CODEX

HYDRA

Olivia Jack

MANUAL DE FUNCIONES

Deme Té

VERSION 9/7/2019

Diseño de **flor de fuego**

Para toda
persona interesada
en usar HYDRA
de Olivia Jack.

¡Dedicado a vos!

Sigamos colaborando.

HYDRA

VISUALES CON CODIGO EN VIVO

Creado por Olivia Jack

Conjunto de herramientas para la visualización en vivo de imágenes en red. Inspiradas en sintetizadores modulares analógicos, estas herramientas son una exploración sobre el uso de la transmisión a través de la web para enrutar fuentes y salidas de video en tiempo real.

Hydra utiliza múltiples framebuffers para permitir la mezcla dinámica, la composición y la colaboración entre corrientes visuales conectadas del navegador. Las transformadas de coordenadas y colores se pueden aplicar a cada salida a través de funciones encadenadas.

NOTA

Experimental / en desarrollo. En este momento solo funciona en Chrome o Chromium, en máquinas con WebGL. Doy la bienvenida a las solicitudes de extracción, así como a los comentarios, ideas y errores en la sección de problemas.

{FUNCIONES BASICAS}

osc(20, 0.1, 0.8).out()

Renderiza un oscilador con parámetros de frecuencia, sincronización y rgb offset

Gira el oscilador 1.5 radianes:

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

Pixelar la salida de la función anterior:

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

s0.initCam()

Inicializa una webcam en el búfer de origen.

src(s0).out()

Render fuente de búfer .

Si tiene más de una cámara conectada, puede seleccionar la cámara usando un índice:

s0.initCam(1)

Inicializa una webcam en el búfer de origen.

src (s0) .out (o0)

// configura la fuente de o0 para representar el búfer que contiene la cámara web.

osc (10, 0.2, 0.8) .diff (o0) .out (o1)

Inicializa un gradiente en el búfer de salida o1, compuesto con el contenido de o0.

render (o1)

render o1 a la pantalla

{WEBCAM KALEIDOSCOPIO}

```
s0.initCam()
```

inicializa una webcam en el búfer de origen s0

```
src(s0).kaleid(4).out()
```

renderizar la webcam a un caleidoscopio

También puedes componer múltiples fuentes juntas:

```
osc(10)
.rotate(0.5)
.diff(osc(200))
.out()
```

De forma predeterminada, el entorno contiene cuatro bufers de salida independientes que pueden mostrar gráficos diferentes. Se accede a las salidas mediante las variables **o0, o1, o2 y o3**.para procesarlas en el búfer de salida o1:

```
osc().out(o1)
render(o1)
```

renderiza el contenido de o1

Si no se especifica ninguna salida en out (), los gráficos se representan en el búfer o0. para mostrar todos los buffers de render a la vez: render()

Los buffers de salida se pueden mezclar y componer para producir lo que se muestra en la pantalla

Las funciones compuestas **blend(), diff(), mult(), add(), mask()** realizan operaciones aritméticas para combinar el color de textura de entrada con el color de textura base, similar a los modos de mezcla de photoshop.

modular (textura, cantidad) utiliza los canales rojo y verde de la textura de entrada para modificar las coordenadas x e y de la textura base. Más sobre modulación en: https://lumen-app.com/guide/modulation/

```
osc (21, 0) .modulate (o1) .out (o0)
osc (40) .rotate (1.57) .out (o1)
```

{FUNCIONES COMO VARIABLES}

Cada parámetro se puede definir como una función en lugar de una variable estática.

Por ejemplo:

```
osc (function () {return 100 * Math.sin (time * 0.1)}). out ()
```

modifica la frecuencia del oscilador en función del tiempo. (El tiempo es una variable global que representa los milisegundos que han pasado desde que se cargó la página).

Esto se puede escribir de manera más concisa usando la sintaxis de es6:

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

{CONEXION A FLUJOS REMOTOS}

Cualquier instancia de Hydra puede usar otras instancias / ventanas que contengan hydra como fuentes de entrada, siempre que estén conectadas a Internet y no estén bloqueadas por un firewall. Hydra usa webrtc (webstreaming en tiempo real) bajo el capó para compartir transmisiones de video entre ventanas abiertas. El módulo incluido rtc-patch-bay administra las conexiones entre las ventanas conectadas, y también se puede usar como un módulo independiente para convertir cualquier sitio web en una fuente dentro de la hidra. (Consulte la fuente de la cámara independiente a continuación, por ejemplo).
Para comenzar, abra HYDRA simultáneamente en dos ventanas separadas. En una de las ventanas, establezca un nombre para el origen del compartimiento de parches dado:

```
pb.setName (“myGraphics”)
```

El título de la ventana debe cambiar al nombre ingresado en setName ().

