

LIVE CREATIVE CODING

INTRODUCCIÓN A LA PROGRAMACIÓN CREATIVA CON GAMUZA

MARÍA JOSÉ MARTÍNEZ DE PISÓN | EMANUELE MAZZA

LIVE CREATIVE CODING

INTRODUCCIÓN A LA PROGRAMACIÓN CREATIVA CON GAmuza

Texto: María José Martínez de Pisón && Emanuele Mazza

GAmuza software: Emanuele Mazza

Dibujos: Carlos Maiques

ISBN: 978-84-608-7606-9

© Plutón Asociación cultural - www.pluton.cc - 2016

Autoedición

Valencia, España.

Formato digital y ejemplos descargables en:

https://github.com/d3cod3/GAmuza-Live_Creative_Coding-Book



Algunos derechos reservados. Reconocimiento - Compartir Igual (by-sa): Se permite el uso comercial de la obra y de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Contenidos

1. Introducción	13
2. ¿Qué es GAmuza?	19
2.1. Configuración Preferencias	23
3. Descripción de la interface	27
3.1. IDE de programación	27
3.1.1. Menú	29
4. Lenguaje de scripting	33
4.1. Elementos básicos de programación: datos, variables y funciones	35
4.1.1. Variables	37
4.1.2. Funciones	41
4.2. Expresiones aritméticas	45
4.2.1. Expresiones relacionales	46
4.3. Estructuras de control	46
4.3.1. Estructuras condicionales y operadores lógicos	47
4.3.2. Expresiones de iteración. Repetición	50
4.4. Estructuras de programación avanzadas	54
4.4.1. Tablas	55
4.4.2. Clases	61
5. Aplicaciones gráficas	69
5.1. Formas geométricas básicas	70
5.1.1. Formas irregulares	76
5.2. Color	80
5.3. Curvas por trigonometría y funciones matemáticas	87
5.4. Transformar: traslación, rotación, escalar	95
5.5. Azar y probabilidad	99
5.5.1. Noise	104
5.6. Gráficos vinculados a eventos de ratón	106
5.7. Gráficos vinculados a eventos de teclado	110

6. Trabajar con archivos: texto, imagen, video y audio	113	13. Protocolos de Comunicación	223
6.1. Textos y tipografía	113	13.1. Comunicación por OSC	223
6.2. Archivos de imagen	117	13.1.1. GAmuza <→ Processing	226
6.2.1. Animación con archivos de imagen y control del tiempo (fps)	118	13.1.2. GAmuza <→ Pure Data	229
6.2.2. Efectos imagen	120	13.2. Comunicación por MIDI. Recibir datos	232
6.3. Archivos de video	122	13.3. Comunicación por DMX. Enviar datos	234
6.4. Archivos de audio	127	13.4. Comunicación por Puerto Serie	236
6.4.1. Espectro sonoro de archivos de audio	130		
7. Módulo Timeline	133	14. Web Data Access	241
8. Interactividad	137	14.1. Data Parsing	241
8.1. Definiciones y referencias de Interactividad	138	14.1.1. Archivos genéricos txt	244
8.2. Interactividad con el sonido	148	14.1.2. Archivos XML	247
8.3. Instalaciones interactivas por Tracking Video	150	14.2. Internet Data	251
9. Sonido	153	14.2.1. HTTP Client, enviar datos a un servidor	254
9.1. Módulo Audio Analysis	155	15. Aplicaciones: proyectos artísticos	259
9.2. Audio input	158	15.1. Binary Ring, Jared Tarbel	259
9.3. Audio Sampling	163	15.2. Adaptación de Run Lola Run Lola Run Lola Run, Daniel Shiffman	263
9.4. Síntesis básica de audio	165	15.3. On Sense, Laboratorio de Luz	266
9.5. Audio Unit Plugins	171	15.4. Roomba, Laboratorio de Luz	270
9.6. Audio Pure Data Synthesis Engine	173	15.5. Folding, Emanuele Mazza y Cristina Ghetti	273
10. Tracking video	179	15.6. Bloomsday, Andrea Canepa	283
10.1. Panel Tracking: por cámara video	179	16. Estructura del código fuente de GAmuza	287
10.1.1. Descripción panel	182	16.1. Integración de openFrameworks	287
10.1.2. Blob detection	187	16.2. Integración de OpenGL	324
10.1.3. Motion	197	16.3. Funciones de Lua	330
10.1.4. Haar	201	16.4. Framework de GAmuza	333
10.1.5. Optical Flow	203		
10.1.6. Trigger Areas	205		
10.2. Panel Tracking: por sensor kinect	209		
10.2.1. Descripción panel	211		
10.2.2. Métodos y funciones para el sensor Kinect	213		
11. Interface Módulo Arduino	217		
12. Módulo Mapper	221		

Índice de ejemplos

Capítulo 4. Lenguaje de scripting

E-4-1	Expresiones aritméticas	45
E-4-2	Expresiones condicionales	48
E-4-3	Estructura repetición: for	50
E-4-4	Estructura for + if	51
E-4-5	Estructura doble for	52
E-4-6	Tabla básica	56
E-4-7	Tabla círculos rebotan	57
E-4-8	Tabla. Noise walker	59
E-4-9	Clase básica	64
E-4-10	Tabla id No clase	66
E-4-11	Clase y tabla 2D	67

Capítulo 5. Aplicaciones gráficas

E-5-1	Rect mode	72
E-5-2	Estructura repetición - círculos	74
E-5-3	Estructura for y color HSB	75
E-5-4	Formas irregulares lineales	76
E-5-5	ofVertex y ofCurveVertex	77
E-5-6	ofCurve y ofBezier	78
E-5-7	Curvas bezier encadenadas	79
E-5-8	Mapear color HSB	85
E-5-9	Tinte imagen	86
E-5-10	Onda sinusoidal	89
E-5-11	Trigonometría. Espiral	91
E-5-12	Duchamp Rotoreliefs	92
E-5-13	Arco de circunferencia	93
E-5-14	Trigonometría: trasladar y girar	94
E-5-15	Duchamp Rotoreliefs movimiento	96
E-5-16	Repetición de translación	97
E-5-17	Random regulado	100
E-5-18	Random walker tradicional	102
E-5-19	Random walker condicionado	103
E-5-20	Modulación noise 1D	104
E-5-21	Modulación noise 2D	105
E-5-22	Interactividad ratón básica	107
E-5-23	mouseDragged y mouseMoved	108
E-5-24	mouseDragged, mouseReleased, mouseMoved	109
E-5-25	Interactividad teclado	110

Capítulo 6. Trabajar con archivos: texto, imagen video y audio

E-6-1 Font unicode random	114
E-6-2 Font curveText	115
E-6-3 Font: texto como formas	116
E-6-4 Cargar imagen - loadImage	117
E-6-5 Tabla imágenes - animación	118
E-6-6 Shaders ejemplo - desenfoque	120
E-6-7 Video Player	122
E-6-8 Imagen webCam + ofPixel()	124
E-6-9 Audio/soundPlayer	128
E-6-10 Análisis simple FFT de archivo audio	130

Capítulo 7. Módulo TimeLine

E-7-1 Módulo Timeline	133
-----------------------	-----

Capítulo 9. Sonido

E-9-1 Audio input - Pitch	158
E-9-2 Audio input - Volumen	159
E-9-3 Audio/audioTrigger	160
E-9-4 Audio input - FFT	161
E-9-5 Audio input - Análisis simple escala Bin	162
E-9-6 Grabación / reproducción sample	164
E-9-7 Generación básica síntesis audio	166
E-9-8 Síntesis audio: Notas musicales	167
E-9-9 Síntesis audio: Cambio tipo de onda	168
E-9-10 Generative audio synth/graphics	169
E-9-11 Audio unites plugins	171
E-9-12 PD Pitch Shifter	175

Capítulo 10. Tracking video

E-10-1 Panel Tracking video	179
E-10-2 ComputerVision - Blobs	189
E-10-3 ComputerVision - Contorno y geometría	191
E-10-4 ComputerVision - Color tracking	194
E-10-5 Interactividad por motion tracking	197
E-10-6 ComputerVision - Haar	201
E-10-7 ComputerVision - Optical Flow	203
E-10-8 ComputerVision - triggerAreas	205
E-10-9 Panel Tracking kinect	209
E-10-10 SensorKinect - Kinect contorno	214

Capítulo 11. Módulo Arduino

E-11-1 Arduino - Servo Motor	218
------------------------------	-----

Capítulo 13. Protocolos de Comunicación

E-13-1 OSC - Envío datos GAmuza <-> Processing	226
E-13-2 OSC - Recibir datos de Pure data	229
E-13-3 OSC - Enviar datos a Pure Data	231
E-13-4 Uso básico de funciones MIDI	232
E-13-5 Comunicación con dispositivo DMX	234
E-13-6 Comunicación con puerto serie: enviar y recibir	238

Capítulo 14. Web data access

E-14-1 Leer datos de archivo txt	244
E-14-2 Leer datos de archivo XML	248
E-14-3 Internet data/http parse data	252
E-14-4 http Form Upload File	255

Capítulo 15. Aplicaciones: proyectos artísticos

E-15-1 <i>Binary Ring</i> - Jared Tarbel	260
E-15-2 Matriz movie 2D desde archivo video	264
E-15-3 <i>On Sense</i> - Laboratorio de Luz	267
E-15-4 <i>Roomba</i> - Laboratorio de Luz	270
E-15-5 <i>Folding</i> - Cristina Ghetti + Emanuele Mazza	274
E-15-6 <i>Bloomsday</i> - Andrea Canepa	284

1. Introducción

La mayoría de las escuelas de arte dan clases de "arte digital", que para ellos significa "cómo utilizar Adobe Photoshop". A pesar de que en estos cursos suelen decir que exploran las posibilidades de un nuevo medio, por lo general exploran poco más que las posibilidades que alguien (es decir, Adobe) ha encontrado conveniente empaquetar en una pieza de software comercial. El hecho es que los ordenadores son capaces de hacer un número inimaginablemente mayor de cosas que cualquier pieza de software específico puede hacernos creer. [...] es mi intención alentar a los artistas visuales a entender y trabajar más allá de las limitaciones impuestas por sus herramientas de software. [...] me gustaría ofrecer una visión de lo que puede lograrse cuando un artista pasa de Photoshop por completo, y hace sus propias herramientas de software con código.¹

En la enseñanza del arte, la importancia de los componentes técnicos, conceptuales, perceptuales y afectivos, se escalan en cada materia para atraer la atención hacia uno u otro rasgo, pero al señalar ese acento no se olvidan las tensiones que deben darse entre esos componentes y que sedimentan el proceso creativo. Con el poso de esa imagen de relaciones en mente, observamos que una parte importante del arte digital está vinculada a los lenguajes de programación creativa, y para subrayar ese marco es fundamental combinar en su enseñanza la capacidad de abstracción que requiere la técnica con el sentido crítico y especulativo que impulsan las humanidades, porque en nuestro campo el código es técnica y lenguaje, es lógica y pensamiento, que se dirige hacia el arte contemporáneo, lo que suma no pocas incertidumbres a los estudiantes que crecen en esta emulsión.

Para reducir tensión a la mixtura, la experiencia acumulada en docencia e investigación se ha vertido en el desarrollo del software GAmuza², una herramienta pensada para impulsar el salto y atraer el aprendizaje. Tras comprobar su funcionamiento en diversos seminarios, en 2012 se incorporó a la enseñanza reglada en el tercer curso de la titulación de grado de la Facultad de Bellas Artes de Valencia, y es el análisis de estas experiencias lo que guía e impulsa este texto³.

En el aula, como en la vida cotidiana, hay una presencia masiva de dispositivos técnicos de comunicación que, además de ser un reflejo comercial del crecimiento de esa industria, indica también la transformación de las relaciones entre las personas y de estas con el mundo. Y ante estos cambios debe surgir el deseo de comprender el papel que la técnica y el lenguaje técnico están jugando en esa transformación.

¹ Golan Levin, (2001) 'Essay for 4x4: Beyond Photoshop with Code' [artículo on-line] <http://www.flong.com/texts/essays/essay_4x4/> [12.05.2012].

² El software GAmuza ha sido desarrollado por Emanuele Mazza como resultado de un proyecto de investigación del Laboratorio de Luz de la Universidad Politécnica de Valencia, financiado por el Ministerio de Ciencia e Innovación Proj.Ref. (HAR2008-02169).

³ Paralelamente se ha llevado un blog que puede crecer sin las limitaciones físicas de estas páginas, <<http://mie.pluton.cc>> en cuya galería se pueden ver algunos de los ejemplos aquí mostrados y muchos otros realizados por los estudiantes.

Durante las últimas décadas del siglo XX el arte se hizo eco del debate entre la prevalencia del lenguaje sobre la imagen o viceversa [picture turn <-> linguistic turn]. Hoy sentimos una polaridad similar entre aquellos debates artístico-filosóficos y el impulso del lenguaje científico, cuya lógica, consciente en parte de sus fallidos principios de verificación, ha ido filtrando las incertidumbres e indeterminaciones de una nueva ciencia que, poco a poco, (o mucho a mucho) va encarnando nuevos esquemas conceptuales; diagramas que dibujan en el horizonte una nueva imagen-mundo, enunciada y representada con el lenguaje numérico, informático.

Según Vilem Flusser, las conexiones que el hombre establece hoy con esas imágenes-mundo, pasan por las "sinuosidades de nuevos campos de relación":

[...] mundos totalmente alternativos se han hecho computables a partir de los números. Estos mundos vivenciables (estéticos) le deben su posible fabricación al pensamiento formal matemático. Lo que tiene como consecuencia que, no solamente los teóricos científicos y los técnicos que aplican sus teorías tienen que aprender el código de este nuevo nivel de conciencia, sino en general todos los intelectuales (y sobre todo los artistas), si es que quieren tomar parte en la empresa cultural del futuro.⁴

Los conocimientos, vivencias y valores que argumenta Flusser pueden ahora llegar hasta el espacio cotidiano y entre las muchas cosas, banales o extraordinarias, que esta información digital ofrece, debemos valorar el esfuerzo de muchos colectivos diluidos por la red que, desinteresadamente, facilitan el acceso a muy distintos campos del saber. Podemos leer investigaciones sobre partículas, discutir la explicación cualitativa del Principio de Incertidumbre de Heisenberg, o las semejanzas entre la teoría de las Catástrofes de René Thom y la del Caos de Ilya Prigogine, entre muchas otras teorías o principios que han incidido en ese cambio de visión y comprensión del mundo. Y así el artista, como explorador intruso en los bosques de la ciencia, busca ciertos claros.

Y la visión lejana del centro apenas visible, y la visión que los claros del bosque ofrecen, parecen prometer, más que una visión nueva, un medio de visibilidad donde la imagen sea real y el pensamiento y el sentir se identifiquen sin que sea a costa de que se pierdan el uno en el otro o de que se anulen.

Una visibilidad nueva, lugar de conocimiento y de vida sin distinción, parece que sea el imán que haya conducido todo este recorrer análogamente a un método de pensamiento.⁵

Los lenguajes de programación, los algoritmos, son pensamiento, y permiten imaginizar sistemas o situaciones complejas de forma dinámica. Más allá del efecto visual o sonoro que puedan producir, su lectura subraya cómo el proceso ha ganado importancia sobre el resultado, dejando visibles los pasos que el pensamiento traduce en términos de programación. No hay límite en el lenguaje, ni en el pensamiento, hay un discontinuo *work in progress* que debería estar impulsado por el marco social

⁴ Vilem Flusser (2005), "La sociedad alfanumérica" en Revista Austral de Ciencias Sociales nº 9, pág. 105.

⁵ María Zambrano (1990), *Claro del Bosque*, Barcelona: Seix Barral, pág. 14.

y cultural, siendo reflejo de una época que no puede ya ordenar linealmente los acontecimientos, no puede reflejar con fijezas lo cambiante y abre redes, ramificaciones e interconexiones.

No hay simples códigos trabajando cuando las personas interactúan con las imágenes, entre ellos está su cultura, o su contexto. Esta es la razón por la que he insistido en la noción de visualización (con especial referencia a lo que hacen los seres humanos), tanto en la creación como en la respuesta, tanto en la generación como en el reconocimiento hay intersecciones entre conocimiento, información y pensamiento⁶.

