21
Technique 5
Grains

SECTION 21.1

Granular Synthesis

Granular synthesis derives from Gabor's theory of acoustic quanta (Gabor 1944). It is painting in sound with a pointillistic style. It is always seen as a computationally expensive method, and something requiring a lot of control data. It has been explored by composers and synthesists (Truax 1988; Xenakis 1971; Roads 1978; Stockhausen) to produce sounds not possible by any other methods. With this method we create a steady state spectrum by combining many short bursts of sound called *grains*. Usually the grains overlap, so the process requires concurrency/polyphony. There are several variations on the basic method, some which have nonoverlapping grains, some which use shorter or longer grains, some employing random distributions of timing and grain size, and some that are more uniform. We will examine a few of the more common approaches now.

A Grain Generator

A grain is simply a waveform modulated by a short envelope. In theory any envelope shape can be used, but it should be time symmetrical. For efficiency, a triangular, trapezoid, or raised cosine window is often the choice but the best is the Gaussian or bell-shaped curve. Figure 21.1 illustrates the main principle of granular synthesis. On the top row you can see a bell curve envelope and the wave packet formed when a continuous sinusoidal waveform is modulated with it.

Keypoint
Granular synthesis uses short packets of sound layered or sequenced to make more complex sounds.

The bottom row shows how three copies of the envelope overlap in time. In a real-time implementation we could actually use two alternating envelopes; at the point where the second envelope reaches a maximum the first is zero, so it can be reused if the source waveform or lookup table can be exchanged

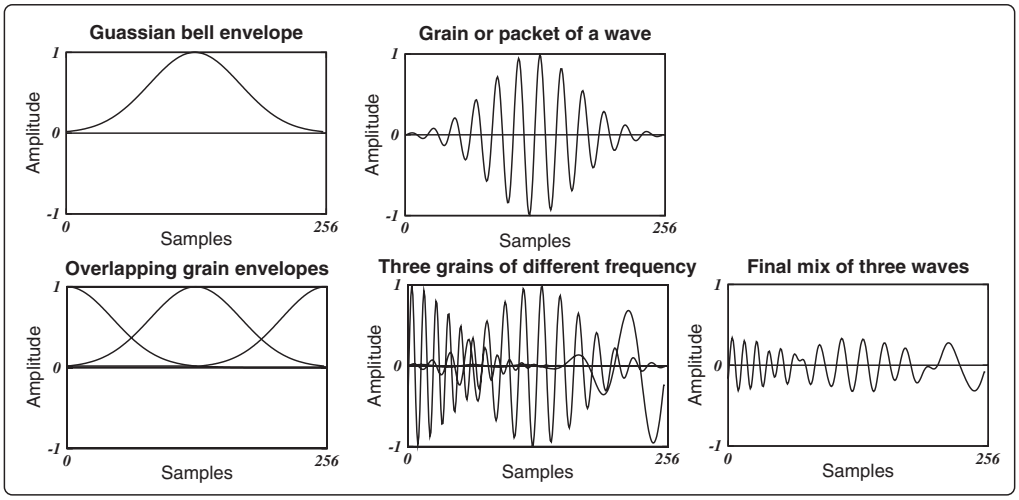

Figure 21.1
Granular synthesis of multiple sources using overlapping grains.

instantly. The second graph on the bottom row shows superimposed plots of three grains at different frequencies, and the final figure shows how they look when mixed into a continuous smooth waveform.

Figure 21.2 further illustrates why we need to do this instead of simply mixing or adjoining short sections of waveforms, and the diagram also provides an experimental framework in Pure Data that you can modify and play with to make offline renders of different grain mixtures. Two waveforms are created with different frequencies in the second row of figure 21.2, and the last graph shows what happens if we mix these. At the midpoints of both tables, where the transition occurs, there is no reason why an arbitrary waveform should have any particular value. If the values are very different when we mix the waves a bump or discontinuity occurs which will result in a click. The bottom row has two tables in which we have enveloped grains. They start and end on zero, and the shape of the Gaussian curve approaches zero asymptotically to the time axis, so we always get a smooth blend. We can move the start and end points of the curve backwards or forwards to get more or less overlap and the transition will remain smooth.

