

# hydra

Conjunto de herramientas\* para *livecoding* (en tiempo real) de  
gráficos con *streaming* de video entre interprete(s)

Toplap Bogotá - 2018

\*cabezas

# hydra

<https://github.com/ojack/hydra> web app source code

<https://hydra-editor-v1.glitch.me> web app

<https://github.com/ojack/hydra-synth> javascript library

<https://github.com/ojack/atom-hydra> atom package

<https://github.com/ojack/hydra-examples> examples

# Repositorio github

<https://github.com/ojack/hydra>



# Herramienta web

<https://hydra-editor-v1.glitch.me>



# hydra

Set of tools for livecoding networked visuals. Inspired by analog modular synthesizers, these tools are an exploration into using streaming over the web for routing video sources and outputs in realtime.

**Conjunto de herramientas para *livecoding* de visuales en red.**

Inspirado por los sintetizadores analógicos de video

Exploración del uso de streaming en internet para rutear entrada y salida de video en **tiempo real**.

Note: experimental/in development. Right now only works on Chrome or Chromium

# hydra local, web app

Javascript, control de los navegadores

Node, javascript para el escritorio

**npm**, node package manager

## Descarga

Consola (requiere conexión, obvio)

**npm install hydra**

## Instalación

Consola (requiere conexión, no tan obvio)

**npm install -d**

## Ejecución:

Consola:

**npm run start**

Browser

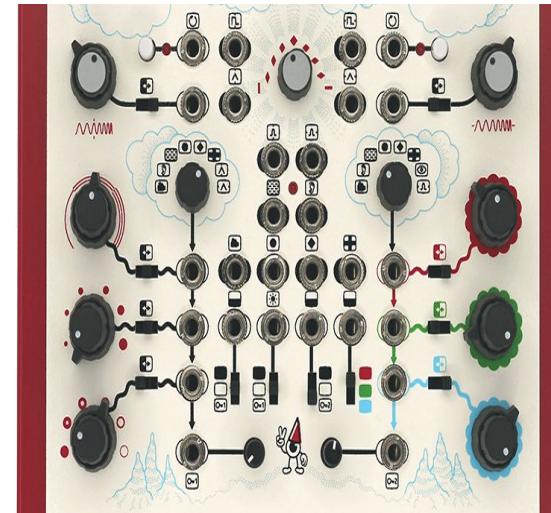
<https://localhost:8000>

# Sintetizadores analógicos de video (hardware)

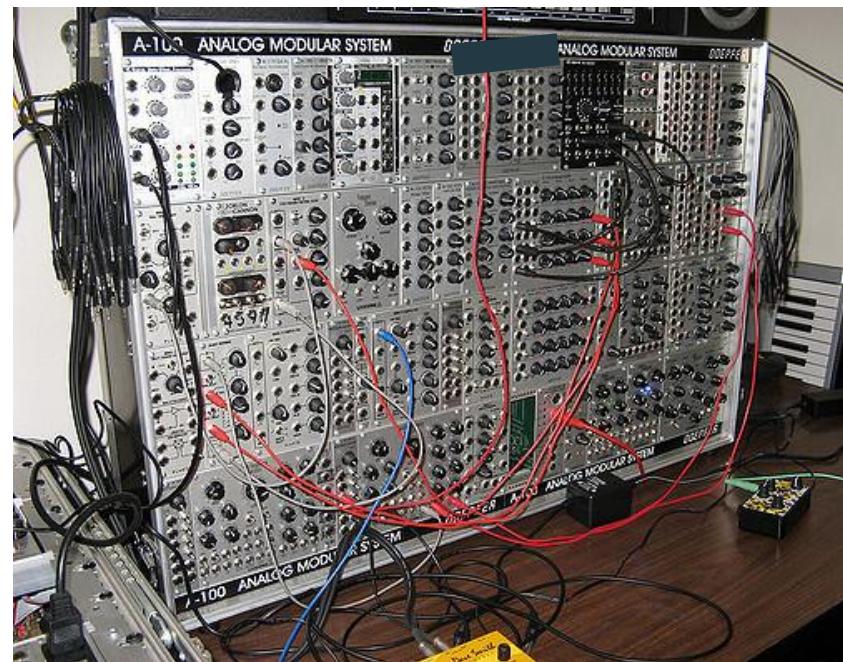


Modular

Semi modular



# Sintetizadores analógicos de audio modulares



# Sintetizadores analógicos de audio



Semi modular



No modular



Semi modular

# Sintetizadores analógicos de audio

Semi modular



Patch bay



No modular

# Sintetizadores digitales

De video:

Computador, hardware (atari xxxx)

Software: Sintetizador digital que simula un sintetizador analógico: <https://lumen-app.com/>

Software: Sintetizador digital que simula un sintetizador analógico: **hydra**

De audio:

Computador, hardware (Jorge Haro, por ejemplo, entre tantos)

# hydra, dependencias

```
"browserify-middleware": "^8.0.0",
"codemirrorexpresshydra-synthmeydap5rtc-patch-baysocket.io
```

# hydra-synth, librería de javascript

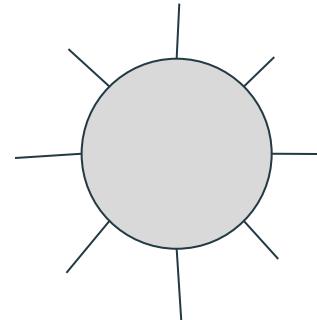
<https://github.com/ojack/hydra-synth>

Dependencias:

```
"budo": "^11.3.2",
"enumerate-devices": "^1.1.1",
"getusermedia": "^2.0.1",
"meydaregl
```

# hydra-synth y glsl

[Glsl](#) wikipedia

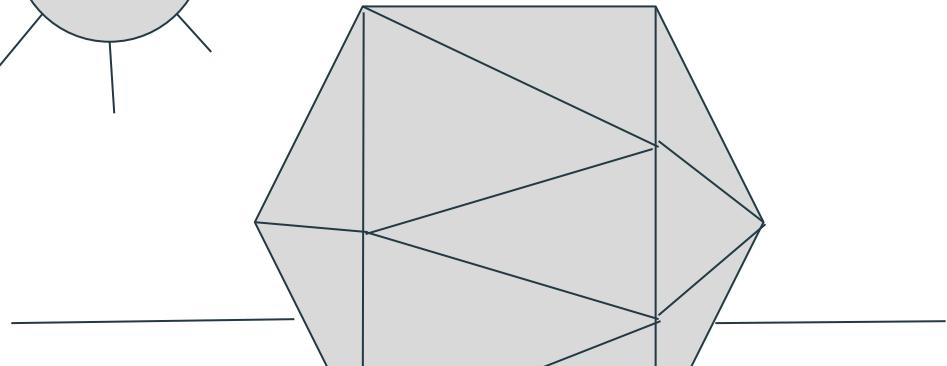


[Webgl](#) wikipedia

Shaders (*shadow, sombra y/o sombreado*)

