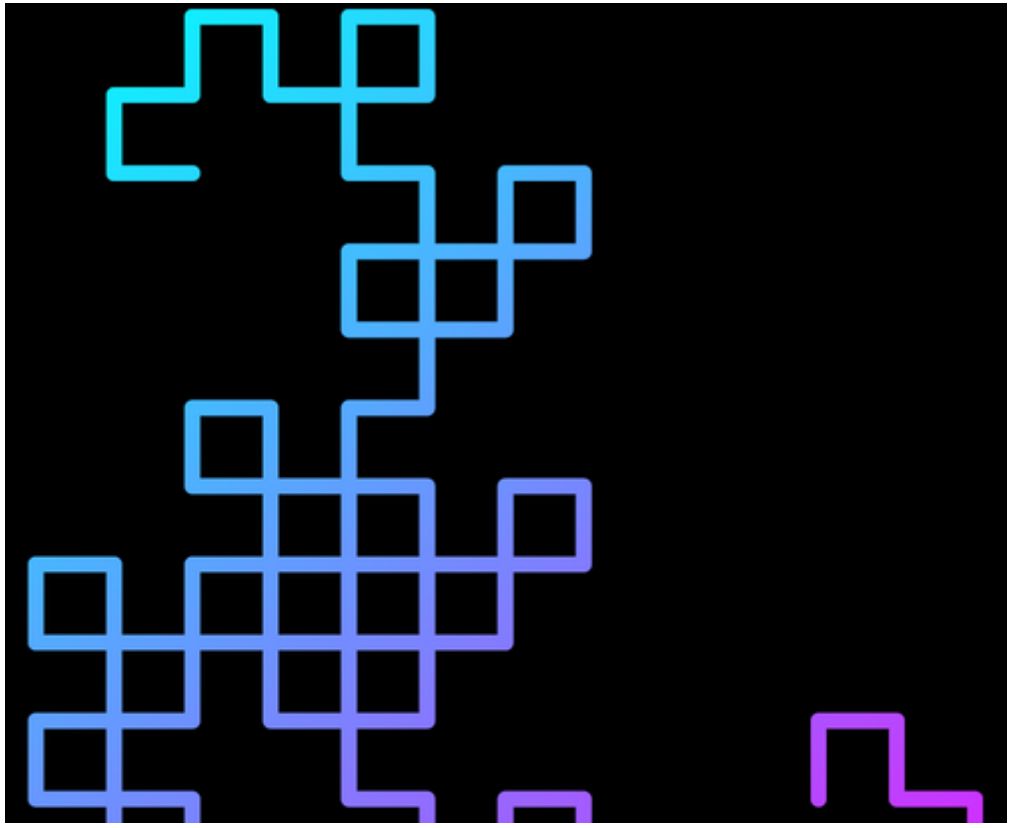
LightPipe

Part 1: Software

Asterix and h8 @ OpenChaos November 2023

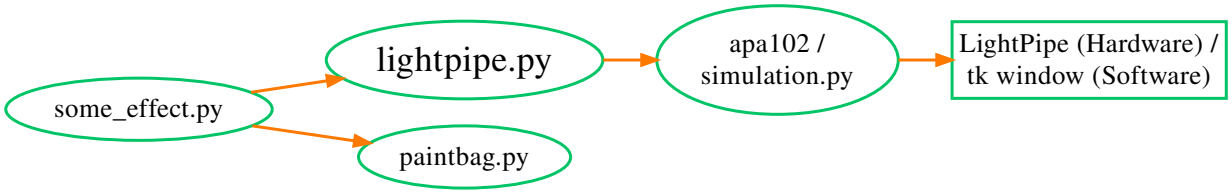


TOC

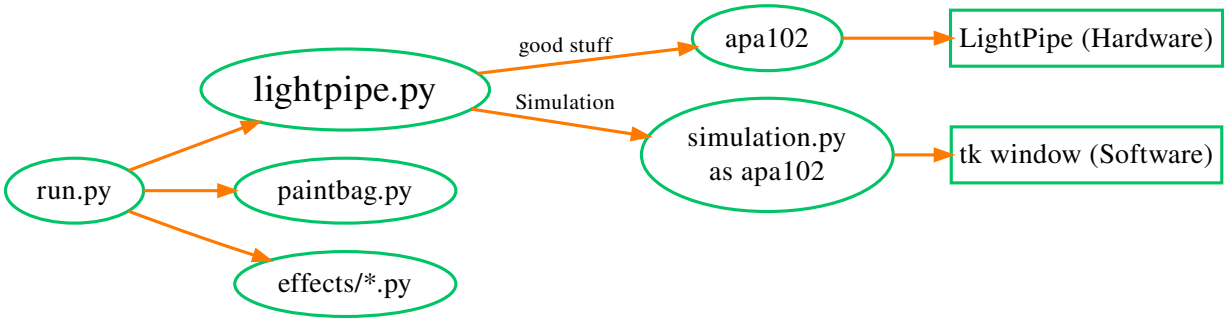
MateLight (2018)

- mostly the same -> much copy-pasted
- refactoring
  - simulation, paintbag
  - different LED-coordinates
- new features
  - more in paintbag
- learning from mistakes
  - effect importing

Simplified overview



Overview


run.py
lightpipe.py
paintbag.py
effects/*.py
good stuff
Simulation
apa102
simulation.py
as apa102
LightPipe (Hardware)
tk window (Software)

lightpipe basics

```
lp = Lightpipe(
     pipes = 1,
    p_size = 16,
    wiring = "",
  brightness = 4,
serial = "simulation",
        )
```

Basic Usage

```
        lp.set_pipe( p=0, color )
lp.set_pixel( p=2, x=3, color, overflow )
    lp.clear( show=False, background )
                lp.show()
```

color

- 3-tuple of int8: `(0, 0, 255)`
- String of hexadecimal notation: `"d511ff"`
- more in paintbag

Simple effect

```
for p in range(lp.pipes):
        for x in range(lp.p_size):
            print(p,x)
            lp.clear(show=False)
            lp.set_pixel(p, x, (255,0,0))