Podemos quedarnos a contemplar la apariencia de esa imagen-mundo como usuarios enganchados a los dispositivos técnicos de comunicación, o saltar la valla, no sin esfuerzo, para aprender el lenguaje que construye de forma diferente esta nueva espacialidad y temporalidad.

Tal vez esta simplificación de situaciones pueda servir para clarificar inicialmente posiciones, pero no podemos plantearnos el desarrollo del pensamiento siguiendo polaridades: pensamiento científico por un lado y filosófico por otro. No. Como veremos, fue Leibniz quien acuñó los términos «función», «variable», «constante» y «parámetro», en un momento en el que, como hoy, el análisis de la lógica pertenecía tanto al campo de la filosofía como al matemático.

Los sistemas de análisis filosófico, desde Moore o Russell, inspeccionaban todo aquello en lo que se detiene el ojo de la mente, fragmentándolo, observando las relaciones de esos pequeños trozos de mundo; desarticulando lo complejo en partes significativas más simples, desde los objetos materiales a los datos de los sentidos, para re-articularlo después por medio de un análisis conectivo. Este mismo sistema de análisis es la base para empezar a desarrollar estructuras complejas con los lenguajes de programación. Nuestro propósito es clarificar cómo el pensamiento traduce las situaciones en algoritmos, y para ello analizaremos estructuras de código significativas, comentadas con imágenes de resultados, explicación del proceso y su relación con referentes artísticos.

La base del pensamiento visual y la percepción espacial se fundamenta en estructuras geométricas cuyo origen también está vinculado a las ciencias ópticas y el estudio de la luz. Paul Virilio nos dice que actualmente hay otra óptica, otra luz y otra transparencia, "trans-apariencia electrónica [que] prepara un nuevo régimen de relaciones"⁷, Michel Serres se sitúa mucho más atrás, para decirnos que la geometría nació de una sombra arrojada en la cambiante superficie de la arena del desierto, ¿cómo fijar un punto allí donde toda huella es borrada por el viento? Y de la luz que arrojó aquella sombra comenta:

⁶ Ron Burnett (2004). *How images think*, Massachusetts: The MIT Press, pág. 203 [texto on-line] [06.08.2012] <<http://www.learndev.org/dl/HowImagesThink.pdf>>

⁷ Paul Virilio, "Todas las imágenes son consanguíneas" [Texto on-line] <<http://www.upv.es/laboluz/2222/textos/virilio.htm>> [07.08.2012]

Cualquier óptico te dirá que la división claro-oscuro no es distinta más que en circunstancias excepcionales. Y, con precisión, cuando no hay atmósfera, cuando no hay fluidos turbulentos [...] En la atmósfera, nunca homogénea, nunca isótropa, los medios locales están distribuidos de forma múltiple, de suerte que los rayos, así puede decirse, buscan fortuna en el mundo. Se quiebran por doquier y ejecutan un recorrido caprichoso. Contornean pues los obstáculos y producen un claro-oscuro, un confuso-distinto. Y es así como se ve lo más comúnmente del mundo. No se necesita, donde sea y siempre, la presencia del sol. La luz se difunde. Si el ver es un modelo del saber, el conocimiento es casi siempre difuso. En absoluto confuso y oscuro, sino multiplicado por franjas y difracciones.⁸

También el sonido y la música tienen una base matemática y generan formas geométricas sinuosas, ondas sujetas a franjas y difracciones similares a las de la luz, pero que oscilan en longitudes distintas y con velocidad diferente.

Si para el ser humano el mundo es lenguaje (o no podemos tener otra imagen del mundo que la que el lenguaje nos brinda), la hibridación y feedback del análisis de los lenguajes visuales, sonoros, lingüísticos e informáticos permite generar otras formas sintácticas de expresión, que como el mundo actual, son más proteicas, cambiantes.

El arte contemporáneo, como generador de la parte expresiva y simbólica de este mundo, ha estado atravesado desde los años 60 por estas nociones de feedback transdisciplinar, buscando nuevas relaciones entre esos lenguajes para modelizar imágenes, sonidos o acontecimientos. No vamos a desarrollar en este texto las teorías vinculadas a estos fenómenos, aunque en algunos momentos aludamos a ellas, lo que queremos subrayar es cómo convergen los modos de comprensión estéticos, filosóficos y científicos, a la espera de que la fascinación que nos procura esta imagen impulse a los estudiantes a superar las dificultades o rechazos que el campo de la programación les genera a priori.

Y en ello va también un reconocimiento, porque nuestro propio impulso debe mucho a la herencia de Processing y openFrameworks, y también a la metodología aportada por Casey Reas, Ben Fry, Daniel Shiffman, Zachary Lieberman, Theo Watson, Arturo Castro y las comunidades de ambos software, por eso los contenidos de este texto van a seguir en parte los pasos aportados por estos y otros autores.

Más allá de las referencias concretas que hay a lo largo del texto sobre la estructura de programación en Processing, su influencia está latente en el deseo de entender y trasmisir el código como algo diferente a un tutorial o manual de instrucciones, articulando datos informáticos con reflexiones estéticas y proyectos artísticos. Con este conjunto de nociones técnicas, teóricas y referenciales esperamos avanzar por un camino que nos lleve a superar la disyuntiva planteada por Lev Manovich.

⁸ Michel Serres, (1991). *El paso del Noroeste*. Madrid: Debate, pág. 48.

Lo que no debemos esperar de Turing-land es un arte aceptable en Duchamp-land. Duchamp-land quiere arte, no la investigación de nuevas posibilidades estéticas de los nuevos medios. La convergencia no va a suceder.⁹

En la técnica subyace la naturaleza de los elementos que propician la interactividad, la generación de formas por el sonido, el movimiento de dispositivos por código..., por eso ella conoce bien cómo se construye la casa del arte digital, sus habitáculos y aperturas al exterior, su extenso jardín y el laberinto que contiene. Desvelar la estructura de la técnica¹⁰ es comprender ese complejo plano constructivo, y siguiendo sus indicaciones podremos empezar a esbozar diagramas, o planificar instrucciones que posibiliten procesos o comportamientos artísticos diferentes.

Para mostrar la relación de esos elementos básicos con los referentes artísticos, recurriremos a minuciosos dibujos que, a modo de ilustraciones botánicas, resaltarán en cada caso la cuestión a analizar, al tiempo que abren una puerta de fuga diferente a la del enmarañado mundo de los derechos de publicación.

Respecto al habitual ¿cómo leer este libro?, el nivel de atención y orden de lectura es algo que depende del bagaje de cada uno. Como dice Derrida, "el lenguaje escrito se dirige inevitablemente a más de una singularidad",¹¹ y si tú, lector particular, estás acostumbrado a utilizar software de programación, es posible que algunas descripciones de código te parezcan obvias; esperamos que eso no consuma tu deseo, piensa que otros posibles lectores se están iniciando, y que tanto GAMUZA como este texto buscan facilitar el aprendizaje.

Para abrir la linealidad de la lectura, se han planteado algunas conexiones cruzadas retomando el tradicional [véase] que remite tanto a referentes como a los ejemplos de código¹². En estos casos cabe el juego de seguir las indicaciones de páginas para observar diferentes maneras de pensar un algoritmo.

Por último señalar de nuevo que, en este contexto, el proceso toma mayor importancia que el objeto acabado, y, aunque ahora tengas ante los ojos este libro materializado, o digitalizado, con una forma concreta, es muy posible que nuestros ordenadores estén tramando ya otra versión de texto y software que esperamos amplíe, limpie y mejore esta.

Iterabilidad, inevitable cualidad de lo digital que como Sísifo remonta una y otra vez su carga.

⁹ Lev Manovich, "The Death of Computer Art" [texto on-line] [12.05.2012] <http://www.manovich.net/TEXT/death.html>.

¹⁰ En el sentido definido por Heidegger, (1994) "La pregunta por la técnica" en *Conferencias y artículos*, Barcelona: Ediciones del Serbal.

¹¹ Safaa Fathy, (1999) *D'Ailleurs, Derrida*. Gloria Films [Documental DVD]

Y Derrida continúa diciendo: "yo sé que cuando eso esté escrito, y formulado en un idioma, y por lo tanto legible, cuando la huella sea descifrable, perderá la unicidad del destinatario, de la destinataria".

¹² Los ejemplos de código están numerados según E-NumCapítulo-orden y están disponibles en: https://github.com/d3cod3/GAMUZA-Live_Creative_Coding-Book/blob/master/GAMUZA_LCC_Examples.zip

2. ¿Qué es GAmuza?

La idea original que impulsó GAmuza era hacer más fácil la programación creativa; entonces saltábamos habitualmente de Processing¹³ a openFrameworks (OF)¹⁴, y el deseo era tener OF con un entorno de programación (IDE) como el de Processing, más un montón de módulos con interfaz gráfica (GUI) para facilitar los pasos programación más técnica como Tracking video, Análisis de audio, comunicación con Arduino, etc...

Por ello GAmuza es un entorno de scripting hecho con OF, inspirado por Processing, uniendo el lenguaje de OF 0.8.4 incluidos sus addon oficiales y algunos no oficiales, todo el lenguaje OpenGL 1.1, y un pequeño framework propio de GAmuza con funciones y módulos GUI para hacer la programación creativa mucho más fácil, todo ello embebido en un entorno de programación Lua ligeramente modificado.

Está pensado especialmente para la enseñanza de arte interactivo y también para la realización de performances audiovisuales en directo. La versión a la que hace referencia este texto es la 1.0.1 para Mac OSX 10.7 o superior, si bien hay una versión anterior (0339) desarrollada para Linux Ubuntu x86 de 32 o 64 bits. Es *open source*, se distribuye bajo licencia MIT y puede descargarse en: <http://gamuza.d3cod3.org/downloads/>

GUI versus Code

En el campo de la enseñanza del arte digital es común ver posiciones que defienden el uso de software libre, pero entre ellos se plantea también otro dilema: utilizar programación gráfica, como Pure Data, o código, línea de comandos, como Processing.

Hay un complicado entrampado de relaciones entre ambas posturas que a veces parecen responder a argumentos de índole casi publicitario: gráficos-amigables versus entornos de programación-hostil. El adjetivo tras el guión indica el tono y posicionamiento del discurso. Ante ese debate GAmuza opta por la hibridación.

Las interfaces gráficas de usuario (GUI) de los módulos y paneles de GAmuza están diseñadas para facilitar procesos de comunicación con dispositivos de audio, vídeo o Arduino, pero no para ocultar conceptos o estructuras del código, por eso incorpora un IDE de programación, estableciendo una situación híbrida que aúna GUI y Code. Esa mayor facilidad en la comunicación del sistema con los dispositivos se ha establecido para que estudiantes, artistas, o gente interesada no experta, pierdan el miedo a algo que comúnmente se califica como demasiado complicado. Pero cuando se comprenden las cosas ya no parecen tan complicadas.

13 Processing <<https://www.processing.org/>> [12.05.2012]

14 openFrameworks <<http://www.openframeworks.cc/>> [12.05.2012]

Por eso...

Tecnología para la gente

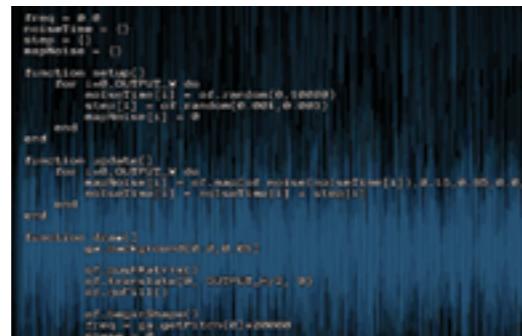
...GAMUZA no oculta los conceptos o estructuras. Comprender el funcionamiento de los dispositivos que usamos habitualmente es el primer paso para no ser esclavo de la tecnología.

Se ha hecho popular publicar ideas personales, intereses, conversaciones, gustos, opiniones, fotos, videos... Todo el mundo está en línea, desde su portátil o smartphone, y en 5 minutos te hacen sentir que eres el webmaster de todas tus ficciones o vidas virtuales diferentes, constantemente conectado con todos tus amigos virtuales; y lo puedes hacer sin conocer la tecnología que estás utilizando. Esto implica muchas cosas, una de ellas es "gente para la tecnología", o en otras palabras, los medios están preparando "gente a la que vender tecnología". GAMUZA es sólo un software, pero creado con un concepto en mente: "Tecnología para la gente", no lo contrario.

GAMUZA se inició como un software de Live coding para programar en directo, un término vinculado tanto a los entornos de programación con un tiempo de compilación *infralive* —'just in time programming' o 'real time'—, como a la realización en directo de audio y vídeo donde la presencia del código como imagen, o sobre la imagen, es uno de los rasgos que lo caracterizan, enfatizando con ello el papel del código como proceso de pensamiento.

Los algoritmos son pensamientos, las motosierras son herramientas¹⁵.

Por eso en las primeras versiones de GAMUZA, hasta build 0.398, el código se escribía directamente en el módulo de Live Coding visualizándose sobre la imagen generada, y aun es así en GAMUZA para Linux. Desde la versión 043x para Mac OSX, el código ya no se muestra en la ventana de salida porque cuenta con un editor independiente para facilitar su uso en la enseñanza de programación.



Ahora, el núcleo central de GAMUZA es el entorno de desarrollo integrado (IDE, Integrated Development Environment), semejante al de Processing, que proporciona un marco de programación muy simplificado porque el lenguaje de script de GAMUZA se basa en LUA¹⁶, que está integrado en OF con una versión modificada del addon ofxLua; una versión parcheada de LUA 5.1 está compilada como biblioteca estática, y el wrapper de OF 0.8.4 junto todos los ofxAddons añadidos, se realiza

¹⁵ Stephen Ramsay, *Algorithms are Thoughts, Chainsaws are Tools*. [video on-line] <<https://vimeo.com/9790850>> [04.09.2012]

¹⁶ The programming language Lua <<http://www.lua.org/>> [11.05.2012]

a través de LUABIND¹⁷, mientras que la unión de OpenGL 1.1 se codifica directamente desde LUA utilizando luaglut. Todo esto se materializa en un entorno de scripting híbrido que permite visualizar las modificaciones del código de forma casi inmediata, sin necesidad de compilar.

Sistema modular y paneles

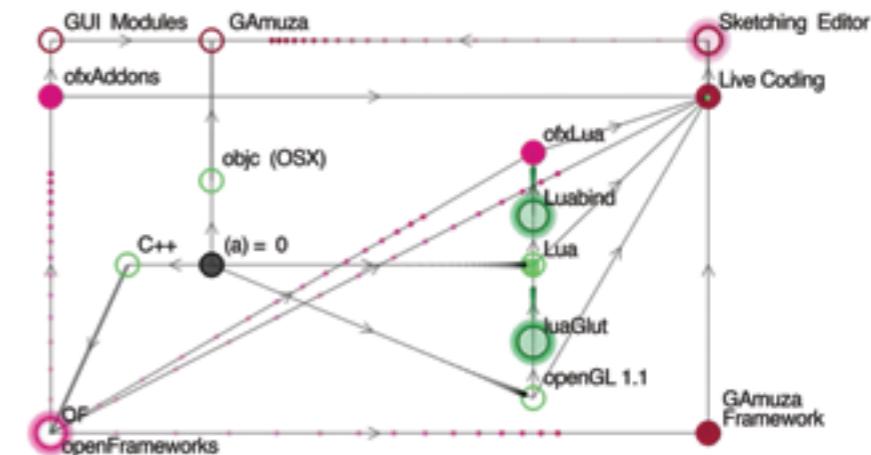
GAMUZA contiene distintos módulos y paneles precodificados que son fácilmente configurables a través de GUI. Los módulos están siempre activos y los paneles se pueden activar, o no, para reducir consumo del equipo. Actualmente los módulos son:

- Timeline
- Audio Analysis (input)
- Arduino
- Mapper

Y los paneles:

- Computer vision
- Sensor Kinect

En resumen, GAMUZA es un software que recoge y coordina de forma particular lenguajes (Lua y C++), plataformas ya existentes (openFrameworks, openCV), e incorpora otras librerías propias, para facilitar los primeros pasos de estudiantes y artistas en el campo de la programación creativa, combinando las dos vías habituales de programación: consola de código textual (IDE) vinculada a las aplicaciones modulares que se ajustan mediante GUI. Retomando un término de Armando Montesinos, GAMUZA más que un collage de lenguajes y plataformas, reconstruye un entorno de programación por medio de "zurcidos", utiliza parches y genera tejido nuevo entrecruzando librerías, sin ocultar los fragmentos. El siguiente esquema permite visualizar, las relaciones y niveles de esas plataformas y lenguajes.



