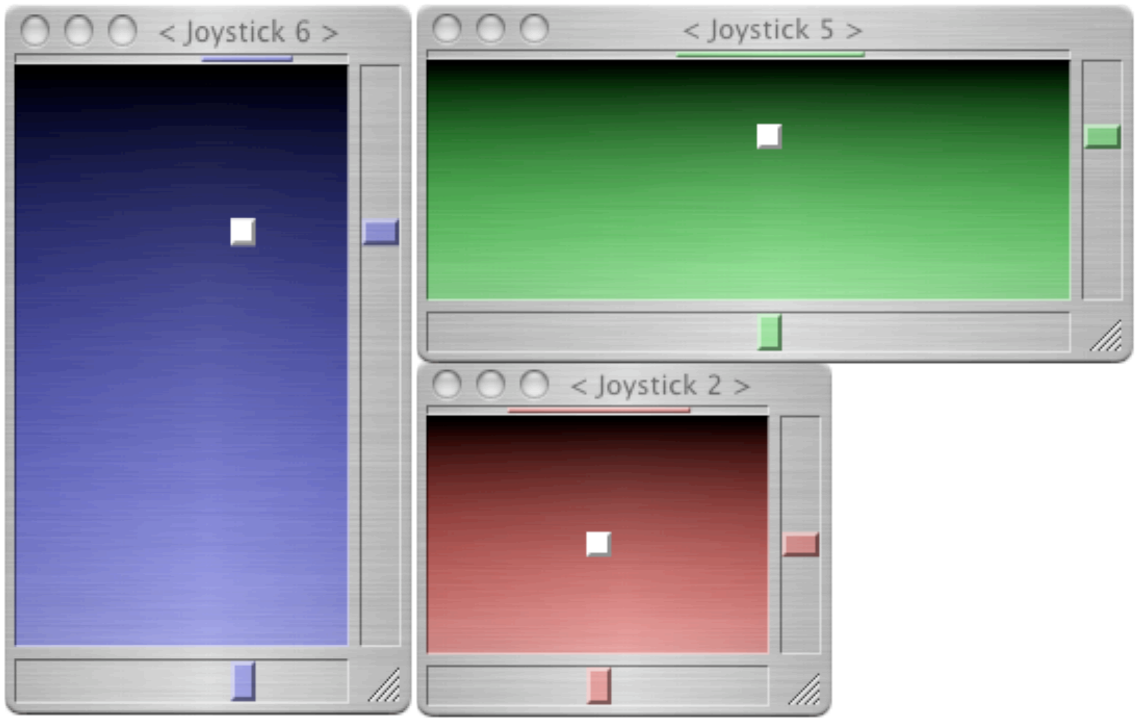


Figure 11: Software Joysticks. Resizing the window effectively rescales the quantization step size of parameters mapped to the X and Y axis.

The software joystick implementation can also be controlled from a hardware joystick module. A MIDI implementation allows for arbitrary mapping from the physical joystick to the position of the thumb on the screen. Alberto de Campo has designed a joystick module that outputs MIDI on sixteen channels from eight joysticks [de Campo]. This module can be seen in Figure 12. There are many ways in which to map from the hardware to the software. One could directly map the physical position of the hardware joystick to the position of the software joystick. The problem with this design is in the 8-bit limitations of the MIDI protocol. With only 128 values, the user will lose resolution with an increase in the size of the joystick on the screen, resulting in quantized steps instead of linear motion. A better implementation requires remapping the 128 MIDI values to the velocity (rate of change) of the thumb on the screen. This allows for more interesting interpolations, and can be customized by resizing the software joysticks.

