

Lamentations I: Remembering Clive Wearing

an opera

michael winter (mexico city, mx; 2017)

Acclaimed musician and musicologist Clive Wearing suffers from one of the worst known cases of amnesia after contracting herpes encephalitis. Wearing's caretakers encouraged him to keep a journal. *Lamentations I: Remembering Clive Wearing* is an opera that sets entries from Clive Wearing's journal. At intervals prescribed by the journal itself, entries are read and accompanied by a sonic and visual flickering / flourish of activity using recordings of Orlando de Lassus' nine *Lamentationes Hieremiae Prophetiae* (as Wearing was an expert in the music of Lassus), sustained noises and tones, as well as lights. The piece intends to reverently reflect the importance of memory on our lives and personal identity.

Setting

The piece should be presented in a very dark, preferably pitch black space.

Selecting journal entries and timings of the waking states / flourishes

The central metaphor of the piece is the act of occasionally waking up into brief states of consciousness from long periods of unconsciousness. The timings of these waking states are derived by the times in Wearing's journal.

In a given performance, at least 5 entries should be selected. It is preferred to select sets of successive entries with timings that occur within a range suitable for the length of the performance, which should last at least 30 minutes but could go on for much longer. There are often long stints between entries. Therefore, multiple series of entries (even from different days / pages) can be selected. As explained in more detail below, any successive groupings of entries will be accompanied by a continuous drone. In the case that an interval between entries is too long to be faithfully observed or a switch is made to another series of entries, the drone fades to silence and then fades back in to indicate the start of the next entry or series of entries. Provided below is an example sequence of times and entries.

0:00	(fade in drone)	
0:03:	(wake)	10:08 am Now I am superlatively awake. First time awake for years. Patience Begins.
0:08	(wake)	10:13 am Now I am overwhelmingly awake. First time for years.
0:16	(wake)	10:21 am A gente liedown.
0:23	(wake)	10:28 am Actually I am now first time awake for years. Patience observedly needed.
0:33	(wake)	10:38 am Actually I am now awake for the first time for years. Patience begins so I may see everything. First thought: I adore Deborah forever.
0:43	(wake)	10:48 am I am now totally, perfectly awake. First time for years. Patience begins fully observed.
0:46	(fade out drone)	
0:49	(fade in drone)	
0:52	(wake)	7:47 am This illness has been like death. Till now. All senses work. First thought: I love darling Deborah forever. Our father.
0:56	(wake)	7:51 am First conscious stroll
1:12	(wake)	8:07 am I am totally, perfectly awake. First time. Breakfast
1:36	(wake)	8:31 am Now I am really completely awake. First time. Patience.
1:40	(wake)	8:35 am Time to see relaxing TV.
1:45	(fade out drone)	

Visual elements

The visual elements of the piece consist of 18 lights that are controlled programatically by a modified version of an algorithm called the rich-get-richer which is programmed in the included SuperCollider application. A more detailed explanation of the algorithm is detailed below; however in short, there are two sets of nine lights each corresponding to one of the nine Lamentations. The SuperCollider application generates live video that can be projected onto a wall though it is possible, or rather preferred, to use the signals to control real lights (e.g. with an Arduino) instead of, or in addition to, the projection. For real lights, incandescent bulbs with tungsten filaments should be used (or covers that produce a similar color). Further, with real lights, the setting can be designed to utilize the entire space by spreading the lights apart.

Sonic elements

Prelude: The piece may be preceded by—and / or accompanied by a program note with—a brief statement about memory that relates to Wearing's condition. These could be quotations by people who have written about memory or Wearing himself (such as Colin Blakemore, Oliver Sacks, or Wearing's wife at the time of the incident Deborah Wearing). A text may also be written especially for a given performance.

Drone: Lorem ipsum...

Readings of the journal entries: The entries should be read in a rather undramatic voice (almost solemnly). Wearing often marked out preceding entries and used quotation marks instead of reiterating the same words in successive entries. For each wake, the entire entry should be read starting with the time and including any words that have been marked out so long as they are legible. They should be read approximately 30 seconds into the waking state once the lights have reached a stable state (which is detailed in the rich-get-richer algorithm description).

***Lamentationes Hieremiae Prophetiae* (quinque vocum) of Orlando de Lassus:** Wearing was an expert in the music of Lassus. *Lamentationes Hieremiae Prophetiae* for five voices consists of nine pieces each used as sources in the opera. A subsets of these sources are played during each wake, filtered through a low pass filter, and attenuated according to a the rich-get-richer.