Esquema de la distribución del código fuente de GAMUZA, representado como una red neuronal

¹⁷ Luabind es una librería que ayuda a crear enlaces entre C++ y Lua. Ver <<http://www.rasterbar.com/products/luabind.html>> [12.05.2012]

Instalación

Como se ha mencionado, GAMUZA está programado para funcionar en mac OSX y Ubuntu Linux x86 de 32 y 64 bits, y hay distintas versiones que tienen requisitos de sistema, procesos de instalación e interfaces diferentes, en este texto vamos a hacer referencia a la versión GAMUZA 0545, para versiones anteriores ver nota al pie¹⁸.

Para instalar GAMUZA 0545, desde <<http://gamuza.d3cod3.org/downloads/>>, descargad la aplicación, abrir (descomprimir) el archivo .zip y arrastrar el ícono de GAMUZA a la carpeta Aplicaciones, y el proceso de instalación ha terminado.

GAMUZA Learning

GAMUZA es también una plataforma on-line de aprendizaje, articulada en cuatro secciones:

Workshops organizados por días de trabajo, con contenidos, sugerencias, exámenes tipo test y ejercicios. <<http://gamuza.d3cod3.org/learning/workshops/>>

- Introduction to GAMUZA
- GAMUZA for Creative Coders
- Prototyping Arduino projects with GAMUZA
- Live audio-visual performances with GAMUZA
- Net data visualization with GAMUZA

Tutoriales, <<http://gamuza.d3cod3.org/learning/tutorials/>>

Ejemplos, organizados por tipologías <<http://gamuza.d3cod3.org/learning/examples/>>

Code - Haiku, es una convocatoria cuatrimestral de "poesía a través de la programación, concebida como un haiku de «formato libre»". Las fechas de la convocatoria se corresponden con los solsticios.

El tema se anuncia un día antes, y hay 24 horas para presentarlo <<http://gamuza.d3cod3.org/learning/code-haiku/>>

¹⁸ Para versiones 0420 OSX 10.6 y 0399 - Ubuntu, ver el archivo http://mpison.webs.upv.es/mie1/text/GAMUZA_0420_manual.pdf
Para la versión 0432 OSX, ver el archivo http://mpison.webs.upv.es/mie1/text/GAMUZA0432_manual.pdf

2.1. Configuración Preferencias

GAMUZA tiene el ajuste de configuración del proyecto en el menú **Preferences**. Para activarlo, clicar sobre el nombre del programa situado en la esquina superior izquierda de la pantalla, se despliega un menú cuyo segundo ítem es **Preferences** (también se puede activar directamente tecleando Comando + coma #,)

La ventana **GA Preferences** tiene 7 pestañas para configurar.

Video Output (Ventana de salida) con las siguientes opciones:

Video Texture Resolution. Es un menú desplegable para seleccionar la resolución de la ventana de salida, se debe elegir según la resolución de los monitores o proyectores que se utilicen como segunda pantalla; las últimas opciones hacen referencia al uso de tarjetas que duplican o triplican el tamaño de la ventana de salida.

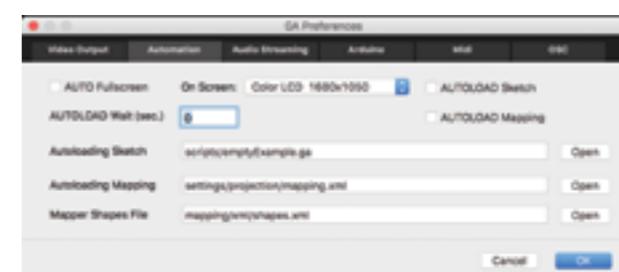


El segundo ítem está relacionado con el anterior y permite poner manualmente la resolución.

El tercer ítem es **Mapping Gird Resolution**, GAMUZA genera una rejilla sobre la imagen de salida para proyecciones de mapping básico. Con este slider se puede elegir el número de nodos de esa rejilla, desde un mínimo de 1 (rectángulo de la proyección tiene un punto de ajuste activo en cada vértice con los que se puede deformar la proyección según las necesidades), a un máximo de 20 (la imagen tiene 400 rectángulos ajustables). En la descripción de la Barra de Menús se señalarán más datos sobre el funcionamiento de esta rejilla.

Automation:

Opción pensada para la presentación de instalaciones. Permite seleccionar un archivo de script previamente guardado, "Autoloading Sketch", abriendo una ventana de diálogo para elegirlo, también si hay un archivo guardado con una configuración del mapping básico



con "Autoloading Mapping", o del Módulo Mapping "Mapper Shapes File". Si se activa AUTO fullscreen, al abrirse el programa, la ventana de salida se pone directamente a pantalla completa, y si se activa AUTOLOAD Sketch, GAMUZA se inicia procesando el archivo de script seleccionado. Igualmente, si se activa AUTOLOAD Mapping, la imagen mostrará las deformaciones que contenga el archivo de configuración de mapping seleccionado. La opción AUTOLOAD Wait (sec.) permite demorar el autoarranque del archivo seleccionado los segundos que se hayan introducido; esta opción es importante cuando el ordenador, al arrancarse automáticamente, debe reconocer las pantallas o proyectores conectados antes de abrir el archivo.

Normalmente las exposiciones duran varios días y mantener técnicamente las piezas de arte interactivo puede ser un problema, si se conjuga la configuración de Automation con las opciones que hay en las Preferencias del Sistema del ordenador para arrancar y apagar el equipo en determinados días y horas, y que GAMUZA se abra automáticamente al iniciar la sesión, la pieza requerirá un mantenimiento mínimo durante la exposición.

Audio Streaming

Input Device, un menú desplegable con los dispositivos de entrada de audio que reconoce el ordenador, en él hay que seleccionar con cuál trabajará GAMUZA.

Output Device, semejante al anterior, hay que seleccionar el dispositivo de salida de audio.



En la opción **Sampling Rate** (Frecuencia de muestreo) hay que seleccionar un número que sea acorde con el dispositivo de entrada de audio. Se puede ver las frecuencias que soporta ese dispositivo en la ventana Tools/Logger que se describe en el capítulo siguiente.

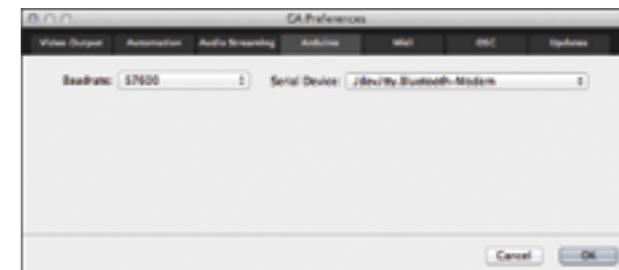
Buffer Size, contiene distintos tamaños de Buffer (cantidad de muestras almacenadas) para seleccionar con cuál de ellos se va a trabajar, 512 es un tamaño estándar válido para la mayoría de los proyectos, pero si se va a trabajar con Bins o FFT tal vez requiera un tamaño mayor. Hay que tener en cuenta que a mayor buffer más recursos de procesador consume.

FFT Windowing (Ventanas de función FFT). Las ventanas son funciones matemáticas que se usan para el análisis y procesamiento de la señal de audio porque evitan las discontinuidades al principio y final de los bloques analizados. En el menú desplegable se puede seleccionar entre: Rectangular, Bartlett, Hann, Hanning o Sine.¹⁹

19 Para conocer las características de estas opciones ver <[http://es.wikipedia.org/wiki/Ventana_\(funcion\)](http://es.wikipedia.org/wiki/Ventana_(funcion))> [25.07.2013]

Arduino

Baudrate (velocidad de transmisión) y Serial Port (puerto serie). Puede utilizarse el software de Arduino para comprobar mediante el Serial Monitor cuál es el puerto asignado y el Baud Rate, para seleccionar en GAMUZA esos mismos datos.



Midi

Aunque no existe un módulo GUI de MIDI, GAMUZA puede recoger datos de estos dispositivos y hay funciones específicas para comunicar con ellos, para ello hay que seleccionar en Preferences el dispositivo



OSC

Contiene los ajustes para el protocolo de transferencia de datos OSC, (Open Sound Control)²⁰. Los datos se introducen directamente en los campos de texto. Los dos primeros **SENDING TO IP** y **PORT** envían datos a un ordenador con el número de IP y puerto que se pongan. El último, **RECEIVING AT PORT** indica el puerto de este ordenador que recibirá los datos enviados desde otro.

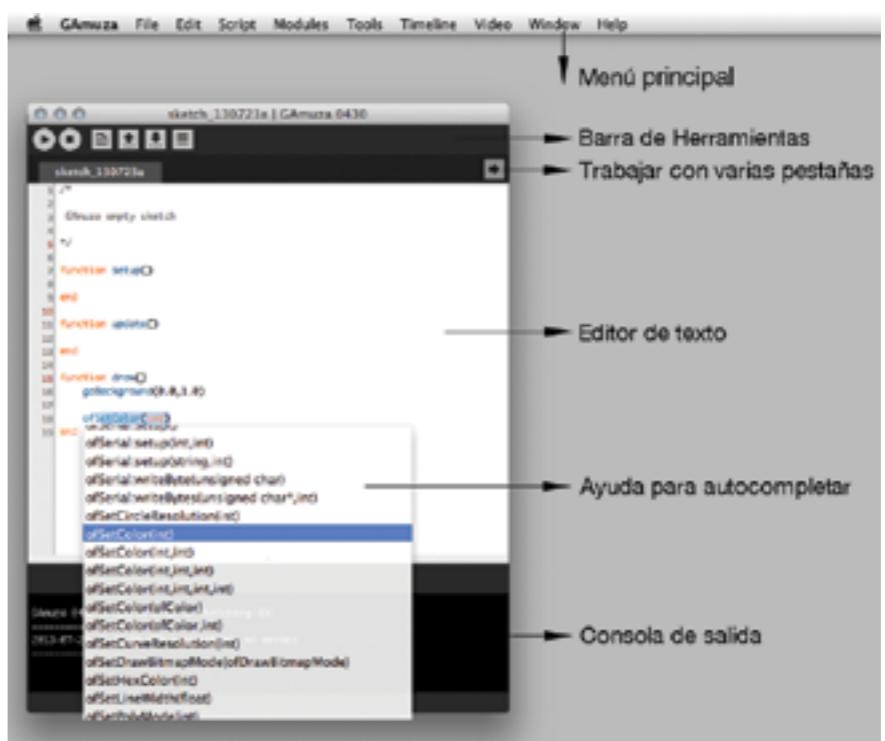


3. Descripción de la interface

3.1. IDE de programación

La IDE (Integrated Development Environment) es un entorno de programación empaquetado como aplicación para facilitar la escritura del código. Al abrir GAmuza.app, aparece un archivo de sketch en el IDE, con un modelo de bloques de programación básico preconfigurado, y la ventana de salida con las dimensiones que tenga asignadas en **Preferences**. Cuando se trabaja con el ordenador conectado a un proyector o a un segundo monitor, esta ventana de salida se arrastra hasta la segunda pantalla, y con las teclas **⌘F** se activa/desactiva la opción pantalla completa.

La interface del IDE de programación sigue directamente el modelo marcado por el PDE Processing, como epígonos de una de sus influencias directas. Principalmente aporta un editor de texto para el código que en su parte superior dispone una barra de herramientas y en su parte inferior una consola de salida. Tiene también un sistema de ayuda para autocompletar el nombre de las funciones (Code Completion) que se activa presionando la tecla Esc después de iniciar el nombre de la función, en el caso de los métodos para las clases hay que escribir el nombre de la clase, después teclear Esc, y aparecen todos los métodos de esa clase.



En la barra de herramientas hay botones que facilitan las labores más usuales: enviar a GAMUZA; limpiar script; nuevo script; abrir script; guardar como; y limpiar la consola de salida.

El editor de texto tiene una columna numerada a la izquierda para facilitar la localización de errores.

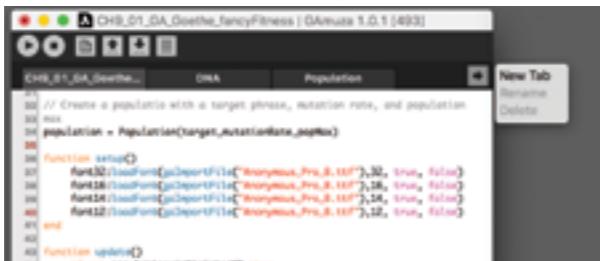
También asigna una serie de colores a términos que reconoce, vinculados a:

- Funciones, clases, métodos y constantes:
- Estructuras programación:
- openGL, funciones y constantes:
- Strings (cadena caracteres entrecomillados)
- Comentarios:
- Y todo lo demás.

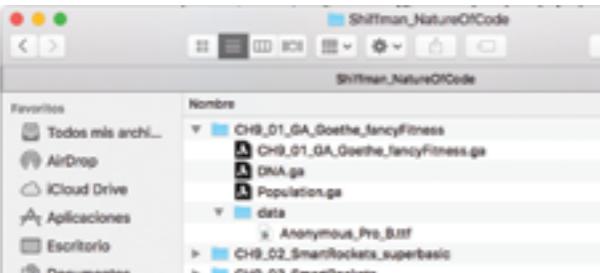
```
ofSetColor(255)
function
glBegin(GL_QUADS)
"texto entrecomillado"
// comentario
x = 125
```

Al guardar el archivo por primera vez, Gamuza genera una carpeta con el mismo nombre que se le dé al script y guarda junto a él otra carpeta denominada **data**, en esta carpeta se deben poner todos los medios que utilicemos: fuentes de texto, videos, audio, imágenes...

El botón con el icono de flecha, situado en la parte superior derecha del IDE, sirve para incorporar, renombrar o borrar, archivos de script vinculados al principal. Estos archivos se presentan como diferentes pestañas en el IDE y son guardados como archivos independientes en su misma carpeta.

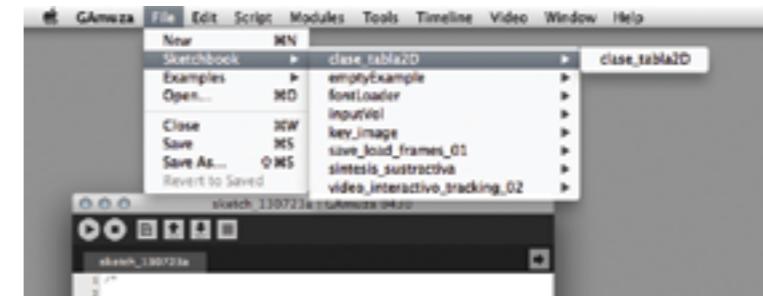


La siguiente imagen muestra la relación de archivos y carpetas de un script con tres pestañas y la carpeta **data** que contiene un archivo de fuentes.