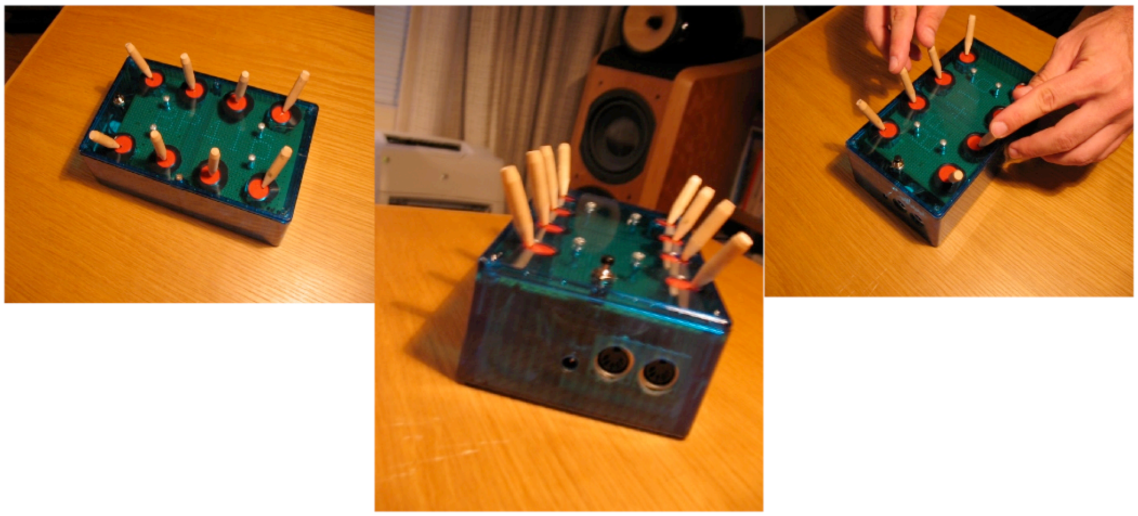


Figure 12: MIDI Joystick Controller (courtesy of Alberto de Campo).

The execution of musically-intuitive interpolations is an essential part of particle-based composition. Two types of interpolation algorithms, constant-time and constant-rate, have been embedded into the control framework. PresetInterpolator is a complex, constant-time interpolation algorithm. A PresetsController instance contains a reference to any number of presets. The user can specify an absolute time constant, within which all registered ZoomSlider's will interpolate from their current value and range to a target value and range. This is a powerful time-based technique, and can be set to an arbitrary length, allowing transformations on multiple timescales. Key features include playing and pausing the interpolator in real-time, saving and loading presets during a morph, and adjusting the number of ZoomSlider updates per second. An example of the PresetsController with the constant-time interpolation algorithm is given in Figure 13.