**Vertex shaders** (coming), [three.js](#), entre otras, [stackgl](#)

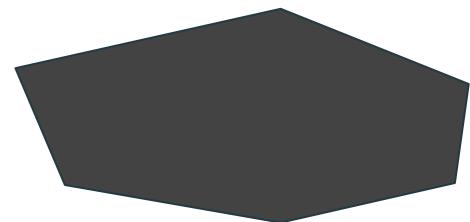
<http://vertexshaderart.com>



## regl

functional abstraction for wegle.

Fragment shaders...



# hydra-synth y glsl

Shaders (*shadow*, sombra y/o sombreado)

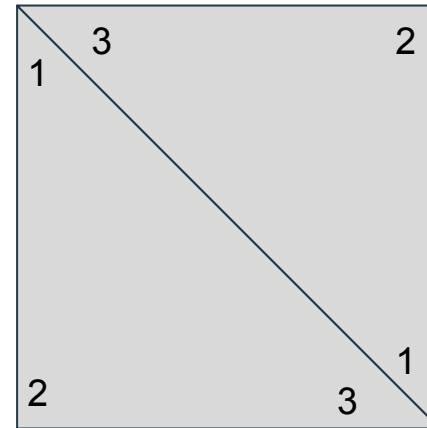
**Vertex shaders** (coming), [three.js](#), [regl](#), [stackgl](#)

<http://vertexshaderart.com>

**Fragment shaders**

<https://www.shadertoy.com/>

<http://glslsandbox.com/>



En shadertoy.com y glslsandbox.com se usa un *vertex shader* con **dos** triángulos. Después se usan los *fragment shaders* para hacer el “**sombreado**” de estos dos triángulos

# Interactive shader format

<https://www.interactiveshaderformat.com/>

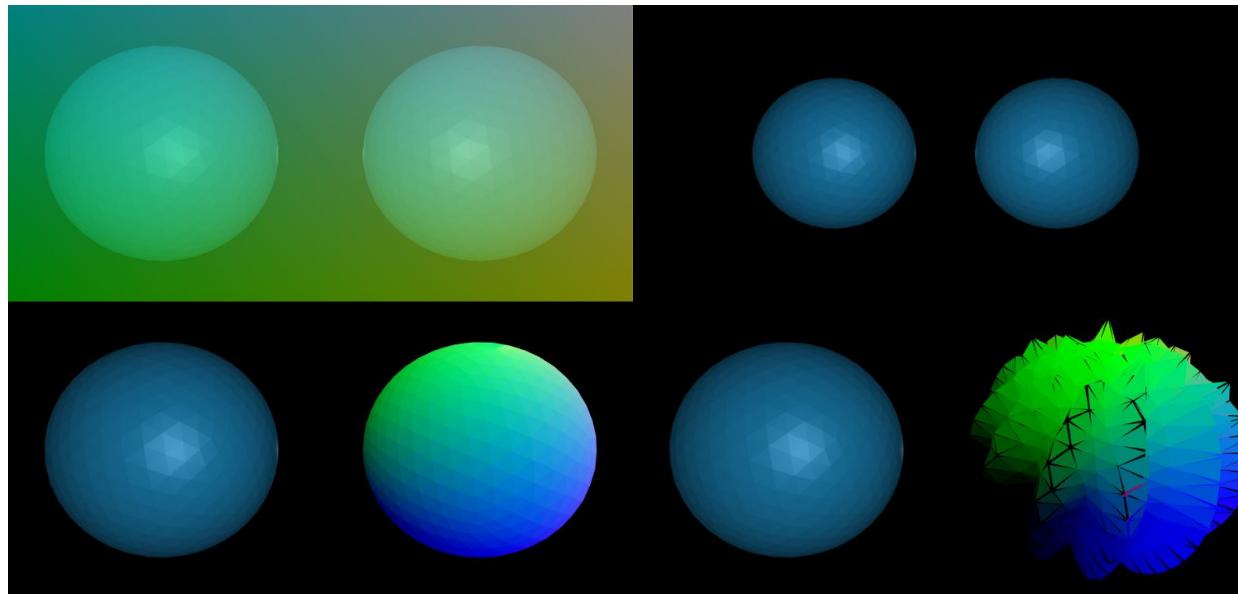
<https://github.com/msfeldstein/interactive-shader-format-js>

<https://www.interactiveshaderformat.com/spec>

There is a mac app

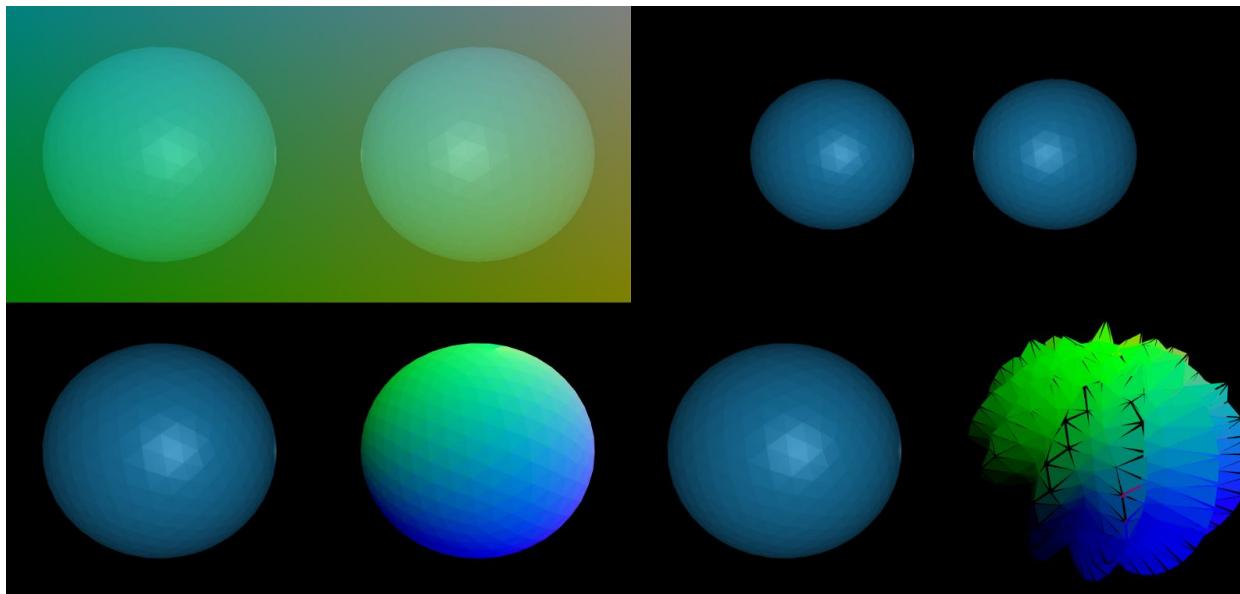
a vertex shader can change the position of vertices in 3D while a fragment shader usually replaces the color of a rendered image.

