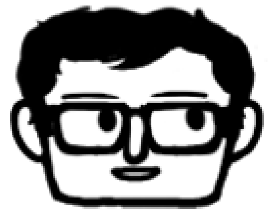
Inspirational tools for the technically minded

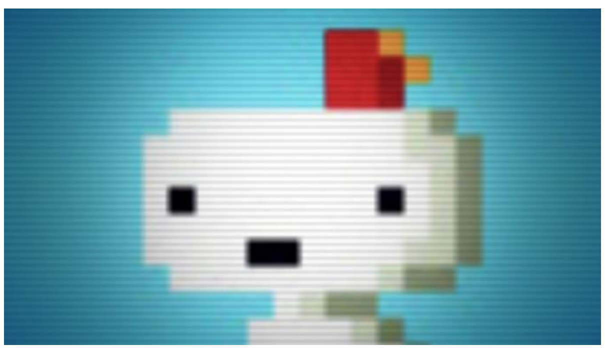
Renaud Bédard

¡Hola!



I’m Renaud.

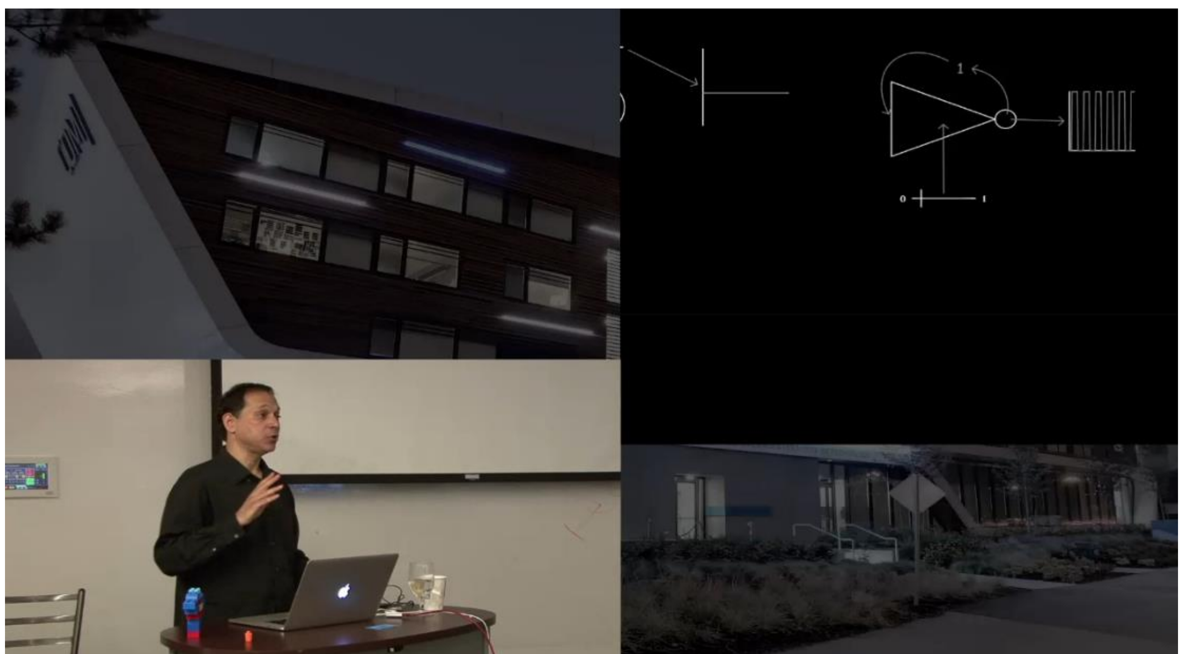
You may remember me from such games as…



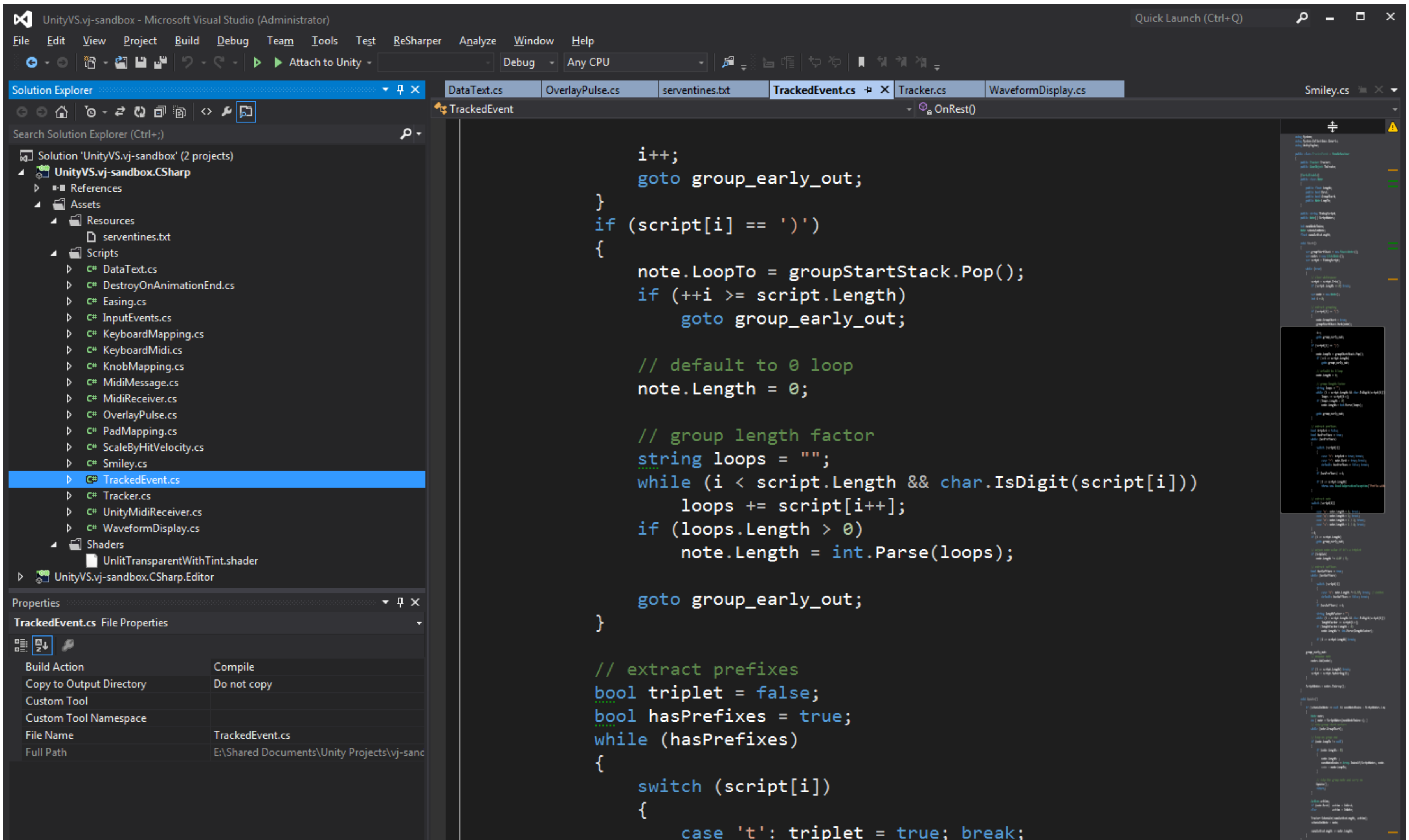
By the way…

- Watch Ken Perlin's talk, especially the stuff on his visual programming system

http://bit.ly/kenperlin_cdm

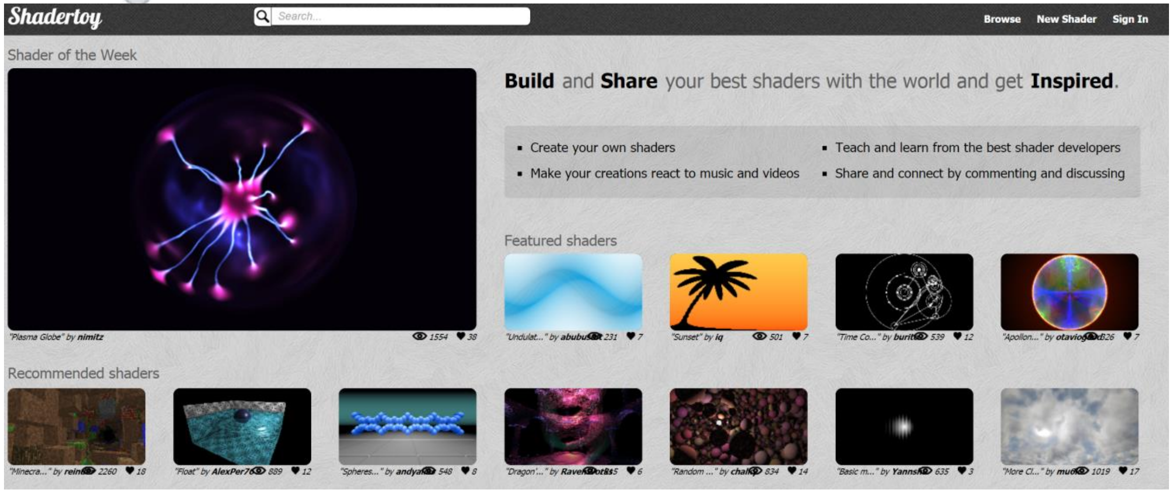


I spend 99% of my time working in something that looks like this



Lately I've been toying with…

ARDUINO AT HEART



And it feels kinda nice!

- Leave the world of algorithms and systems, but not completely
- The same engineering reflexes applied to other domains

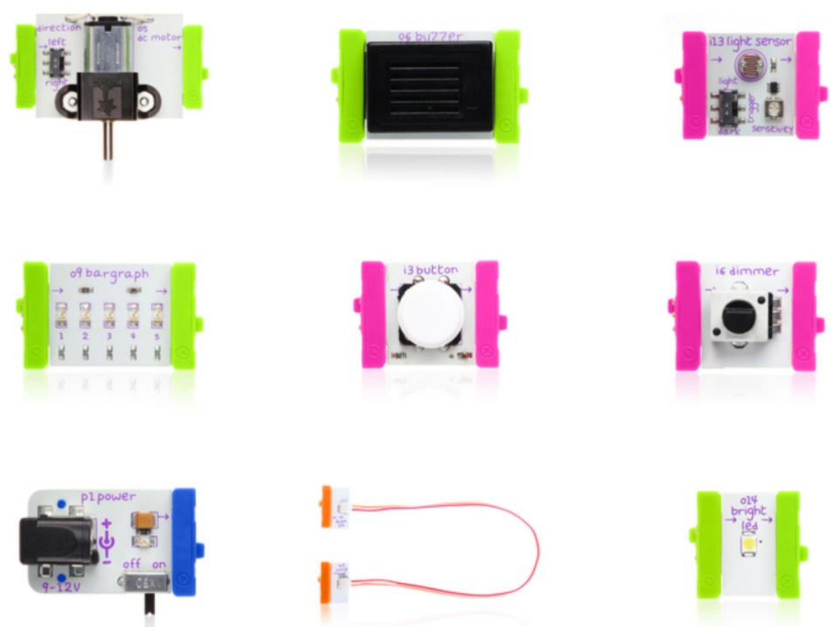
- Get shiny, inspiring results… fast!
- And learn about different technologies along the way

littleBits

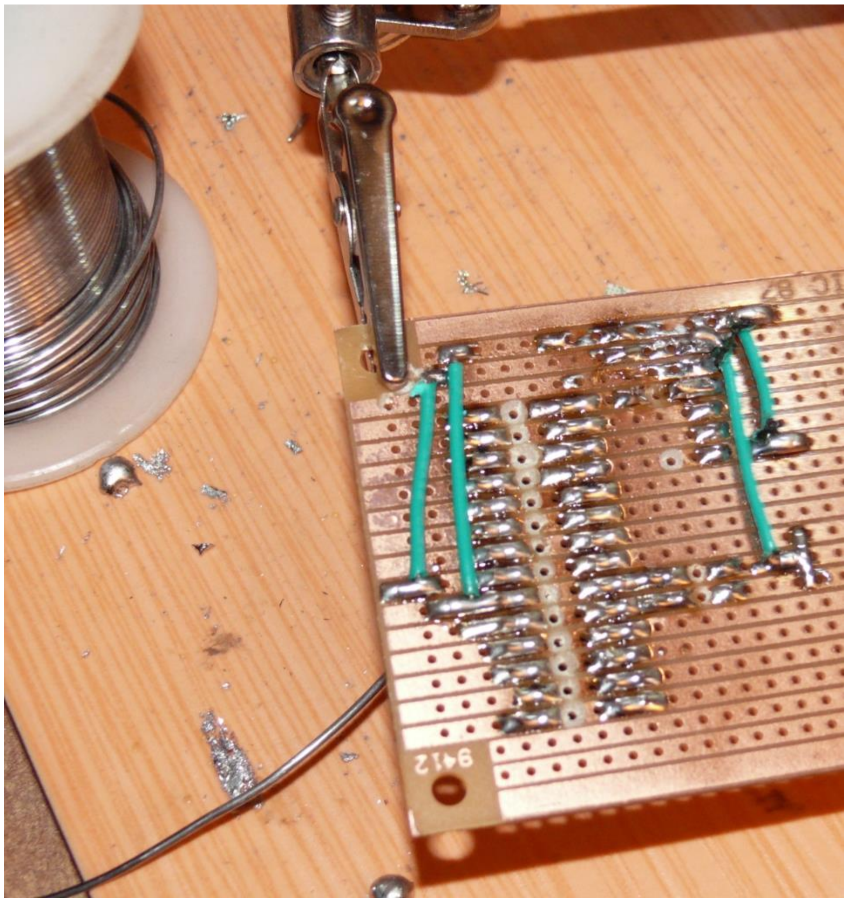
a littleBit of geeky fun!