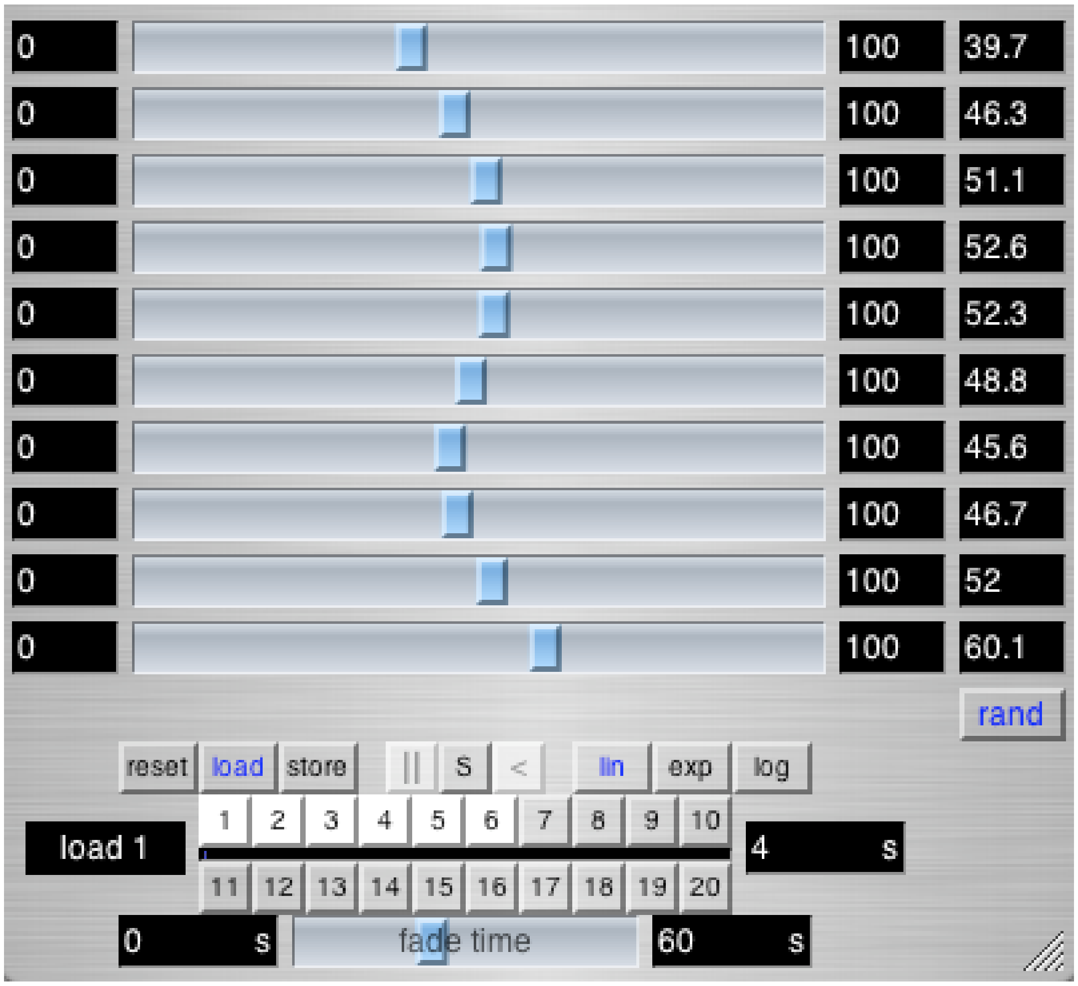


Figure 13: A prototype of a controller designed for constant-time interpolation.

One problem that was encountered during the coding of this feature had to do with the interpolation of the ZoomSlider ranges. If a ZoomSlider's range changes direction, there is a possibility that the current value will be outside the range of the lower and upper bounds for some percentage of the interpolation time. If the user pauses during such an occurrence, or stops an interpolation prematurely, the system will highlight the current value to show that it is out of range. This value will still be output. Another solution would be to clip the current value to the closest range boundary, but this would result in an audible step, which might be objectionable. A preference has been added to allow the user to select which action will be taken.

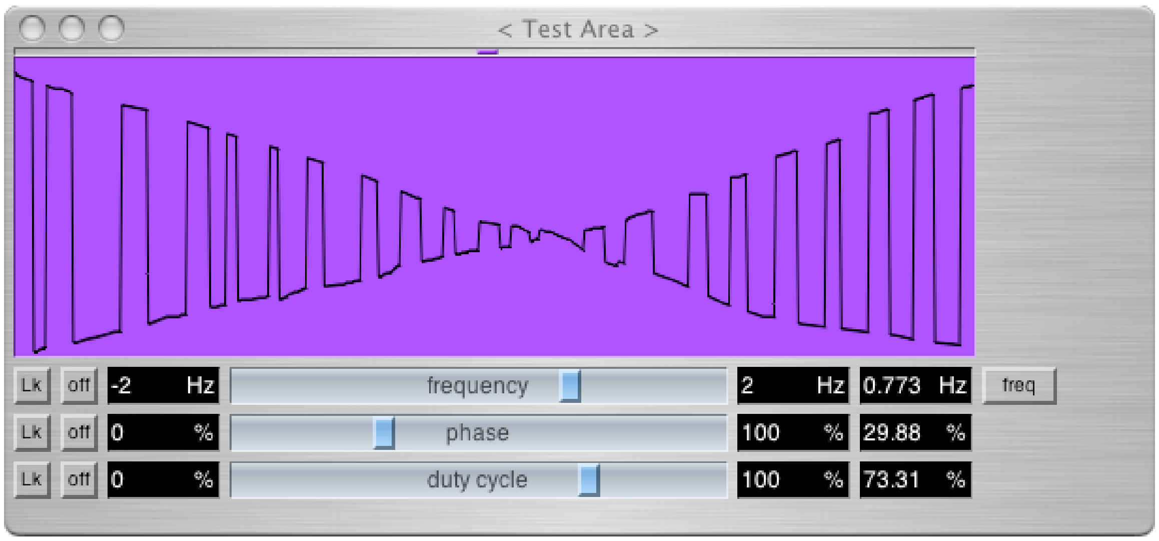
The constant-rate interpolation algorithm built into 'Emission Control' is a completely new concept. Loosely based on the 'TILT' feature found in pinball games, a 'Tilt' instance updates all registered ZoomSliders at a selectable constant-rate by some maximum amount in a single direction, as if the user interface has been bumped, shaken or jolted from one side. This technique creates fascinating, user-controllable unidirectional randomness. Since the algorithm is directly manipulating the position of multiple ZoomSliders with completely configurable ranges by slightly different amounts, a very natural-sounding parametric modulation occurs. Other features of the algorithm include linear or logarithmic slider motion, an edge-bouncing factor for simulating back and forth motion, and a deceleration factor.