            lp.show()
            time.sleep(0.5)
    lp.set_pipe(0, (0,255,0))
    lp.show()
```

lightpipe advanced

more features in the lightpipe core

Position Overflow handling

- adjacent: draw overflowing pixels on all adjacent pipes. nyi
- border: replace overflowing values with max or min.
- discard: ignore overflowing values.
- flow: like adjacent, but only either up- or downstream. nyi
- modulo: overflow into the same pipe.
- next: overflow into the (linear) next pipe.

Overflow code

```
if mode == "border":
    x = max(0, min(x, self.p_size-1))
if mode == "discard":
    #
    #
    pass
if mode == "modulo":
    x = x % self.p_size
if mode == "next":
    p += x // self.p_size
    x = x % self.p_size
return p,x
```

Demo Time

Paste multiple colors

```
lp.fill(p, x, colors, overflow)
```

```
257 for i, color in enumerate(colors):
258     self.set_pixel(p, x+i, color, overflow)
```

All methods with part pixel

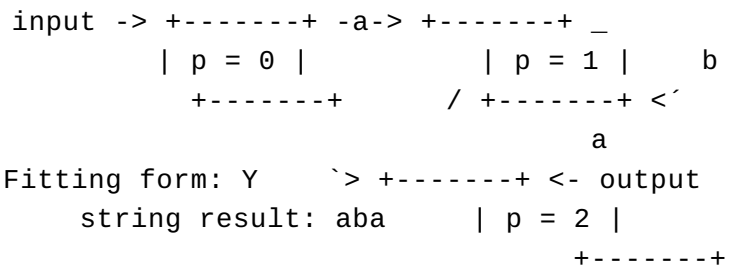
```
lp.set_pixel_0( p, x, color, overflow )
lp.set_pixel_y( p, x, color, y, overflow )

lp.fill_0(p, x, colors, overflow)
lp.fill_y(p, x, colors, y, overflow)
```