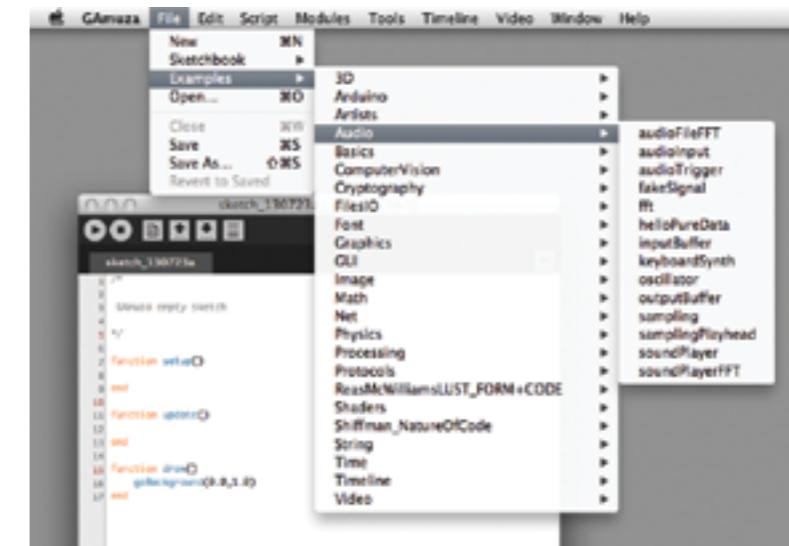


Los tres archivos de script aparecen con la extensión .ga; si el archivo principal se vuelve a guardar, se actualiza todo directamente, pero si se da a "guardar como" y se aloja en otro sitio, hay que volver a copiar el archivo de fuentes en la nueva carpeta **data** que se genere.

3.1.1. Menú



En el primer menú, **File**, se accede a las operaciones habituales para abrir un archivo de script nuevo (**New**) o que ha sido previamente guardado (**Open**). Entre ambas opciones se encuentran dos ítems que tienen un carácter diferente; el primero, **Sketchbook**, da acceso directamente a la carpeta GAMUZA situada en Documentos. El segundo, **Examples**, contiene numerosos ejemplos que vienen con el programa, ordenados por categorías. Todos están comentados mostrando el uso del código y responden a una amplia tipología de prácticas.



Los tres siguientes ítems del menú **File** corresponden a las tareas usuales de cerrar (**Close**), guardar (**Save**) y guardar como (**Save As**). El último, recupera la versión anterior guardada de ese script (**Revert to Saved**). Junto a los ítems del menú se muestra el atajo de teclado para poder hacer esa tarea sin tener que acceder al menú.

El contenido del menú desplegable **Edit** es el similar al de la mayoría de los software: **Undo** (⌘Z, deshacer), **Redo** (⇧⌘Z, redo), **Cut** (⌘X, cortar), **Copy** (⌘C, copiar), **Paste** (⌘V, pegar), **Select All** (⌘A seleccionar todo), **Find** (⌘F, encontrar y reemplazar), y por último **Special Characters** (caracteres especiales) que abre el Visor de caracteres de Mac.

En el menú **Script**, la opción **Compile Script** (⌘K) equivale al botón play (Send to GAmuza) de la barra de herramientas del IDE, y **Clear Script** al segundo botón, stop. El tercer ítem **Clear Console** equivale al último de los botones y borra el contenido de la consola de salida.

El primer ítem del menú **Modules** es **Preview** (⇧⌘0) y activa/desactiva la ventana del previo de imagen. Los siguientes ítems corresponden a los módulos GUI del **TimeLine** (⇧⌘1), **Análisis de audio** (⇧⌘2), **Arduino** (⇧⌘5) y **Mapper** (⇧⌘6) que se analizarán con detalle en los capítulos 8, 9, 11 y 12.

El menú **Tools** da acceso a las herramientas de GAmuza:

Color Correction (⇧⌘K) cuenta con tres bloques de efectos preprogramados para ajustar, mediante sliders, la apariencia de color de la imagen en la ventana de salida:

- **Gamma correction**: ajusta la "sensación" de contraste de la imagen si se percibe blanquecina o "lavada".
- **Brightness**: ajusta el brillo de la imagen.
- **Saturation**: satura o desatura el color de la imagen.
- **Contrast**: ajusta el contraste.
- **Film Bleach**: emula una técnica de revelado analógico que produce contraste alto y saturación baja. También llamado Bleach Bypass.
- **Film 80's Technicolor**: emula el rango de color de los TV de tubo catódico de los años 80's.
- **Force Black&White**: Estos tres últimos filtros están relacionados y el ajuste de uno repercute en los otros. Este aporta otra manera más flexible para desaturar.
- **White Exposure**: ajusta la sensación de luz.
- **White Diffusion**: emula el uso de filtros de gelatina White Diffusion para suavizar sombras.

Color Selector (⇧⌘C), ayuda a conocer los parámetros de un determinado color en los modelos RGB y HSB.

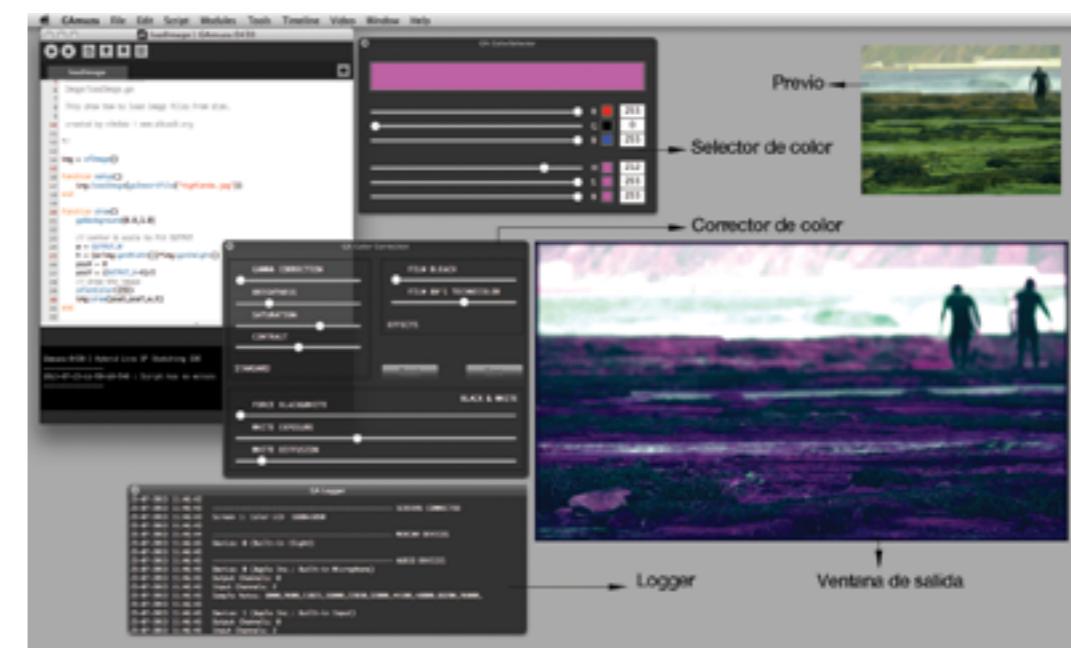
Grid Viewer, permite incorporar una rejilla a la ventana de salida, mostrando distintas opciones: Rejillas predefinidas (Proporción áurea, Ejes centrales, Tercios) o definiendo el ancho y alto de la rejilla en píxeles.

Logger, abre una ventana que muestra todos los dispositivos que reconoce el ordenador: pantallas, cámaras, tarjetas de sonido, puertos serial, Arduino, conexión OSC, así como si el Pure Data Synthesis

Engine está iniciado, el listado de plugins de audio (Audio Unit) disponibles en el sistema, y si la tarjeta gráfica del equipo soporta el uso de shaders.

Archive sketch, crea un archivo comprimido .zip con el script y todos sus contenidos.

Por último, **Export to HTML**, recoge el script con todos los ficheros utilizados, un par de pantallazos de la ventana de salida, una copia del setting de los módulos, y además crea un archivo .html con una página de documentación del script automáticamente maquetada, actuando como un sistema de documentación automática del trabajo.



Los ítems que contiene el menú **Timeline** son los habituales cortar, copiar, pegar, pero el atajo de teclado varía, en lugar de la tecla comando ⌘, utiliza Control ⌃. Su uso se describirá en el capítulo dedicado al Timeline.

En el menú **Video** aparecen opciones para:

Basic Grid Mapping. **Edit Mapping** (⇧⌘E), para mostrar/ocultar la rejilla de mapping sobre la ventana de salida. Las opciones a seleccionar son:

Reset Mapping, Deja la rejilla ortogonal eliminando todas las deformaciones (⇧⌘R).

Keyboard Control (⇧⌘M), pasa el control de ajuste de los nodos de la rejilla del ratón al teclado y viceversa.

Manual Edit Point ON (**↑⌘←**) Activa un nodo para poder moverlo.

Manual Edit Point OFF (**↑⌘↖**) Desactiva el nodo para poder pasar a otro.

Go to North Point (**⌘↑**) Si el nodo está activo, lo desplaza hacia arriba, si se ha desactivado pasa el control al nodo superior.

Go to South Point (**⌘↓**) Si el nodo está activo, lo desplaza hacia abajo, si se ha desactivado pasa el control al nodo inferior.

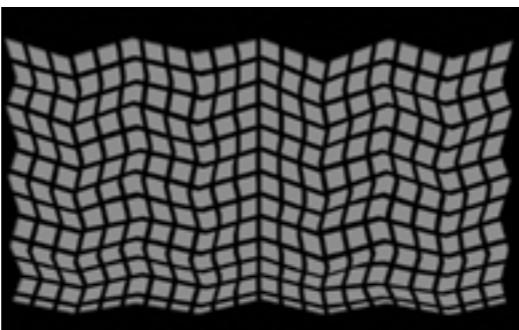
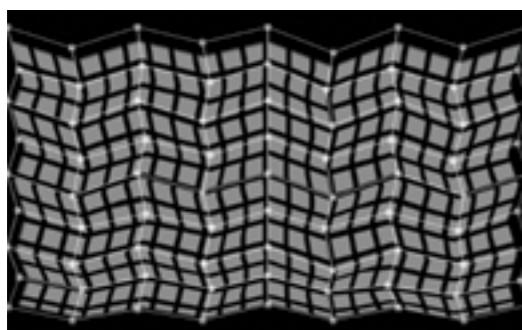
Go to East Point (**⌘→**) Si el nodo está activo, lo desplaza hacia la derecha, si se ha desactivado pasa el control al nodo de la derecha.

Go to West Point (**⌘←**) Si el nodo está activo, lo desplaza hacia la izquierda, si se ha desactivado pasa el control al nodo de la izquierda.

Load Mapping. Para cargar una configuración de mapping previamente guardada.

Save Mapping. Para guardar la configuración actual.

Los ajustes de mapping se realizan proyectando sobre una superficie irregular, y la ventana de salida está en el proyector, pudiendo quedar algunas zonas con poca visibilidad desde el punto de vista en que se encuentre el ordenador; para estos casos es importante tener el control de los nodos con el teclado, lo que permitirá ver cómo se ajusta la deformación de la imagen a las superficies aunque el nodo de esa zona esté en un lugar poco visible para clicar con el ratón.



Mapper Module. El Módulo Mapper tiene opciones mucho más flexibles para mapear que Basic Grid Mapping y se describe en el capítulo 12. Las opciones de este menú son:

Activar/desactivar los ajustes realizados. **Resetear.** Cargar (load) una configuración de Mapper previamente guardada. **Guardar** la configuración actual.

Preview. Para seleccionar el tamaño de la ventana del previo. Las opciones son: QVGA (320 x 240), nHD (640 x 360), VGA (640 x 480), SVGA (800 x 600) y FWVGA (854 x 480)

Toggle Full Screen (**⌘F**). Para activar/desactivar la ventana de salida a pantalla completa.

Toggle Show/Hide Cursor (**⌘P**). Para mostrar u ocultar el cursor.

4. Lenguaje de scripting

El lenguaje de scripting de GAMUZA está basado en LUA, se integra openFrameworks a través de una versión modificada del addon ofxLua. Una versión parcheada de LUA 5.1 se ha compilado como librería estática, y la unión de openFrameworks 0.8.4 con todos los ofxAddons se realiza a través LUABIND²¹, mientras que la unión con OpenGL 1.1 se codifica directamente desde LUA utilizando luaglut²².

Todo este entramado de lenguajes se ha realizado tras observar la dificultad de muchos estudiantes de artes visuales que, aun deseando utilizar código de programación para sus prácticas, les resulta difícil. Somos conscientes de que programar no implica solo recordar los detalles de la sintaxis, sino que se necesita desarrollar una lógica particular junto a la capacidad de análisis y comprensión de conceptos abstractos complejos.

Para enseñar a programar con este lenguaje nos situamos en un proceso intermedio que empieza por diseccionar y comprender el código, para que posteriormente se puedan relacionar esas unidades en estructuras cada vez más complejas, hasta llegar a escribir programas propios. Para ello se clarifican métodos de análisis que traduzcan el problema en estructuras significativas secuenciales que ayuden a diseñar la estructura de código.

[...] los expertos no sólo tienen más información, sino que la tienen mejor organizada. En lugar de percibir y recordar piezas individuales de información, procesan la información como grupos significativos, por lo que su percepción es más eficiente y su capacidad de recuerdo tiene un rendimiento mucho más alto.²³

Programar es diseñar algoritmos, lo que implica pensar en objetos abstractos que se planifican de forma lógica para generar procesos, o comportamientos, mediante una serie de instrucciones consecutivas. Las instrucciones tienen que ser entendidas por el sistema, por lo que no pueden ser ambiguas. De ahí que se aconseje hacer previamente un esquema mediante diagramas de flujo, o de otro tipo, siguiendo un lenguaje "pseudohumano", enlazando las instrucciones en secuencias que sigan un orden, como los que aparecen junto a algunos de los ejemplos a lo largo de este texto. Lo que implica idear y pensar de forma abstracta siguiendo estructuras lógicas.

En el campo del arte estamos más acostumbrados a pensar en imágenes o estructuras para expresar una idea que en algoritmos y lógica, por eso los software de código creativo están orientados a visualizar las formas o imágenes en pantalla rápidamente, utilizando un lenguaje de programación muy simplificado.

21 luabind · GitHub, <<https://github.com/luabind/luabind>> [29.12.2014]

22 LuaDist/luaglut · GitHub, <<https://github.com/LuaDist/luaglut>> [29.12.2014]

23 Katherine B. McKeethen et al., (1981) "Knowledge Organization and Skill Differences in Computer Programmers" *Cognitive Psychology* 13(3), pág. 307. [Texto on-line] <<http://spider.sci.brooklyn.cuny.edu/~kopec/research/sdarticle11.pdf>> [12.05.2014].

El proceso de trabajo en un proyecto de arte que requiera programación creativa sigue inicialmente pasos semejantes a cualquier proyecto artístico:

- Analizar qué es lo que se quiere expresar y cuáles son los recursos que mejor lo pueden reflejar.
- Planificar las distintas partes en que se pueda dividir ese proceso con una secuencia de etapas más simples. Hacer esa planificación en un bosquejo sobre papel ayuda a ver las relaciones y orden de esos pasos. El dibujo es el mejor y más rápido aliado del pensamiento y ese esquema o bosquejo permite ver la finalidad de cada una de esas etapas simples, qué datos necesita de entrada y cuáles produce como salida.

Un buen bosquejo en papel del software debe incluir una serie de imágenes que muestren cómo funciona la estructura narrativa de la pieza, de forma semejante al guión gráfico de una animación. Además de las imágenes que genera en pantalla, los bocetos a menudo contienen diagramas de flujo del programa, los datos y anotaciones para mostrar cómo las formas se mueven e interactúan. Los programas también pueden planificarse utilizando combinaciones de imágenes, esquemas formales y descripciones de texto²⁴.

- Escribir cada una de esas fases en archivos separados y comprobar que funcionan, corrigiendo errores si los hubiera.
- Combinar los distintos archivos comprobando que no hay errores de lógica entre ellos, ni cálculos innecesarios y que los datos de salida son los que se buscaban.
- Documentar el código con comentarios, hacer una breve explicación al inicio del planteamiento y en los argumentos de entrada y salida.

Para comprender el sentido de estos procesos es muy importante visualizar referencias que muestren dónde y cómo se aplican o se han aplicado procedimientos semejantes en resultados concretos. Lo que en el campo del arte entendemos como referentes artísticos, o contextualización referencial.