One of the most important requirements for high-level control of particle parameters is a generalized modulation scheme. The latest interface built into 'Emission Control' is the ECShaperMatrix class hierarchy and its associated graphical views. This system is loosely based on the Electronic Analog Computers of the past [Slater], as well as the modulation matrix designs of modern synthesis and processing systems [Roads, Sasso].

In systems such as CloudGenerator and Native Instruments Reaktor, the modulation model is based on statistical distributions of parameters around center values within lower and upper bounds. Truax has described this probability distribution as a 'tendency mask', within which the user can specify the slope of the lower and upper bounds of a statistical distribution as a function of time. For example, in CloudGenerator the user can specify the beginning and ending lower and upper bounds of the frequency content of a 'cloud' of particles. In Native Instruments Reaktor, all particle-processing parameters have a center value and associated jitter (or deviation) parameter, around which the parameter will be randomly distributed. This same model has been used in Symbolic Sounds Kyma, ConstQ, and the Creatovox Project [Roads, de Campo].

In systems such as PulsarGenerator and Propellerheads Reason, the modulation model is extended to allow reading from arbitrary wavetables for parameter modulation. While Reason allows for simple geometric waveshapes such as linear or curved segments, PulsarGenerator allows for arbitrarily complex waveshape specifications, including hand-drawn curves and soundfiles loaded from memory.

'Emission Control' combines and extends these models, allowing for system-wide routing of complex mixtures of control signals to one or more destinations. At any point in time, the user can create a new modulation matrix known as a 'Shaper'. Every instance has four slots that can hold a modulator, each of which is mixed or multiplied together by some variable amount and routed to a modulation destination. Some of these modulators include low-frequency oscillators, multi-segment envelope generators, MIDI events, and hand-drawn re-scalable wavetables. The resulting control signal mix is sent to the modulation input of a chosen parameter, and then scaled to an appropriate value in order to produce the requested parametric variations. The user can control the center value of a parameter, and simultaneously vary the amount of modulation around that value. This extends the purely statistical distribution model, allowing for more deterministic, controlled transformations from many types of control signal sources. A master duration control allows the user to synchronize all modulators to the same clock, allowing for rhythmic and cyclic variations. An example of the ECWavetable user interface is given in Figure 14.



