Plinth Manual

Tobias Brodel - Audiokinetic Experiments Lab, RMIT

May 13, 2015

Contents

1 Introduction

This piece of equipment, known as *The Plinth,* is an audioreactive, MIDI-programmable lighting design unit. Initially conceived by Darrin Verhagen and built by Stuart McFarlane it consists of a strip of 34 RGBLEDs, controlled by an Arduino and driven by Pure Data. It is designed to produce tight, syncretic audiovisual experiences with a mininum of latency. The composer is not expected to be a programmer, all it takes to operate this instrument is a basic knowledge of MIDI. The reference implementation in AkE uses Ableton Live but in theory any MIDI-compliant DAW or instrument should be able to interface with *The Plinth.* Bug reports are welcome.

2 Getting Started

2.1 Setup

The Plinth is intended to be a modular system, able to plug and play with a wide variety of hardware and software. However before we can get our hands on those sweet blinkenlights we must perform a few setup tasks. You will want to download "plinth-setup.zip" from [link].

2.1.1 Saffire Setup

First you'll need to setup the Focusrite Saffire PRO 40 soundcard attached the workstation in AkE. Open the .zip you downloaded and locate a file called "plinth-saffire.pro40". By opening this file you should have loaded the appropriate settings for the soundcard. The main things to check are that DAW1 and DAW2 are routed to Line Outputs 7 and 8 respectively and that DAW9 and DAW10 are going to Line Outputs 9 and 10. This will ensure that the mix gets to the headphones (7/8) and isolated audio sends can reach the plinth (9/10).

2.1.2 Using Live

Inside the "plinth-setup" folder there should be a "plinth-live Project" Ableton Live session. Open it on the workstation in AkE, then open preferences (Cmd+,). Navigate to the Audio tab and select "Saffire (20 in, 20 out)" for both "Audio Input Device" and "Audio Output Device". Remaining in the "Audio" tab, clink the "Output Config" button and enable all sends from 1-10. Now navigate to the MIDI tab and enable "Output: Pro40(MIDI)" by clicking the "Off" button under the "Track" column. It should now be highlighted yellow and read "On".

In the session you should find six MIDI tracks:

1. Pixel
2. Routine

3. Luminosity
4. Delay
5. Pixel Method
6. Luminosity Method

Let's not worry too much about what they do for now. First, make sure the workstation MIDI is connected to *The Plinth.* Now set *Routine* to 1, *Luminosity* to 127 and play some notes around middle-C (60) on the *Pixel* track. If you see lights then you're in business.

There's also an audio track, make sure that is being sent to channels 9/10 on the Saffire.

2.1.3 Using Pro Tools

Another option is to open the "plinth-protools" session. Check under "Setup->Playback Engine" that "Saffire" is selected. Then open "Setup->I/O" go through every tab clicking the "Default" button. The tracks should be set up identically to Ableton Live.

2.1.4 Using Other DAWs

If you're using something other than Ableton Live or Pro Tools then you have to set these tracks up yourself. You'll need one regular MIDI signal on channel 1 to voice notes from and five control-change (CC-data) signals to control the other parameters of the instrument. These CC signals need to communicate on channels 1-5 respectively and use parameter 6 (Data Entry).

In addition you will need a stereo audio send. In the examples above I've isolated that from the main mix. I'm sending it out from channels 9/10 on the AkE workstation, if you're using your own hardware or know what you're doing then any send *should* work.

2.1.5 MIDI-only Setups

Some setups may restrict you to a MIDI-only situation. In this case simply make sure that *Pixel Method* is set to 0 and *Luminosity Method* is set to between 0 and 83 for reasons soon to be elucidated.

2.2 Controls

Now you have the MIDI set up, let's test that it does what it says on the tin.

2.2.1 Pixel

The *Pixel* track should select a pixel to illuminate. It is triggered by sending a note between 60 and 93. Depending on what the *Pixel Method* track is set to it may do nothing, make sure that this track is set to 0.

2.2.2 Routine

This track switches between one of seventeen behaviours programmed into the Arduino. Making sure that CC data is being sent over channel 1 with the parameter set to 6 the LEDs should respond to this signal. A value of 0 should turn off all LEDs, a value of 1 should illuminate one pixel and a value of 10 should turn all lights on to the same colour.

2.2.3 Luminosity

CC data sent on channel 2, parameter 6 should control the luminosity, the brightness of the LEDs. This is dependent on the setting of the *Luminosity Method* channel, make sure that is set to 0.

2.2.4 Delay

Some routines respond to a *Delay* argument, most however do not. At the moment this breaks most things, I would recommend leaving it set to 0. This CC signal runs over channel 3, parameter 6.

2.2.5 Pixel Method

This track decides how pixels are selected, with the *Pixel* MIDI track, or via audio. If set between 0-63 the *Pixel* track will select LEDs like usual, otherwise (64-127) audio amplitude data will determine the pixel. Communicates on channel 4, parameter 6.

2.2.6 Luminosity Method

Much like the above channel, this signal controls the way luminosity is manipulated. 0-42 uses the *Luminosity* track, 43-83 uses velocity data from the *Pixel* track and 84-127 uses audio amplitude data. Communicates on channel 5, parameter 6.

2.3 The Pixel Parameter

Astute readers will have surmised that there are four parameters being controlled here. *Routine, Luminosity* and *Delay* are reasonably obvious, however *Pixel* bears some explanation.

