generator, hierarchical dust, and necklaces

michael winter (los angeles, ca; 2016)
for cristian alvear

general remarks (which can be used / excerpted for progam notes)

In March of 2016, the power went out in the industrial part of Downtown Los Angeles where I was living. One of the neighboring buildings, a fish distribution company, used an incredibly loud generator as backup. In an effort to capture the moment, I recorded the sound with a handheld video camera including the view of a Los Angeles sunset over the warehouse.

generator, hierarchical dust, and necklaces is an installation-performance piece consisting of several elements: the video and audio field recordings mentioned above (the 'generator'), an electronic manipulation that filters the sound of the generator (using a technique I call 'hierarchical dust'), a synthesized visualization of the hierarchical dust algorithm (juxtaposed with the video field recordings), and a set of patterns played on the guitar (which are analogous to mathematical objects called 'necklaces').

The form of the piece consists of one or more cycles / swells. Throughout each cycle, the digital filter oscillates between unstable and stable states. During an unstable state, the filter is triggered rapidly to change configurations (that is, which bands of the frequency spectrum it is filtering). In a stable state, a given filter configuration is held, and then faded out and back in as the guitar, which is tuned based on the 60 cycle hum of the generator, plays a specified repeated pattern.

The trajectory of the cycle is a swell articulated by loudness and the amount the filter can change every time it is triggered: at first, the bands of the filter have a high probably of being allowed to pass, but over time, each band has a higher and higher probability of being stopped.

Th guitar part was designed specifically to accompany / augment the installation intermittently which allows the piece to be played in concert or in an installation / exhibition setting with longer durations where the performer can enter and exit freely between cycles.

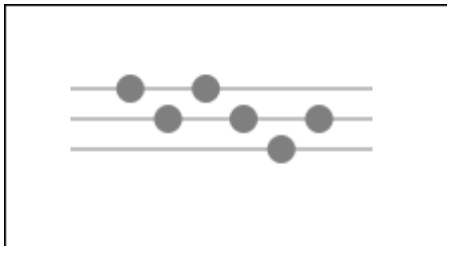
instructions for the guitarist

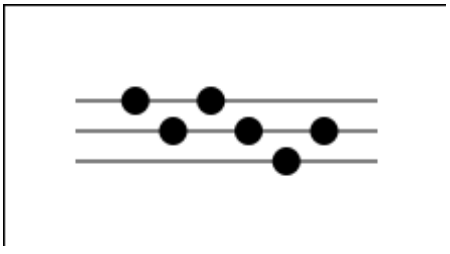
The lowest string, VI, of the guitar is tuned to the fundamental of the generator (which is a 60 cycle hum). The V string is tuned a perfect fifth above and the IV string is tuned a just minor 7th above the V string (this can be tuned from the 7th harmonic of the V string. The two highest strings, II and I, should be tuned approximately one octave higher than the IV string. This will allow all of these strings to be played open. The III string will have be stopped such that it is near the I and II string. Thus, the resulting notes of the highest three strings will all be slightly detuned from each other at approximately one octave higher than the IV string. This tuning brings the guitar substantially lower than usual, which contributes to the desired effect.

During the unstable states, the guitarist is free to improvise freely on all 6 notes but should do so in an extremely sparse manner (averaging a tone every few seconds).

During the stable states. The SuperCollider program (with user-controlled variables that are explained in more detail below) will display a plucking pattern (a necklace) in tablature-form to be realized on the highest three strings.

As illustrated below, the unstable and stable states are distinguished visually by the program in a window titled 'necklaces window'. During the unstable states, the upcoming necklace will be given in gray. Once a switch to a stable state occurs the necklaces turns black and the performer should play the pattern until it fades out (marking the return to an unstable state).