Desde la otra ventana, inicie “myGraphics” como una secuencia de origen.

```
s0.initStream (“myGraphics”)
```

renderizar a pantalla:

```
s0.initStream (“myGraphics”)
src (s0) .out ()
```

Las conexiones a veces tardan unos segundos en establecerse; Abre la consola del navegador para ver el progreso. Para listar las fuentes disponibles, escriba lo siguiente en la consola:

```
pb.list ()
```

{RESPUESTA DE AUDIO}

EXPERIMENTAL

La funcionalidad FFT está disponible a través de un objeto de audio al que se accede mediante “a”. El editor utiliza https://github.com/meyda/meyda para el análisis de audio.
Para mostrar las bandejas FFT,

```
a.show()
```

Establecer el número de fft bins:

```
a.setBins (6)
```

Acceda al valor del contenedor de la izquierda (frecuencia más baja):

```
a.fft [0]
```

Usa el valor para controlar una variable:

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

Es posible calibrar la capacidad de respuesta cambiando el valor mínimo y máximo detectado. (Representado por líneas borrosas sobre el pie).
Para establecer el valor mínimo detectado:

```
a.setCutoff (4)
```

La configuración de la escala cambia el rango que se detecta.

```
a.setScale (2)
```

El fft [] devolverá un valor entre 0 y 1, donde 0 representa el límite y 1 corresponde al máximo.
Puede establecer el suavizado entre las lecturas de nivel de audio (valores entre 0 y 1). 0 corresponde a ningún suavizado (más rápido, tiempo de reacción más rápido), mientras que 1 significa que el valor nunca cambiará.

```
a.setSmooth (0.8)
```

Para ocultar la forma de onda de audio:

```
a.hide ()
```

{COLOR}

.contrast(cantidad)

cantidad (1.6 por defecto)

Mayor valor de cantidad hace mayor contraste

.color(r, g, b)

Colorear la textura

.colorama(cantidad)

(0.005 por defecto)

Desplazar los valores de HSV

.invert(cantidad)

(1.0 por defecto)

Invierte colores

.luma(compuerta, tolerancia)

compuerta (0.5 por defecto)

tolerancia (0.1 por defecto)

.thresh(threshold, tolerance)

threshold :: float (default 0.5)

tolerance :: float (default 0.04)

{COORD}

.kaleid(nSides)

nSides :: float (4.0 por defecto)
Efecto caleidoscopio con repetición nSides.

.rotate(angle, speed)

angle(10.0 por defecto)
speed (0.0 por defecto)

.scale(size, xMult, yMult)

tamaño ()
xMult (1.0 por defecto)
yMult (1.0 por defecto)

.pixelate(x, y)

pixelX :: float (20.0)
pixelY :: float (20 por defecto).
Pixela la textura con segmentos pixelX y segmentos pixelY.

.scrollX(scrollX, speed)

scrollX (0.5 por defecto)
speed(0.0 por defecto)

.scrollY(scrollY, speed)

scrollY (0.5 por defecto)
speed :: float (0.0 por defecto)
https://github.com/ojack/hydra/blob/master/docs/funcs.md

.repeat(x,y)

.repeatX()

.repeatY()

{COMBINECOORD}

```
.modulate(textura, cantidad)
.modulateHue (color, cantidad)
.modulateKaleid (nSides)
.modulatePixelate(multiplo, desfazaje)
.modulateRotate (multiple, offset)
.modulateScale (multiple, offset)
.modulateScrollX (multiple, scrollX, speed)
.modulateScrollY (multiple, scrollY, speed)
.modulateRepeat()
```

{SOURCES}

osc(frequency,sync,offset)
Frecuencia (60.0 por defecto)
sincro (0.1 por defecto)
compensación (0.0 por defecto)

render(output buffer)
default: o0

shape(sides, radius, smoothing)
Lados (3.0)
Radio (60.0)
Sutileza (0.01)
Genera una forma geometrica

solid(r, g, b, a)
Genera un solido (RGBA)

gradient(velocidad)
Genera un gradiente.

noise(scale, offset)
Genera Ruido Perlin

```
voronoi(scale,speed,blending)
```
Genera formas voronoi.

```
.out(output buffer)
```
output buffer osc: o0,o1,o2,o3 src: s0,s1,s2,s3

SECUENCIAS DE PARÁMETROS

```
osc([80, 100, 200, 50], 1 ))
.out(o0)

osc([80, 100, 200, 50].fast(0.2),1))
.out(o0)
```

VARIABLES GLOBALES

Algunas variables útiles que se definen globalmente,y puede ser utilizado dentro de funciones como un parámetro.

<TIME>
```
osc(({time})=>Math.sin(time))
.out(o0)
```

<MOUSE>
.x Posición X del mouse
.y Posición Y del mouse

```
osc(() => mouse.x).out(o0)
```
Controla la frecuencia del oscilador con la posición del Mouse.

Diseño de **flor de fuego**