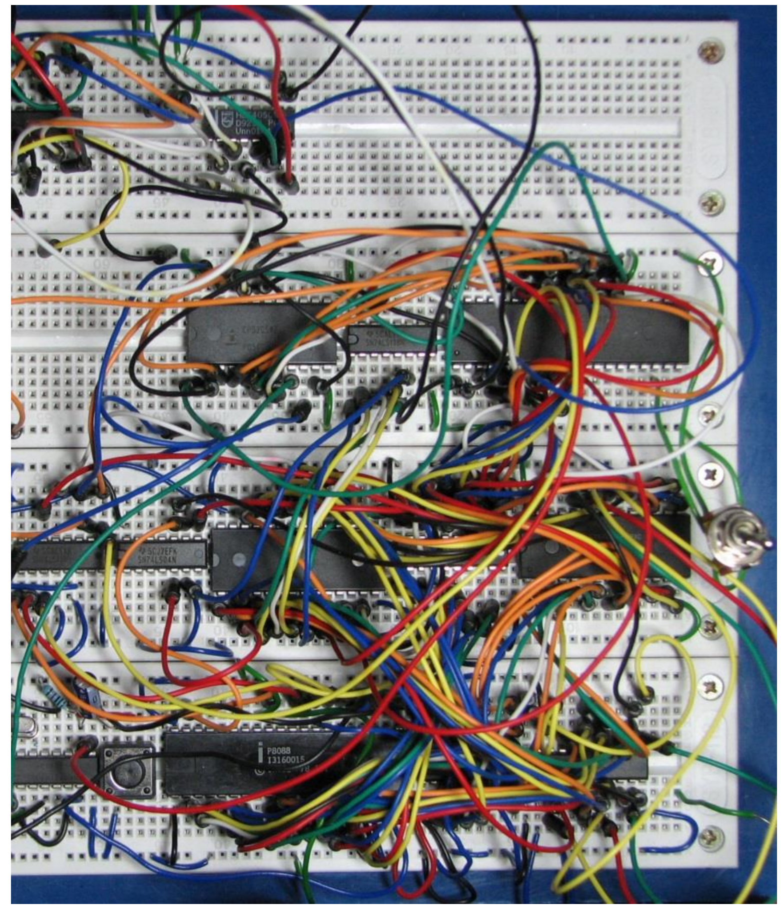
- Magnetized LEGO-like modules
- Quick and easy way to prototype electronic circuits
- Can be a learning tool
- Growing module library
- Sensors, light, sound…

- Even a dev kit to make your own modules!



Electronics without the hassle



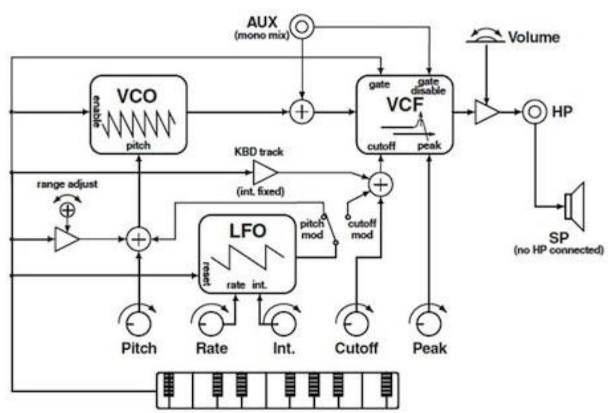


(this scares me too)

"But why should I care?"

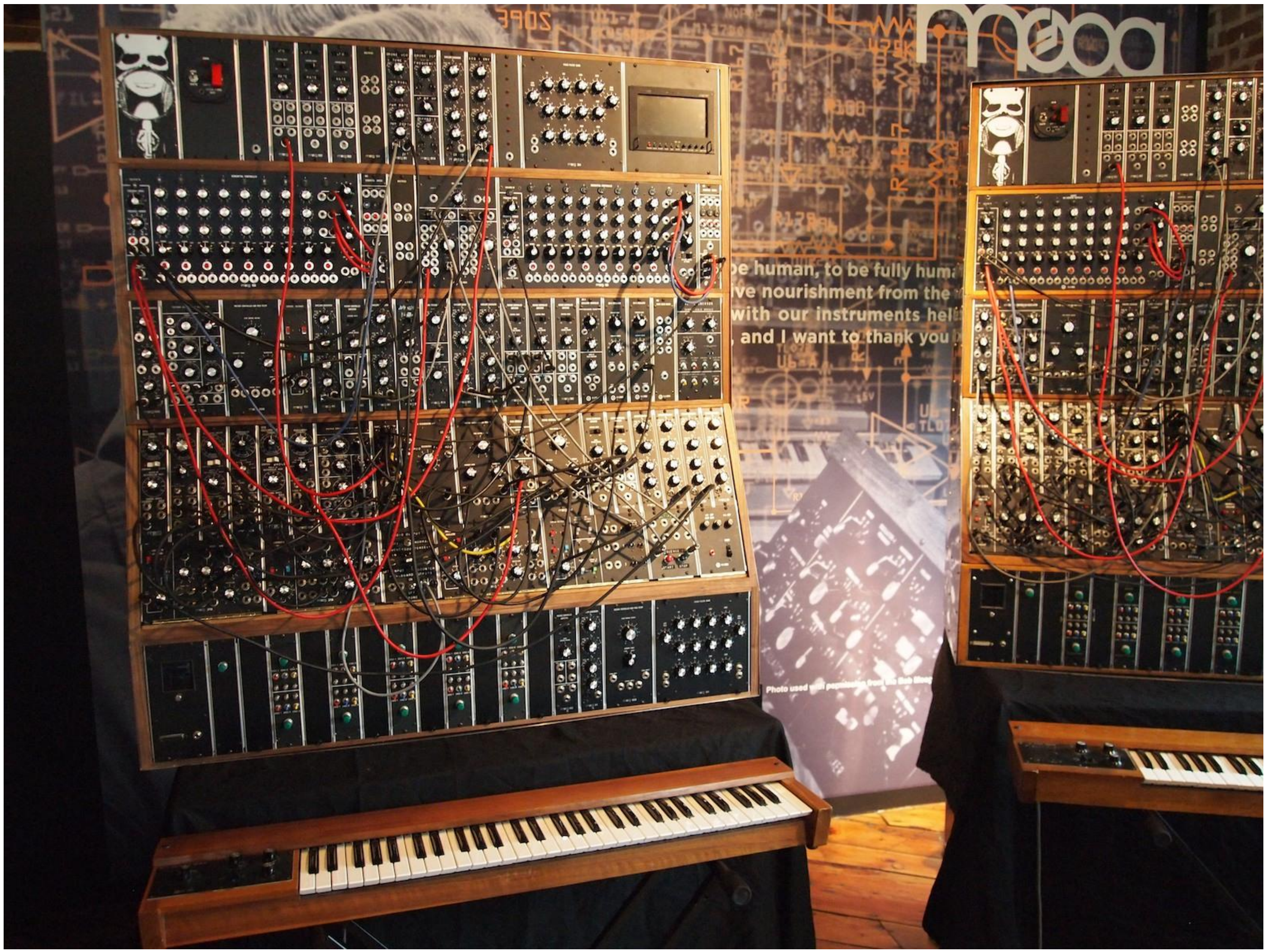
- Similar to node-based programming
- Can make you interested in signal processing
- Can be your first step into electronics and embedded systems programming

- A rare glimpse into the analog world



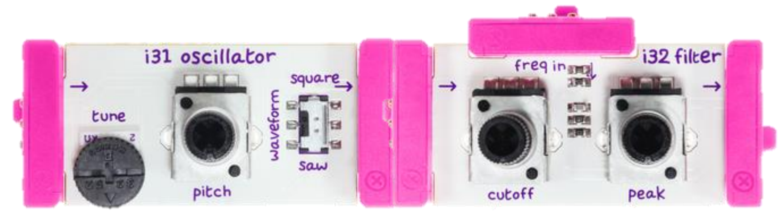
How I got into it

Modular synthesizers



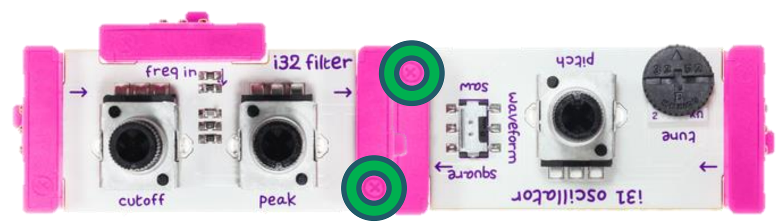
Zero-risk

- This will not blow up



- Neither will this!

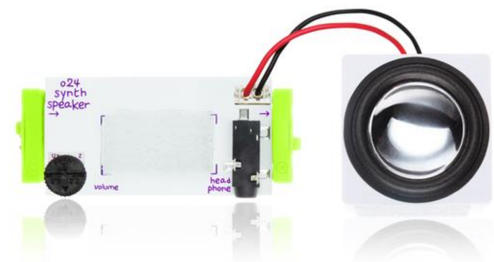
- This probably would
  - But it won't let you do it, because *magnets*



The basics

Power source
(USB or
9V DC source)

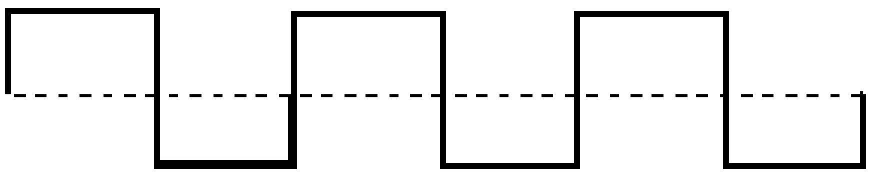
Sound input or generator
(or any combination of other bits)



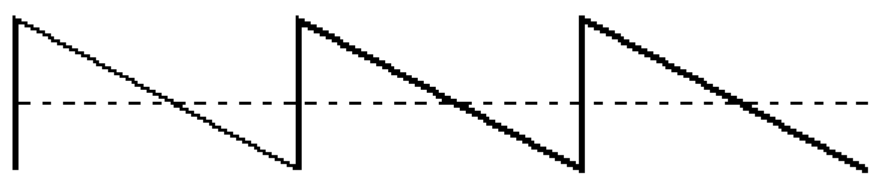
Output (speaker)