This image shows the four possible variations.



## Types of shaders

In the upper left a postprocessing shader adds a color gradient to the rendered image. To the right of it, a vertex shader reduces the render area. The two bottom images show material shaders; the left one only alters the color while the right one changes the position of the vertices. Since shaders are always composed of both vertex and fragment parts, the last example also changes the color



# hydra-synth y glsl, efectos

Shaders (*shadow*, sombra y/o sombreado), pero también efectos!

Todos (algunos de) los efectos de seriously.js  
efectos se pueden ‘traducir’ a hydra!!!!!!:

## Seriously.js effects

Por ejemplo, kaleidoscope effect **seriously.js style**

```
void main(void) {
    if (segments == 0.0) {
        gl_FragColor = texture2D(source, vTexCoord);
    } else {
        vec2 centered = vTexCoord - 0.5;

        //to polar
        float r = length(centered);
        float theta = atan(centered.y, centered.x);
        theta = mod(theta, TAU / segments);
        theta = abs(theta - PI / segments);

        //back to cartesian
        vec2 newCoords = r * vec2(cos(theta), sin(theta)) + 0.5;
        gl_FragColor = texture2D(source, mod(newCoords - offset, 1.0));
    }
}
```

## **kaleidoscope effect **hydra style****

```
kaleid: {
    type: 'coord',
    inputs: [
        {
            name: 'nSides',
            type: 'float',
            default: 4.0
        }
    ],
    glsl: `vec2 kaleid(vec2 st, float nSides){
        st -= 0.5;
        float r = length(st);
        float a = atan(st.y, st.x);
        float pi = 2.*3.1416;
        a = mod(a,pi/nSides);
        a = abs(a-pi/nSides/2.);
        return r*vec2(cos(a), sin(a));
   }`
},
```

# Custom-effect-definitions a la bbc VideoContext

## Custom effect definitions

```
var effectDefinition ={  
    title:"",           //A title for the effect.  
    description: "",    //A textual description of what the effect does.  
    vertexShader : "",  //The vertex shader  
    fragmentShader : "", //The fragment shader  
    properties:{        //An object containing uniforms from the fragment shader for  
mapping onto the effect node.  
    },  
    inputs:["u_image"]   //the names of the uniform sampler2D's in the fragment shader which  
represent the texture inputs to the effect.  
};
```

# hydra-synth y glsl, efectos

Shaders (*shadow*, sombra y/o sombreado), pero también efectos!

Algunos de los efectos de [three.js](#)  
efectos se pueden ‘traducir’ a hydra!!!!!!:

[https://threejs.org/examples/#webgl\\_postprocessing](https://threejs.org/examples/#webgl_postprocessing)

[https://threejs.org/examples/#webgl\\_postprocessing\\_glitch](https://threejs.org/examples/#webgl_postprocessing_glitch)

<https://github.com/mrdoob/three.js/blob/dev/examples/js/postprocessing/GlitchPass.js>

[https://threejs.org/examples/#webgl\\_postprocessing\\_rgb\\_halftone](https://threejs.org/examples/#webgl_postprocessing_rgb_halftone)

[https://threejs.org/examples/#webgl\\_postprocessing\\_masking](https://threejs.org/examples/#webgl_postprocessing_masking)

[https://threejs.org/examples/#webgl\\_postprocessing\\_pixel](https://threejs.org/examples/#webgl_postprocessing_pixel)

[https://threejs.org/examples/#webgl\\_postprocessing\\_advanced](https://threejs.org/examples/#webgl_postprocessing_advanced)

# hydra-synth y glsl, efectos

Shaders (*shadow*, sombra y/o sombreado), pero también efectos!

The book of shaders

# Hydra-synth, aplicaciones:

hydra, aplicación web, y atom-hydra *son*  
aplicaciones/clientes de hydra-synth

Integrar [extramuros](#) [:] o [estuary](#) [:]) y hydra

**Hacer estas diapositivas en hydra**

Instalación: Consola:

```
npm install hydra-synth
```

Ejecución/uso: Código:

```
const Hydra = require('hydra-synth');
hydra = new Hydra([opts]);
hydra.osc().out();
```

## Options

```
canvas: null, // canvas element to render
to. If none is supplied, a canvas will be
created and appended to the screen
pb: null, // an instance of rtc-patch-bay
to use for networking
autoLoop: true, // if true, will
automatically loop using
requestAnimationFrame. If set to false, you
must implement your own loop function using
the tick() method
makeGlobal: true, // if false, will not
pollute global namespace
numSources: 4, // number of source buffers
to create initially
numOutputs: 4, // number of output buffers
to use. Note: untested with numbers other
than 4. render() method might behave
unpredictably
```

# Atom-hydra, paquete para atom

<https://github.com/ojack/atom-hydra>

```
"atom-message-panel": "^1.3.0",
"dgram": "^1.0.1",
"hydra-synth": "^1.0.12",
"is-image": "^2.0.0",
"meyda": "^4.1.3",
"osc-min": "^1.1.1",
"p5": "^0.6.1",
"rtc-patch-bay
```

atom editor de texto de github, es decir, de microsoft >[:{}>, escrito en node/javascript; aplicación de escritorio.

Inspiración: Veda for atom, más general: **fragment shaders**.