The strip of LEDs on the plinth is an array of 34 pixels, each capable of producing any 24-bit RGB colour. By default each LED is assigned a default base hue which informs the behaviours of any given routine. There are twelve base colours which traverse the visible electromagnetic spectrum in a clockwise fashion, from 0-33 in groups of three:

- Pixels 0-2: Red
- Pixels 3-5: Orange

- Pixels 6-8: Yellow
- Pixels 9-11: Chartreuse
- Pixels 12-14: Green
- Pixels 15-17: Green-Cyan
- Pixels 18-20: Cyan
- Pixels 21-23: Blue-Cyan
- Pixels 24-26: Blue
- Pixels 27-29: Violet
- Pixels 30-32: Magenta
- Pixel 33: Red-Magenta

3 The Routines

This section of the manual details the bahaviour of the seventeen routines programmed into the Arduino.

3.1 Routine 0: Off

This routine turns all LEDs off, regardless of any other parameters.

3.2 Routine 1: Single

This routine illuminates a single pixel. Its luminosity is variable. Unaffected by *Delay.*

3.3 Routine 2: Spectrum

Turns all LEDs on at their base colour. Luminosity variable. Unaffected by *Delay.*

3.4 Routine 3: Spectrum Chase

Traverses entire plinth in clockwise sequence until all LEDs are illuminated, starting with *Pixel*. Luminosity variable. The interval between pixels can be varied using *Delay* (experimental).

3.5 Routine 4: Double Spectrum Chase

Similar to above but travels in both directions around the circle, starting with *Pixel.* Luminosity Variable. Affected by *Delay* (experimental).

3.6 Routine 5: Diad

Illuminates LED at *Pixel* and LED directly opposite in their respective base colours. Luminosity variable. Unaffected by *Delay.*

3.7 Routine 6: Triad

Illuminates equilateral triangle based upon LED at *Pixel* at each each pixel's default colour. Luminosity variable. Unaffected by *Delay.*

3.8 Routine 7: Triad Chase

Illuminates a series of triangles of increasing distance, based upon LED at *Pixel* at each pixel's default colour. Luminosity variable. Affected by *Delay* (experimental).

3.9 Routine 8: Tetrad

Illuminates four LEDs in equal distances, based on LED at *Pixel* at each pixel's default colour. Luminosity variable. Unaffected by *Delay.*

3.10 Routine 9: Skinny Tetrad

Illuminates four LEDs in unequal distances, based on LED at *Pixel* at each pixel's default colour. Luminosity variable. Unaffected by *Delay.*

3.11 Routine 10: Mono

Illuminates all pixels to the colour of *Pixel.* Luminosity variable. Unaffected by *Delay.*

3.12 Routine 11: Hemisphere

Illuminates all pixels. Half beginning at *Pixel* set to that LED's base colour. Other half set to its complementary. See *Diad.* Luminosity variable. Unaffected by *Delay.*

3.13 Routine 12: Trisect

Like *Hemisphere* but split into thirds. See *Triad.* Luminosity variable. Unaffected by *Delay.*

3.14 Routine 13: Quadrant

Like previous two routines but split into quarters. See *Tetrad.* Luminosity variable. Unaffected by *Delay.*

3.15 Routine 14: Outsider

All LEDs illuminated at one eighth of given luminosity. Colour set by default at *Pixel*. Another LED chosen at random is set to complementary colour (see *Diad*) and illuminated at given luminosity. Affected by *Delay* (experiemntal).

3.16 Routine 15: Outsider Triad

Like *Outsider* but with two random pixels. Hues set as in *Triad.* Luminosity variable. Affected by *Delay* (experimental).

3.17 Routine 16: Outsider Tetrad

Like previous two routines but with three random pixels. See *Tetrad.* Luminosity variable. Affected by *Delay* (experimental).

4 Coding

The sources to all of the software running the plinth are available at `https://github.com/TobyJamesJoy/ake-plinth`. There are three main elements to the code, the Arduino sketch, the PD patches and this manual. Currently work is underway to build a virtual model of the plinth in the Unity game engine so students can develop works offsite. Everything is BSD licensed, feel free to check out the code, patches are welcome.

4.1 Arduino

The Arduino code is written in Kernel Normal Form (KNF, see `http://www.openbsd.org/cgi-bin/man.cgi/OpenBSD-current/man9/style.9`). This is an attempt to keep the code conformant to an ANSI-C style. I want to avoid C++isms so as to make everything portable. In the future we may choose to move away from Arduino to for instance plain C running on an Atmel chip or something like a Raspberry Pi. Also KNF is the only sane coding style ever. The code links to "LPD8806.h", found at `https://github.com/adafruit/LPD8806` and "SPI.H" that ships with the Arduino IDE.

4.2 Pure Data

The PD component of this comprises of two patches, "daw_ctl.pd" and "arduino.pd", they do pretty much what it says on the tin. "daw_ctl.pd" accepts the MIDI, CC and audio signals from the workstation's DAW and parses them. The results are then handed to "arduino.pd" in the form of four sends: "rout", "pix", "lum" and "del". "arduino.pd" parses this data again and communicates it via serial bus to Arduino to generate the routines.

If you're hacking on this at home please be mindful that the plinth runs vanilla PD from the Debian repos, this again is for portability's sake. In the

future we'll probably having this running from a low-cost SoC, for which there probably won't be pd-extended binaries. As a result many objects rely on the maxlib, cyclone and comport externals. If you're testing on Debian install everything like so:

```
apt-get install puredata pd-comport pd-cyclone pd-maxlib
```

I find it's good practice to know which objects are from vanilla and which are external so I don't start PD with any -lib flags. As such you'll need to specify the library name when declaring new objects (e.g. "cyclone/drunk" and "maxlib/scale").

4.3 Manual

The source for this manual too is available on github, it's written in LaTeXusing the texlive package from the Debian repos. If any of this could be clearer or if you think I've left something out, please send me a patch!