-2 Hz frequency 2 Hz 0.773 Hz

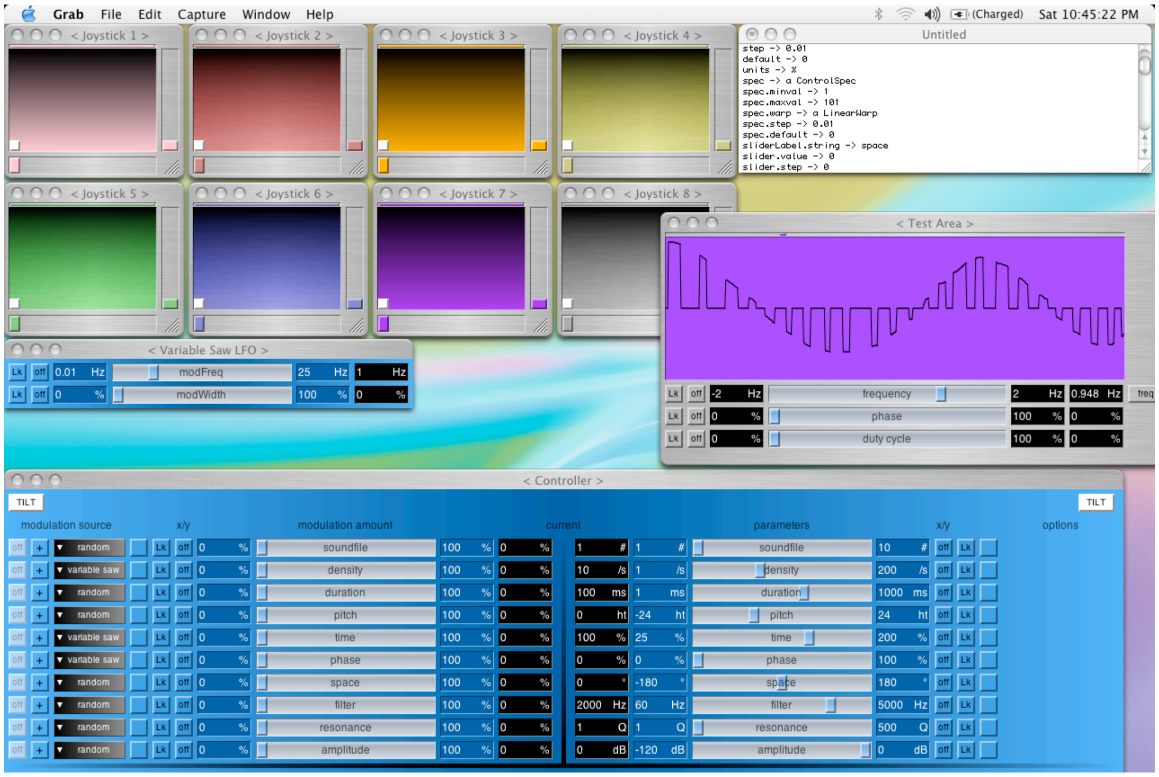
0 % phase 100 % 29.88 %

0 % duty cycle 100 % 73.31 %

Figure 14: Drawable Wavetable for use in the ShaperMatrix.

3.3 Assessment

Building a generalized control scheme and user interface for particle synthesis and processing is a complex task. 'Emission Control' is an attempt to embed a generalized particle-processing algorithm within a simple and intuitive transformation environment. Unlike its predecessors, 'Emission Control' is not linked to any underlying particle model. It is an open-ended system, defined directly through its usage. A user can work simultaneously on multiple timescales, setting constraints on parameters controlled by high-level interpolation devices and a matrix of modulators. Figure 15 shows the current state of the GUI.



```
step -> 0.01
default -> 0
units -> %
spec -> a ControlSpec
spec.minval -> 1
spec.maxval -> 101
spec.warp -> a LinearWarp
spec.step -> 0.01
spec.default -> 0
sliderLabel.string -> space
slider.value -> 0
slider.step -> 0
```

0.01 Hz	modFreq	25 Hz	1 Hz
0 %	modWidth	100 %	0 %

-2 Hz	frequency	2 Hz	0.948 Hz
0 %	phase	100 %	0 %
0 %	duty cycle	100 %	0 %

modulation source	x/y	modulation amount			current		parameters		x/y
random	off	0 %	soundfile	100 %	0 %	1 #	1 #	soundfile	10 #	off
variable saw	off	0 %	density	100 %	0 %	10 /s	1 /s	density	200 /s	off
random	off	0 %	duration	100 %	0 %	100 ms	1 ms	duration	1000 ms	off
random	off	0 %	pitch	100 %	0 %	0 ht	-24 ht	pitch	24 ht	off
variable saw	off	0 %	time	100 %	0 %	100 %	25 %	time	200 %	off
variable saw	off	0 %	phase	100 %	0 %	0 %	0 %	phase	100 %	off
random	off	0 %	space	100 %	0 %	0 °	-180 °	space	180 °	off
random	off	0 %	filter	100 %	0 %	2000 Hz	60 Hz	filter	5000 Hz	off
random	off	0 %	resonance	100 %	0 %	1 Q	1 Q	resonance	500 Q	off
random	off	0 %	amplitude	100 %	0 %	0 dB	-120 dB	amplitude	0 dB	off

Figure 15: The current state of the Emission Control GUI.

4. Future Work

4.1 Code Optimizations

Particle-based synthesis and processing routines are CPU-intensive. In order to create a real-time implementation, code must be optimized at both the signal-processing and interface layers. These optimizations must also be able to scale with the growing trend in the audio industry toward polyphonic, multichannel output.

The current prototype successfully executes on a 2004 Macintosh Powerbook G4 clocked at 1.2 Ghz. While this implementation allows for real-time, multichannel processing, the processor is always on the verge of overload due to the dynamic nature of the sound particle model. For instance, as the particle overlap increases, there is a non-linear increase in CPU usage. This is not a desirable characteristic. Even though the particle scheduler is a simple overlap and add linear mixer, the overhead of the

Supercollider language creates a non-linear increase in overall CPU usage. This problem could be solved by either writing optimizations into the Supercollider 3 open-source library, or directly porting to a stand-alone system.