Oscillator

- The oscillator bit has 2 modes
  - Square
  - Sawtooth

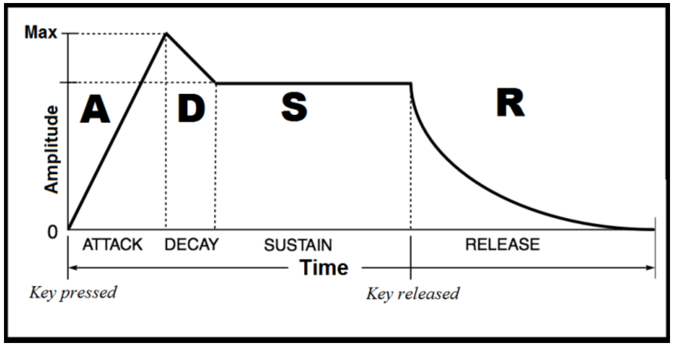


square

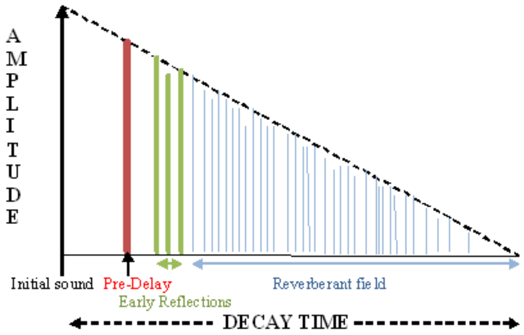


sawtooth

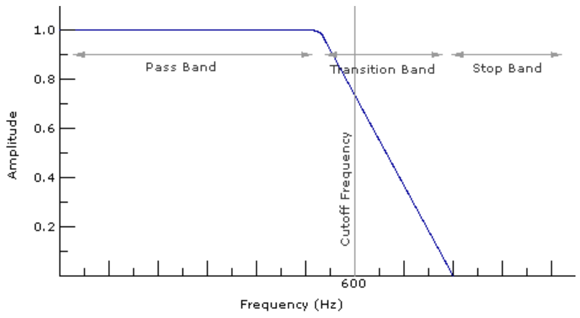
To make things more interesting…



Envelope for pseudo ADSR control



Delay for some nice sound reverberation



Low pass filter with a cutoff frequency of 600Hz

A resonant low-pass filter with an input for the cutoff frequency

Gating, tone control and sequencing

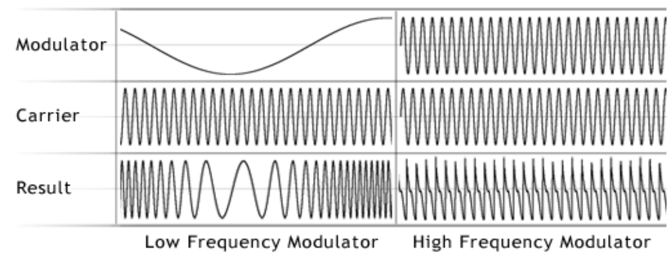
4-step sequencer with per-step tone control
(can be fed a step, or run with an internal clock)

A variety of sensors and buttons
(light, pressure, bend, switch…)

Even a cute little one-octave keyboard!

Go wild!

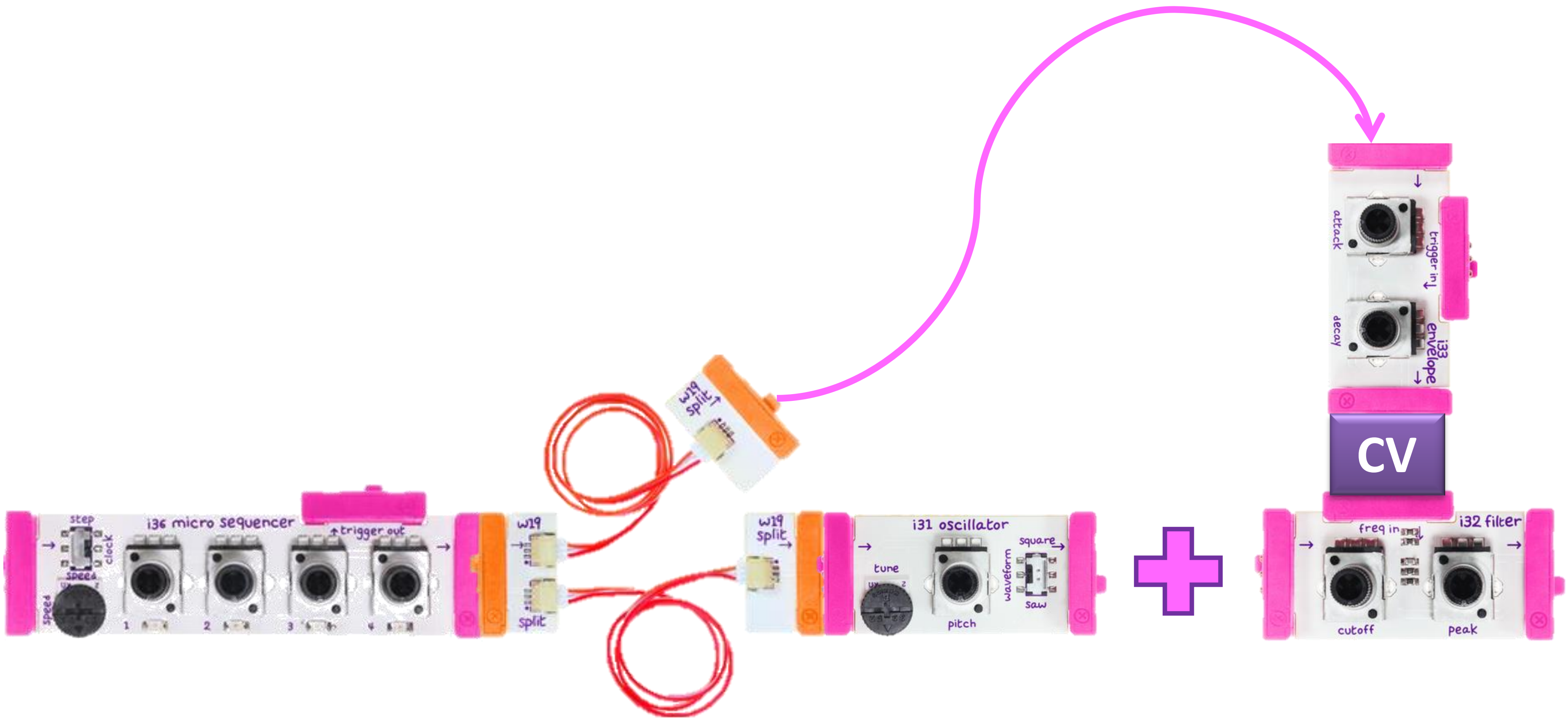
- If you feed an oscillator to another oscillator, the first oscillator oscillates the tone of the second
  - You’ve just made an FM synth!



- Depending how you order your bits, you can control tone, amplitude, filter cutoff...
  - The only rule : respect polarity

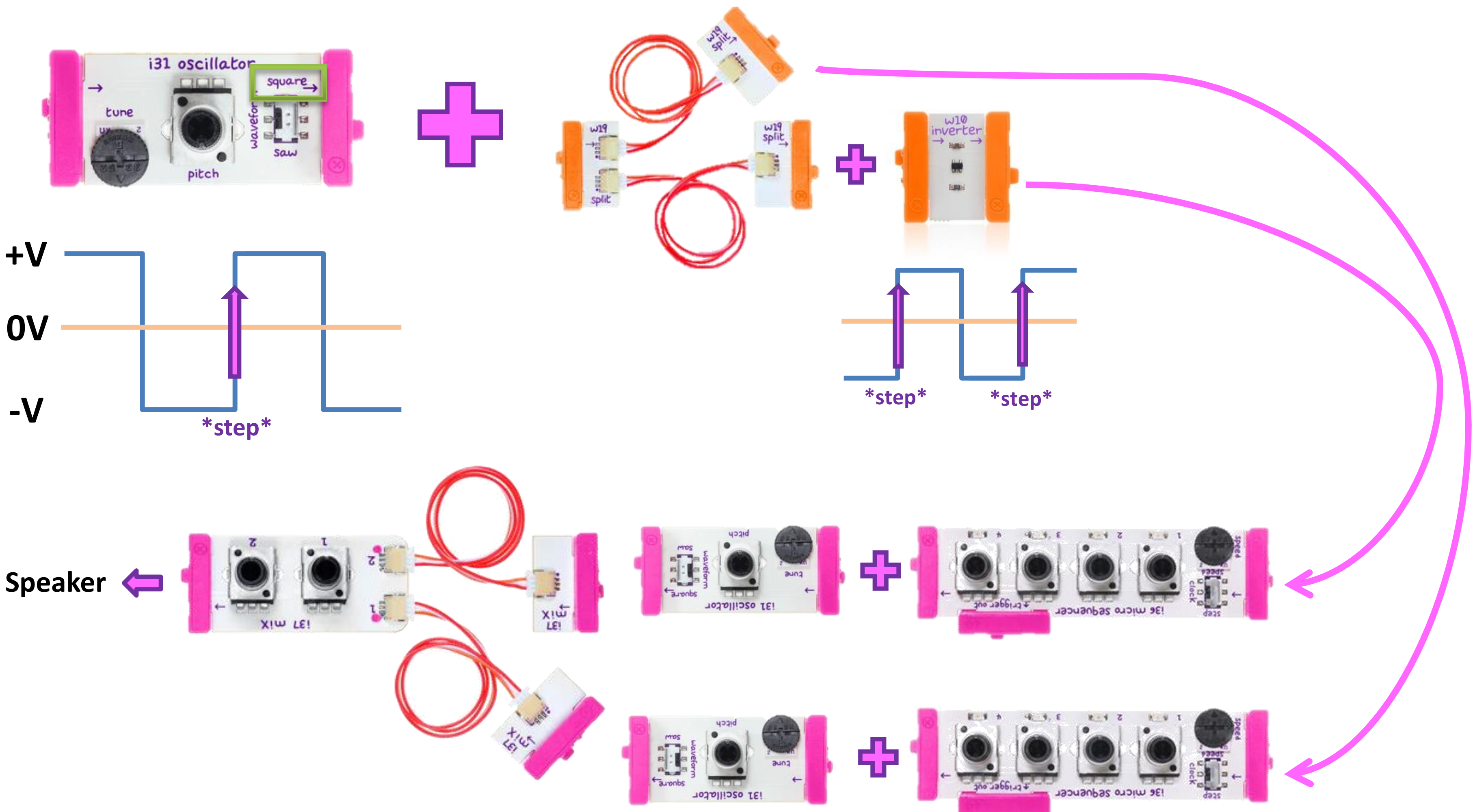
The fun part

(TB-303 style acid filter)



The fun part, part *dos*

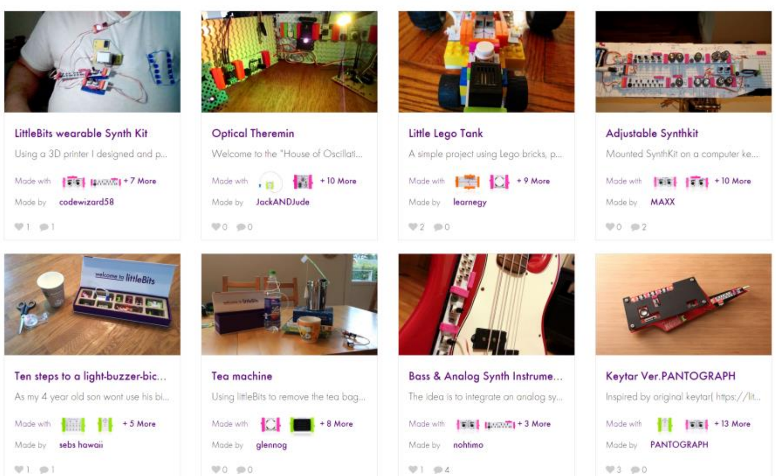
(8-step sequencer)