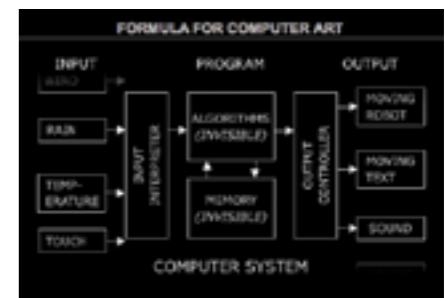
Aun conscientes de esta necesidad referencial, para iniciar el aprendizaje hay algunas estructuras, funciones y expresiones básicas que deben conocerse en su conjunto antes de trabajar con ellas, por eso se recomienda alternar la lectura de este capítulo con los siguientes para comprender su aplicación gráfica. Tras pasar estos peldaños, entraremos ya en las cuestiones de interactividad con el uso de los módulos y paneles GUI que aporta GAmuza, con ellos y la base adquirida de programación se pueden plantear y resolver proyectos de arte digital e interactivo diferentes.

²⁴ Casey Reas y Ben Fry, (2007) *Processing, a Programming Handbook for Visual Designers and Artist*. Cambridge (MA): The MIT Press., pág. 145.

4.1. Elementos básicos de programación: datos, variables y funciones

Un dato por sí mismo no constituye información, es el procesamiento de los datos lo que nos proporciona información²⁵.

Como anticipo nos detendremos en la Fórmula giratoria del *Computer art* que propone Jim Campbell²⁶, con elementos de entrada (input), procesamiento, y salida (output). Es común que las piezas de arte digital recojan datos del entorno, ya sean del contexto físico exterior, del propio planteamiento del autor, o de la interacción del espectador. Estos datos entran al ordenador (input) mediante un interprete (sensores, teclado, ratón...) son procesados de forma invisible (algoritmo, código) y producen una respuesta sensitiva (output).



DATOS (INPUT) → PROCESAMIENTO → (OUTPUT) INFORMACIÓN

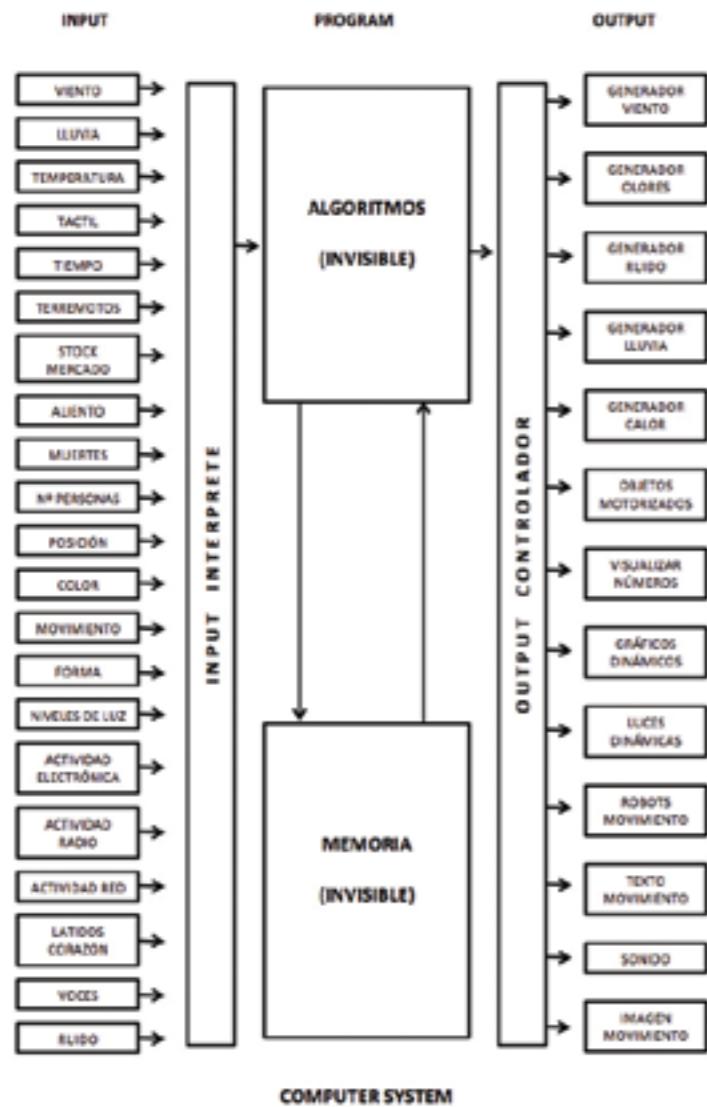
Siguiendo esta fórmula, los datos de entrada al sistema pueden estar relacionados con el viento, lluvia, temperatura, tiempo, terremotos, stocks de mercado, niveles de luz..., pero más allá del origen de los datos, es necesario saber cuál es su naturaleza. Así, los datos del viento pueden ser numéricos (velocidad km/h), letras (direcciones N, S, E, O), palabras (nombres locales de los vientos), imágenes (fotografías, gráficos, etc) o vectores (dirección o intensidad)

La base del software es el procesamiento de datos y necesita conocer su naturaleza porque la cantidad de memoria que debe reservar es muy diferente para un número o para una cadena de caracteres y mucha más para una imagen o para una gran cantidad de números organizados en matrices. La memoria retiene la información de los datos para que puedan ser utilizados repetidas veces. Este espacio reservado de la memoria está vinculado con las **Variables**.

En la página siguiente se muestra un gráfico que despliega sobre el papel la fórmula *Computer art* de Jim Campbell, fijando su movimiento.

²⁵ Definición de Dato, [enciclopedia on-line] <<http://es.wikipedia.org/wiki/Dato>> [28.06.2012]

²⁶ "Formula for Computer art", <http://www.jimcampbell.tv/portfolio/miscellaneous_references/> [28.06.2012]



4.1.1. Variables

Una variable es como un contenedor para almacenar datos que después pueden ser reutilizados muchas veces en el programa y cuyo valor puede cambiar. El sistema necesita reservar determinada cantidad de bits de memoria para que esa variable esté siempre disponible. Para comprender cómo se estructuran las variables en programación básica —cómo se identifican y diferencian los datos—, vamos a comparar su uso en GAMUZA y Processing, por ser uno de los software más utilizados y documentados en este tipo de aplicación creativa. En Processing, cada variable tiene tres partes:

- identificación del tipo de datos que guarda (`int`, `float`, `boolean`,...)
- nombre (el que decida el programador)
- valor (se asigna un valor inicial)

Processing necesita hacer constar esa identificación, distinguiendo entre:

```
int      // Número entero
float    // Número con coma flotante
boolean  // Un dato que sólo puede ser verdadero o falso.
PImage   // Imagen
char     // Carácter. El valor se define con apostrofes: 'a'
string   // Cadena de caracteres. Se define con comillas "antes"
PFont   // Fuentes tipográficas
```

En GAMUZA (en Lua) a las variables no se les asigna el tipo, porque es un lenguaje "dinámicamente tipado", si bien el programador debe ser consciente del tipo de datos que alberga esa variable. Siempre hay que declarar la variable antes de utilizarla, en GAMUZA además hay que asignarle un valor que en el proceso del programa puede cambiar; en Processing es opcional, puede declararse y posteriormente darle un valor, como muestra la segunda variable en el siguiente ejemplo:

Processing <code>int miTiempo = 50;</code> <code>float y;</code> <code>boolean b = true;</code> <code>PImage myImage;</code>	GAMUZA <code>miTiempo = 50</code> <code>y = 0.0</code> <code>b = true</code> <code>myImage = ofImage()</code>
--	---

Cada programador decide cómo nombrar las variables, pero algunos hábitos han generado una cuasi-norma:

- Se recomienda poner a las variables un nombre relacionado con su contenido, para que sea más fácil leer la programación y se reduzca la necesidad de comentarios.
- Los nombres de las variables suelen comenzar con una letra minúscula, y si hay varias palabras en el nombre, la primera letra de cada palabra adicional está en mayúscula (como `miTiempo`)

Hay también algunas normas para nombrar variables:

- El nombre de una variable no puede empezar con números
- No puede coincidir con los términos propios del lenguaje de programación.

Palabras reservadas en GAMUZA (Lua)²⁷ son:

```
and      break      do      else      elseif      end      false
for      function    if       in       local      nil      not
or       repeat     return   then    true      until   while
```

Para evitar confusiones, los nombres de las variables tampoco deberían coincidir con otros nombres que reconoce el sistema como `angle` o `int`. Si al escribir una variable se colorea, siempre puede readaptarse para mantener su sentido, como por ejemplo: `_angle`

En GAMUZA, para modificar el tipo de dato de una variable se pueden utilizar las siguientes funciones de Lua o de openFrameworks:

`toString()` el argumento que acoge entre los paréntesis puede ser de cualquier tipo de datos y lo convierte en un string (cadena de caracteres) con un formato razonable.

Para un mayor control en la conversión de datos numéricos a string es aconsejable usar `string.format()` dado que incorpora códigos para especificar el tipo de dato, los principales códigos son:

```
string.format("%s", 100)           // %s devuelve un string
100
string.format("%c", 76)           // %c devuelve un carácter, char
L
string.format("%f", math.pi)       // %f devuelve un float como string
3.141593
string.format("%i,%i",-100,100)   // %i número entero con su signo como string
-100, 100
```

Otras funciones útiles de openFrameworks para convertir otros tipos de dato son²⁸:

```
string.byte('L') //devuelve el código numérico de los caracteres
76
ofToFloat("3.14") // Convierte un string (float-string) en un valor float
3.14
ofToInt("100") // Convierte un string (int-string) en un valor int
100
ofToBool("true") //Convierte un string (bool-string) en un booleano real
true
```

Otra cuestión importante en relación a las variables es el alcance o ámbito de la variable, es decir, dónde puede ser utilizada en relación a dónde se ha creado o declarado, porque ello determina su ámbito de aplicación —dónde se puede acceder a ellas dentro del programa. Las variables declaradas dentro de un bloque se denominan locales y sólo son válidas dentro de ese mismo bloque o dentro de otros bloques encerrados en él. Las variables declaradas al iniciar el script —antes del `setup()`— se denominan globales y se pueden utilizar en cualquier parte. Si las variables están declaradas dentro del `setup()` sólo se pueden utilizar en el bloque `setup()`, igualmente, si se han declarado en el `draw()` sólo son válidas en el bloque `draw()`. El siguiente ejemplo muestra el alcance de las variables dentro de una estructura de código que se describe en el siguiente apartado.

```
d = 50          // d y val son variables globales,
val = 0         // se pueden usar en cualquier lugar
function setup()
end
function update()
val = d * 0.7  // sus valores se actualizan en el bloque update()
d = d + 1
end
function draw()
gaBackground(0.0,1.0)
ofSetColor(val)
y = 150         // y es variable local, se puede usar solo en draw()
ofLine(0, y, d, y) // d e y son llamadas desde draw()
y = y - val    // el valor de y se relaciona con el de val
ofLine(0, y, d, y)
if d > 50 then
  x = 10        // x es variable local solo se puede usar en bloque if
  ofLine(x, 40, x+d, 40)
end
ofLine(0, 50, d, 50)
ofLine(x, 60, d, 60) // ERROR x no se puede usar fuera del bloque if
end
```

Hay otro tipo de variables que se denominan variables del sistema o constantes, tienen un nombre asignado por el propio software que le otorga ya su valor. En GAMUZA se reconocen:

27 Lua 5.1 Reference Manual, [texto on-line] <<http://www.lua.org/manual/5.1/manual.html>> [30.07.2012]

28 Más información en <<http://openframeworks.cc/documentation/utils/ofUtils/>> [30.07.2012]

De sistema

Ancho de la ventana de salida **OUTPUT_W**
 Alto de la ventana de salida **OUTPUT_H**

Tracking video

Máximo nº de blobs detectados **MAX_BLOBS**
 Nº columnas de optical flow **OPTICAL_FLOW_COLS_STEP**
 Nº filas optical flow **OPTICAL_FLOW_ROWS_STEP**

Análisis captura de audio

Nº de bandas FFT en la captura de audio **FFT_BANDS**
 Tamaño del buffer de audio **BUFFER_SIZE**
 Nº de canales de entrada de audio **AUDIO_INPUT_CHANNELS**
 Nº de canales de salida de audio **AUDIO_OUTPUT_CHANNELS**

Generación sonido

Tipos de onda **GA_SIN, GA_COS, GA_SAW, GA_TRI, GA_RECT, GA_NOISE, GA_PINK, GA_BROWN, GA_PHASOR, GA_PULSE**
 Notas musicales **DO_N, DOB_N, RE_N, REB_N, MI_N, FA_N, FAB_N, SOL_N, SOLB_N, LA_N, LAB_N, SI_N**; siendo N un valor que puede ir de 0 a 8

OSC

Puerto de envío de datos OSC **OSC_SENDING_PORT**
 Puerto recepción de datos OSC **OSC RECEIVING_PORT**
 Mensaje tipo **int** **OSC_INT**
 Mensaje tipo **float** **OSC_FLOAT**
 Mensaje tipo **string** **OSC_STRING**

Trigonometría

Radianes **PI, TWO_PI, HALF_PI, FOUR_PI**

De OpenFrameworks

Las asociadas a las funciones y clases de openFrameworks incluidas en GAMUZA.²⁹

De OpenGL

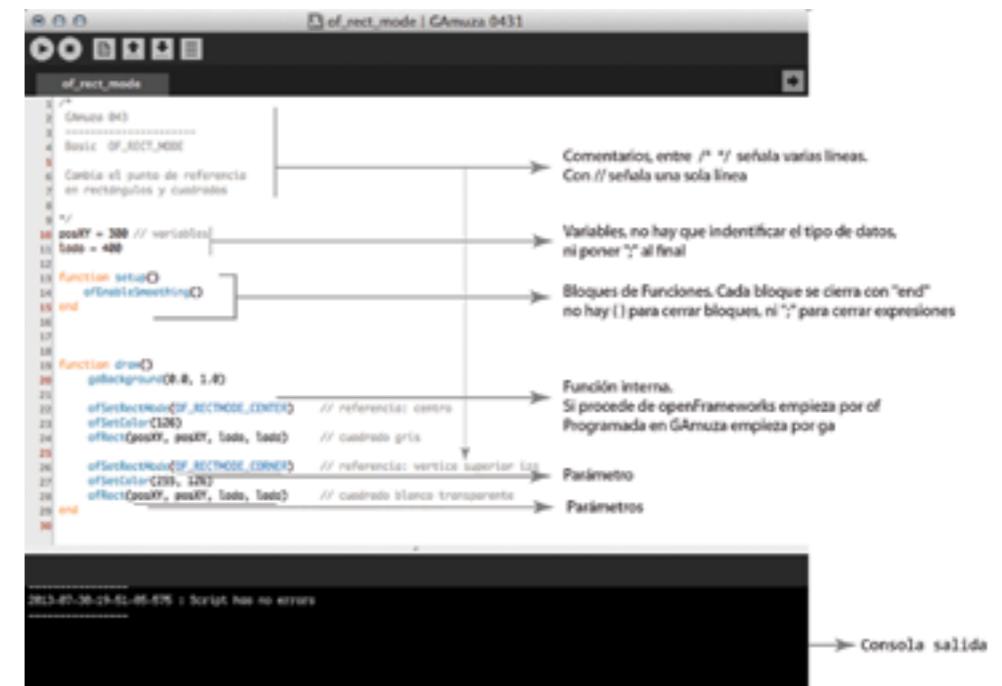
Las asociadas a las funciones y clases de OpenGL incluidas en GAMUZA³⁰. Ver apartado 16.2. Integración de OpenGL

4.1.2. Funciones

Fue Leibniz quien en el siglo XVII acuñó los términos «función», «variable», «constante» y «parámetro». Durante los siglos XIX y XX, a partir de la Teoría de Conjuntos se amplió el rango de aplicación de las funciones matemáticas, estableciéndose la actual definición de función como: la correspondencia entre los elementos de dos conjuntos dados, no necesariamente numéricos.

En programación, una función es un grupo de instrucciones que se ordenan con un objetivo concreto y que se efectúa al ser llamada desde otra función o procedimiento. Las funciones deben tener un nombre único en su entorno de programación, una lista de parámetros de entrada asignada y pueden devolver un tipo de datos como resultado.