wiring String

- driver gets LightPipe wiring information
- can calculate adjacency of pipes
- The `wiring` string consists only of `a` *nother pipe* and `b` *ack to previous pipe*

ascii visulisation



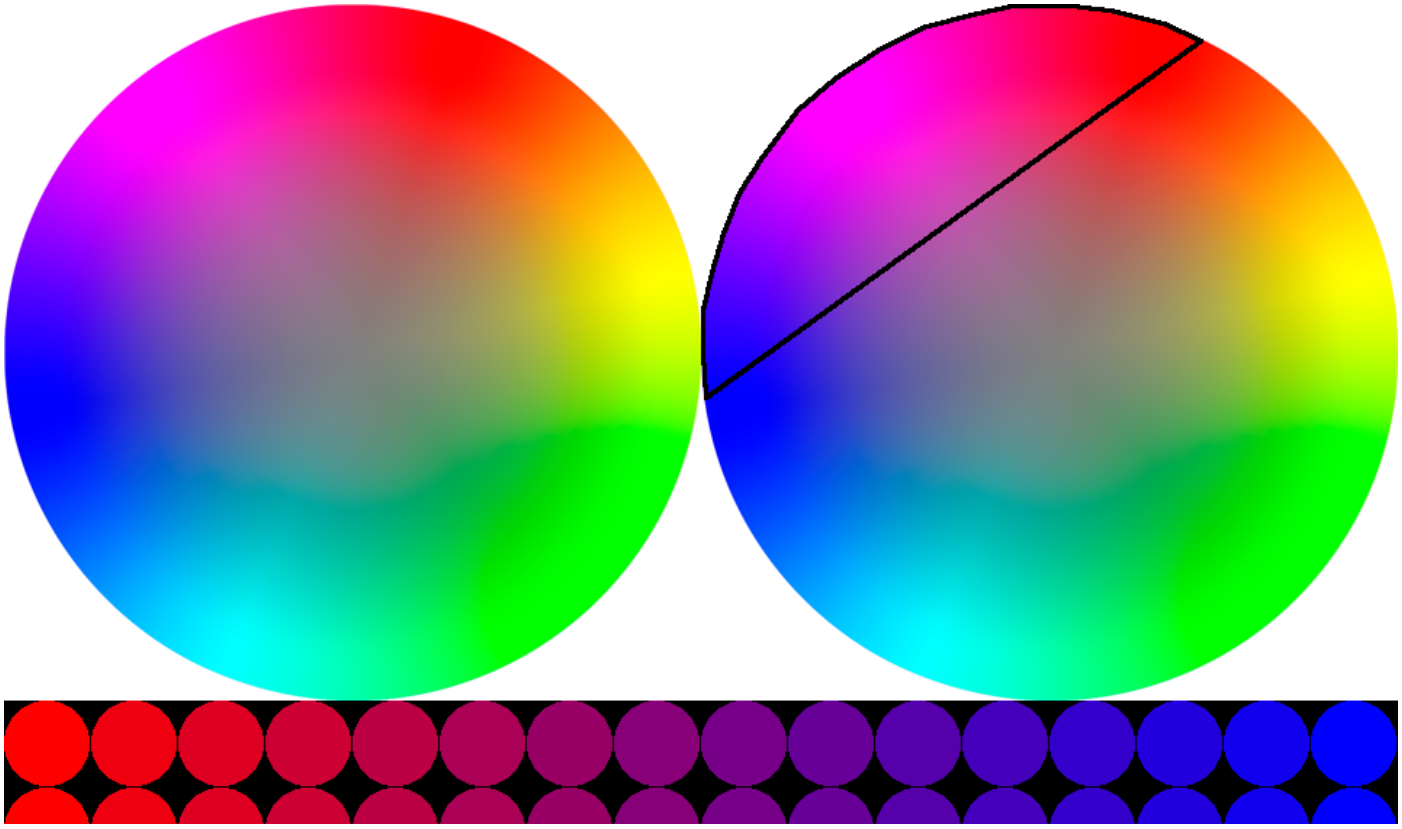
Tree lut

- `lp.get_adjacent_pipes(p) -> set(p_ids)`
- based on wiring string, a k-tree look-up-table can be implemented
- nyi, but math is done