# Tiempo real<sup>+</sup> y livecoding

Función V / entorno >	web*	atom*
línea	Ctrl + enter	Shift + enter
bloque	Alt + enter	Ctrl + enter
todo	Ctrl + shift + enter	-
Esconder código	Ctrl + shift + enter	-
Guardar!	Ctrl + S	

+Algoritmos que se puedan correr en tiempo real, en la tarjeta graficadora, en paralelo

\*All code can be run either from the in-browser text editor or from the browser console.

! Guarda una url que se puede compartir

# Empezar a cacharrear con hydra

Basic-functions

Passing functions as variables or control voltage

Desktop capture, requiere extensión de chrome

Connecting to remote streams, [rtc-patch-bay](#)

Audio Responsiveness (experimental), meyda librería de javascript.

<https://github.com/ojack/hydra-examples>

Documentación insipiente

# Variabes y funciones de acceso global

a	Análisis de audio, meyda librería de javascript
o0, o1, o2, o3	4 Buffers de salida, output buffers, texturas
render(oX)	Muestra el buffer X, X->0, 1, 2 o 3. <b>render()</b> Muestra los 4 buffers
s0, s1, s2, s3	Fuente de Buffer, <i>source buffer</i> , buffer fuente, texturas
src(s)	Entrada de buffer, <i>buffer input o input buffer</i>
pb	<u><a href="#">rtc-patch-bay</a></u>
bpm()	<i>Beats per minute</i> , pulsos por minuto de la variable time
msg()	<i>osc message manager (?)</i> administrador de mensajes osc.
time, mouse	

# Fuente de Buffer vs entrada de Buffer o buffer de entrada o Buffer fuente

Source buffer: Matemáticas que producen una textura

Buffer source src : gradient, noise, osc, shape, solid, src

Source Buffer (input?) s0, s1,...

```
s0.initCam(0) // initialize  
a webcam in source buffer s0  
src(s0).out() // render  
source buffer s0
```

Otras entradas de buffer, buffer inputs?, input buffers?

sX, canvas, video, or image

src(s0)

src(o0)

sX, sharing camera, screen capture, and  
remote streams

# Fuente de Buffer vs entrada de Buffer o buffer de entrada o Buffer fuente

In hydra-synth (and in hydra -web app- and in atom-hydra)

## Using Custom Sources

Any canvas, video, or image element can serve as a source in addition to the built-in source functions for sharing camera, screen capture, and remote streams. Video and images must be fully loaded before being passed to hydra.

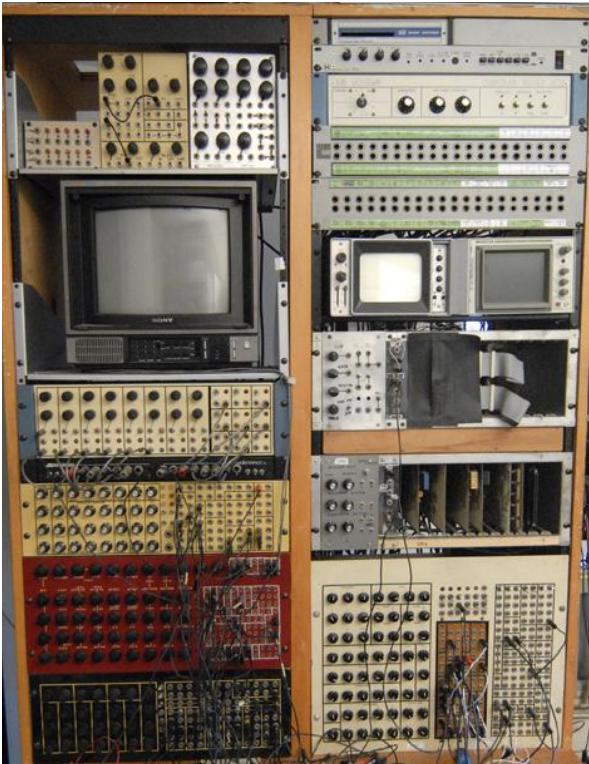
Add a custom source:

```
s0.init({  
    src: <canvas, video, or image element>,  
    dynamic: true // optional parameter. Set to false if using a static image or something that will not change  
})
```

You can add new source buffers once hydra has been initialized:

```
let src = hydra.newSource()  
src.init({ src: canvasEl })
```

# Analogía entre código y hardware



Código:

```
osc(20, 0.1, 0.8).rotate(0.8).pixelate(20,  
30).out()
```

```
osc(20, 0.1, 0.8)  
.rotate(0.8)  
.pixelate(20, 30)  
.out()
```

**Argumento de función-> perilla**

**Punto -> cable**

**Argumento de función -> cable**

# Documentación insipiente

All of the available functions for transforming coordinates and color, as well as compositing textures, correspond directly to a snippet of **fragment** shader code..

## API

For updated list of functions, see [composable-glslTransforms.js](#) file in hydra-synth

Mostrar app

# Oscilador (sinusoidal)

Elemento básico de muchos tipos de síntesis

`osc(freq, sync, offset)`

```
vec4 osc(vec2 _st, float freq, float sync, float offset){
```

```
    vec2 st = _st;
```

```
    float r = sin((st.x -offset/freq +time*sync)*freq)*0.5 + 0.5;
```

```
    float g = sin((st.x +time*sync)*freq)*0.5 + 0.5;
```

```
    float b = sin((st.x+offset/freq +time*sync)*freq)*0.5 + 0.5;
```

```
    return vec4(r, g, b, 1.0);
```

```
}
```



# Oscilador (sinusoidal)

Elemento básico de muchos tipos de síntesis

```
vec4 osc(vec2 _st, float freq, float sync, float offset){  
    vec2 st = _st;  
    float r = sin((st.x -offset/freq +time*sync)*freq)*0.5 + 0.5;  
    float g = sin((st.x +time*sync)*freq)*0.5 + 0.5;  
    float b = sin((st.x+offset/freq +time*sync)*freq)*0.5 + 0.5;  
    return vec4(r, g, b, 1.0);  
}
```



# Oscilador (sinusoidal)

Elemento básico de muchos tipos de síntesis

```
vec4 osc(vec2 _st, float freq, float sync, float offset){  
    vec2 st = _st;  
    float r = sin((st.x -offset/freq +time*sync)*freq)*0.5 + 0.5;  
    float g = sin((st.x           +time*sync)*freq)*0.5 + 0.5;  
    float b = sin((st.x+offset/freq +time*sync)*freq)*0.5 + 0.5;  
    return vec4(r, g, b, 1.0);  
}
```



# Oscilador (sinusoidal)

El oscilador digital de hydra tiene una “perilla” que el analógico no tiene

```
vec4 osc(vec2 _st, float freq, float sync, float offset){  
    vec2 st = _st;  
    float r = sin((st.x -offset/freq +time*sync)*freq)*0.5 + 0.5;  
    float g = sin((st.x +time*sync)*freq)*0.5 + 0.5;  
    float b = sin((st.x+offset/freq +time*sync)*freq)*0.5 + 0.5;  
    return vec4(r, g, b, 1.0);  
}
```



# Oscilador (sinusoidal)

Funciones como variables, Passing functions as variables

Simulación de VC (*voltage control*): ¡control de voltaje digital!