GAMUZA, como todo software, requiere una terminología específica y, para facilitar el aprendizaje, el nombre de las funciones se ha mantenido lo más próximo posible al utilizado en otros lenguajes de programación creativa como Processing y openFrameworks. En las funciones debemos diferenciar entre los bloques de funciones, las funciones internas del software y funciones escritas por cada programador, pero para no complicar las definiciones vamos a ir señalando sus diferencias con ejemplos que muestren la estructura de programación. Empecemos viendo cuál es la estructura básica de un programa en el IDE.



```

of_rect_mode
of_rect_mode
  // Comentarios, entre /* */ señala varias líneas.
  // Con // señala una sola línea

  posXY = 300 // variables
  base = 400

  function setup()
    ofSetRectMode(OFP_RECT_MODE_CENTER)
  end

  function draw()
    ofBackground(0,0,100)
    ofRect(posXY, posXY, base, base) // referencial: centro
    ofRect(posXY, posXY, base, base) // cuadro gris
    ofRect(posXY, posXY, base, base) // referencial: vértice superior izq
    ofRect(posXY, posXY, base, base) // cuadro blanco transparente
  end
}

2013-07-30-21-51-075 : Script has no errors
  
```

The screenshot shows the GAMUZA IDE interface with a script titled "of_rect_mode". The code is annotated with arrows pointing to specific parts of the script:

- Annotations for comments: "Comentarios, entre /* */ señala varias líneas. Con // señala una sola línea".
- Annotations for variables: "Variables, no hay que indentificar el tipo de datos, ni poner ";" al final".
- Annotations for functions: "Bloques de funciones. Cada bloque se cierra con "end" no hay () para cerrar bloques, ni ; para cerrar expresiones".
- Annotations for internal functions: "Función interna. Si procede de openFrameworks empieza por of. Programada en GAMUZA empieza por ga".
- Annotations for parameters: "Parámetro" and "Parámetros".
- Annotation for the console output: "Consola salida".

29 Información en <http://ofxfenster.undef.ch/doc/ofConstants_8h.html> [15.08.2013]

30 Información en OpenGL Documentation <<http://www.opengl.org/sdk/docs/man/xhtml/>> [26.08.2012]

El sistema lee el código desde arriba hacia abajo y lo primero que aparece en la imagen son los comentarios. En GAmuza, los comentarios de línea se señalan con // igual que en Processing y C++. Son dos signos consecutivos, pueden ponerse más de dos, pero si las dos primeras barras están más separadas el sistema no lo reconoce como comentario y da error, se puede detectar el fallo si el texto no adopta el color gris que el IDE les asigna por defecto.

Cuando el comentario ocupa varias líneas (bloque de comentario), se señalan al principio con /* y al final del comentario con */.

Los comentarios son textos que el sistema ignora, no los procesa, pero son importantes para el programador. El código de programación puede ser muy breve y fácil de leer o tremadamente extenso, por lo que es fundamental acostumbrarse a comentarlo bien y también a ordenar y tabular las líneas de código según niveles para facilitar su lectura y revisión.

Tras los comentarios, están las variables globales y después los bloques de funciones que estructuran y contienen el código, estas líneas de código pueden ser funciones internas, procedimientos o expresiones, según las acciones que deban realizar. Por defecto, el nombre de las funciones aparece de color azul en el IDE y los términos que marcan la estructura de programación en naranja.

En GAmuza el final de cada bloque de función se señala con el término **end**, no usa signos de corchetes {} que marquen principio y final.

Los bloques básicos son:

```
function setup() // solo se ejecuta una vez
end
function update() // donde se actualizan las variables, se lee en loop
end
function draw() // donde va el código a dibujar, se lee en loop.
end
```

En el primer bloque, **function setup()**, cuando se arranca el programa, el código que contiene se procesa solo una vez (ciclo único). Contrariamente, el código que se escribe dentro de los bloques **function update()** y **function draw()**, el sistema los lee en loop (ciclo infinito), de manera que cuando llega a la última línea, genera un frame en la ventana de salida y empieza de nuevo por la primera línea de código, y así continuamente hasta que se apaga el programa.

Además de los bloques **setup()**, **draw()** y **update()**, existen otros bloques usuales en programas interactivos:

```
function mouseDragged() // cuando la acción se da al arrastrar el ratón
end
function mouseMoved() // el código actúa cuando se mueve el ratón
end
function mousePressed() // el código actúa cuando se aprieta un botón
end
function mouseReleased() // actúa cuando se suelta el botón
end
function keyPressed() // el código actúa cuando se presiona una tecla
end
function keyReleased() // el código actúa cuando se suelta una tecla
end
```

Después de los bloques, el siguiente elemento importante son las funciones internas. En GAmuza, los nombres de estas funciones tienen diferentes prefijos según su origen:

- of** Empiezan por of las funciones que provienen de openFrameworks. El 98% de las funciones de openFrameworks está disponible en GAmuza³¹ y mantienen su nombre original. Ver apartado 16.1 Integración de openFrameworks.
- ofx** Empiezan por ofx las funciones que provienen de las api ofxAddons³².
- gl** Corresponde a las funciones de OpenGL, ver capítulo 16.2 Integración de OpenGL.
- ga** Empiezan por ga las funciones del framework de GAmuza, ver capítulo 16.4 Framework de GAmuza.
- pd** Son funciones propias de GAmuza que se han programado para comunicar con el PURE DATA SYNTHESIS ENGINE, ver capítulo 16.4 Framework de GAmuza.
- au** Son funciones propias de GAmuza vinculadas con AUDIO UNIT PLUGINS, ver capítulo 16.4 Framework de GAmuza.

En el capítulo 16. Estructura del código fuente de GAmuza, se muestra en distintos apartados todas las funciones que soporta GAmuza.

Las funciones terminan con dos paréntesis que pueden albergar, o no, parámetros³³. Los parámetros se sitúan entre los paréntesis, separados por comas, y afectan el modo en que actúa la función. En las funciones de GAmuza que acogen parámetros, su valor numérico siempre tiene un rango entre 0.0 y

³¹ Documentación de openFrameworks ver <<http://www.openframeworks.cc/documentation>> [03.06.2012]

³² ofxAddons. Un directorio de extensiones y librerías para openFrameworks. <<http://ofxaddons.com/>> [03.06.2012]

³³ Algunos textos los denominan argumentos. En nuestro caso mantenemos la nomenclatura utilizada en: Casey Reas y Ben Fry, (2007) *Processing, a Programming Handbook for Visual Designers and Artists*. Cambridge (MA): The MIT Press.

1.0; algunas funciones no tienen parámetros y otras pueden albergar más o menos parámetros según la situación, como:

```
gaBackground(0.0, 1.0) //escala grises [de 0.0 (negro) a 1.0 (blanco)], alpha
gaBackground(0.0, 0.0, 1.0, 1.0) // (R, G, B, alpha) fondo azul opaco
```

Además de las funciones internas propias del software, cada programador puede hacer sus propias funciones, desarrolladas como un bloque, para después ser llamadas en alguno o varios de los bloques del sistema. Estas funciones pueden escribirse en el script principal o añadir pestañas a la interface del IDE guardándose entonces como archivo independiente vinculado al principal.

4.2. Expresiones aritméticas

Las cualidades visuales de una forma se pueden expresar numéricamente de modo que sus proporciones deriven de operaciones aritméticas. Estas operaciones pueden ser muy complejas para aquellos que disfruten con el campo de las matemáticas, o simples para programadores más visuales. En cualquier caso son un instrumento útil para la composición de formas, colores o sonidos, combinando variables con operadores aritméticos. Los operadores aritméticos básicos en GAMUZA provienen del lenguaje Lua, y son: + (suma), - (resta), * (multiplicación), / (división), ^ (exponencial), %(modular)

La función de los operadores +, -, *, / y ^ son conocidos por todos, el operador % es menos usual, calcula si un número es proporcional a otro, modular, es decir, si el número a la izquierda del operador % se divide por el que está a su derecha da como resultado un número entero. Es útil cuando se quiere trabajar con números múltiplos de 2, 3, 4,...

Es común utilizar expresiones matemáticas para modificar el valor de las variables o para expresar los parámetros de una función. Cuando la expresión aritmética se sitúa en el bloque `update()`, en cada vuelta del loop se aplica de nuevo la actualización del valor, incrementándolo o reduciéndolo sucesivamente. El siguiente ejemplo muestra la animación de una línea horizontal hasta que desaparece de la ventana. [Véase apartado 4.3.1. Estructuras condicionales, cómo regular el comportamiento ante estos límites]

```
/*
GAMUZA 043      E-4-1
-----
Expresiones aritméticas
*/
y = 20          // variable global
function setup()
end
function update()
    y += 20      // se puede escribir también y = y + 20
end              // y actualiza el valor de y, sumando 20 cada frame
function draw()
    gaBackground (1.0, 0.3) //fondo negro transparente, se ve el movimiento de las líneas
    ofSetColor(0)
    ofSetLineWidth(4)
    ofLine(0,y,OUTPUT_W,y) // función línea, recoge el valor actualizado en el update
end
```



4.2.1. Expresiones relacionales

Sintaxis

> (mayor que)	< (menor que)
>= (mayor o igual a)	<= (menor o igual a)
== (igualdad)	!= (desigualdad)
true	false

Una expresión relacional está compuesta por dos valores que se comparan con un operador relacional que evalúa si la relación establecida entre los datos es verdadera o falsa (**true** y **false**).

Expresión	Resultado
12 > 8	true
60 < 40	false

La expresión de igualdad se hace con **==**, porque el signo **=** aislado, significa la asignación de un valor a la variable, como en **x = 14**.

4.3. Estructuras de control

Las expresiones relacionales se utilizan en las estructuras de control, estas estructuras amplian mucho las opciones para procesar las instrucciones escritas en el código según las condiciones que establezcan, pudiendo también modificar el orden en el que se activan. Hay dos categorías de estructuras de control:

Estructuras condicionales: permiten que se realicen determinados conjuntos de instrucciones, en función de que se verifique, o no, una o varias condiciones enunciadas.

Estructuras iterativas o de repetición: permiten que se realice un conjunto de instrucciones repetidas veces, ya sea un número predeterminado de veces, o hasta que se verifique una determinada condición.

4.3.1. Estructuras condicionales y operadores lógicos

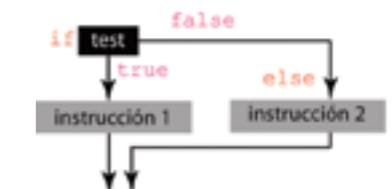
Sintaxis

Condicionales **if, elseif, else**

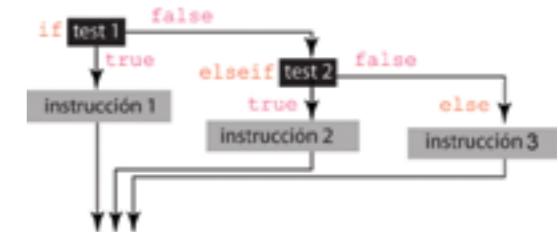
La expresión condicional simple **if**, permite a un programa seguir comportamientos diferentes, dependiendo de que los valores de sus variables cumplan unas condiciones u otras, esta comprobación se obtiene a través de una expresión relacional (test), de modo que las acciones pueden tener lugar, o no, sólo si una condición específica se cumple: si la condición se cumple (si el test tiene como resultado **true**), realiza la instrucción, si no se cumple se la salta. Para plantear el test se utilizan expresiones relacionales y operadores lógicos.



La expresión condicional doble **else** amplía las posibilidades de la estructura **if**, de manera que cuando la expresión asociada es falsa, el código del bloque **else** se ejecuta en su lugar.



La expresión condicional múltiple **elseif**, amplía aun más las opciones de la condicional. Si no se cumple la condición **if**, pasa a evaluar la condición **elseif**, y si tampoco se cumple puede aun establecer un **else** para efectuar una tercera instrucción si las dos anteriores no se cumplen. Pueden haber varios **elseif** encadenados.



Las expresiones condicionales utilizan **operadores lógicos** para combinar dos o más expresiones relacionales y para invertir los valores lógicos.

Sintaxis

Operadores lógicos **or, and, not**

Los operadores lógicos permiten formular más de una condición de forma simultánea. El operador **not**, invierte el valor lógico de las variables booleanas, cambiando verdadero a falso, y falso a verdadero. La estructura condicional **if** en GAMUZA, a diferencia de otros software, no utiliza **()** ni **{}**.

```

if not b then      // si b es falso, (if b == false) entonces
  instrucción      // haz tal cosa
end                // end, cierra el bloque if
  
```

```

if b == true then // si b es verdadero (if b then), entonces
end
if a > 5 or b < 30 then // si a es mayor que 5 ó b es menor que 30
    instrucción // solo hace falta que cumpla una de las condiciones
end
if a < 5 and a > 30 then // si a es mayor que 5 y es menor que 30
    instrucción // hace falta que se cumplan las dos condiciones
end

```

La estructura `if` con operadores lógicos más las opciones `elseif` y `else` amplían mucho la capacidad del código en situaciones en que las variables son cambiantes.

```

if a == b then // si a es igual a b, entonces
    instrucción 1 // haz instrucción 1
elseif a != c then // si no, pero si a es distinto de c, entonces
    instrucción 2 // haz instrucción 2
else // si no se cumple ninguna condición anterior
    instrucción 3 // haz instrucción 3
end

```

En el siguiente ejemplo se reasignan nuevos valores a las variables cuando las formas alcanzan los límites de la ventana de salida, generando un loop en la animación. Retomamos el ejemplo de la página 45, aplicando control a esos límites, cuando la línea llega al final de la pantalla, su coordenada y vuelve a valer 0.

```

/*
GAMUZA 043          E-4-2
-----
Control del movimiento con
Expresiones condicionales
*/
y = 0
inc = 20

function setup()
end

function update()
    if y < OUTPUT_H then // si y es menor que la altura de pantalla
        y += inc // (y = y + inc) suma 20 a su valor
    else
        y = 0 // y vale 0
    end
end

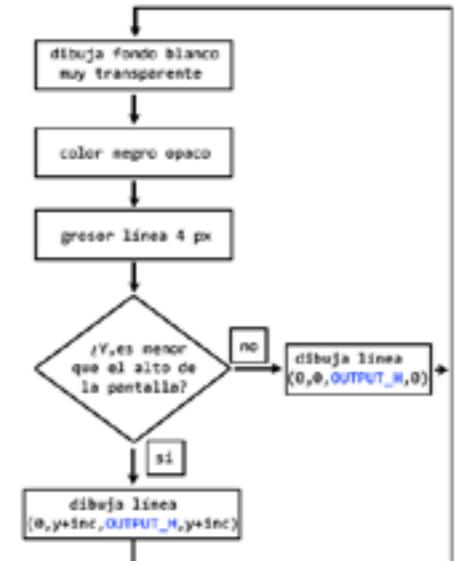
```



```

function draw()
gaBackground(1.0,0.2)
ofSetColor(0) // color negro
ofSetLineWidth(4) // grosor línea
ofLine(0, y, OUTPUT_W, y)
// recoge el valor de y actualizado
end

```



4.3.2. Expresiones de iteración. Repetición

Sintaxis

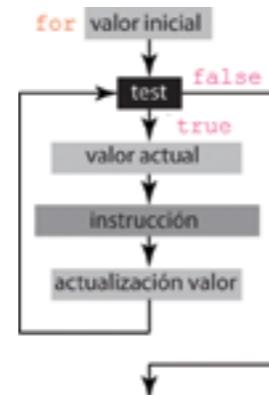
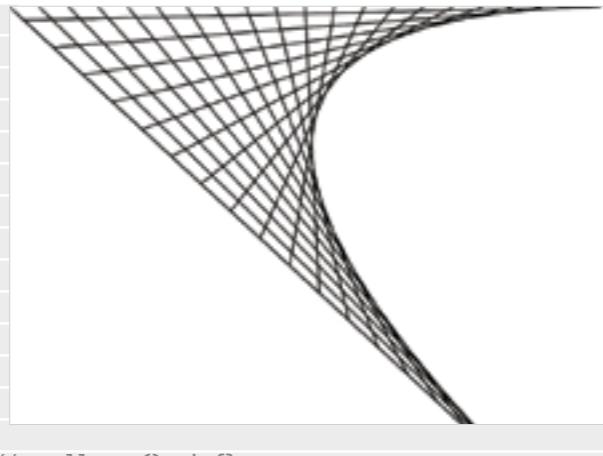
Repetición **for**

Las estructuras iterativas se utilizan para realizar acciones repetitivas en una o varias instrucciones, hasta que determinada expresión relacional deje de ser cierta. Cuando se sabe de antemano el número de repeticiones necesario, se utiliza la estructura **for**, si no se sabe el número de repeticiones se puede utilizar la estructura **while** [véase su uso acompañado con un **for** en el ejemplo E-14-1].