‘Cloud’ and ‘Emission Control’ are currently written in two different programming languages. The signal-processing code is compiled and dynamically linked to the synthesizer in the Supercollider runtime environment. Furthermore, all GUI processing and control is written as high-level object-oriented code that must be interpreted in this same environment. Any minor change in the parameter relationships between the inner signal-processing loop and user interface could cause the system to become unstable under certain circumstances. The best solution would be to join the two pieces of code together into a single, low-level process. This will be discussed further in section 4.3.

Many low-level optimizations are possible within the current design. These include using memory-lookups rather than direct computation, making separate versions of the unit generators for more specific uses, lowering the total number of unit generator instantiations, and increasing the speed of the internal DSP loop.

Most computation within ‘Cloud’ is optimized for real-time output through the direct use of tables stored in memory. For example, all gain compensation functions use table-lookups. However, all major envelope shapes are calculated on a per-grain basis. This choice was made in order to correctly set the durations of multi-segment envelopes, which would be harder to design using pre-configured tables. Using tables for all types of envelopes will greatly reduce the number of calculations in any given particle generation period.

Separating out various subsets of the generator will also increase the speed of the process. For example, ‘Cloud’ currently determines the slope of the trigger input signal to calculate a weighted gain for the output particle amplitude, independent of the direct amplitude input. This equation is redundant if the input is already supplied. Using a ‘switch’ or other selection structure in the code will only create more complexity. Moving this and other special cases to their own generators will solve this redundancy.

Once an instance of ‘Cloud’ is instantiated, the running unit generator produces some measurable amount of CPU load. Determining the trade-off between multiple simultaneous instantiations and a single instantiation with greater CPU load will help to determine possible load optimizations. In general, the overhead of constructing a new unit generator for every stream of particles outweighs the simplicity of mixing multiple trigger sources together and feeding them into the same generator. However, measurements must be made to fully define this relationship. One problem that currently exists in the single generator model is that only one particle can be constructed per trigger in time. This limits the inter-onset delay to the duration of one sample, which is not a limitation when multiple generators are used in parallel. One solution would be to add an input parameter to set the number of particles to generate and output in parallel when a trigger is received.

A final low-level optimization would be to shorten the size and/or decrease the complexity of operations within the internal DSP loop. Most linear operations have already been combined as much as possible. However, there is still a switch statement that determines what type of interpolation to use on the sample within a particle. In order to maintain the highest sound quality, the algorithm could selectively choose the type of interpolation internally, dependent on either the current CPU usage or the re-sampling frequency for the particle. In the latter case, linear interpolation could be used ‘only’ when the re-sampling rate is not an integer-multiple of the original rate. Linear interpolation takes about twice as many multiples and adds per sample in the inner loop.

Computation at the control and GUI layer can be optimized in a number of ways. The most obvious problem that currently exists is the dependency on interpreted object-oriented code. Rewriting this layer in a lower level language could increase the throughput considerably! Two other interfaces that could be optimized are the parameter interpolation algorithms and the communication channel between the synthesis specification in the language and the SC3 Server application.

The constant-time and constant-rate interpolation algorithms spawn multiple, parallel threads within the operating system. Combining these into a single event-dispatching thread would decrease the spawning overhead significantly. An example of this can be found in the Java 2.0 v1.4.2 Swing Library implementation. Here, a general-purpose, pre-existing timer thread handles all GUI updates, thus foregoing the need to maintain and handle a linked list of running threads. This would not only lower the average CPU usage, it would also remove the CPU spikes that occur when threads are spawned in large groups.

A final update could help to lower the required throughput from the synthesis language to the SC3 Server. Sending synthesis control messages as bundles would decrease the throughput and processing latency simultaneously. Even greater returns would result from optimizing the unit generator graph function created on the SC3

Server. Minimizing the branches of the graph tree will maximize throughput from the input to the output.

4.2 Code Additions

Many additions to the design of 'Cloud' have been planned. These include an implementation for a delay line with feedback paths, a score generator and reader with optional visualizations, and psycho-acoustic enhancements to the perceived particle streams.

Building a delay line into the current implementation will allow for more advanced control schemes. For example, a delay tap could be inserted into the particle stream, allowing the user to specify both the time position of the tap, and the amount of the output to feed back into the input. This would allow for an advanced form of recursive, higher-order granulations. Furthermore, this could be extended to allow for multiple parallel feedback paths.