Toy model of a 60 cycle hums: One of the sonic elements is a toy model of a 60 cycle hum. This is generated by an interpolating oscillator reading (60 times per second) a buffer filled with random sample values between -1 and 1. The buffer is then passed through a low pass filter. Nine of these models are instantiated: one corresponding to each of the Lamentations.

Sustained tones / ensemble: Lorem ipsum...

SuperCollider program

Lorem ipsum...

Rich-get-richer algorithm

Lorem ipsum...

version generated: 2017.11.15

remembering_clive_wearing_main.scd

```
1 (
2 // MAIN LAUNCH
3 /*
4 Loads necessary files and definitions
5 When starting over, best to reboot server and interpreter
6
7 No GUI, just keycodes:
8 <ctrl + f> = enter full screen, <esc> = exit full screen
9 <ctrl + r> = run synth, <ctrl + p> = pause synth
10 <ctrl + s> = trigger start / fade in, <ctrl + e> = trigger end / fade out, <ctrl + t> = trigger wake
11
12 When automate == 1 (true), the trigger keys are disabled and the following lists in the SynthDef are used:
13 startTimes, wakeTimes, and endTimes
14
15 This launches the synth in a paused state to give the user time to put the visuals in full screen.
16 To start the synth, <ctrl + r> must be executed
17 */
18 var automate = 1;
19 var dir = thisProcess.nowExecutingPath.dirname;
20 ^fadeInTrigBus = Bus.control(s); ^wakeTrigBus = Bus.control(s); ^fadeOutTrigBus = Bus.control(s);
21 "remembering_clive_wearing_visuals.scd".loadRelative(true, {
22     "remembering_clive_wearing_synthdef.scd".loadRelative(true, {
23         var fileCount = 0, samples;
24         PathName(dir + "/" + "../audio").files.postln;
25         samples = PathName(dir + "/" + "../audio").files.sort({arg a, b;
26             a.fileName.toLower < b.fileName.toLower }).collect({|file|
27             Buffer.read(s, file.fullPath, action: {
28                 if(fileCount == 8, {^flicker = Synth.newPaused(\flicker,
29                     [\automate, automate, \bufs, samples,
30                     \fadeInTrigBusNum, ^fadeInTrigBus,
31                     \fadeOutTrigBusNum, ^fadeOutTrigBus,
32                     \wakeTrigBusNum, ^wakeTrigBus]), {});
33                 fileCount = fileCount + 1;
34             });
35         });
36     });
37 });
38 )
```

```

1 (
2 // SYNTHDEF
3 SynthDef(\flicker, { |automate = 1, bufs = #[0, 1, 2, 3, 4, 5, 6, 7, 8],
4   fadeInTrigBusNum = 0, fadeOutTrigBusNum = 2, wakeTrigBusNum = 1|
5   //----- Vars -----
6   // fadeInTimes, fadeOutTimes, wakeTimes according to example in score with 1 minute delay
7   // start times / fade ins
8   var fadeInTimes = [1, 50] * 60;
9   // end times / fade outs (must be same length as fadeInTimes with fadeInTimes[i] < fadeOutTimes[i])
10  var fadeOutTimes = [47, 106] * 60;
11  // These are the times to wake up with a flourish of activity and the reading of an entry in the journal
12  var wakeTimes = [4, 9, 17, 24, 34, 44, 53, 57, 73, 97, 101, 106] * 60;
13  // These are the frequency ratios of the ensemble parts
14  var freqRatios = [2, 3/2, 5/4, 7/4, 11/8, 13/8, 17/16, 19/16, 23/16];
15  // Triggers
16  var fadeInTrigs, wakeUpTrigs, fadeOutTrigs;
17  // Rich-get-richer vars
18  var pulse, state, runningPulseCount, norms, wealthGainLag, probs, selects;
19  // Control signal vars
20  var energy, switches, switchesSmoothed;
21  // Sonification vars
22  var lamentations, hums, ensemble, drone, fadeIn, fadeInOutEnv, wakeEnv, fadeOut;
23  // Timed trigger
24  var timedTrigger = {|times| Changed.kr(EnvGen.kr(Env.step({|i| i % 2} ! (times.size + 1),
25    times.differentiate ++ [0.01]), Impulse.kr(0)));};
26  // Monitor time
27  var sTrig, sCount, secs, mins;
28
29  //----- Triggers -----
30  // Triggers for fadeInTimes
31  fadeInTrigs = Select.kr(automate, [InTrig.kr(fadeInTrigBusNum), timedTrigger.value(fadeInTimes)]);