La estructura **for** se articula con tres declaraciones: el valor inicial de la variable, el test con una expresión relacional y la actualización del valor de esa variable. Las declaraciones dentro del bloque **for** se ejecutan de forma continua, generando un loop dentro de ese bloque mientras el test sea verdadero (**true**). El valor inicial de una variable se verifica en el test, la actualización modifica su valor después de cada iteración del bloque y la aplica a la instrucción. Cuando el test tiene como resultado **false**, el loop termina y el sistema continúa con las líneas de código que estén después del bloque **for**. Cuando la actualización del valor es sumar 1, suelen ponerse solo dos declaraciones: valor inicial de la variable y test, asumiendo la estructura que en cada actualización suma 1. GAMUZA, igual que en las estructuras condicionales, no utiliza **()** ni **{}** en la estructura **for**, y la cierra con **end**. Algunos ejemplos comentados.

```
/*
GAMUZA 043 E-4-3
-----
estructura for
*/
function setup()
  ofEnableSmoothing()
end

function draw()
  gaBackground(1.0, 1.0)
  ofSetColor(0)
  ofSetLineWidth(3)
  for i = 0, OUTPUT_W, 50 do // no lleva () ni {}
    ofLine(i, 0, OUTPUT_W-i, OUTPUT_W-i)
  end // cierra el for con end
end
```



La estructura **for** se repetirá en loop hasta que la variable **i** tenga el valor **OUTPUT_W**, ancho de la ventana de salida, y en cada vuelta el valor de **i** sumará 50. Con esa lista de valores aplicada a la coordenada **x** se van dibujando líneas en cuyo primer par de coordenadas, **x** responde a ese incremento de 50, e **y** siempre vale 0, es decir, recorre el eje de las X situando un punto cada 50 px.

En el segundo par de coordenadas de las líneas, **x** e **y** tienen el mismo valor por lo que los puntos con sus dos coordenadas iguales se sitúan siempre en una diagonal de 45°, y como este valor se va reduciendo simultáneamente a como crece la **x** primera, lo que recorre el segundo par de coordenadas es la primera línea dibujada por el **for**, reproduciendo así la representación plana de un paraboloide hiperbólico.

La estructura repetitiva **for** puede combinarse con la condicional **if** para reajustar la condición ya establecida en el test de **for**. En este caso la condición del **if** señala cómo funciona el operador aritmético **%** que vimos en la página 45.

```
/*
GAMUZA 043 E-4-4
-----
estructura for más if
*/
function setup()
end

function draw()
  gaBackground(0.0,1.0) // fondo negro opaco
  ofSetLineWidth(3) // grosor línea 3 px
  for a = 0, OUTPUT_W, 13 do
    if(a % 2) == 0 then // si a es par, entonces
      ofSetColor(255) // color blanco
    elseif(a % 5) == 0 then // si no es par, pero es múltiplo de 5
      ofSetColor(255, 0, 0) // color rojo
    else // para el resto de opciones
      ofSetColor(0,255,0) // color verde
    end
    ofLine(a, 0, a, OUTPUT_H) // dibuja las líneas con los valores
  end // que a obtiene en el for para coordenada X
end
```

Como los valores que *a* va obteniendo en el loop del **for** tienen que cumplir otras condiciones, se ha elegido el número 13, raro, para tener un rango más diverso, pero siempre generará un pattern:

0, 13, 26, 39, 52, 65, 78, 91, 104, 117, 130, 143, 156, 169, 182, 195...,

Par, impar, par, impar..., por lo que dibuja una línea blanca cada dos. El primer múltiplo de 5 que no sea par es el 6º y luego cada 10. De este modo el **for** dibuja las líneas verticales cada 13 píxeles, y la condicional **if** determina los colores.

La estructura iterativa **for** genera repeticiones en una dimensión, pero si se anida un **for** dentro de otro **for** se generan repeticiones en dos dimensiones, como muestra el siguiente ejemplo.

```
/*
GAmuza 043          E-4-5
-----
estructura for+for retícula 2D

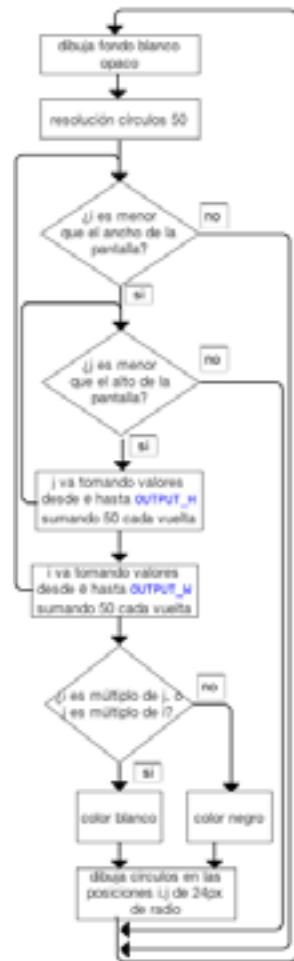
*/
inc = 20

function setup()
end

function draw()
    gaBackground(1.0,1.0)
    ofSetCircleResolution(50)
    for i=0, OUTPUT_W, inc do
        for j=0 , OUTPUT_H, inc do
            if i % j == 0 or j % i == 0 then //si i es proporcional a j o viceversa
                ofSetColor(255)           //color blanco
            else
                ofSetColor(0)            // color negro
            end
            ofCircle(i, j, inc/2-1)
        end
    end
end
```

En este doble **for**, cuando la variable *i* adopta el primer valor, *i* = 0, la lectura del código pasa al segundo **for** para asignar el primer valor a *j*, *j*=0, luego *j*=20, *j*=40,... hasta que *j* llega al valor de la altura de la ventana de salida, con ello ha generado las coordenadas (0,0) (0,20) (0,40)... (0,OUTPUT_H). Después vuelve al primer **for** e *i* adquiere el valor 20 y de nuevo pasa al segundo **for** para asignar valores a *j* desde 0 hasta el valor de la altura de ventana de salida, generando las

coordenadas de la segunda columna de círculos (20,0) (20,20) (20,40)..., así hasta que completa toda la retícula cuando *i* alcanza el valor del ancho de ventana y *j* el alto: (OUTPUT_W, OUTPUT_H). Si quitamos el bloque condicional **if** del código y dejamos solo el color negro, la ventana de salida mostraría una retícula regular de círculos, porque los huecos que se perciben corresponden a círculos blancos que se confunden con el fondo cuando *i* es múltiplo de *j* o viceversa.



4.4. Estructuras de programación avanzadas

Las tablas y las clases en el lenguaje de scripting derivan de la programación orientada a objetos (POO) que es un modelo de programación, una forma particular de pensar y escribir el código, en la que se utilizan los elementos básicos vistos anteriormente: variables, expresiones y funciones..., pero organizados de modo más próximo al lenguaje verbal. El hecho de ser un modelo implica que tiene su propia lógica.

La POO nos hace pensar las cosas de una manera más abstracta y con una estructura muy organizada, que sigue un proceso de descomposición del problema en partes bastante particular. Los métodos tradicionales de programación estructurada descomponen la complejidad en partes más simples y fáciles de codificar, de modo que cada sub-programa realiza una acción. En la POO no se descompone el problema en las acciones que tiene que hacer el programa, sino en objetos, por lo que debemos pensar en el posible escenario de esos objetos para definir sus características y comportamiento. De ahí que el elemento básico de esta estructura de programación no son las funciones sino los **objetos**, entendidos como la representación de un concepto que contiene información necesaria (datos) para describir sus **propiedades** o atributos y las operaciones (métodos) que describen su **comportamiento** en ese escenario. Por eso implica un método distinto de enfocar los problemas que empieza por la **abstracción**, entendida en este caso como la capacidad mental para representar una realidad compleja en los elementos separados simplificados (objetos), ver aisladamente sus necesidades de definición y comportamiento, y también sus relaciones (**mensajes**) y comportamiento conjunto.

Si nos detenemos a pensar sobre cómo se nos plantea un problema cualquiera en la realidad podremos ver que lo que hay son entidades (otros nombres que podríamos usar para describir lo que aquí llamo entidades son "agentes" u "objetos").

Estas entidades poseen un conjunto de propiedades o atributos, y un conjunto de métodos mediante los cuales muestran su comportamiento. Y no sólo eso, también podremos descubrir, a poco que nos fijemos, todo un conjunto de interrelaciones entre las entidades, guiadas por el intercambio de mensajes; las entidades del problema responden a estos mensajes mediante la ejecución de ciertas acciones.³⁴

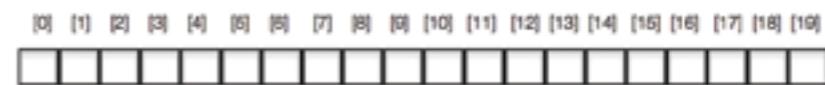
Esta forma particular de estructurar el código favorece su reutilización al compartimentarse en módulos independientes. Recordad la cita de Katherine B. McKeithen³⁵ respecto a la importancia de organizar la información para procesarla como grupos significativos, algo así sucede en la POO. Nos centraremos en dos elementos que tienen un papel importante en este juego del pensar diferente, Tablas y Clases; señalando la particularidad de esos elementos en GAMUZA, pues la hibridación de plataformas entre openFrameworks y Lua, plantean otra forma de aproximarse a la programación orientada a objetos.

³⁴ Luis R. Izquierdo (2007) "Introducción a la programación orientada a objetos", pág. 2. [texto on-line] <<http://luis.izqui.org/resources/ProgOrientadaObjetos.pdf>> [25.07.2012].

³⁵ Katherine B. McKeithen et al., "Knowledge Organization and Skill Differences in Computer Programmers" *Cognitive Psychology* 13(3), pág. 307. [Texto on-line] <<http://hdl.handle.net/2027.42/24336>> [Consultado: 12.05.2012]

4.4.1. Tablas

Las tablas mantienen cierta relación con la noción de array que en programación y matemáticas está vinculada a matriz, y significa un conjunto ordenado consecutivamente de elementos, del mismo tipo de datos, almacenados bajo el mismo nombre y para los que el sistema reserva memoria. El orden consecutivo asigna a cada elemento un ID, este ID es una forma numérica consecutiva que empieza por 0 y lo identifica de manera que se puede acceder a ese elemento en la programación individualmente. Los datos pueden ser de cualquier tipo pero no se pueden mezclar distintos tipos en un mismo array. Hay arrays de una dimensión (lineales) o de dos dimensiones (matrices) que podemos visualizarlo como una fila de contenedores que acogen cada uno de ellos un dato. En la siguiente imagen vemos cómo un array de 20 elementos se ordena hasta la posición 19, porque empieza a contar desde 0:



Los arrays de dos dimensiones constituyen una matriz de filas y columnas. Por ejemplo, la ventana de salida podemos verla como un array cuyo número de elementos es igual a los píxeles de la imagen y su dimensión es el resultado de multiplicar el ancho por el alto. Al igual que los array lineales, empieza a contar por cero y el primer elemento tiene el ID [0,0]

[0,0]	[0,1]	[0,2]	[0,3]	[0,4]
[1,0]	[1,1]	[1,2]	[1,3]	[1,4]
[2,0]	[2,1]	[2,2]	[2,3]	[2,4]
[3,0]	[3,1]	[3,2]	[3,3]	[3,4]

Vamos a ver cómo GAMUZA a través del lenguaje Lua utiliza lo que se denominan tablas (librería table de Lua), que pueden hacer las mismas operaciones que los arrays con mayor flexibilidad porque las tablas permiten utilizar datos de diferente tipo y simplifican mucho el proceso de programación. Además, las tablas en Lua no se entienden como lineales o de dos dimensiones, sino que una tabla lineal puede albergar múltiples dimensiones de datos.

La sintaxis para declarar una tabla es:

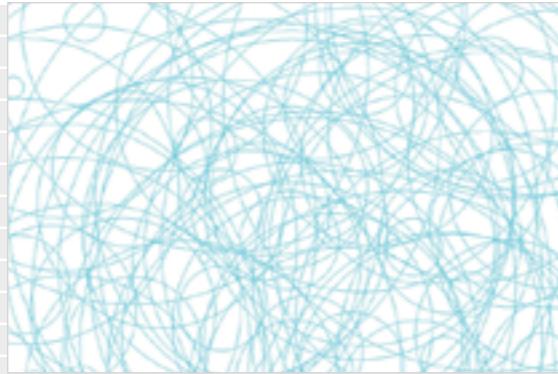
```
nombreTabla = {}
```

Para crearla pueden enumerarse los elementos que la constituyen entre {}, definir cual es el número de elementos por medio de una variable, o simplemente crear la tabla y todos los objetos utilizando una expresión **for** que recorrerá la tabla y asignará valores a todos los datos. Una vez declarada, los componentes de la tabla se expresan con paréntesis cuadrados: **nombreTabla[x]**, donde x corresponde al ID de ese elemento. Pasamos a describirlo comentado el código de algunos ejemplos.

```
/*
GAmuza 043      E-4-6
-----
Basics/tabla_circulosFijos.ga
creado por n3m3da | www.d3cod3.org

*/
myTable = {}

function setup()
    for i = 0, 100 do // establece nº objetos de la tabla
        myTable[i] = {} // crea otra tabla en la tabla
        myTable[i].x = ofRandomf()*OUTPUT_W // asigna valores cada coordenada x
        myTable[i].y = ofRandomf()*OUTPUT_H // asigna valores cada coordenada y
        myTable[i].diam = ofRandom(0,OUTPUT_W/2) // y valores a los diámetros
    end
end
function draw()
    gaBackground(1.0,1.0)
    ofSetCircleResolution(60)
    ofSetColor(0,255,255) // color cyan
    ofNoFill() // no rellenar
    for i = 0, 100 do // dibuja los 100 círculos
        ofCircle(myTable[i].x, myTable[i].y, myTable[i].diam)
    end
end
```

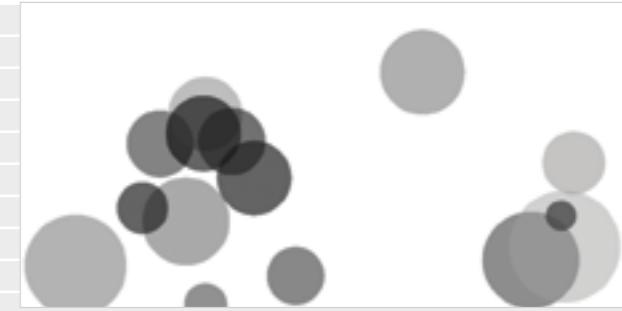


La variable `i` de la estructura `for` va adoptando consecutivamente el ID de cada círculo de la tabla. Con la primera línea del código: `myTable[i] = {}`, se crea una nueva tabla para cada uno de los elementos de la tabla inicial, lo que permite definir sus propiedades sin hacer nuevos arrays, simplemente en el mismo `for` se va asignando valores a cada una de esas propiedades utilizando el operador `".."` junto al código que representa cada objeto en la tabla inicial: `myTable[i]`.