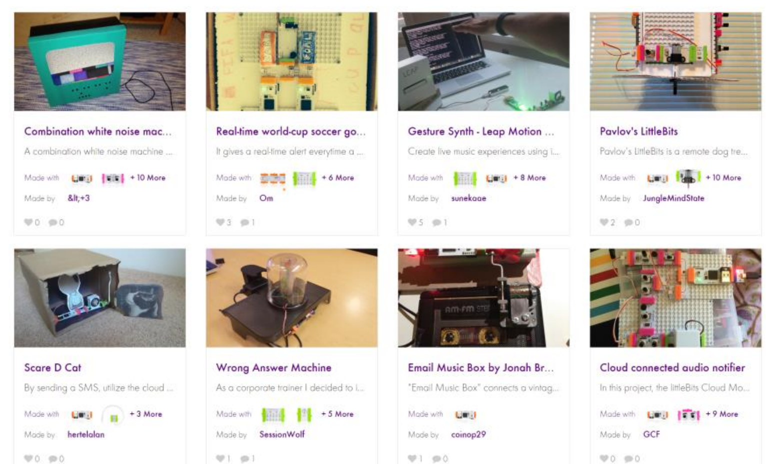


And you could also…

- Use a sawtooth oscillator to do kick drum sounds
- Use a sharp envelope on a noise sample to get a hi-hat
- Drive a filter to self-oscillation and get a sine oscillator
- Use an inverter as a bitcrush/distortion effect

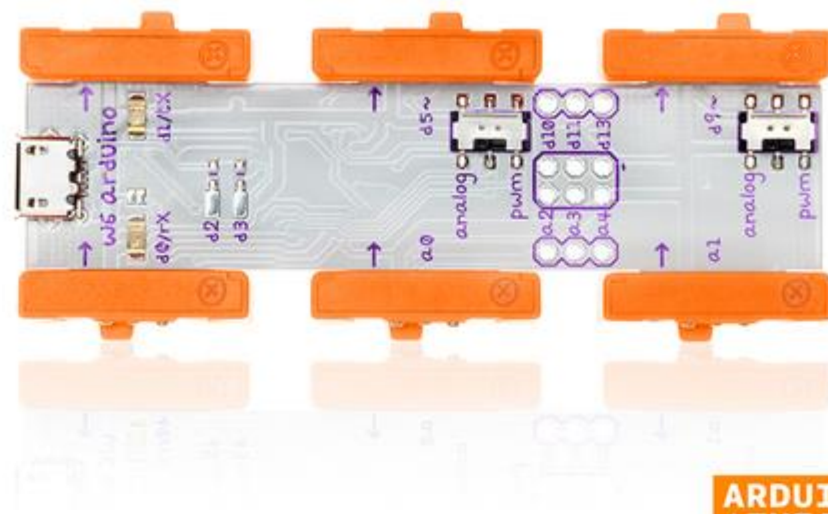
Projects can be shared on the littleBits website





But wait, there’s more!

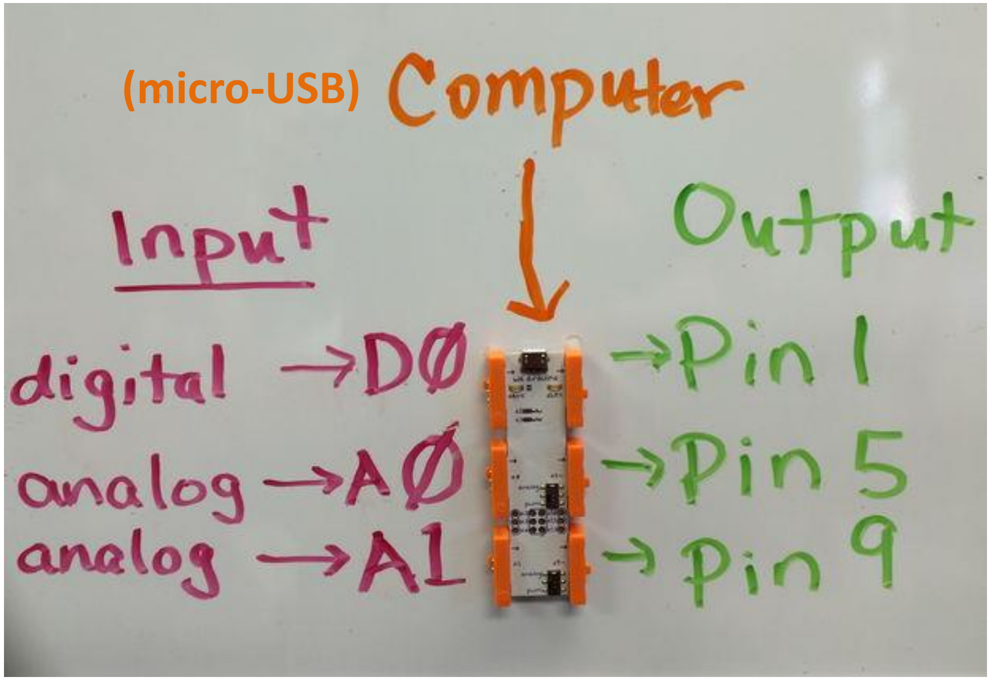
It’s *programmable*!



ARDUINO
AT HEART

ARDUINO + littleBits

- 16Mhz, 2.5KB RAM, 32KB flash memory
- Programmed in AVR C++, through the Arduino IDE
- Interconnects with any and all littleBit modules



Arduino Sketches

- API and code flow inspired by Processing

```
// sketch-global variables go here

void setup() {
    // one-time initialization
}

void loop() {
    // main loop, runs in real time as fast as possible
    unsigned long timeSinceStarted = millis();

    int digitalInput = digitalRead(0); // digital, 0 (false) or not 0 (true)
    int analogInput = analogRead(0); // analog, 10-bit resolution (0 to 1024)

    digitalWrite(1, 1); // digital output to pin 1
    analogWrite(5, 128); // analog output, 8-bit resolution (0 to 255)
}
```

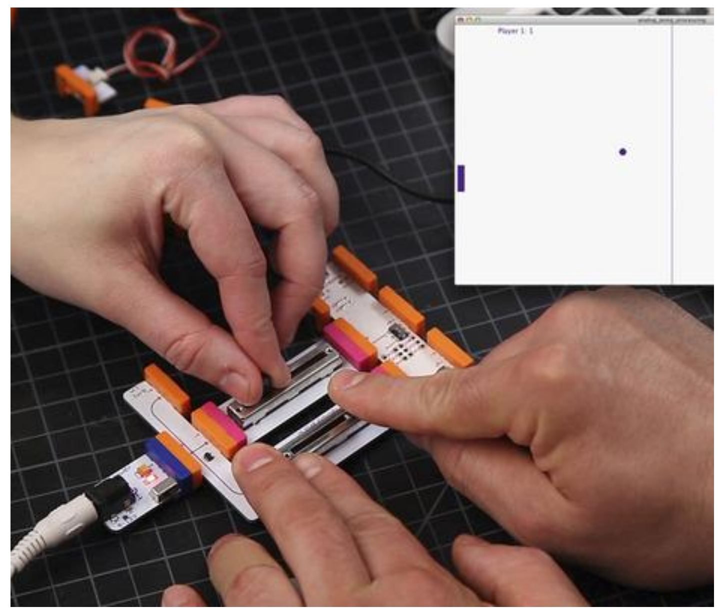
So what do I do with it?

- Use the dimmer and slide dimmers to control input values

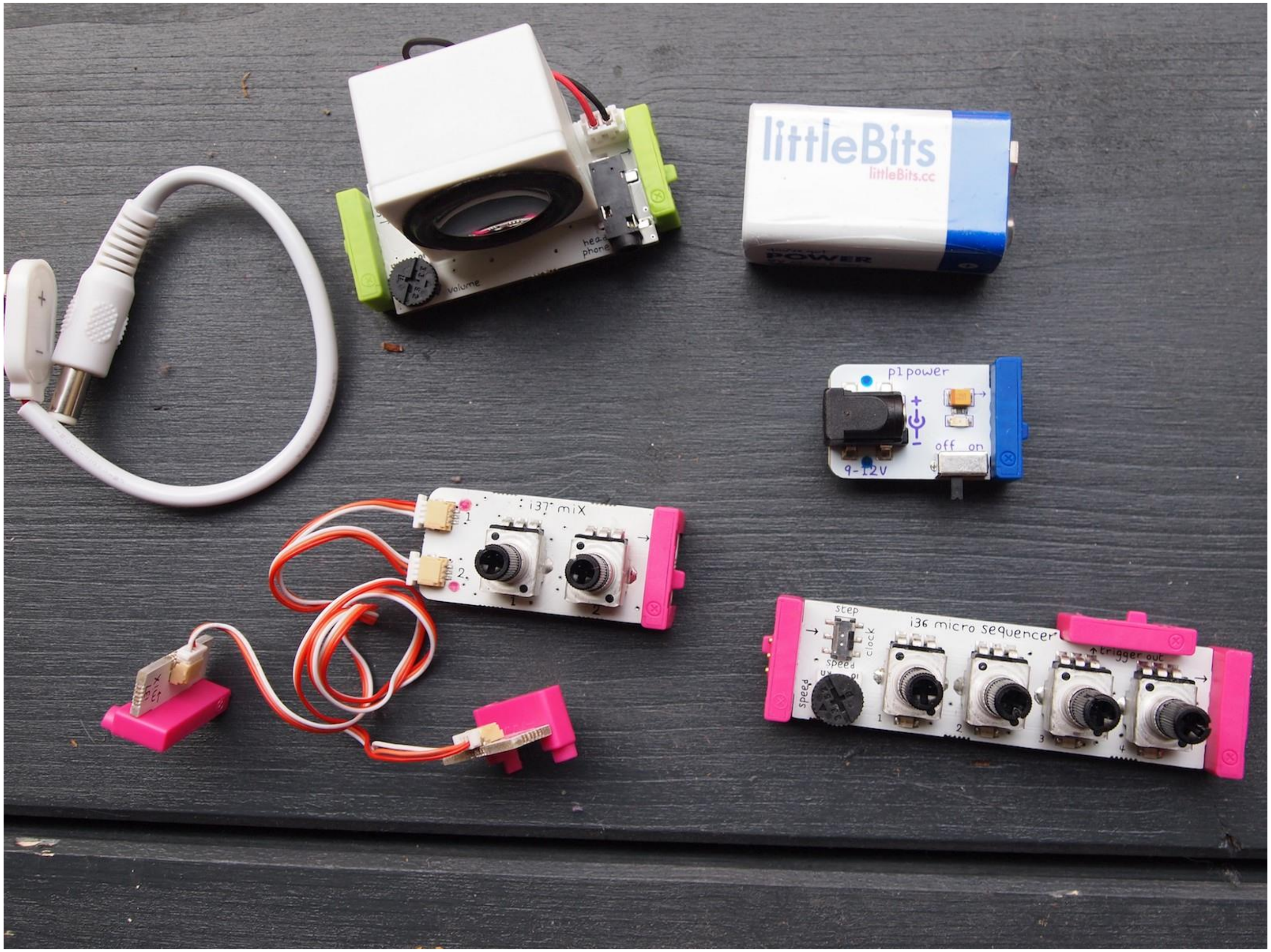
- Use the onboard memory to record/playback stuff
- Anything you can think of, really