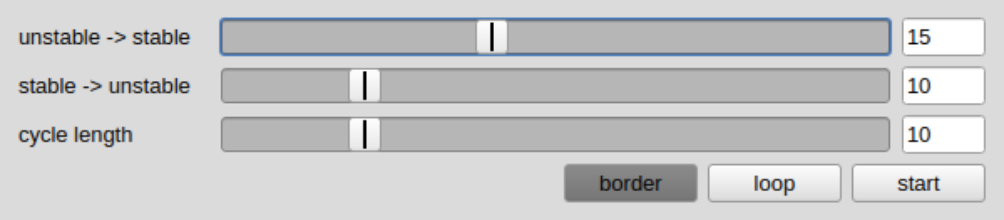


The peak volume of the generator should be quite loud. While the generator fades in and out in order to bring the guitar to the foreground during the stable states, it will likely be necessary to amplify the guitar.

SuperCollider program structure

The structure of the application is hopefully straightforward and does not warrant much explanation. The application launches three windows: 1) the 'hierarchical dust window' which, as explained in the following section, should be projected next to the video field recordings; 2) the 'necklaces window' which, as explained in the previous section, displays the plucking pattern that the guitarists plays during stable states; and 3) a graphical user interface (gui) that controls variables that the a user can manipulate. The gui is shown below along with an explanation of each function.

To launch the application, execute `generator_hierarchical_dust_and_necklaces_main.scd` in SuperCollider after booting the server (on linux, this is achieved by pressing cmd+enter with the cursor anywhere within the code block).



unstable → stable slider: This is the probability that after 15 seconds within an unstable state that a transition back to a stable state will occur. That is, every second after 15 seconds, there is a 1 in x chance that a transition will occur. The higher the number, in general, the longer the unstable states will persist. This variable goes into effect in realtime.

stable → unstable slider: This is the probability that after 15 seconds within a stable state that a transition back to an unstable state will occur. That is, every second after 15 seconds, there is a 1 in x chance that a transition will occur. Note that the transition back will take an additional 15 seconds such that the shortest possible duration of a stable state is 30 seconds. The higher the number, in general, the longer the stable states will persist. This variable goes into effect in realtime.

cycle length slider: This is the length in minutes that a cycle will last. This variable goes into effect at the end of each cycle.

border button: This button adds / removes the window decorations from the 'hierarchical dust window' (explained in more detail below).

loop button: This button allows the cycles to continue looping for performances with more than one cycle such as if the piece is installed for longer periods of time.

start button: This button (re)starts all processes after a 4 second delay.

The primary source code for the application is appended at the end of this score and can be downloaded from a git repository at `https://www.github.com/mwinter80/generator_hierarchical_dust_and_necklaces`. The whole package with the video and audio files is available upon request or can be downloaded at `http://www.unboundedpress.org/code_releases/generator_hierarchical_dust_and_necklaces_source.zip`. Note that moving the files or changing the filenames will break the application. The generation of this document (using LaTex) contains a version date in order to help track changes and the git repository will also detail commit changes. The piece was last tested on SuperCollider version 3.8.0.

video playback

The video field recordings should be played in random order and projected for the audience next to the output of the 'hierachical dust window'. Examples of 3 frames are provided on the following page. Note that this may need to be done using two computers / projectors: one that outputs the SuperCollider generated video from the 'hierarchical dust window' and one that plays back the video field recordings.

Another options is to run them from the same computer placing the 'hierarchical dust window' next to the program that is used to playback the video field recordings. Note that to do this, the window decorations of the desktop window manager need to be removed (which is not always possible with some operating systems and window managers). The SuperCollider program has a button to remove the window decorations from the 'hierarchical dust window' and some video playback program such as VLC have a similar option. There are also window managers such as OpenBox which allow users to toggle on and off window decorations. The example frames on the following page were made using the aforementioned technique with one computer.