Figure 21.3 gives the three subpatches required to implement figure 21.2. The first shows how a signal expression is used to obtain the Gaussian curve from an exponential function. The second shows how we can take any segment of waveform, multiply it by the curve, and store it in a temporary new wavetable as a grain. In practice we might use a source of sampled audio in the wavetable and give an offset to choose different start points from the source material. Blending the grains is an interesting issue, and in the third part of figure 21.3 you can see a non-real-time solution that fills another array with a crossfade between two grains. For creating very dense granular sounds it's often best to

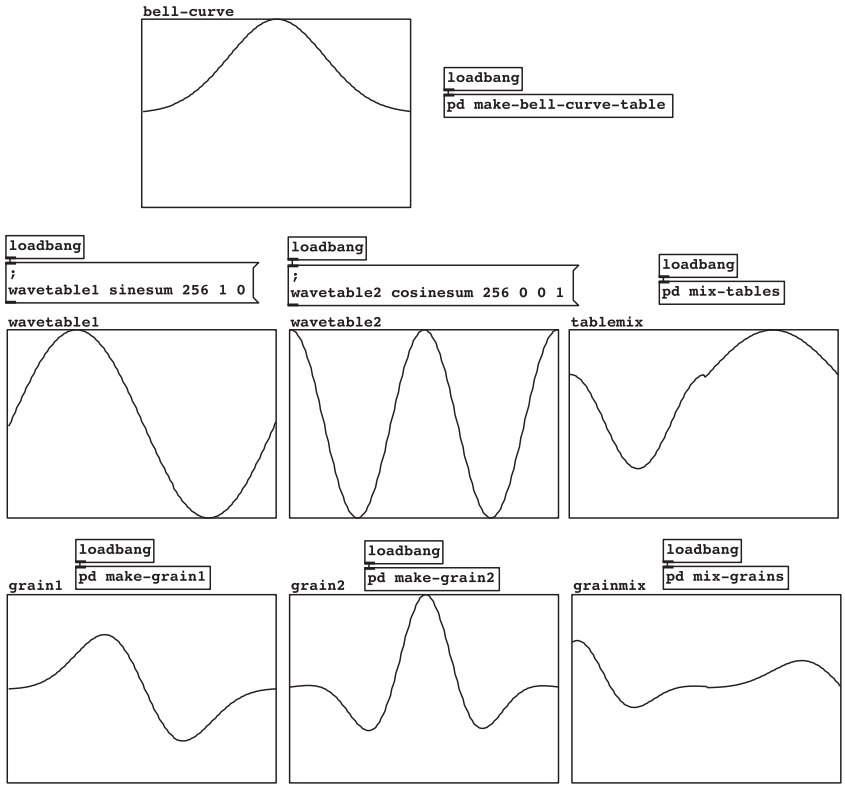

Figure 21.2
A Gaussian bell curve envelope used to form grains that can be mixed without clicking.

prerender textures, which is why I have demonstrated this offline approach. In Pure Data we can do something like the Csound `grain` opcode and make a grain *compositor* that overlays thousands of grains into the same array by repeatedly passing over and mixing, much like a sound-on-sound technique. The problem with this approach is that noise accumulates from digital errors, so in a moment we will look at how to combine grains in parallel and in real time.

Types of Granular Synthesis

There are many ways that grains can be combined. We can choose different durations and amplitudes for each grain envelope. We can choose different time distributions and overlaps. And we can also choose what waveform to put in each grain. Together, these possible parameters lead to several techniques that each have strengths for making different kinds of sounds.