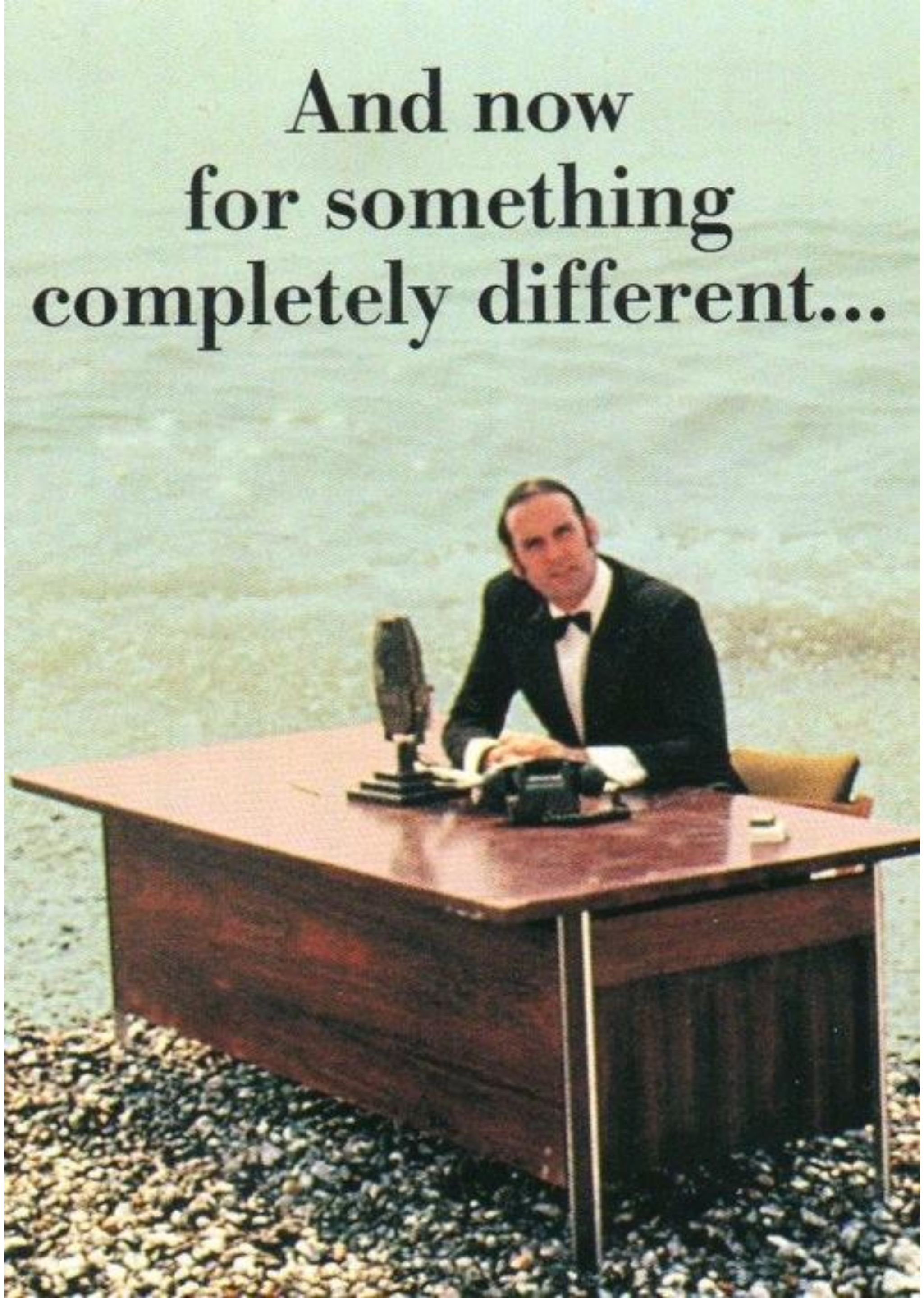
…or connect it to a computer and use it as a controller!

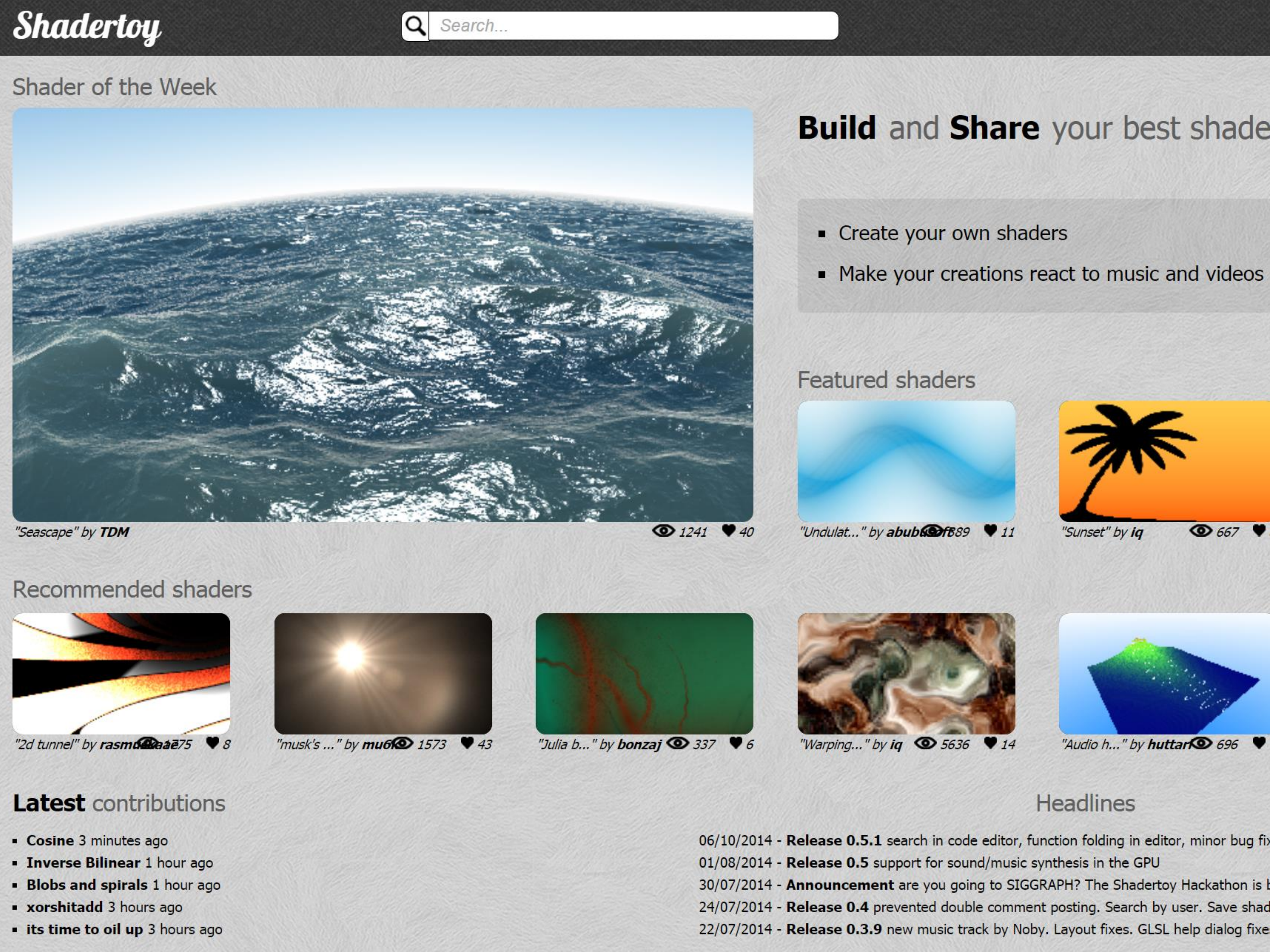
- Using `Serial.write()` and `Serial.read()` (through a COM port)



Demo time!

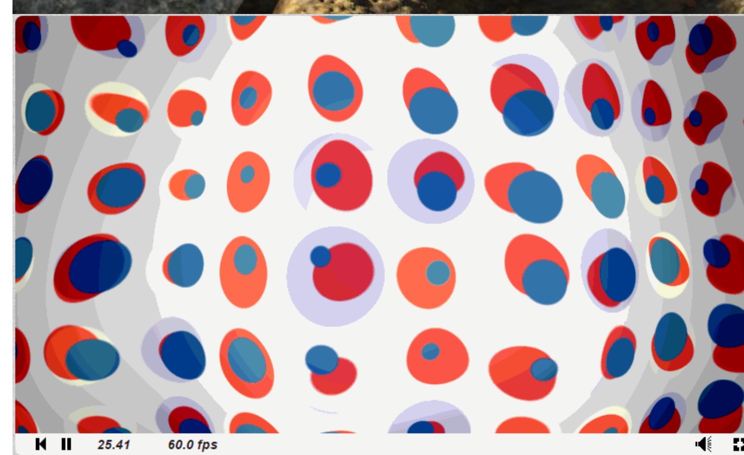

littleBits
littleBits.cc
POWER
p1 power
9-12V
off on
i37 mix
i36 micro sequencer
step
clock
speed
trigger out
volume


And now
for something
completely different...


Shadertoy
Search...
Shader of the Week
"Seascape" by TDM
1241
40
Build and Share your best shade
Create your own shaders
Make your creations react to music and videos
Featured shaders
"Undulat..." by
11
"Sunset" by iq
667
Recommended shaders
"2d tunnel" by
8
"musk's ..." by
1573
43
"Julia b..." by bonzaj
337
6
"Warping..." by iq
5636
14
"Audio h..." by
696
Latest contributions
Cosine 3 minutes ago
Inverse Bilinear 1 hour ago
Blobs and spirals 1 hour ago
xorshitadd 3 hours ago
its time to oil up 3 hours ago
Headlines
06/10/2014 - Release 0.5.1 search in code editor, function folding in editor, minor bug fi
01/08/2014 - Release 0.5 support for sound/music synthesis in the GPU
30/07/2014 - Announcement are you going to SIGGRAPH? The Shadertoy Hackathon is
24/07/2014 - Release 0.4 prevented double comment posting. Search by user. Save shad
22/07/2014 - Release 0.3.9 new music track by Noby. Layout fixes. GLSL help dialog fixe

Shadertoy

- Common ground to write & share pixel shaders on the Web
- Learn from genius-level graphics programmers from around the world

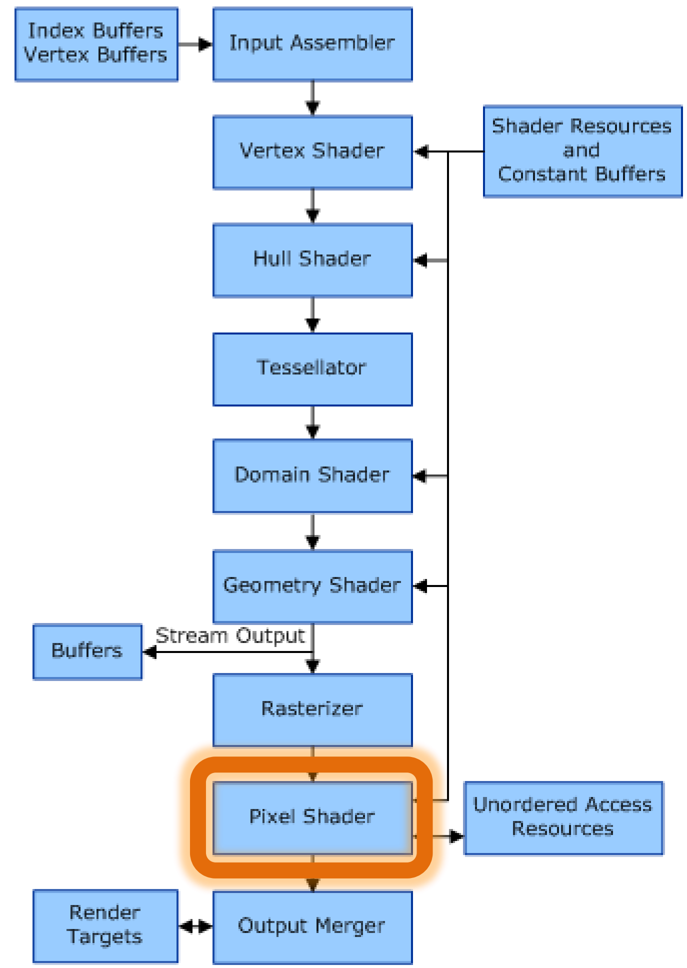


The challenge :

- No geometry
  - Only a fullscreen quad
- Limited set of input textures
- Very limited external state
  - Keyboard, mouse, time, resolution
- No persistence of state between frames
  - Everything is computed all the time

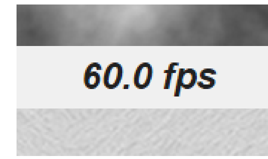
Pixel Shaders, eh?