Accompanying the sonic with the visual elements of the piece, while optional, is highly preferred.

document generated: 2017.08.31

generator_hierarchical_dust_and_necklaces_main.scd

```
(
// MAIN LAUNCH
~dir = thisProcess.nowExecutingPath.dirname;
"generator_hierarchical_dust_and_necklaces_synthdef.scd".loadRelative(true, {
	"generator_hierarchical_dust_and_necklaces_visuals.scd".loadRelative(true, {
		Buffer.read(s, thisProcess.nowExecutingPath.dirname +/+ "../audio/generator.wav", action: {
			|buf|
			~buf = buf;
			{~generateVisuals.value(buf)}.defer;
		});
})});
)
```

generator_hierarchical_dust_and_necklaces_synthdef.scd

```
(
SynthDef(\hierarchical_dust, {
	arg stable = 10, unstable = 10, buf = 0, loop = 0, cycle_len = 10;
	var local_in, hold, change, state, latch, hierarchical_dust, generator, env_master, env_spectrum, env_vol,
	    chain, spectrum_mult;

	// Feedback in state
	local_in = LocalIn.kr(2, 0);
	// Make each state last at least 15 seconds, however these variables could be different / manipulated in the
	    array [15, 15, 15]
	// That is, they could be turned into user variables
	// Note the second two account for the fade in and fade out of the necklace, so it will be at least 30 seconds
	    long
	hold = PulseCount.kr(Impulse.kr(1), Changed.kr(local_in[0])) > Select.kr(local_in[0], [15, 15, 15]);
	// Change state trigger
	change = TWChoose.kr(Impulse.kr(1), [0, 1], [Select.kr(local_in[0] > 0, [unstable, stable]), 1].normalizeSum) *
	     hold;
	// Change state
	state = Stepper.kr(change + TDelay.kr(local_in[0] > 1, 15), 0, 0, 2);

	// Monitor
	Poll.kr(Impulse.kr(1), hold, \hold);
	Poll.kr(Impulse.kr(1), change, \change);
	Poll.kr(Impulse.kr(1), state, \state);

	// Trigger for filter changes
	hierarchical_dust = (
		Impulse.kr(8) *
		(TRand.kr(0, 1, Impulse.kr(8)) <= 0.5)  *
		(TRand.kr(0, 1, Impulse.kr(1)) <= 0.75) *
		(state <= 0)
	);

	// Playback the soundfile
	generator = PlayBuf.ar(1, buf, BufRateScale.kr(buf), 1, 0, 1);
	latch = Impulse.kr(Latch.kr((60 * cycle_len).reciprocal, local_in[1]));
	env_master = EnvGen.kr(Env.sine(Latch.kr(60 * cycle_len, latch)), latch * (loop + Impulse.kr(0)));
	Poll.kr(Impulse.kr(1), env_master, \env);

	// Feedback out state
	LocalOut.kr([state, latch]);

	// Gate bins of the FFT
	env_spectrum = pow(env_master, 3) * 0.75;
	chain= FFT(LocalBuf(128).clear, generator);
	spectrum_mult = { |i| TRand.kr(0, 1, hierarchical_dust) > Latch.kr(env_spectrum, hierarchical_dust)} ! 64;
	chain = chain.pvcalc(64, {|mags, phases| [mags * spectrum_mult, phases] });

	// Output
	env_vol = pow(env_master, 2);
	Out.ar([0,1], IFFT(chain).dup * env_vol * (1 - EnvGen.kr(Env.asr(15, 0.9, 15, \sine), state % 2)));

	// Send info to Visuals
	SendTrig.kr(Impulse.kr(24), 0,
		1 - EnvGen.kr(Env.sine(1/6.0, env_vol), Select.kr(state > 0, [hierarchical_dust, state - Delay1.kr(
		    state) < 0])));
	SendTrig.kr((state - Delay1.kr(state) < 0) * PulseCount.kr(Changed.kr(local_in[0])) >= 1, 1);
	SendTrig.kr(Changed.kr(state), 2, state > 0);
	SendReply.kr(Impulse.kr(24), '/tr', spectrum_mult, 3);
}).send(s);
)
```

generator_hierarchical_dust_and_necklaces_visuals.scd

```
(
~generateVisuals = {
	arg buf;
	var control_window, width_cw = 600, height_cw = 100,
	stable_slider, unstable_slider, cycle_slider, stable_val, unstable_val, cycle_val,
	border = true, border_button, loop_button,start_button,
	hierarchical_dust_window, width_hdw = 600, height_hdw = 600, shade_hd = 1, spectrum_mult_hd = Array.fill(128,
	    {1}), state_hd = 0,
	necklaces_window, width_nw = 400, height_nw = 600, shade_n = 0, reset_hd_window,