paintbag

contains color helper functions

- colors
- **random_color(saturated=True)**
- **rgb_tuple(color)**
- rgb2hex(color)
- hsv2rgb(h, s, v)
- check_{saturation,hue}(color)
- create_gradient(color1, color2, n)
- saturate_gradient(gradient)
- **cgs(color1, color2, n, clockwise=False)**

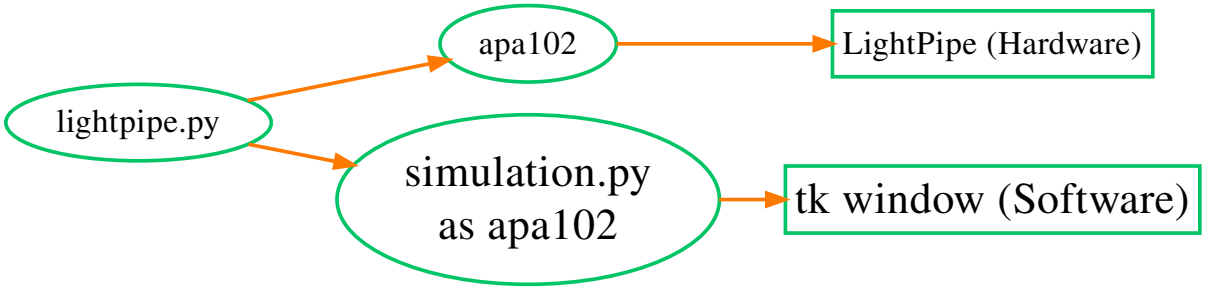


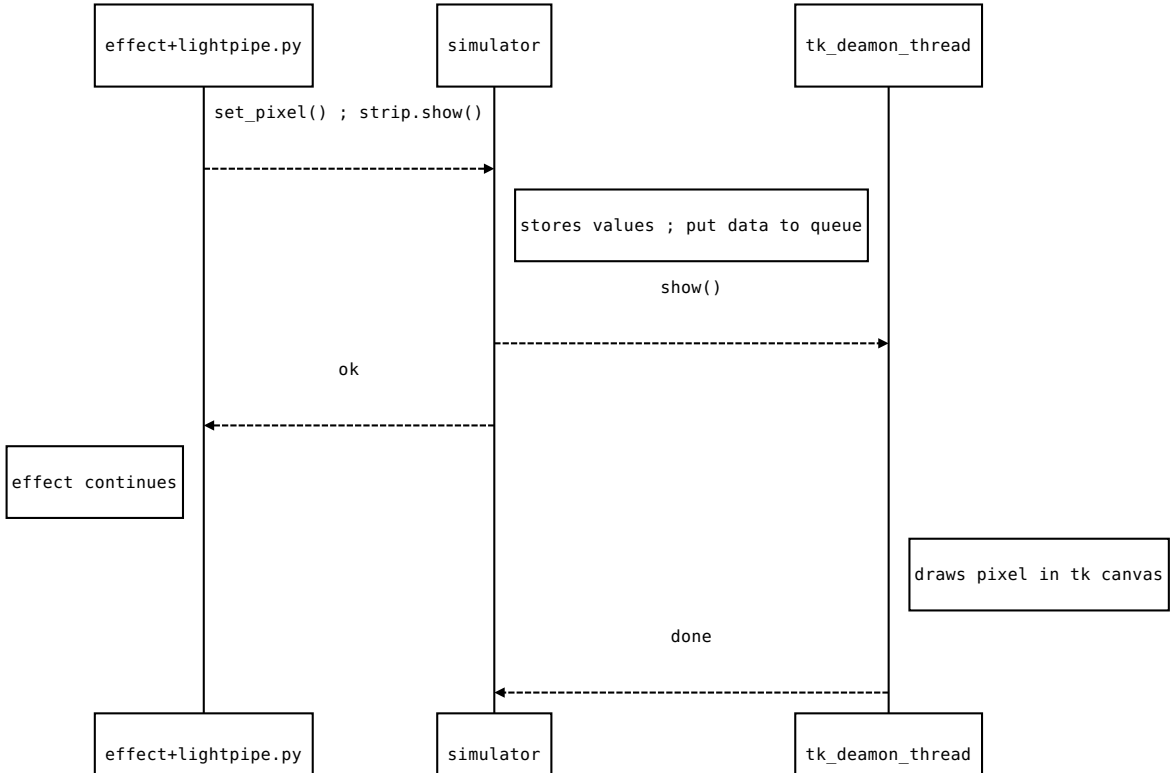
Beautiful effect

```
for p in range(lp.pipes):
        grad = pb.cgs(
                pb.random_color(),
                pb.random_color(),
                lp.p_size,
            )
        lp.fill( p, 0, grad )
    lp.show()
    time.sleep(5)
```