Synchronous granular synthesis and PSOLA

Two familiar effects are *time-stretching* and *pitch-shifting*, which can be seen as two sides of a common process called pitch synchronous overlap and *add*

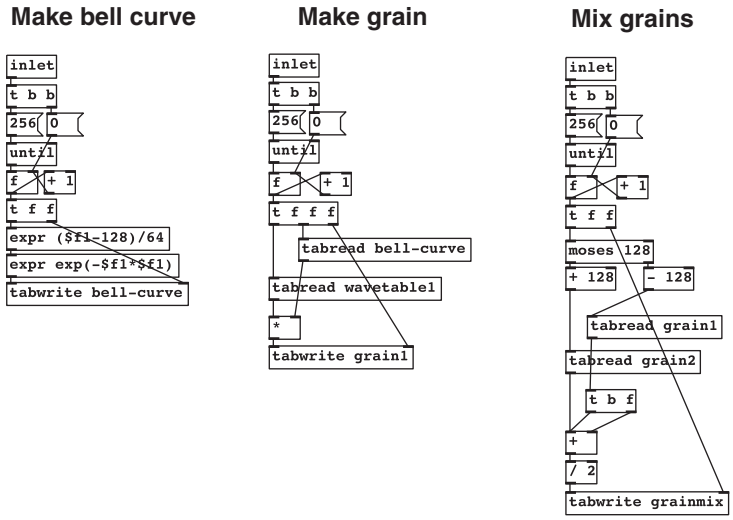

Figure 21.3
Operations on tables for granular synthesis.

(PSOLA). A sound can be divided into short segments that overlap and then each is played back in sequence so that the original sound is obtained. To time stretch a sound we add extra copies of the grains by changing the length and overlap duration so that the total length is greater or less than the original. In this case all the grains are taken from the same source (as seen in the first part of figure 21.4) and played back at the same original pitch, but the position they are chosen from slowly increments through the source file. For pitch shifting the playback rate of each grain waveform is altered and the grain envelope parameters are then chosen to obtain the original sound sample length. Both methods can add undesirable artifacts, a pitched quality at the frequency of the grain stream. Choosing the best frequency, duration, and overlap depends on the source material and most commercial time-stretch and pitch-shift plugins use an algorithm that analyses the sound first to choose the best parameters. The effect can be lessened by adding jitter, some random fluctuation to the grain sequence, which brings us to the subject of asynchronous granular synthesis.

Asynchronous granular synthesis

The second part of figure 21.4 illustrates a technique more commonly used to create sustained versions of dynamic sounds by randomly selecting grains from around a certain point in the source waveform and then mixing them randomly in time. Often the location in the source file is chosen from a Gaussian distribution, or maybe completely at random. Another technique is to use *random walks* around some point in the file or the *zigzag* method where the time direction reverses occasionally. This works well for turning strings, voices, and noisy sounds into rich textures. Although the grains themselves may be completely

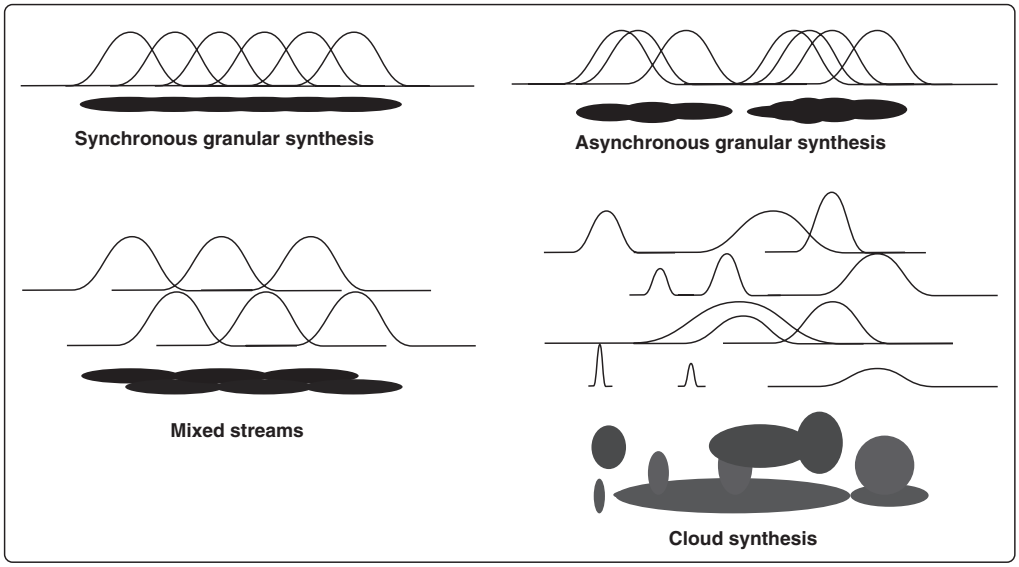

Figure 21.4
Types of granular synthesis.