Ejemplo de **argumento de una función -> cable**

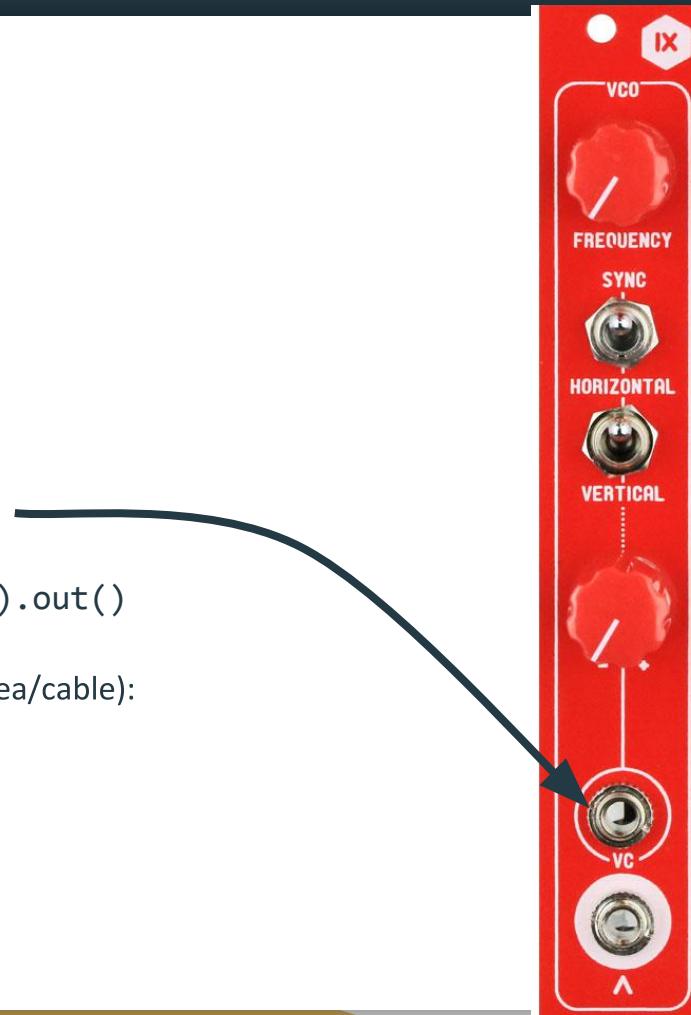
```
osc(300,(time) => (100 * Math.sin(time * 0.1))).out()
```

También es posible sincronizar/controlar con audio (mic\* o línea/cable):

```
osc(10, 0, () => (a.fft[0]*4)).out()
```

Ejemplo en hydra-examples (3)

\*pide autorización inicial



# Oscilador (sinusoidal)

Funciones como variables Passing functions as variables

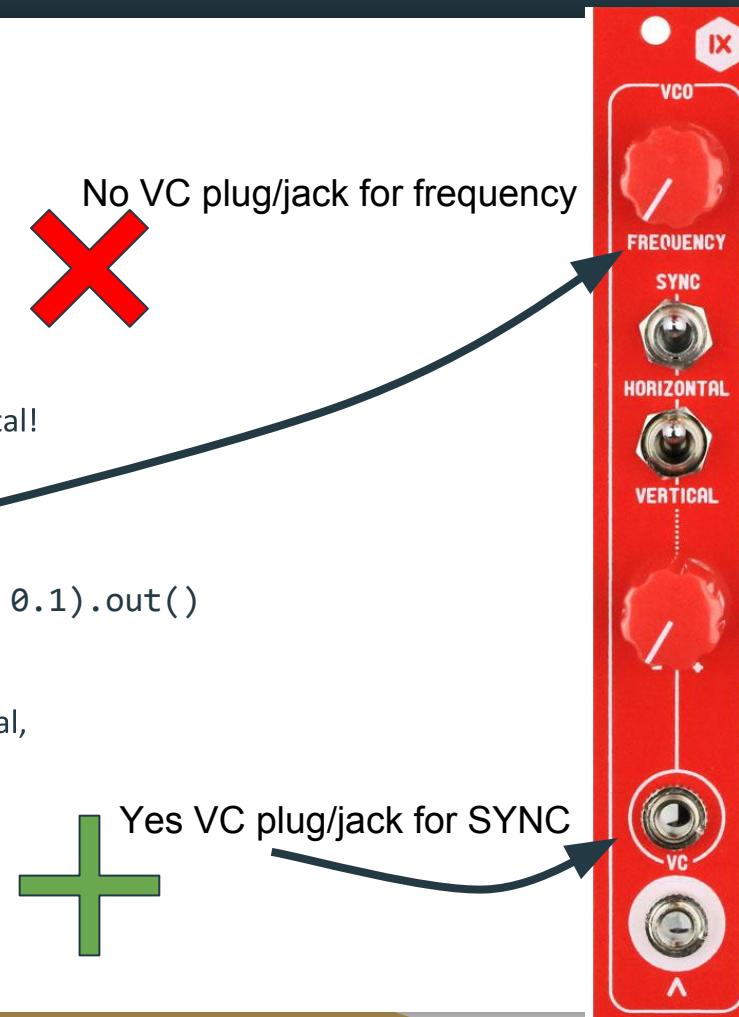
Simulación de VC (*voltage control*): ¡control de voltaje digital!