Simulation

- no LightPipe? -> use simulation
- written in python-tk ○ optional dependency of driver
- replaces the apa102 driver


apa102
LightPipe (Hardware)
lightpipe.py
simulation.py
as apa102
tk window (Software)


effect+lightpipe.py
simulator
tk_deamon_thread
set_pixel() ; strip.show()
stores values ; put data to queue
show()
ok
effect continues
draws pixel in tk canvas
done
effect+lightpipe.py
simulator
tk_deamon_thread

If you are interested?

Warning:
more complex code below

simulator thread

```
self.q = queue.Queue(maxsize=2)
self.simulator_ready = False
self.simulation = threading.Thread(
        target = self._simulator_daemon,
        daemon = True,
    )
self.simulation.start()
while not self.simulator_ready:
    #
    time.sleep(0.1)
```

simulator deamon function

```
42  def _simulator_daemon(self):
43      self.simulator = Simulator(
44              pipes = self.pipes,
45              p_size = self.p_size,
46              q = self.q,
47          )
48      self.simulator_ready = True
49      self.simulator.run()
```

simulator show

```
def show(self):
    #
    #
    #
    #
    data = str(self.leds)
    self.q.put(data)
    self.simulator.main.event_generate("<<show>>")
```

tk event

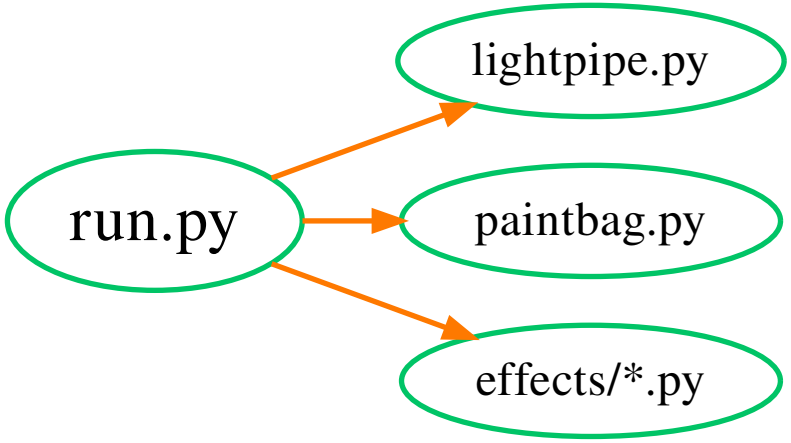
```
111         self.main.bind("<<show>>", self.show)
```

```
116     def show(self, event):
117         data = self.q.get()
118         leds = ast.literal_eval(data)
119         #
120         for pos, color in enumerate(leds):
121             self.set_pixel(pos, color)
122         self.q.task_done()
```