```

```
	// All the possible necklaces
	necklaces = [[[2,0,1],[2,1,0]],
		[[2,0,1,0],[2,1,0,1],[2,2,0,1],[2,1,2,0],[2,2,1,0]],
		[[2,0,1,0,1],[2,1,0,1,0],[2,2,0,1,0],[2,0,2,0,1],[2,1,2,0,1],[2,2,1,0,1],
			[2,2,2,0,1],[2,0,2,1,0],[2,1,2,1,0],[2,1,2,2,0],[2,2,1,2,0],[2,2,2,1,0]],
		[[2,0,1,0,1,0],[2,1,0,1,0,1],[2,2,0,1,0,1],[2,0,2,0,1,0],[2,1,2,0,1,0],[2,2,1,0,1,0],
			[2,2,2,0,1,0],[2,0,2,1,0,1],[2,0,2,2,0,1],[2,1,0,2,0,1],[2,1,2,1,0,1],[2,1,2,2,0,1],
			[2,2,0,2,0,1],[2,2,1,2,0,1],[2,2,2,1,0,1],[2,2,1,0,2,0],[2,0,2,1,2,0],[2,0,2,2,1,0],
			[2,1,0,2,2,0],[2,1,2,1,2,0],[2,1,2,2,1,0],[2,1,2,2,2,0],[2,2,1,2,1,0],[2,2,1,2,2,0],
			[2,2,2,1,2,0],[2,2,2,2,1,0]],
		[[2,0,1,0,1,0,1],[2,1,0,1,0,1,0],[2,2,0,1,0,1,0],[2,0,2,0,1,0,1],[2,1,2,0,1,0,1],
			[2,2,1,0,1,0,1],[2,2,2,0,1,0,1],[2,0,1,2,0,1,0],[2,0,2,1,0,1,0],[2,0,2,2,0,1,0],
			[2,1,0,2,0,1,0],[2,1,2,1,0,1,0],[2,1,2,2,0,1,0],[2,2,0,2,0,1,0],[2,2,1,2,0,1,0],
			[2,2,2,1,0,1,0],[2,2,2,2,0,1,0],[2,0,1,2,1,0,1],[2,0,1,2,2,0,1],[2,0,2,0,2,0,1],
			[2,0,2,1,2,0,1],[2,0,2,2,1,0,1],[2,0,2,2,2,0,1],[2,1,0,2,1,0,1],[2,1,0,2,2,0,1],
			[2,1,2,0,2,0,1],[2,1,2,1,2,0,1],[2,1,2,2,1,0,1],[2,1,2,2,2,0,1],[2,2,0,2,0,1,0],
			[2,2,0,2,2,0,1],[2,2,1,0,2,0,1],[2,2,1,2,1,0,1],[2,2,1,2,2,0,1],[2,2,2,0,2,0,1],
			[2,2,2,1,2,0,1],[2,2,2,2,1,0,1],[2,2,2,2,2,0,1],[2,0,2,0,2,1,0],[2,0,2,1,2,1,0],
			[2,0,2,1,2,2,0],[2,0,2,2,1,2,0],[2,0,2,2,2,1,0],[2,1,0,2,1,2,0],[2,1,0,2,2,1,0],
			[2,1,0,2,2,2,0],[2,1,2,0,2,2,0],[2,1,2,1,2,1,0],[2,1,2,1,2,2,0],[2,1,2,2,2,1,0],
			[2,1,2,2,2,1,0],[2,1,2,2,2,2,0],[2,2,0,2,2,1,0],[2,2,1,2,1,2,0],[2,2,1,2,2,1,0],
			[2,2,1,2,2,2,0],[2,2,2,1,2,1,0],[2,2,2,1,2,2,0],[2,2,2,2,1,2,0],[2,2,2,2,2,1,0]]],

	binary_map = [[0, 0], [0, 1], [1, 0]], binary_rep,
	necklace_len_count = Array.fill(necklaces.size, {1}),
	necklace_count = all {: Array.fill(n.size, {1}), n <- necklaces },
	synth, osc_func, necklace_func, necklace, necklace_transition, necklace_fade_in, necklace_fade_out, run = true;

	// Generate the necklaces
	necklace_func = {var necklace_len_index, necklace_index;
		necklace_len_index = ({|i| i} ! necklaces.size).wchoose(necklace_len_count.normalizeSum);
		necklace_len_count = necklace_len_count + 1;
		necklace_len_count[necklace_len_index] = 0;

		necklace_index = ({|i| i} ! necklace_count[necklace_len_index].size).wchoose(necklace_count[
			necklace_len_index].normalizeSum);
		necklace_count[necklace_len_index] = necklace_count[necklace_len_index] + 1;
		necklace_count[necklace_len_index][necklace_index] = 0;
		necklace = necklaces[necklace_len_index][necklace_index];
	};

	// Fade necklaces in and out
	necklace_fade_out = Routine {
		26.do({ arg i; shade_n = (25-i).abs/25.0; (1.0/25).wait;});
		necklace_fade_out.yieldAndReset;
	};
	necklace_fade_in = Routine {
		2.wait; necklace_func.value; 26.do({ arg i; shade_n = i/25.0; (1.0/25).wait;});
		necklace_fade_in.yieldAndReset;
	}.play;
	necklace_transition = Routine {
		necklace_fade_out.play; 1.wait; necklace_fade_in.play;
		necklace_transition.yieldAndReset;
	};

	// Get messages from SynthDef
	osc_func = OSCFunc({ arg msg, time;
		switch(msg[2],
			0, {shade_hd = msg[3]},