Ejemplo de **argumento de una función -> cable**

```
osc((time) => (100 * Math.sin(time * 0.1))), 0.1).out()
```

Por supuesto, debido a que estamos en el entorno digital,  
se puede hacer control por voltaje digital de elementos  
que en el hardware no están: de todas las “perillas”

No VC plug/jack for frequency



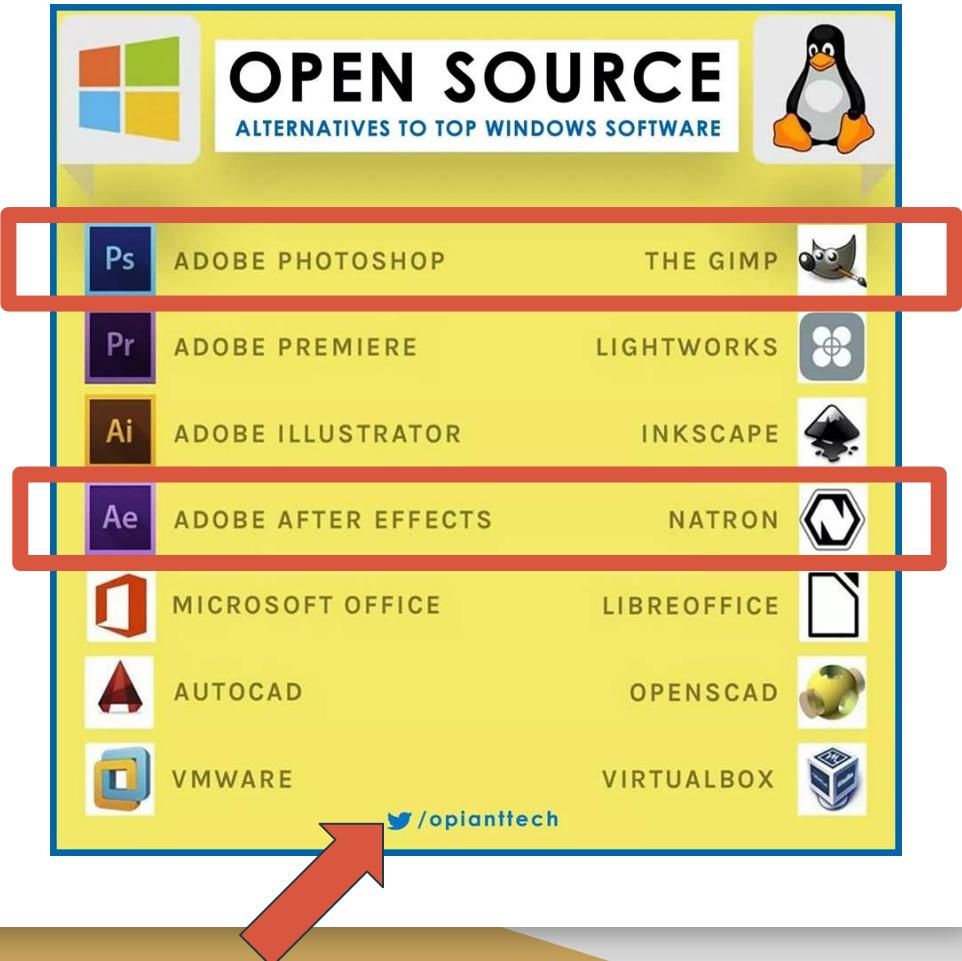
Yes VC plug/jack for SYNC

# hydra e imagen y video

The composite functions `blend()`, `diff()`, `mult()`, and `add()` perform arithmetic operations to combine the input texture color with the base texture color, similar to [the gimp](#) / [natron](#) blend modes.

[Ejemplo de blending](#) en hydra-examples (1)

[Ejemplo de video](#) en hydra-examples (5)



# Feedback, retroalimentación o acople

Space-Time Dynamics in Video Feedback (1984). video and paper about using analog video feedback to model complex systems.

Space-Time Dynamics in Video Feedback (1984). [video](#) and [paper](#) by Jim Crutchfield about using analog video feedback to model complex systems.

Buffer can be an input to itself:

```
osc(40, 0.1, 1).modulate(src(00), 0.1).scale(1.1).rotate(0.04).out(00)
```

(Ejemplo de cuando un **argumento de una función -> cable**)

Ejemplo\* en hydra-examples (2).

\*Aún no está.

# Feedback, retroalimentación o acople Cámara

`s0.initCam(0)` cámara incorporada al computador (eventualmente)

Feedback: apuntar otra cámara (web) a la pantalla del computador.

Analógico-digital, lente de la cámara y medio-ambiente y filtros analógicos (lentes u otros elementos difractores, por ejemplo)

`s0.initCam(1)` cámara conectada por usb, p. ej.

# Desktop capture

Instalar extensión de chrome en la [carpeta correspondiente](#) en hydra github. Otra cabeza de hidra.

Entrada de :

1. Otra aplicación, incluyendo video
2. Otro sitio web en otra pestaña/ventana del navegador.

También se puede usar para crear **feedback**, retroalimentación o acople

render screen tab:

```
s0.initScreen()  
src(s0).out()
```

# Connecting to remote streams

Todas las instancias de hydra que estén llamando al mismo servidor desde la misma red sin proxy pueden usarse entre sí como fuente, `src`. Otra cabeza de hidra.

Esto se logra usando *webstreaming* en tiempo real, real time *webstreaming*, web-rtc en el mundo javascript.

[rtc-patch-bay](#): (module built on top of SimplePeer)

administra las conexiones entre las ventanas.

Se puede usar como un módulo en sí para convertir cualquier sitio web en fuente/entrada/input de hydra.

Ventana 1

```
pb.setName("myGraphics")
```

```
pb.list()
```

Ventana2

```
s0.initStream("myGraphics")
src(s0).out()
```

# Sincronización con audio

Passing functions as variables

¡Control de voltaje digital!

[meyda](#), librería de análisis de audio

Descriptores de audio, otro ejemplo más en c++:

Sincronización/control con audio (mic\* o línea/cable):

```
osc(10, 0, () => (a.fft[0]*4)).out()
```

[Ejemplo](#) en hydra-examples (4)

\*en web pide autorización inicial



# Sincronización con audio Con tidal usando osc

Por medio de osc (open sound control), reemplazo/evolución de midi, (ejemplo\* midi en hydra-examples (10))  
Protocolo de transmisión de datos con mayor resolución que el midi.