32  // Triggers for fadeOutTimes
33  fadeOutTrigs = Select.kr(automate, [InTrig.kr(fadeOutTrigBusNum), timedTrigger.value(fadeOutTimes)]);
34  // Triggers for wakeTimes
35  wakeUpTrigs = Select.kr(automate, [InTrig.kr(wakeTrigBusNum), timedTrigger.value(wakeTimes)]);
36
37  //----- Rich-get-richer Algorithm -----
38  // Update resolution
39  pulse = Impulse.kr(30);
40  // State of consciousness: asleep or awake; a wake up lasts 60 seconds plus a bit of a tail
41  state = EnvGen.kr(Env.sine(60), wakeUpTrigs);
42  // Binary representation of which element was selected
43  selects = LocalIn.kr(9);
44  // Running sum of the times each of the 9 elements has been selected over 120 pulses
45  runningPulseCount = RunningSum.kr(pulse * selects * (state > 0), (ControlRate.ir / 30) * 120);
46  // Normalize the counts over 121 pulses (adding a 1 in the wake states so probs is always > 1)
47  norms = ((0.925 * (state > 0) + 0.075 + runningPulseCount) / 129);
48  // Goes from 0 to 1 over several pulses after an element is chosen ending with 1 to 4 lights on favoring 2 or 3
49  wealthGainLag = pow((PulseCount.kr(pulse, selects) /
50    TWChoose.kr(wakeUpTrigs, [1, 2, 3, 4], [1, 2, 2, 1], 1)).clip, 4);
51  // Probabilities such that the rich get richer except directly after an element is selected
52  probs = {|i| pow(norms[i] * wealthGainLag[i], state * 4)} ! 9;
53  // Select an element
54  selects = TWindex.kr(pulse, probs, 1);
55  // Feedback binary representation
56  LocalOut.kr({|i| (i <= selects) * (selects <= i)} ! 9);
57
58  //----- Control signals -----
59  // The norms are basically the amount of energy to each element
60  energy = norms;
61  // In a toy model manner, this mimics voltage to the system
62  switches = {|i| TRand.kr(0, 1, pulse) < energy[i]} ! 9;
63  // Smooths the signal such that the lag time is greater as the element gets richer
64  switchesSmoothed = {|i| Lag2.kr(switches[i], 0.25 + (0.75 * energy[i]))} ! 9;
65
66  //----- Sonification -----
67  // Playback of the Lassus Lamentations with a LPF as to not overwhelm to overall sound
68  // Each of the Lamentations as a 2 in 3 chance of sounding
69  lamentations = {|i| LPF.ar(PlayBuf.ar(2, bufs[i], 1, wakeUpTrigs,
70    TIRand.kr(0, BufFrames.ir(bufs[i]), wakeUpTrigs), 1), 2880) *
71    switchesSmoothed[i] * TWChoose.kr(wakeUpTrigs, [0, 1], [1, 2], 1)} ! 9;
72  // Toy model of an electric hum
73  hums = {|i| var buf = Array.fill(256, {1.0.rand2}).as(LocalBuf.clear);
74    LPF.ar(Osc.ar(buf, 60, 0), 960) * switchesSmoothed[i]} ! 9;
75  // Sustained tones based on energy to that element
76  ensemble = {|i| SinOsc.ar(240 * freqRatios[i], 0) * energy[i]} ! 9;
77  // Drone in sleep state that gets attenuated in wake state
78  drone = {|i| var harm = pow(2, 2 - (i / 3).trunc), amp = (1 - (0.75 * energy.sum)) * (1 / pow(harm, 2));
79    SinOsc.ar(60 * harm + TRand.kr(-3, 3, switches[i]), 0, amp)} ! 9;
80
81  //----- Mix -----
82  // Fade ins (10 secs) / outs (30 secs)
83  fadeInOutEnv = EnvGen.kr(Env.asr(10, 1, 30, \sine),
84    Latch.kr(Select.kr(fadeOutTrigs, [1, 0]), fadeInTrigs + fadeOutTrigs));
85  // Fade wake sounds in and out based on total energy in the system
86  wakeEnv = pow(0.8 * energy.sum + 0.2, 4);
87  // Final mix currently set to output stereo where the multiplier is the final output level
88  // Send to separate channel for more control of the individuals sounds (e.g. with a mixer)
89  // The lamentations of a 50 / 50 chance of sounding at all
90  Out.ar([0,1], Mix.new(lamentations) * fadeInOutEnv * wakeEnv * 0.75 * TIRand.kr(0, 1, wakeUpTrigs));
91  Out.ar([0,1], Mix.new(hums) * fadeInOutEnv * wakeEnv * 0.075);
92  Out.ar([0,1], Mix.new(ensemble) * fadeInOutEnv * wakeEnv * 0.125);
93  Out.ar([0,1], Mix.new(drone) * fadeInOutEnv * 0.25);