- A "simple" program that runs for every pixel rasterized by the past stages
  - Input
    - Parameters via host application
    - Interpolated vertex shader output
  - Output
    - Usually color (at least 1 target)
- Evaluated in a vacuum; no knowledge of neighboring pixels
  - Massively parallel as a result



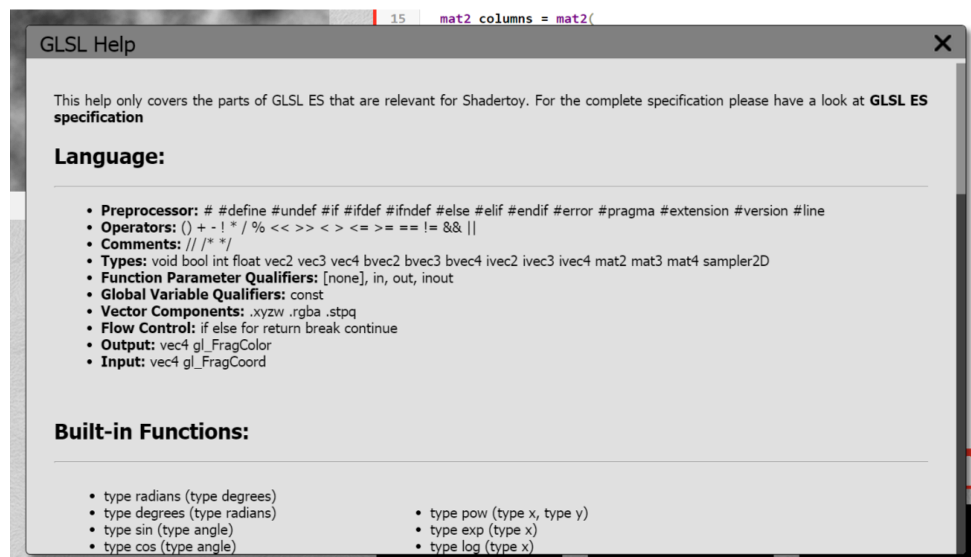
Learning with Shadertoy

- It's real-time!
  ALT+ENTER to compile & commit changes



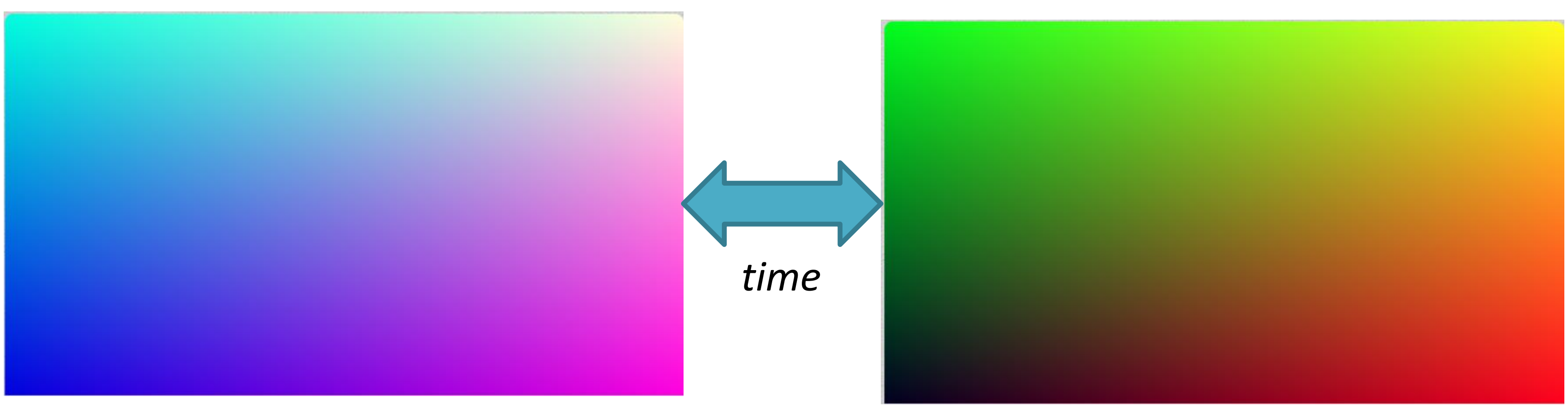
- Syntax highlighting and inline error log

- GLSL Cheat Sheet
  one click away



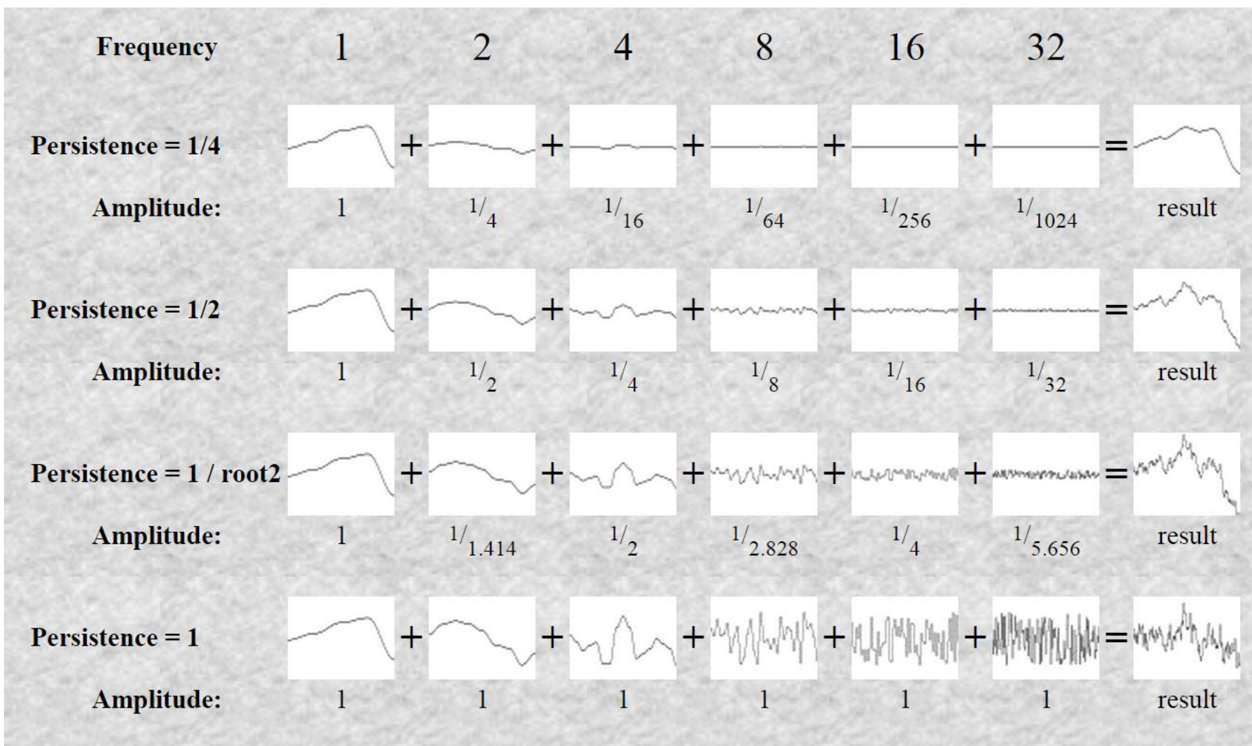
Baby’s first Shadertoy

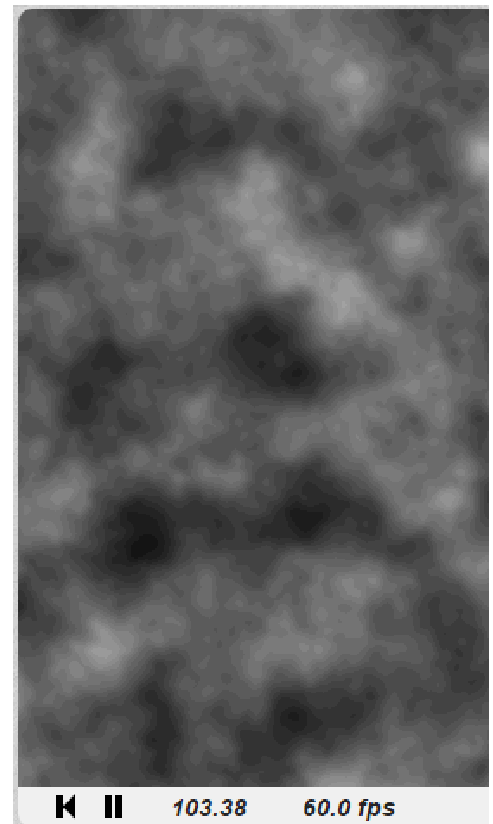
```
void main(void)
{
    vec2 uv = gl_FragCoord.xy / iResolution.xy;
    float t = 0.5 + 0.5 * sin(iGlobalTime);
    gl_FragColor = vec4(uv, t, 1.0);
}
```



Next up : Fractal Noise

- Good old Photoshop Clouds
- A simple yet interesting procedural effect to get started

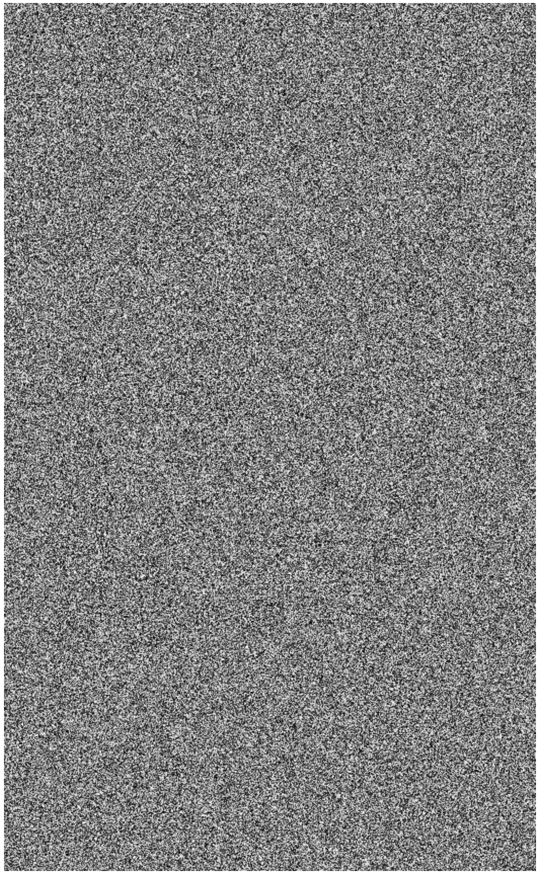




1. 2D Noise Function

- There is no PRNG on a GPU …but we can simulate one!
  Keep the fractional part of a rapidly changing, irregular function

```
float random(vec2 point)
{
  float value =
    sin(dot(point,
        vec2(12.9898, 78.233))) *
    43758.5453;

  return fract(value);
}
```



2. 2D Value Noise

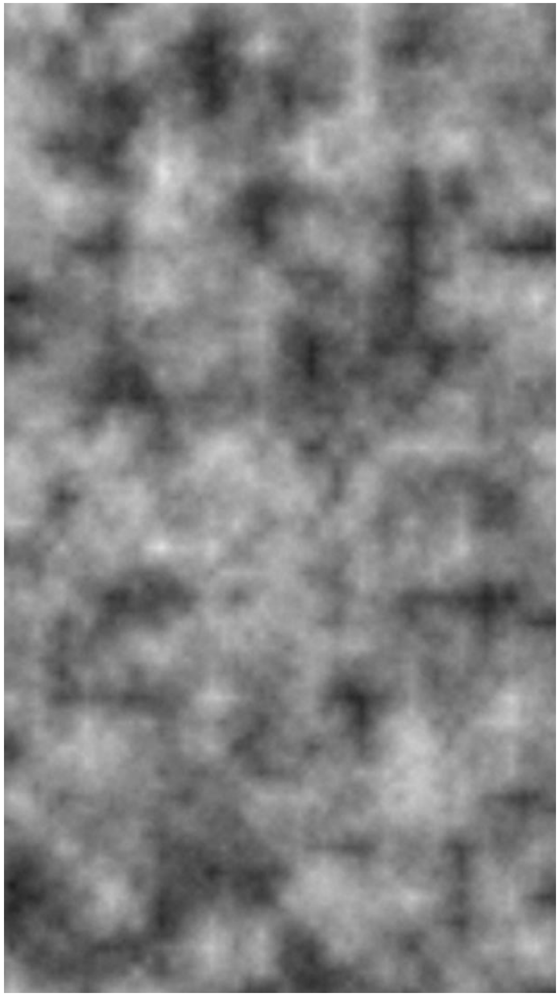
- Returns a smooth, continuous noise function
  by using bilinear interpolation on discrete samples of the 2D noise function
  …basically, zooming into it

```
#define V2_X vec2(1.0, 0.0)
#define V2_Y vec2(0.0, 1.0)

float valueNoise(vec2 point)
{
  vec2 integer = floor(point), remainder = fract(point);

  mat2 columns = mat2(
    random(integer),        random(integer + V2_X),
    random(integer + V2_Y), random(integer + 1.0));

    vec2 row = mix(columns[0], columns[1], remainder.yy);
    return mix(row[0], row[1], remainder.x);
}
```