¡Más control de voltaje digital! ¡Y más sincrónico! Mejor que los descriptores, o con otra expresividad. Otra cabeza pa la hidra.

osc-min, api en javascript para transferencia i/o de mensajes mediante protocolo osc.

Tidal, [tidalcycles.org](http://tidalcycles.org) y supercollider. No es análisis del sonido sino sincronización sintética, la misma fuente de sonido se sincroniza con las gráficas (p. Ej. ryo i keda, alva noto). **Sin** audio (mic\* o línea/cable).

Ejemplo 1 osc en hydra-examples (8). Hay que añadir el siguiente código de inicialización a supercollider.

\*Aún no está

# Sincronización con audio

## Con tidal usando osc

Por medio de osc (open sound control), reemplazo/evolución de midi,  
Tidal, [tidalcycles.org](http://tidalcycles.org). Tomado de tidalcycles.org:

Permite tocar música con un *timing* muy flexible, mediante un lenguaje de programación especializado en:

1. Descripción de patrones como secuencias de pasos (polifónicos y polimétricos).
2. Generadores de patrones continuos, p. ej. ondas sinusoidales, diente de sierra.
3. Un rango amplio de transformación de patrones.

# Computación física Con arduino usando osc

Por medio de osc (open sound control).

Ejemplo 2 en hydra-examples

Arduino: ¡Otra cabeza pa' la hidra!

Node se puede comunicar con Arduino y similares (esp32, fpga, wemos, por ej.) mediante la librería serial-port, estos mensajes se pueden convertir a osc y enviarlos a hydra.

Una librería de más alto nivel que serial-port: johnny-five.

# Azúcar sintáctica, syntactic sugar

[x,y,z,...] secuencia de valores

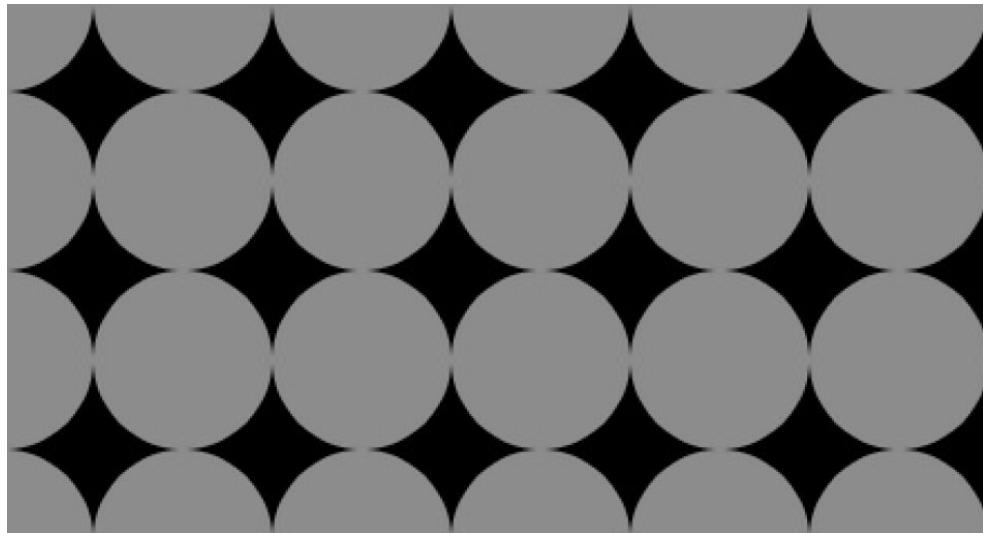
¡Como un secuenciador!

`osc([50,20,100].fast(0.5), 0.4, 0.5);`



# hydra + p5.js

hydra incluye p5.js: ¡otra cabeza de hydra!



Secure | https://alpha.editor.p5js.org

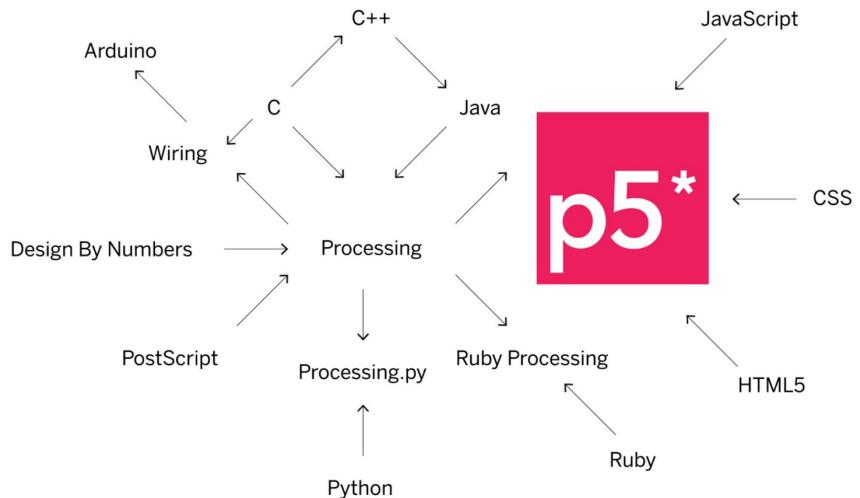
p5\*

File ▾ Edit ▾ Sketch ▾ Help & Feedback ▾

sketch.js\*

```
1< function setup() {
2   createCanvas(480, 120);
3   noStroke();
4 }
5
6< function draw() {
7   background(0);
8   for (var y = 0; y <= height; y += 40) {
9     for (var x = 0; x <= width; x += 40) {
10       fill(255, 140);
11       ellipse(x, y, 40, 40);
12     }
13   }
14 }
15 }
```