			1, {necklace_transition.play},
			2, {state_hd = msg[3]},
			3, {spectrum_mult_hd = msg[3..]}
	)},'/tr', s.addr);

	reset_hd_window = {
		arg border = true, isLaunch = true;
		// Create window for projection
		hierarchical_dust_window = Window("hierarchical dust window",
			if(isLaunch,
				{Rect(100, Window.availableBounds.height - height_hdw, width_hdw, height_hdw)},
				{hierarchical_dust_window.bounds}), true, border);
		hierarchical_dust_window.background = Color.white;
		hierarchical_dust_window.front;

		// Animate
		hierarchical_dust_window.drawFunc = {
			Pen.use {
				Pen.color = Color.gray(1-shade_hd);
				Pen.addRect(Rect(0, 0, hierarchical_dust_window.bounds.width, hierarchical_dust_window.
					bounds.height));
				Pen.perform(\fill);

				Pen.color = Color.gray(shade_hd);
				if(state_hd == 1, {
					binary_rep = (all {: binary_map[x], x <- necklace }).flatten;

					{ |i|
						Pen.line(
							hierarchical_dust_window.bounds.width-if(binary_rep[i] ==
								1,{40},{30})@(((hierarchical_dust_window.bounds.height-40)
									/ 128) * i + 150),
							hierarchical_dust_window.bounds.width-20@(((
								hierarchical_dust_window.bounds.height-40) / 128) * i +
								150))
```

```
                                        } ! binary_rep.size;});

                                { |i| if(spectrum_mult_hd[i] == 1,
                                        {Pen.line(
                                                20@(((hierarchical_dust_window.bounds.height-40) / 128) * i + 150),
                                                40@(((hierarchical_dust_window.bounds.height-40) / 128) * i + 150))})}
                                                     ! 64;
                                Pen.stroke;
                        };
                };
        };
        reset_hd_window.value;

        // Create window for score
        necklaces_window = Window("necklaces window",
                Rect(width_hdw + 100, Window.availableBounds.height - height_nw, width_nw, height_nw));
        necklaces_window.background = Color.white;
        necklaces_window.onClose = { run = false; hierarchical_dust_window.close; control_window.close; ~synth.free };
        necklaces_window.front;

        // Animate
        necklaces_window.drawFunc = {
                Pen.use {
                        Pen.color = Color.gray(if(state_hd == 1,{0}, {0.5}), shade_n);
                        { |i| Pen.line((200-75)@((i-1)*15 + 300), (200+75)@((i-1)*15 + 300)) } ! 3;
                        Pen.stroke;

                        { |i| Pen.addOval(
                                Rect(200 - 75 + ((i+1) * (150.0 / (necklace.size + 2)) + 4),
                                        (2-necklace[i]-1) * 15 + 300 - 7, 14, 14)) } ! necklace.size;
                        Pen.perform(\fill);
                        Pen.stroke;
                };
        };

        // Refresh
        { while { run } { hierarchical_dust_window.refresh; necklaces_window.refresh; 24.reciprocal.wait; } }.fork(
            AppClock);

        // Create window for user controls
        control_window = Window.new("control window",