3. Fractal Iteration

- Accumulate value noise samples,
  each iteration halving weight and doubling detail

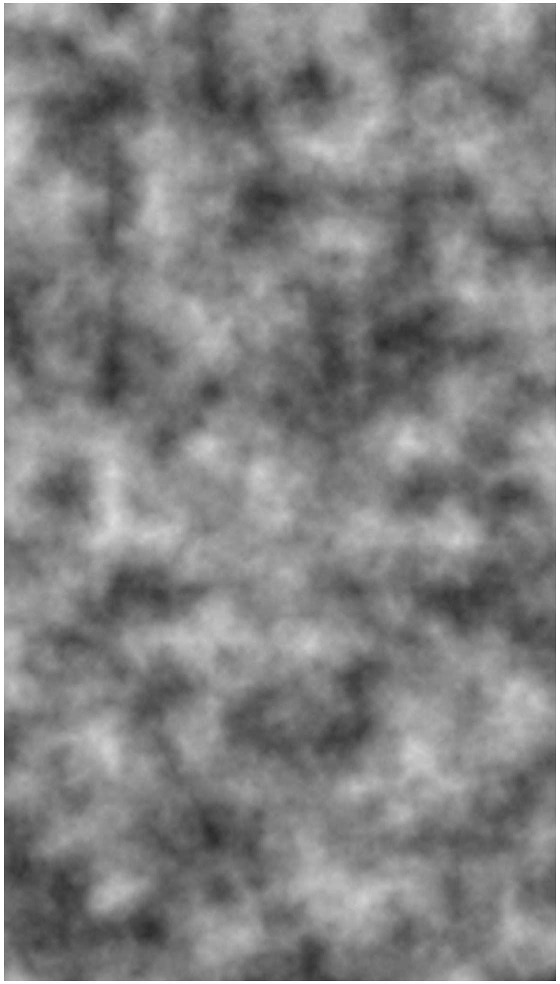
```
#define OCTAVES 16

float fractalNoise(vec2 point)
{
    float result = 0.0;
    float contribution = 0.5;

    for (int i = 0; i < OCTAVES; i++)
    {
        result += valueNoise(point) * contribution;
        contribution /= 2.0;
        point *= 2.0;
    }

    return result;
}
```



3.1. Fluffier Clouds! FLUFFIER!

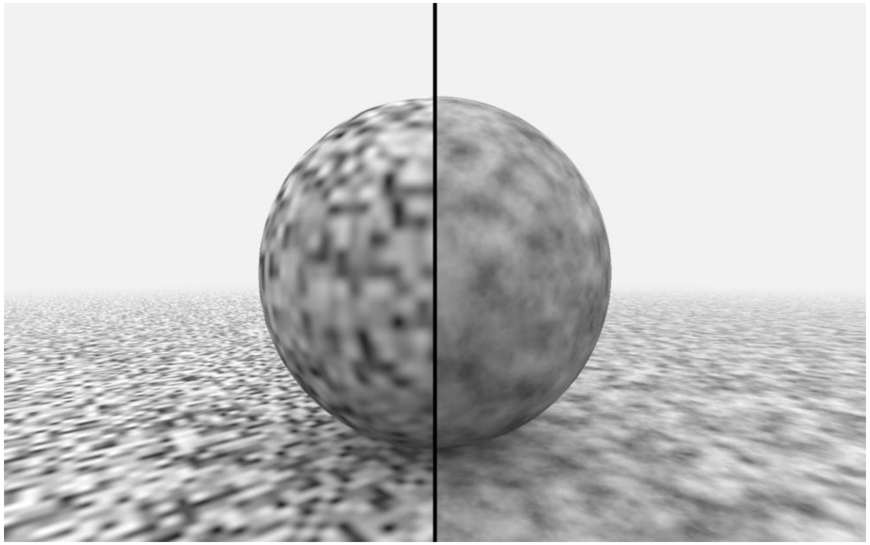
- Rotating at every iteration by an irregular amount hides the repeating diamond pattern

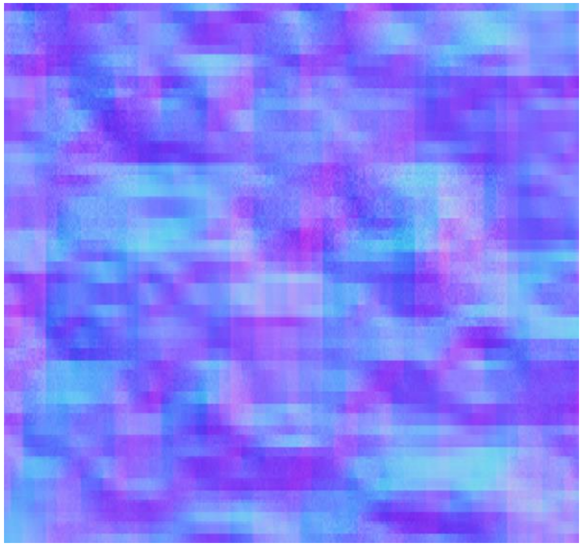
```
#define DEG_TO_RAD 0.0174532925
#define sind(x) sin(DEG_TO_RAD * x)
#define cosd(x) cos(DEG_TO_RAD * x)
#define THETA 30.0

float fractalNoise(vec2 point)
{
    // ...
    mat2 rotationMatrix = mat2( cosd(THETA), sind(THETA),
                               -sind(THETA), cosd(THETA));

    for (int i = 0; i < OCTAVES; i++)
    {
        // ...
        point *= rotationMatrix;
    }
}
```



We could go much further
(but in the interest of time…)

- Change the per-iteration contribution and get coarser or smoother noise
- Use time (or mouse drag) to scroll the noise values on the screen
- Use 3D noise and animate over time
- …calculate derivatives and transform to a normal map!



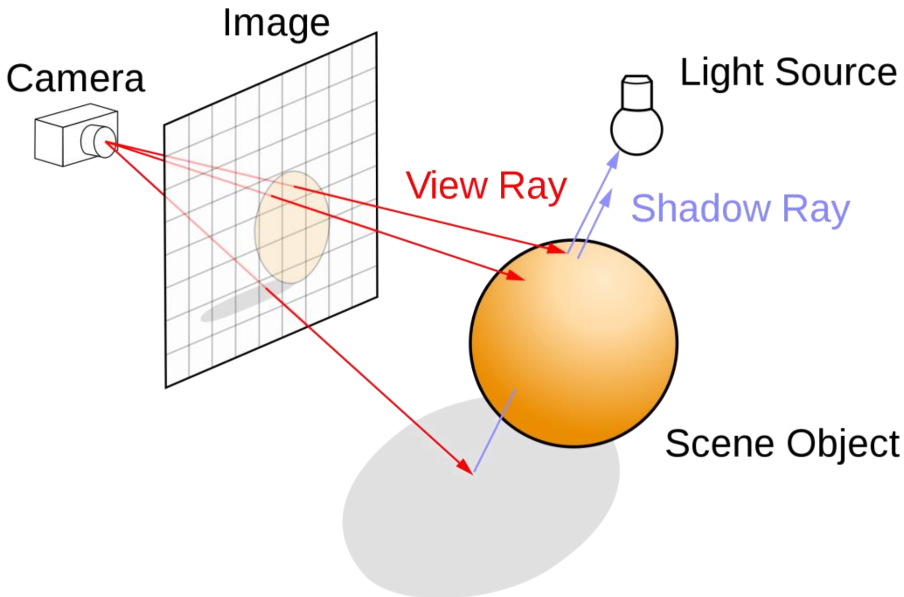


Okay, but... how do I do *THAT*?

iq's "Canyon"

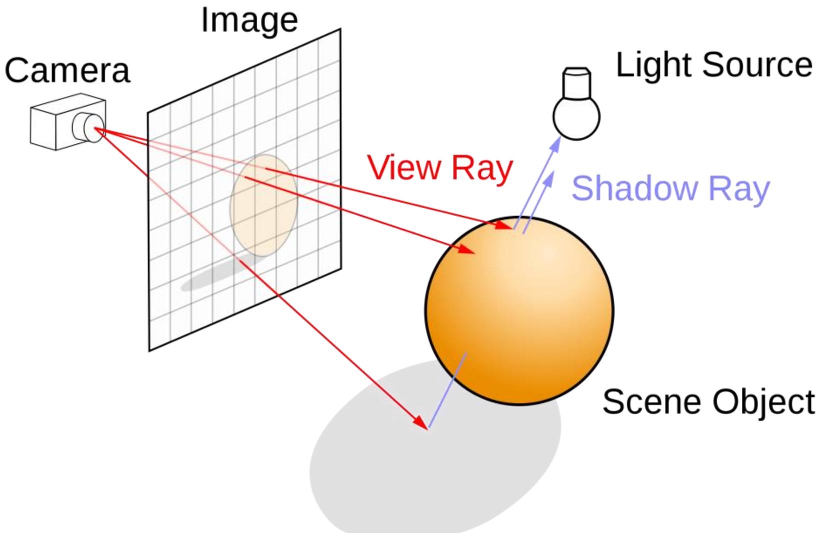
Approach #1 : Raytracing

- Remember : The shader encodes a function describing the world/scene for a defined pixel
- The pixel can represent a ray that's cast from the camera in 3D space



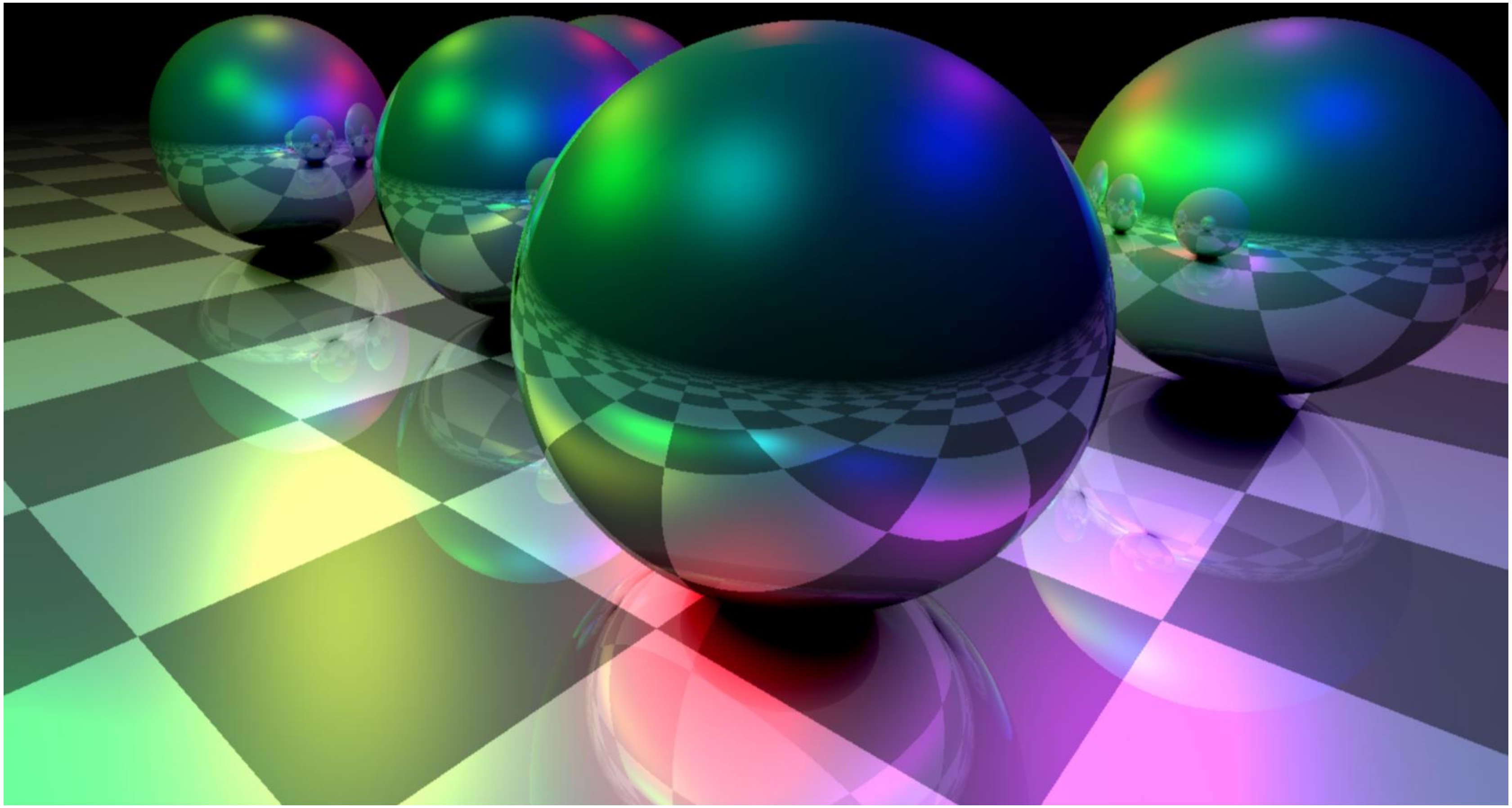
Raytracing, continued

- Test every object in the scene for intersection
  - Keep closest hit
- Return surface properties (color, normal, material)
- Bounce rays for reflection
- Regular lighting equations apply (accumulate lights)