aperiodic, the resulting texture retains the pitch of the original source material. Notice below the envelope graph I have drawn an artistic depiction of the resulting texture. In this case its amplitude will warble up and down since the superposition of grains whose contents are in phase will produce peaks while those places where the grain density is low or the packet contents are out of phase will cause quiet spots. Asynchronous granular synthesis is often improved by the use of some light reverb to spread out the unpredictable amplitude contours.

Sound Hybridisation

Granular techniques offer a useful kind of cross synthesis if we combine grains from two or more source waveforms. This can be done statistically, or it can be more carefully controlled by sequencing. A "round robin" or shuffling algorithm can give rise to sounds that take on the quality of two or more others when they are combined at the granular scale. The third diagram in figure 21.4 depicts two interleaved streams of grains from different sources. Taking this concept to the limit we can design entirely new sounds from fragments of existing ones in a way that mixing and splicing by hand would make impossible. The last part of figure 21.4 shows what is sometimes called *cloud* synthesis, because we take bits from many streams and combine them with different grain density, duration, overlap, randomness, or spacial position.

A Granular Texture Source

Let's explore further with a quick practical. We will make a tool that can be used to create continuous layers of sound from a short sample. It works well for voice, string, brass, and other pitched sources. Starting with figure 21.5 we see an abstraction that provides the main function, a flexible grain generator we shall call `grainvoice`. It relies on two arrays, which will be globally visible in the parent patch: one to store the source waveform, called `source-array`, and another to hold the grain envelope curve, called `grain-env`. The latter is fixed at 2,048 points, but the former may be resized to fit any sound sample we supply. The core component is a `vline~` object, which receives a message to create a line going from 0.0 to 1.0 over a certain time interval. This time interval is the grain duration, which is substituted into the second element of the second list. The line segment simultaneously addresses both tables, and their results are multiplied.

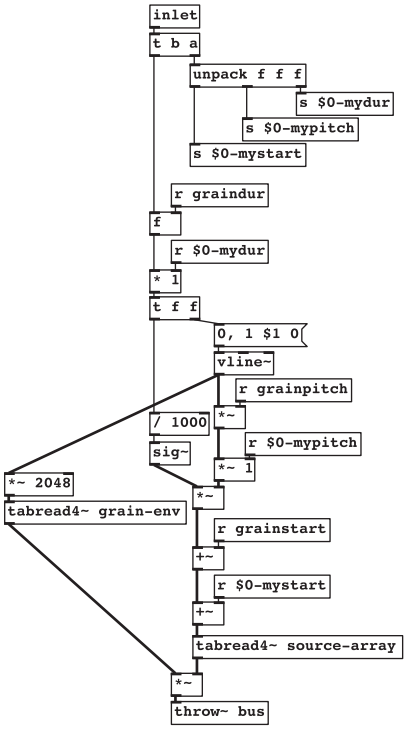

Figure 21.5
A real-time grain generator.

Parameters `grainpitch`, `graindur`, and `grainstart` control the sound. These are given in two forms. First, global versions set the pitch, duration, and start point for all the grain generators in the entire patch. These are modified by local versions (prefixed $0-my) which set the parameters unique to a

voice instance. To obtain the envelope index we multiply by the table size of 2,048. To get the table index we need to multiply by the sample rate. In this example the sample rate is 44,100, so you should load a compatible sound file or make the patch adaptive using [samplerate~]. Each grain voice uses [throw~] to send its output to a summation point in the main patch.