```

```

94 //----- Visualization Control -----
95 // Send signals to visualizer (these could be sent and scaled appropriated to control real lights)
96 { |i| SendTrig.kr(pulse, i, switchesSmoothed[i] * fadeInOutEnv) } ! 9;
97 { |i| SendTrig.kr(pulse, i + 9, energy[i] * fadeInOutEnv * wakeEnv) } ! 9;
98
99 //----- Monitor Time -----
100 sTrig = PulseDivider.kr(pulse, 30); sCount = PulseCount.kr(sTrig);
101 secs = sCount % 60; mins = (sCount / 60).trunc;
102 Poll.kr(sTrig, mins + (secs / 100), "time (min.secs)");
103
104 }).send(s);
105 )

```

remembering_clive_wearing_main.scd

```

1 (
2 // VISUALS
3 // Init vars and window
4 var projectionWin, scoreWin, osc.func, run = true, blend = {0} ! 27;
5 projectionWin = Window.new("Lamentations I: Remembering Clive Wearing", Rect(500, 500, 750, 500)).front;
6 projectionWin.background = Color.black;
7
8 // Keybinds (these can be change if conflicting with system keybinds)
9 projectionWin.view.keyDownAction = { |doc, char, mod, unicode, keycode, key|
10 case
11 // <ctrl + f> = enter full screen
12 { mod == 262144 && key == 70 } { projectionWin.fullScreen }
13 // <esc> = exit full screen
14 { mod == 0 && key == 16777216 } { projectionWin.endFullScreen }
15 // <ctrl + r> = run synth
16 { mod == 262144 && key == 82 } { ~flicker.run }
17 // <ctrl + p> = pause synth
18 { mod == 262144 && key == 80 } { ~flicker.run(false) }
19 // <ctrl + s> = start / fade in
20 { mod == 262144 && key == 83 } { ~fadeInTrigBus.set(1) }
21 // <ctrl + e> = end / fade out
22 { mod == 262144 && key == 69 } { ~fadeOutTrigBus.set(1) }
23 // <ctrl + t> = trigger wake
24 { mod == 262144 && key == 84 } { wakeTrigBus.set(1) }
25 };
26
27 // Get control signals from SynthDef
28 osc.func = OSCFunc.new({ arg msg, time; blend[msg[2]] = msg[3]}, '/tr', s.addr);
29
30 // Draw the window
31 projectionWin.drawFunc = {
32 var projectionRect = projectionWin.view.bounds;
33 { |i| var outerLen, vPad, outerSquare;
34 outerLen = projectionRect.width / 2;
35 vPad = projectionRect.height - outerLen;
36 outerSquare = projectionRect.insetBy(outerLen * i, vPad / 2).resizeTo(outerLen, outerLen);
37 outerSquare = outerSquare.insetBy(outerLen * 0.05,
38 outerLen * 0.05).resizeTo(outerLen * 0.9, outerLen * 0.9);
39 { |j| var innerLen, innerSquare;
40 innerLen = outerSquare.width / 3;
41 innerSquare = outerSquare.insetBy(innerLen * (j % 3),
42 innerLen * (j / 3).trunc).resizeTo(innerLen, innerLen);
43 Pen.addOval(innerSquare);
44 Pen.fillRadialGradient(innerSquare.center, innerSquare.center,
45 (innerSquare.width / 16), (innerSquare.width / 2),
46 Color.black.blend(Color.new255(255, 214, 170, 255), pow(blend[(i * 9) + j], 0.5)),
47 Color.black);
48 } ! 9 } ! 2
49 };
50 projectionWin.refresh;
51
52 // Refresh function
53 { while { run } { projectionWin.refresh; 30.reciprocal.wait; } }.fork(AppClock);
54 )

```