- A bit too code-heavy for a PowerPoint slide
- Very clear implementation by McZonk :

http://bit.ly/Raytrace_ShaderToy



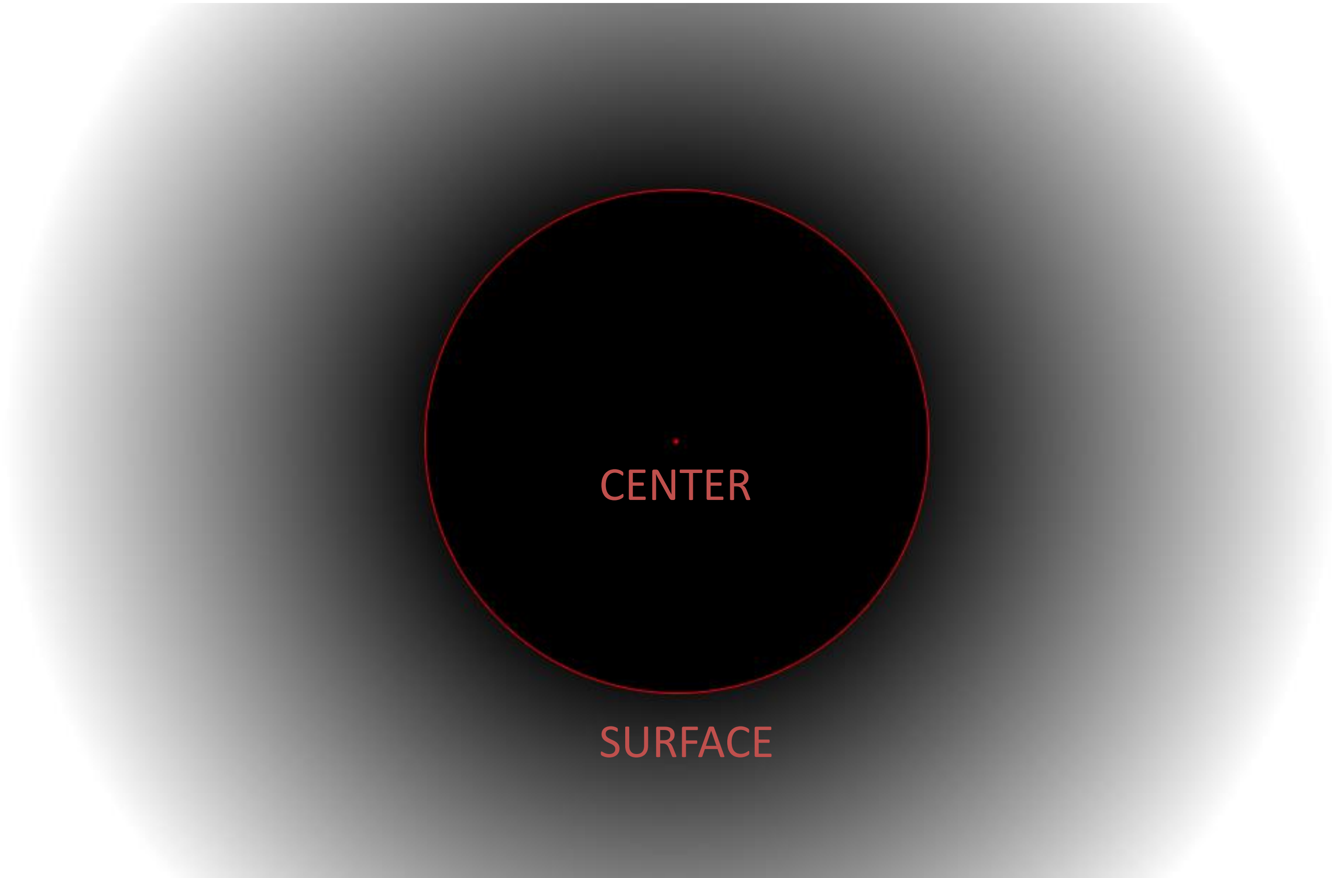
Caveats

- Looping through scene members is slow
  - May need partitioning

- Limits possible shapes
  - Ray-plane, ray-sphere, sure…
  - For complex geometry, test every triangle…?
  - Doing it properly is not a ShaderToy-sized problem

Approach #2 :
Raymarching Distance Fields

- Objects defined as distance functions
- How far is the object's surface from point *p*?



Raymarching, continued

- Iteratively get closer to intersecting surface
- Start at camera position, walk along ray direction

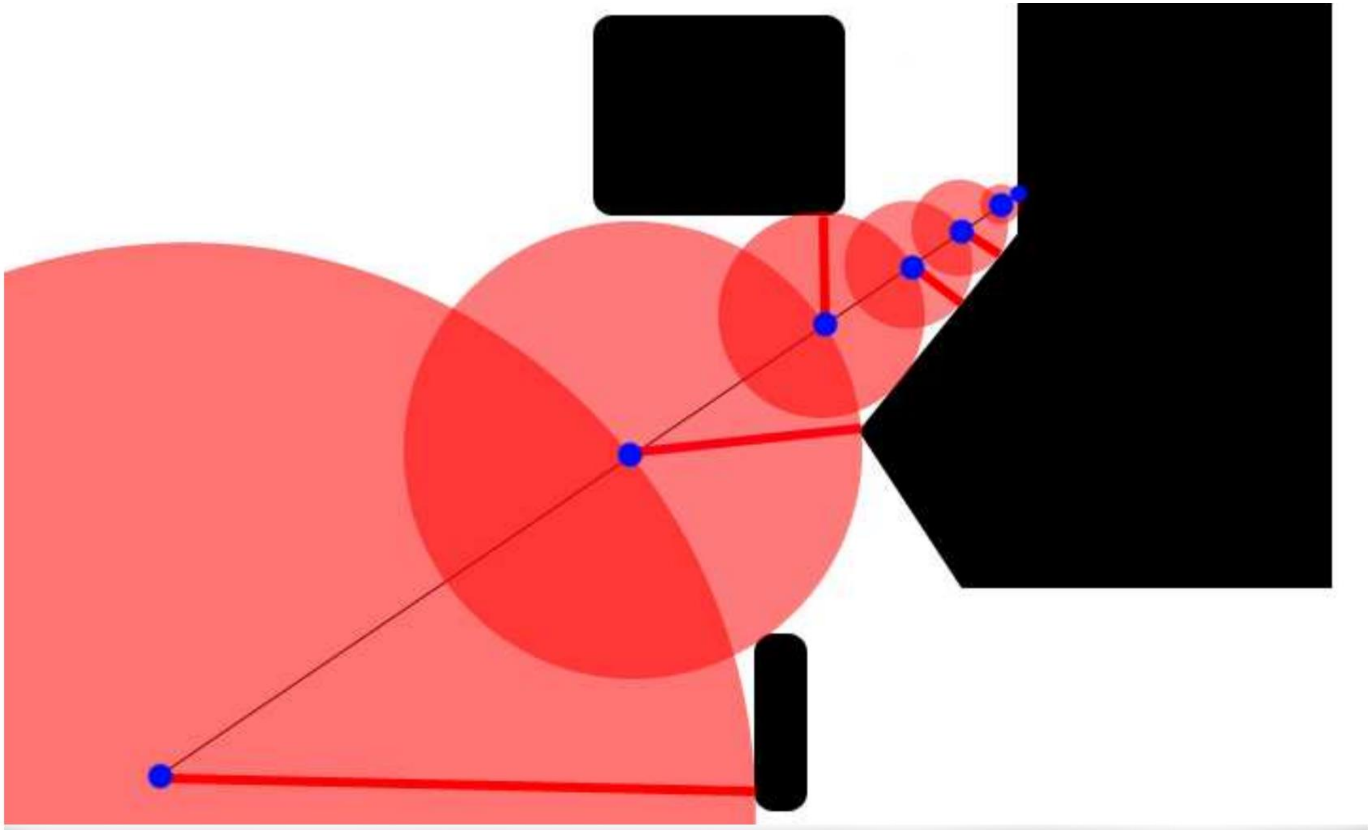


Image credit : RGBA demogroup, NVScene 2008 presentation

Constructing a scene

- Anatomy of a distance function
  - Input : Position at which we are marching
  - Input : Object parameters (dimensions, etc.)
  - Output : Distance to the surface

```
float sphere(vec3 p, vec3 offset, float radius)
{
    return length(p - offset) - radius;
}
```

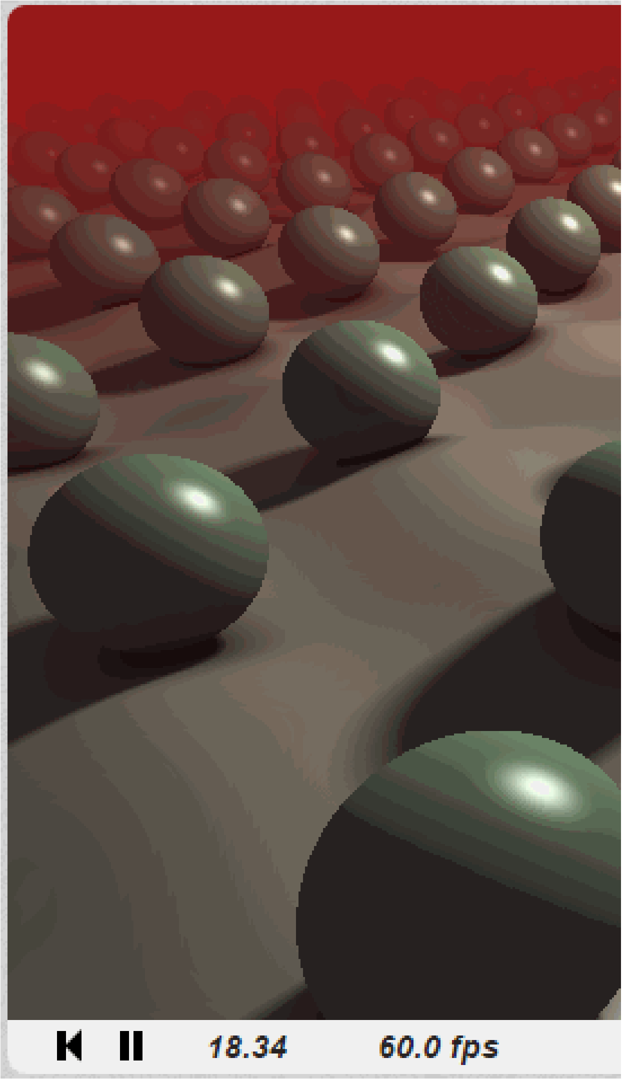
- To combine two distance functions, keep the smallest distance!

Operations on distances

- Duplicate objects infinitely by transforming the incoming position

```
vec3 q = mod(p, f) - 0.5 * f;
return shape(q /*, ... */);
```

- Boolean/CSG operations with `min()` and `max()`
- Displace surfaces by simply adding distances!



For more info…

- Lots of great raymarching building blocks on Iñigo Quílez’s website :

  http://bit.ly/iq_Raymarching

- Search for "raymarching" in ShaderToy
  - Most 3D ShaderToys use that



I guess what I’m trying to say is…

- Play and code are not mutually exclusive
- Shaders are awesome
- You’re more creative than you think!

Fin.

THANKS FOR LISTENING! <3
Questions?