                Rect(100, Window.availableBounds.height - height_hdw - height_cw - 30, width_cw, height_cw));
        control_window.onClose = { run = false; hierarchical_dust_window.close; necklaces_window.close; ~synth.free };
        control_window.front;
        unstable_val = TextField().fixedWidth_(50).string_("15");
        stable_val = TextField().fixedWidth_(50).string_("10");
        cycle_val = TextField().fixedWidth_(50).string_("10");
        unstable_slider = Slider(control_window).orientation_(\horizontal).action_({
                var scaled_val = (unstable_slider.value * 25 + 5).trunc;
                ~synth.set(\unstable, scaled_val);
                unstable_val.string = scaled_val;});
        unstable_slider.value = 0.4;
        stable_slider = Slider(control_window).orientation_(\horizontal).action_({
                var scaled_val = (stable_slider.value * 25 + 5).trunc;
                ~synth.set(\stable, scaled_val);
                stable_val.string = scaled_val;});
        stable_slider.value = 0.2;
        cycle_slider = Slider(control_window).orientation_(\horizontal).action_({
                var scaled_val = (cycle_slider.value * 25 + 5).trunc;
                ~synth.set(\cycle_len, scaled_val);
                cycle_val.string = scaled_val;});
        cycle_slider.value = 0.2;
        border_button = Button(control_window).states_([["border", Color.black], ["border", Color.black, Color.grey]]).
            value_(1).action_(
                {|v| hierarchical_dust_window.close; reset_hd_window.value(if(v.value == 1,{true},{false}), false)});
        loop_button = Button(control_window).states_([["loop", Color.black], ["loop", Color.black, Color.grey]]).
            action_(
                {|v| ~synth.set(\loop, v.value)});
        start_button = Button(control_window).states_([["start", Color.black]]).action_(
                {|v| Routine{
                        state_hd = 0;
                        necklace_fade_in.play;
                        ~synth.free;
                        (1).wait;
                        ~synth = Synth.newPaused(\hierarchical_dust, [\buf, buf]);
                        (1).wait;
                        {stable_slider.valueAction = stable_slider.value;
                        unstable_slider.valueAction = unstable_slider.value;
                        cycle_slider.valueAction = cycle_slider.value;
                                loop_button.valueAction = loop_button.value;}.defer;
                        (2).wait;
                        ~synth.run;
        }.play;});
        control_window.layout = VLayout(
                HLayout([StaticText().string="unstable -> stable", stretch: 1], [HLayout(unstable_slider, unstable_val)
                    , stretch: 4]),
                HLayout([StaticText().string="stable -> unstable", stretch: 1], [HLayout(stable_slider, stable_val),
                     stretch: 4]),
                HLayout([StaticText().string="cycle length", stretch: 1], [HLayout(cycle_slider, cycle_val), stretch:
                    4]),
                HLayout(nil, nil, nil, border_button, loop_button, start_button)
        );
};
)
```