Currently, a table containing the input parameters to the particle processor is output to a file. Reading this back into the algorithm as a timed score would result in a repeatable, modifiable algorithmic control scheme. This could be further modified to provide the user with visualizations of the particle scattering algorithms in both time and space.

Psycho-acoustic enhancements to offset the panning, scattering, and spatial envelopment algorithms are also possible. A binaural spatial model using head-related transfer functions (HRTFs) would be a great alternative to the polygonal and spherical loudspeaker distributions currently in use [Senova, McAnally, Martin]. Plugging in a simple pair of headphones would allow the user to hear a proper stereo mix-down of the actual spatial scattering process. A model of sound source occlusion would provide efficient simulation of objects such as walls, tunnels, or other particle clouds within the environment, resulting in a more controllable virtual sound space [Farag, Blauert]. Many more enhancements are possible.

4.3 Software / Hardware Ports

A Macintosh OSX software port of the prototype is in the planning stages. The ‘Cloud’ and ‘Emission Control’ signal-processor and interface libraries will be re-implemented on top of the CoreAudio Framework. The GUI will be written using the Cocoa or Java Swing API, depending on the real-time capabilities of the libraries and their measured latencies. This port should considerably decrease the processor load, increase the polyphony and channel count on less powerful CPUs, and attain much larger particle transformations.

A hardware prototype is also being considered. Such a system would replace the software DSP and GUI algorithms with fabricated chips and physical controllers. This could include an array of motorized faders or infinite rotary encoders, LCDs, and a set of onboard joysticks. The inclusion of a MIDI keyboard would open the system up to a future market of sound designers and electronic musicians composing sound with a pre-configured, completely customizable particle synthesis instrument.

5. Conclusion

In this paper, it has been shown that particle-based sound synthesis and processing is a powerful historical and modern technique for sound design and music composition. Through the unification of various particle synthesis models, a new prototype system has been developed that extracts the most common features of other designs and combines them into a more comprehensive architecture. 'C loud' is a generalized sound particle algorithm that efficiently generates and distributes multi-dimensional particle streams and clouds in both time and space. 'Emission Control' is a generalized control scheme and user interface for composing all types of particle transformations, defined or undefined, with deep 'analog-style' control and flexibility. These developments, combined with future efforts to optimize, extend, and further develop the system architecture, can be understood as a viable alternative or addition to the existing family of synthesis and processing techniques.

References and Bibliography

Bencina, R. 2001 "Implementing Real-Time Granular Synthesis." Unpublished manuscript. Revised and updated version "Implementing Real-Time Granular Synthesis" in GreenBaum (Ed), Audio Anecdotes, A.K. Peters, Natick. Awaiting publication.

Cavaliere S., Piccialli A. 1997. "Granular synthesis of musical signals." In C. Roads, S.T. Pope, A. Piccialli, G. De Poli, eds. 1997. Musical Signal Processing. Swets & Zeitlinger B.V., Lisse, the Netherlands. pp. 155-186.

Chowning, J. 1971. "The Simulation of Moving Sound Sources." *Journal of the Audio Engineering Society* 19(2-6).

Dodge, C., Jerse, T. 1997. *Computer Music: Synthesis, Composition, and Performance*. Second Edition. New York: Schirmir. Chapter 6 "Subtractive Synthesis."

Dutilleux, P., De Poli, G., Zolzer, U. 2002. *DAFX - Digital Audio Effects*. Zolzer, U. (Ed), John Wiley and Sons. Chapter 7 "Time-Segment Processing."

Farag, H., Blauert, J., Alim, O. 2003. "Psychoacoustic Investigations On Sound-Source Occlusion." *Journal of the Audio Engineering Society* 51(7/8).

Fischman, R. 2002 "Application of Mathematical Models to the Generation of Organic Musical Structure and Discourse in Composition: Research Summary." *Proceedings of the 2002 International Computer Music Conference*. Göteborg, Sweden. San Francisco: International Computer Music Association. pp. 516-521.

Gabor, D. 1946. "Theory of communication." Journal of the Institute of Electrical Engineers Part III, 93: 429-457.

Gabor, D. 1947. "Acoustical quanta and the theory of hearing." Nature 159 (4044): 591-594.