# hydra + p5.js



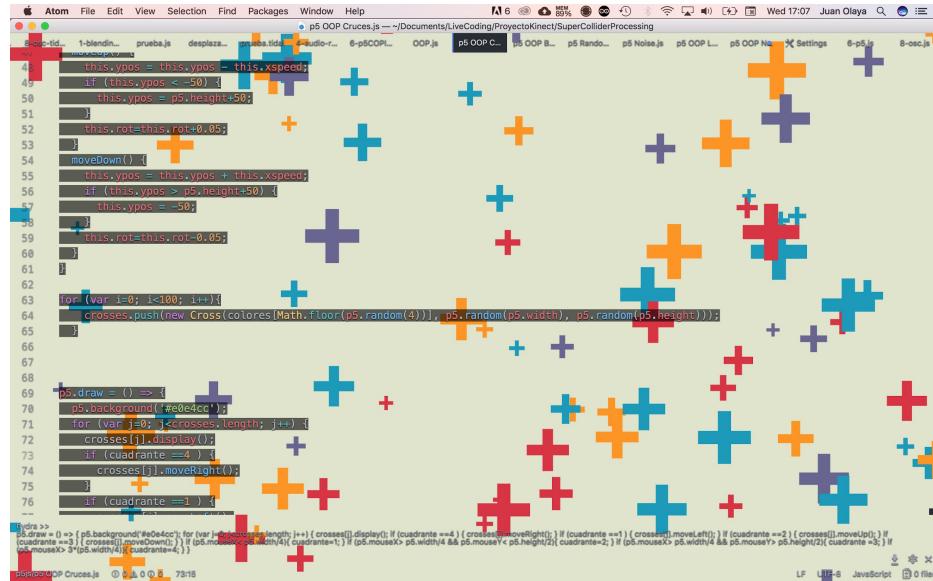
# Programación Orientada a Objetos en Hydra

## PROS:

1. Crear figuras geométricas
2. Crear muchas copias de los objetos
3. Controlar cada objeto y sus estados
4. Implementar diferentes comportamientos  
(e.g. simulación de física)

## CONTRA:

El tiempo de implementación es mayor



The screenshot shows an Atom code editor window with several tabs open, including "p5 OOP Cruces.js". The main editor area contains the following Processing.js code:

```
46  this.ypos = this.ypos - this.xspeed;
47  if (this.ypos < -50) {
48    this.ypos = p5.height+50;
49  }
50  this.rot=this.rot-0.05;
51
52  moveDown() {
53    this.ypos = this.ypos + this.xspeed;
54    if (this.ypos > p5.height+50) {
55      this.ypos = -50;
56    }
57    this.rot=this.rot-0.05;
58
59  }
60
61
62  for (var i=0; i<100; i++){
63    crosses.push(new Cross(colores[Math.floor(p5.random(4))], p5.random(p5.width), p5.random(p5.height)));
64  }
65
66
67
68
69  p5.draw = () => {
70    p5.background('#e0e0cc');
71    for (var j=0; j<crosses.length; j++) {
72      crosses[j].display();
73      if ((cuadrante == 4) || 
74          (cuadrante == 1) || 
75          (cuadrante == 3)) {
76        crosses[j].moveRight();
77      } else {
78        crosses[j].moveLeft();
79      }
80    }
81  }
82
83  cuadrante () => { p5.background('#e0e0cc'); for (var i=0; i<crosses.length; i++) { crosses[i].display(); if (cuadrante == 4) { crosses[i].moveRight(); } if (cuadrante == 1) { crosses[i].moveLeft(); } if (cuadrante == 3) { crosses[i].moveDown(); } if (p5.mouseX>p5.width/4 && p5.mouseY < p5.height/2) { cuadrante=2; } if (p5.mouseX>p5.width/4 && p5.mouseY> p5.height/2) { cuadrante=3; } if (p5.mouseX < p5.width/4 && p5.mouseY < p5.height/2) { cuadrante=4; } if (p5.mouseX < p5.width/4 && p5.mouseY> p5.height/2) { cuadrante=1; } }
84
85  mouse() {
86    cuadrante();
87  }
88
89  mouseDragged() {
90    cuadrante();
91  }
92
93  mouseReleased() {
94    cuadrante();
95  }
96
97  mouseMoved() {
98    cuadrante();
99  }
100}
```

# Programación Orientada a Objetos en hydra

## Object-Oriented Programming (OOP) with p5js + Processing

This repository is associated with the course: Object-Oriented Programming (OOP), which has been taught in the university [School of Arts and Letters](#) (Bogotá-Colombia) by Juan Olaya since the semester 2016.2 to the present.

DOI [10.5281/zenodo.1403871](https://doi.org/10.5281/zenodo.1403871)

For those courses we use the Java library [Processing](#) and its JavaScript version [P5js](#). The following are the course steps:

### Course steps

- Step 1: One Instance + Function move right
- Step 2: Two Instances + Functions move right/left
- Step 3: Multiple Instances + ArrayList + Functions move right/left
- Step 4: Multiple Instances + ArrayList + Functions move right/left + Rebound Function
- Step 5: Multiple Instances + ArrayList + Rebound Function + Location Vector + Multiple Constructors
- Step 6: Multiple Instances + ArrayList + Rebound Function + Velocity Vector + Gravity Vector
- Coding Challenge 1: Class Car showing a X
- Coding Challenge 2: Class Node showing a network

### Step 1: One Instance

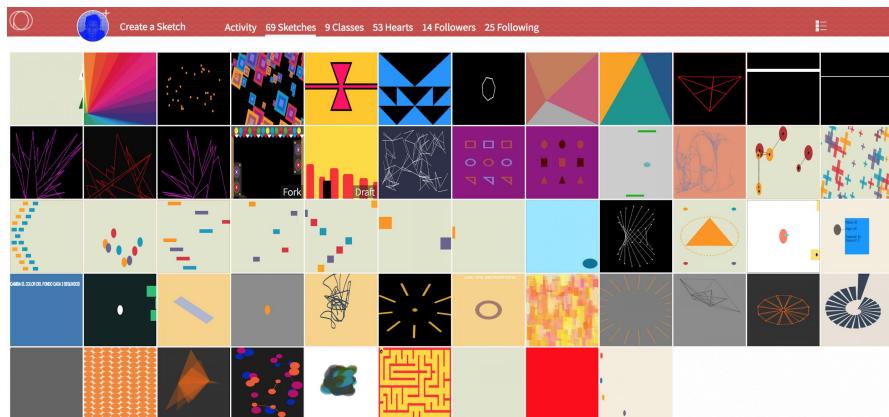
One Instance + Function move right. [Download Class Car](#).

[See on OpenProcessing](#)



<https://github.com/JuanOlaya/OOP-Processing>

# Programación Orientada a Objetos en Hydra



<https://www.openprocessing.org/user/65585>

# Programación Orientada a Objetos (POO) en hydra

Trabajos futuros -> Integración con:

1. Arduino + OSC + Hydra (p5.js + POO)
2. Hydra (p5.js + POO) + Shaders
3. Detección y reacción con beat de sonido

# hydra + p5.js



[The coding train](#)



[p5js.org](https://p5js.org)



[Ejemplo en hydra-examples](#)