Four instances of the grain voice are used in figure 21.6. The main patch consists of five groups of objects, so let's deal with each group in turn. At the top left is a file loader comprising [openpanel], and a message to tell [soundfiler] to load the given file reference into array `source-array` (resizing as necessary). Beneath is a subpatch to fill the grain envelope table. Keen eyes may notice the Gaussian bell function has been replaced by a raised cosine window, sometimes

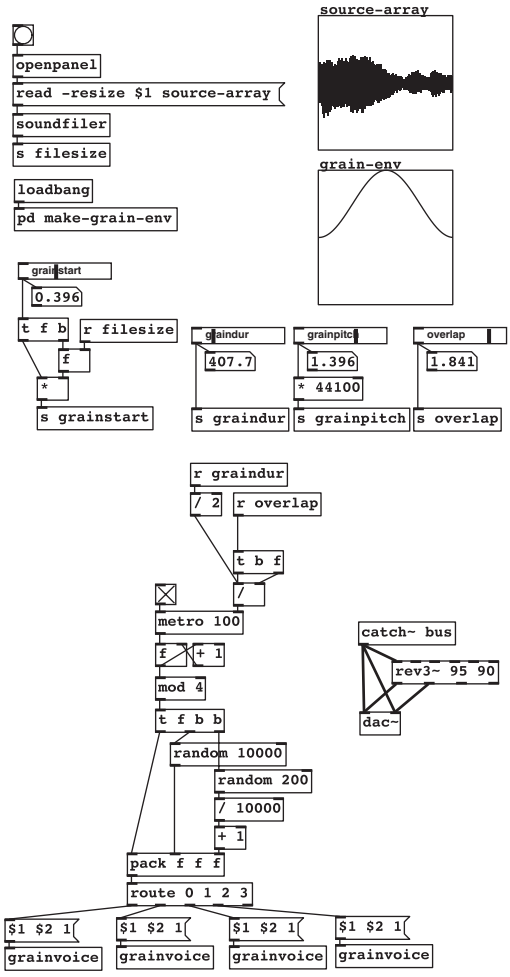

Figure 21.6
A sustained texture pad using four overlapping grain generators.

called a *Hanning window*, which is computed as $0.5 + \cos(x)/2$ between $-\pi$ and π. To the right of these subpatches are graphs of the two tables.

In the middle of the patch is a set of controls. Notice that soundfiler returns the size of the file loaded, in samples, which is broadcast to `filesize`. The first control uses this to scale the `grainstart` parameter so that 0.0 is always the start of the file and 1.0 is always the end. Grain duration is simply given in milliseconds, with a slider range between 10ms and 2000ms. The grain pitch is centered on 1.0, which plays back at the usual 44.1kHz. Moving this slider left or right of the middle slows or speeds up the sample replay. Finally there is an `overlap` parameter, which we shall examine in a moment. It ranges between 1.0 and 2.0.

The main part of the patch is at the bottom. It is a round-robin sequencer based on a metro driving a counter which prepends a number between 0 and 3 to a list via pack. These two-element lists containing a couple of random numbers are then distributed by route to four possible voices. The metronome period is calculated in accordance with the grain duration, but here is where we also involve the `overlap` parameter. With overlap set to 2 the clock period is 1/4 of the grain duration so the first grain will finish in time to be retriggered. For smaller values there will be less grain overlap. This changes the density of the texture. You may like to play with the random values that are substituted into the local grain voice parameters. These give a start offset of up to $10,000$ samples and a pitch variance of 2 percent, providing a thick chorusing effect. More focussed textures can be obtained by reducing the pitch and timing variation of the grains, whereas more chaotic, "fat" sounds result from bigger variations. To sweeten the patch I have included a copy of Miller's rev3~ reverb at the output.

SECTION 21.2

Time and Pitch Alteration

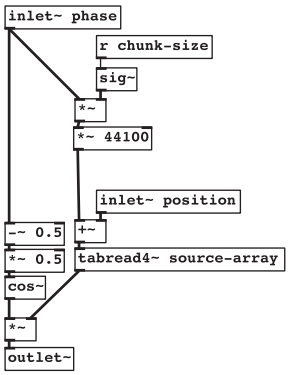

Figure 21.7
Another grain generator.