Gabor, D. 1952. "Lectures on communication theory." Technical Report 238, Research Laboratory of Electronics. Cambridge, Massachusetts: Massachusetts Institute of Technology.

Kendall, G. "A 3-D Sound Primer: Directional Hearing and Stereo Reproduction." Invited paper. *The Computer Music Journal*, Vol. 19, No. 4. Cambridge, Massachusetts: MIT Press.

Lippe, C. 1994. "Real-time Granular Sampling Using the IRCAM Signal Processing Workstation." Contemporary Music Review, Vol. 10, United Kingdom, Harwood Academic Publishers, pp. 149-155.

Loomis, J., Klatzky, L., Golledge, G. 1999. "Auditory Distance Perception in Real, Virtual, and Mixed Environments." In Y. Ohta & H. Tamura eds., *Mixed Reality: Merging real and virtual worlds*. pp. 201-214. Tokyo: Ohmsha, 1999.

Otis, A., Grossman, G., Cuomo, J. 1968. "Four sound-processing programs for the Illiac II computer and D/A converter." Experimental Music Studios Technical Report Number 14. Urbana: University of Illinois. Paper by Otis, A. "Program 3. Time Rate Changing."

Roads, C. 1978a. "Automated granular synthesis of sound." Computer Music Journal 2(2): 61-62. Revised and updated version printed as "Granular synthesis of sound" in C. Roads and J. Strawn, eds. 1985. Foundations of Computer Music. Cambridge, Massachusetts: MIT Press. pp. 145-159.

Roads, C. 1985. "Granular synthesis of sound." In C. Roads and J. Strawn, eds. 1985. Foundations of Computer Music. Cambridge, Massachusetts: MIT Press. pp. 145-159.

Roads, C. 1991. "Asynchronous granular synthesis." In G. De Poli, A. Piccialli, and C. Roads, eds. 1991. Representations of Musical Signals. Cambridge, Massachusetts: MIT Press. pp. 143-185.

Roads, C., Jaroslaw, G. 1995. "'Analog' Control of Digital Audio Signal Processing." Les Ateliers UPIC and Département Musique, Université de Paris VIII. Paris, France.

Roads, C. 1996. *The Computer Music Tutorial*. Cambridge, Massachusetts: MIT Press.

Roads, C. 2001. "Sound composition with pulsars." *Journal of the Audio Engineering Society* 49(3).

Roads, C. 2002. *Microsound*. Cambridge, Massachusetts: MIT Press.

Roads, C., and J. Alexander. 1995. Cloud Generator Manual. Distributed with the program Cloud Generator. Internet: www.create.ucsb.edu.

Rossum, D. “Making digital filter sound ‘analog’.” *Proceedings of the 1992 International Computer Music Conference*. Strange, A. (Ed) San Francisco: International Computer Music Association. pp. 30-33.

Rumsey, F. 2002. “Spatial Quality Evaluation for Reproduced Sound: Terminology, Meaning, and a Scene-Based Approach.” *Journal of the Audio Engineering Society* 50(9).

Senova, M., McAnally, K., Martin, R. 2002. “Localization of Virtual Sound as a Function of Head-Related Impulse Response Duration.” *Journal of the Audio Engineering Society* 50(1/2).

Slater, D. 1998. “Chaotic Sound Synthesis.” *The Computer Music Journal*, Vol. 22, No. 2. pp. 12-19. Cambridge, Massachusetts: MIT Press.

Stockhausen, K., 2000. *Stockhausen on music: Lectures and Interviews*. Maconie, R. (Ed) Marion Boyers Publishers Inc. New York, NY.

Xenakis, I. 1992. Formalized Music. Revised edition. New York: Pendragon Press.

Acknowledgements

First and foremost, I would like to thank Curtis Roads for his support, enthusiasm, and love of electronic music. Without his help, none of this would be possible. Special thanks goes to my Graduate Committee (Professor Curtis Roads, Professor JoAnn Kuchera-Morin, and Professor Jack Loomis) and the Media Arts and Technology Faculty and Staff, who have made this research possible. My family and friends have been there for me through every moment. Garry Kling and Lance Putnam gave me incredible help and incite during the long hours on the 4th floor of South Hall. Laili Zandieh has been my love, light, and best friend. Finally, I would like to thank those who have an undying passion for something in their life, and will stop at nothing to achieve their dreams.