reverse calc coordinates

```
124 def set_pixel(self, pos, color):
125     pipe_pos = pos % (self.p_size * 3)
126     p = pos // (self.p_size * 3)
127 
128     if pipe_pos >= self.p_size and pipe_pos < 2*self.p_size:
129         x = self.p_size - pipe_pos % self.p_size -1
130     else:
131         x =                       pipe_pos % self.p_size
132     y = 3*p + pipe_pos // self.p_size
```

Running effects



Import all

```
3  import glob
4  import importlib
```

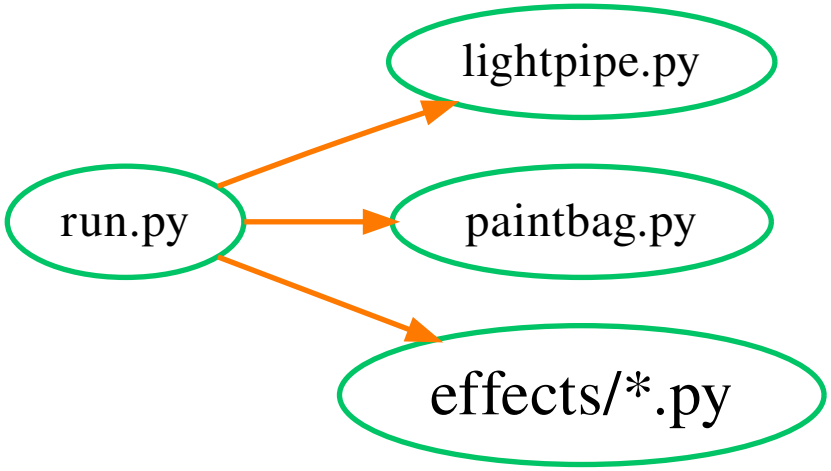
```
11 ls = [_[8:-3] for _ in glob.glob("effects/*.py")]
12 effect_dict = {}
13 for ef in ls:
14     effect_dict[ef] = importlib.import_module(f"effects.{ef}")
```

Execute all

```
from inspect import signature
```

```
f = effect_dict[ef].effect
#
if len(signature(f).parameters) == 1:
    try:
        f(lp)
    except:
        #
elif len(signature(f).parameters) == 2:
    try:
        f(lp, pb)
    except:
        #
```

my_effect.py



my_effect.py

```
 5  def effect(lp):
 6      color = (255,255,0)
 7      while True:
 8          now = time.localtime( time.time() )
 9          d = lp.p_size * (now[3] * 60 + now[4]) // (24*60)
10
11          lp.clear(show=False)
12          lp.fill(0, 0, d*[color])
13          lp.show()
14          time.sleep(100)
```

my_effect.py

```
26 if __name__ == "__main__":
27     import lightpipe
28     lp = lightpipe.Lightpipe()
29     effect(lp)
```

Future Plans

until 37c3

- `config` via file
- `tree_lut` based on `wiring`
- allow- or blocklist in `run.py`

Future Plan: 38c3

- **construct layer**
  - between lightpipe and effect
  - to build specialised effects for specific constructs
- state (maybe)
  - save & reload LED colors
- **color calibration**

Future Plan: 39c3

- 3D-printing like driver & effects
  - driver gets fitting information,
  - iterates over existing (x,y,z) LEDs and calls:
  - `effect(t, (x,y,z) ) -> color`

Future is not now

- Now write effects together?
- Slides: https://md.cccgoe.de/p/pFp4fAYF6 (https://md.cccgoe.de/p/pFp4fAYF6)
- Code: https://git.cccgoe.de/lightpipe (https://git.cccgoe.de/lightpipe)