Here is a two-phase PSOLA effect adapted from a Pure Data help example. With it you can change the pitch or playback rate of a sound sample stored in a table. The core component is the grain generator seen in figure 21.7 of which there are two. A `position` signal value slowly sweeps through the file, but added on top of this is a more rapidly moving phasor that arrives on the `phase` inlet. This is scaled by `chunk-size`. Each cycle of the phasor produces a raised cosine window that modulates the chunk of sample data read from the source table. So, this grain generator uses a computed function instead of a lookup table for its envelope. Notice also that it's fixed to work at a 44.1kHz sampling rate. Below in figure 21.8 you can see two copies which are summed. Each is driven by two signals, a position value obtained from the vline~ line generator on the

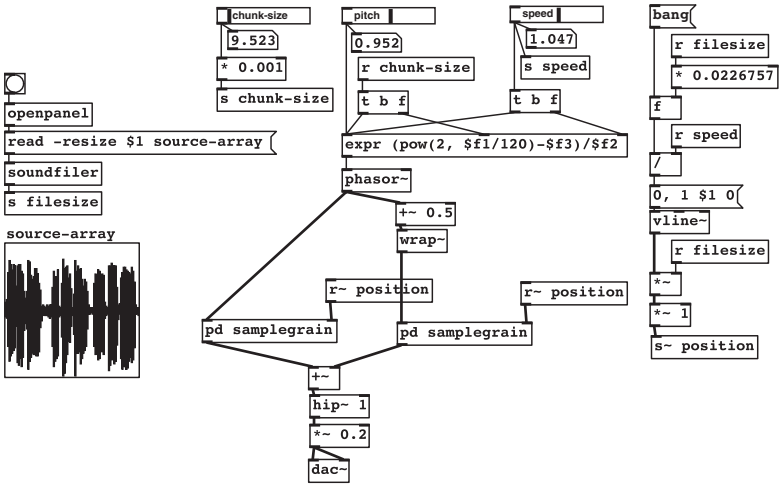

Figure 21.8
Time stretch and pitch shift using overlapping grains on opposite phases.

right-hand side, and a phasor for the grains. One of the phasors is made 180° out of phase with the other.

On top of the patch is an expression that computes the grain length from the playback speed, pitch, and chunk size controls. To use the patch, first load a sample, which will return its file size to the line generator for use in computing the line length. Then set up the speed and pitch controls. Pitch is in cents around a middle zero position, while speed is a factor with one (normal speed) in the centre.

References

Textbooks

Roads, C. (1996). *The Computer Music Tutorial*. MIT Press.
Tolonen, T., Valimaki, V. and Karjalainen, M. (1998). *Evaluation of Modern Sound Synthesis Methods*. Technical report 48, Laboratory of Acoustics and Audio Signal Processing, Dept. Electrical Engineering, Helsinki University of Technology.
Roads, C. (2004). *Microsound*. MIT Press.
Xenakis, I. (1971). *Formalized Music: Thought and Mathematics in Composition*. Indiana University Press. (2nd ed., Pendragon, 2001.)

Papers

Cavaliere, S., and Aldo Piccialli, A. (1997). "Granular synthesis of musical signals." In *Musical Signal Processing* (ed. Curtis Roads et al.). Routledge.
De Poli, G., and Piccialli, A. (1991). "Pitch-synchronous granular synthesis."

In *Representations of Musical Signals*, 187–219, ed. Giovanni de Poli, Aldo Piccialli, and Curtis Roads, MIT Press.
Gabor, D. (1947) "Acoustical quanta and the theory of hearing." *Nature* 159, no. 4044: 591–594.
Jones, D., and Parks, T. (1988). "Generation and combinations of grains for music synthesis." *Comp. Music J.* 12, no. 2: 27–33.
Keller, D., and Truax, B. "Ecologically-based granular synthesis." <http://ccrma.stanford.edu/∼dkeller/.pdf/KellerTruax98.pdf>.
Miranda, E. (1995). "Granular synthesis of sounds by means of a cellular automaton." *Leonardo* 28, no. 4: 297–300.
Roads, C. (1978). "Automated granular synthesis of sound." *Computer Music Journal* 2, no. 2: 61–62.
Truax, B. (1988). "Real-time granular synthesis with a digital signal processor." *Comp. Music J.* 12, no. 2: 14–26.