En la función `draw()`, además de las funciones internas para el color y no relleno, otra estructura `for` dibuja todos los círculos recogiendo los datos de las propiedades definidas en el `setup()`. No hay función `update()` porque en el ejemplo las formas están fijas, no hay comportamiento. En el siguiente ejemplo sí se incluye este bloque con comportamientos, en el que también se incrementa el número de propiedades de cada uno de los objetos de la tabla.

```
/*
GAmuza 043      E-4-7
-----
Tabla círculos rebotan
*/
circulos ={} // declara la tabla
maxCirculos = 20

function setup()
    for i = 0, maxCirculos-1 do // recorre todos los objetos
        circulos[i] = {} // crea otra tabla en la tabla
        circulos[i].posX = ofRandom(30, OUTPUT_W) // coordenadas X al azar
        circulos[i].posY = ofRandom(OUTPUT_H/2) // coordenadas Y al azar
        circulos[i].diam = ofRandom(15, 100) // diámetro círculos
        circulos[i].velX = 0.000000 // velocidad X
        circulos[i].velY = 0.000000 // velocidad Y
        circulos[i].gY = 0.00184562 // gravedad Y
        circulos[i].gX = ofRandom(-0.018326, 0.044326) // gravedad X
        circulos[i].color= ofRandom(0, 100) // gris azar para cada objeto
        circulos[i].transp = 220 // transparencia
    end
end
function update()
    for i=0, maxCirculos-1 do
        circulos[i].posX += circulos[i].velX // posición + velocidad
        circulos[i].posY += circulos[i].velY
        circulos[i].velX += circulos[i].gX // velocidad + gravedad
        circulos[i].velY += (circulos[i].gY*circulos[i].diam)
        if circulos[i].posX <= 0 or circulos[i].posX >= OUTPUT_W-10 then
            circulos[i].velX *= -0.972773
        end
        if circulos[i].posY >= OUTPUT_H or circulos[i].posY <= 0 then
            circulos[i].velY *= -0.962722 // invierte dirección al tocar los bordes
        end
    end
end
function draw()
    gaBackground(1.0, 1.0)
    ofSetCircleResolution(60) // dibuja todos los círculos
    for i= 0, maxCirculos-1 do // transparencia vinculada al diámetro
        ofSetColor(circulos[i].color, circulos[i].transp - circulos[i].diam)
        ofCircle(circulos[i].posX, circulos[i].posY, circulos[i].diam)
    end
end
```



En el bloque `setup()` se asignan las **propiedades** de los objetos utilizando el operador `".."` junto al nombre de la tabla y el identificador de cada uno de los objetos que contiene `[i]`

```
nombreTabla[i].Propiedad
```

El `for` recorre la tabla, reserva la memoria que el sistema necesita para estos datos y les asigna las propiedades definidas, que en este caso son: posición X y posición Y (para localizar el centro de cada círculo), radio (tamaño), velocidad X y velocidad Y (para el movimiento de los círculos) gravedad X y gravedad Y (para representar los rebotes y un movimiento menos lineal), color y transparencia.

Los valores de `i` van desde `0` a `19` (`maxCirculos-1`), en 20 pasos porque empieza a contar desde `0`. Los valores de las propiedades que se asignan inicialmente pueden después cambiar, pues un objeto puede tener distintos valores en momentos diferentes, pero siempre es el mismo objeto.

Los **comportamientos** se establecen en el bloque `update()`, donde se actualizan los datos de las variables que definen las **propiedades**. Con un nuevo `for`, a las coordenadas de posición de cada objeto se le suman las de velocidad; y a las de velocidad, las de gravedad en la coordenada de las X; y las de gravedad por el diámetro del círculo en la coordenada Y. Este incremento produce un movimiento más físico de peso en el círculo. Después se ajustan los rebotes a los límites de la ventana mediante bloques condicionales `if`, invirtiendo la dirección del movimiento al multiplicar el valor de su velocidad por un número negativo con muchos decimales que no llega a `1` para simular un efecto de gravedad. Finalmente, en el bloque `draw()` se dibujan todos los círculos mediante un nuevo `for`.

El siguiente ejemplo, más complejo, está basado en la noción de Random Walker, utiliza varias tablas, la función `ofNoise()` [Véase apartado 5.5.1 Noise] y `ofMap()` [Véase descripción en página 98].

```
/*
GAmuza 043          E-4-8
-----
Tabla Noise Walker
Shiffman_NatureOfCode/CH0_noiseWalk
creado por n3m3da | www.d3cod3.org
*/
walkers = []
noiseTimeX = []
noiseTimeY = []
stepX = []
stepY = []
numW = 300
```



```
function setup()
    for i = 0,numW-1 do
        walkers[i] = {}
        walkers[i].x = OUTPUT_W/2
        walkers[i].y = OUTPUT_H/2
        noiseTimeX[i] = ofRandom(0,10000)
        noiseTimeY[i] = ofRandom(7000,14000)
        stepX[i] = ofRandom(0.001,0.003)
        stepY[i] = ofRandom(0.001,0.003)
    end
end

function update()
    for i=0,numW-1 do
        mapX = ofMap(ofNoise(noiseTimeX[i]),0.15,0.85,0,OUTPUT_W,true)
        mapY = ofMap(ofNoise(noiseTimeY[i]),0.15,0.85,0,OUTPUT_H,true)

        walkers[i].x = mapX
        walkers[i].y = mapY
        noiseTimeX[i] += stepX[i]
        noiseTimeY[i] += stepY[i]
    end
end

function draw()
    gaBackground(0.0,0.01)
    ofNoFill()

    for i=0,numW-1 do
        if math.fmod(i,7) == 0 then // Devuelve el resto de la división de i por 7
            ofSetColor(255)           // y redondea el cociente a cero.
        else
            ofSetColor(0)
        end
        ofRect(walkers[i].x,walkers[i].y,1,1)
    end
end
```

Algunas funciones internas vinculadas a las tablas³⁶

`table.getn(myTable)` Para saber el número de elementos de una tabla

`table.foreach(myTable, myFunction)` Recorre cada índice de la tabla y le asigna la función (el segundo parámetro dentro del corchete).

`table.insert(myTable, [position], value) // ejemplo (myTable, 5, true)`

Añade un valor al final de la tabla si solo tiene 2 parámetros. Si se señala insertar el valor en una posición exacta los índices siguientes se desplazan en consecuencia.

`table.remove(myTable, [pos]) // ejemplo (myTable, 3)`

Elimina un elemento de una tabla y mueve hacia arriba los índices restantes si es necesario. Si no se asigna posición borra el último elemento.

4.4.2. Clases

En una clase, el concepto más destacado es el de **objeto** que representa cualquier cosa, real o abstracta, y es el componente conceptual del problema planteado. En programación, un objeto se define en dos partes: 1) **Constructor**: una serie de datos que constituyen el estado particular (propiedades) del objeto y que se formalizan mediante un conjunto de variables. 2) **Update**: comportamiento de los objetos que se formaliza mediante **métodos** (funciones de objetos) relacionados con las variables.

Como se ha visto en los ejemplos anteriores, el objeto círculo tiene una serie de **propiedades** declaradas con variables según lo que requiera la situación planteada: coordenadas del centro, radio o diámetro, color, transparencia, grosor del trazo..., de ellas, las que se decidan como necesarias para definir el objeto serán los parámetros del constructor.

El **comportamiento** quedará regulado por los **métodos** que programemos para que esos objetos círculo sitúen sus coordenadas fijas o en movimiento, sigan una dirección u otra, reboten con los límites de la ventana, etc.,

Definición de Clase

Estableciendo una relación algo simplista podemos entender las clases aludiendo a la Teoría de las ideas (o Teoría de las formas) de Platón. Las clases se corresponderían al mundo inteligible donde están las estructuras o modelos (las ideas) en las cuales se basan las cosas físicas concretas (los objetos) que podemos percibir en el mundo sensible.

Por ejemplo, la clase círculos puede contener todos los objetos circulares, con sus distintos tamaños, colores, etc., y sus distintos comportamientos. El objeto círculo es una **instancia** de la clase círculos. Cada objeto círculo tiene un estado particular independiente, pero todos tienen en común el hecho de tener una variable color, radio, etc. E igualmente deben tener en común la posibilidad de adoptar determinados comportamientos, por lo que la clase debe declarar o implementar los métodos que regulan el comportamiento de los objetos que contiene.

En resumen, una clase es como una plantilla que define las propiedades y los métodos que son comunes para todos los objetos que agrupa, por lo que para definirlos siempre hay un bloque dedicado al **constructor** que lleva el mismo nombre que la clase y define los parámetros necesarios para crear o instanciar un objeto de la clase.

Herencia

Los objetos de una clase pueden pertenecer a una clase mayor que, por ejemplo, alberga figuras geométricas desplegadas en distintas clases. Si en nuestro trabajo planteamos distintos niveles, un círculo debe responder tanto a las propiedades de la clase círculos como a la de la clase figuras planas, por lo que compartirá esos atributos y comportamientos con objetos de otras posibles clases, como la clase triángulos o la clase cuadrados.

³⁶ Más información en "Lua for Beginners" [texto on-line] <<http://lua.gts-stolberg.de/en/table.php>> [29.07.2012]

Esta herencia no limita las posibilidades de particularización de la clase círculos, más bien el orden de pensamiento es inverso al establecido aquí. La herencia permite definir nuevas clases partiendo de las existentes, de modo que heredan todos los comportamientos programados en la primera y pueden añadir variables y comportamientos propios que la diferencian como clase derivada o subclase. Y así se puede establecer un árbol de herencias tan ramificado como se necesite.

Mensajes

La imagen siguiente, además de reflejar un diagrama de herencias también muestra con una flecha que va de la clase triángulos a la clase rectángulos como se establece un mensaje. Es a través de los mensajes como se produce la interacción entre los objetos, permitiendo así programar comportamientos más complejos.

Por ejemplo, cuando un objeto triángulo cumple una determinada condición, puede enviar un mensaje a un objeto rectángulo para que realice alguno de los comportamientos que su clase (la del rectángulo) tiene definidos, en el mensaje se pueden incorporar determinados parámetros para ajustar el comportamiento.

Un mismo mensaje puede generar comportamientos diferentes al ser recibido por objetos diferentes. A esta característica se le denomina **polimorfismo**.

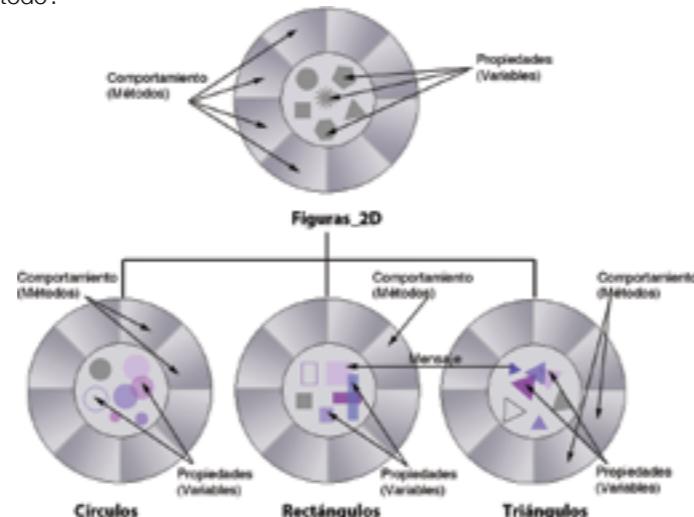
Es conveniente hacer diagramas de las clases con las propiedades y comportamientos de los objetos antes de empezar a escribir el código, para planificar cuestiones como:

¿Cuántas clases necesitamos definir y qué herencia se establece entre ellas?

¿Qué propiedades necesitan los objetos?

¿Cuáles son los métodos que ofrecen?

¿Dónde implementamos cada método?



Lua does not have a class system, but its powerful metaprogramming facilities makes defining classic objects straightforward.³⁷

En el lenguaje Lua, la noción de clases también está relacionada con el funcionamiento de las tablas y metatablas, porque las tablas tienen un estado y una identidad (**self**) que es independiente de sus valores, como hemos mencionado antes, dos objetos con el mismo valor son objetos diferentes, un objeto puede tener distintos valores en momentos diferentes, pero siempre es el mismo objeto. Al igual que los objetos, las tablas tienen un ciclo vital que es independiente de aquello que lo creó, o donde se crearon. Hemos visto que las tablas asignan propiedades a cada elemento mediante el operador ". ", ahora veremos cómo se asocian métodos a los objetos de una clase mediante el operador ":".

La sintaxis para declarar las clases es:

```
class 'nombreClase'
```

Se hace el **constructor** con una función que lleva el nombre de la clase y se inicializa con

```
function nombreClase:_init (parámetro1, parámetro2, ...)
```

Nombre de la clase, dos puntos ":" y dos guiones bajos "__" seguidos de **init** con los parámetros que definen la construcción del objeto, entre paréntesis y separados por ", ".

La clase tiene su propio método **update()** que se escribe:

```
function nombreClase:update()
```

donde se pone el código para actualizar los valores de las variables y programar el comportamiento de los objetos.

Y su método **draw()** que contiene el código de las formas que va a dibujar.

```
function nombreClase:draw()
```

Después, se pueden crear los objetos de la clase dándole los valores concretos a los parámetros definidos en el constructor.

```
ob1 = nombreClase(valorParámetro1, valorParámetro2, ...)
```

La herencia se implementa utilizando el concepto de metatablas, que actúan como superclases y la función **setmetatable()**. El código **__index** almacena la metatable de la tabla. En el siguiente código la tabla Rectangulo deriva de la tabla Formas³⁸:

```
Rectangulo = {}
setmetatable(Rectangulo, {__index = Formas})
```

Algunos ejemplos comentados de clases simples.

³⁷ Lua-users wiki: Simple Lua Classes [texto on-line] <<http://lua-users.org/wiki/SimpleLuaClasses>> [25.07.2012]

³⁸ Marcelo da Silva Gomes, "The Shape Example in Lua" [texto on-line] <<http://onestepback.org/articles/poly/gp-lua.html>> [25.07.2012]

```

/*
GAmuza 043          E-4-9
-----
Clase básica
*/
// -----
// ----- class foco

class 'foco'
    //constructor
function foco:_init(x, r, s, g) //define parámetros y propiedades del objeto
    self posX = x // posición X
    self.radio = r // diámetro
    self.speed= s // velocidad
    self.grav = g // gravedad
end

function foco:update()           // actualiza parámetros para el comportamiento
    self posX += self.speed
    self.speed += self.grav*self.diam
        // condicionales: cuando llegue a los topes cambia de dirección
    if self posX + self.radio > OUTPUT_W or self posX - self.radio < 0 then
        self.speed *= -0.999722
    end
end

function foco:draw()            // función para dibujar la forma del objeto
    ofCircle(self posX, OUTPUT_H/2, self.diam)
end

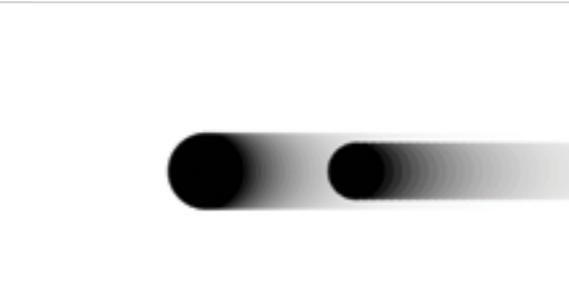
// -----ARCHIVO PRINCIPAL-----

foco1 = foco(80, 80, 3.5, 0.00184562) // crear 2 objetos de la clase foco
foco2 = foco(425, 60, 15.5, 0.00184562)

function setup()
    ofEnableSmoothing()
end

function update()
    foco1:update()
    foco2:update()
end

```



```

function draw()
gaBackground(1.0, 0.1)
ofSetCircleResolution(50)
ofSetColor(0)
foco1:draw()
foco2:draw()
end

```

Después de declarar la clase: `class 'foco'` se define el **constructor**, para ello deben conocerse los parámetros necesarios para definir las **propiedades** concretas de los objetos que queremos crear. En el ejemplo, el código del constructor `function nombreClase:_init (parametro1, parámetro2,...)` corresponde a: `function foco:_init(x, d, s, g)`, y dentro de él, utilizando el operador `"."` se define cada propiedad de los objetos con la identidad (`self`).

Las clases tienen sus propios métodos `update()` y `draw()` para determinar los **comportamientos** y dibujar los objetos que se asocian a la clase mediante el operador `:`

```

function foco:update()
function foco:draw()

```

Dentro del método `:update()`, para definir el **comportamiento** de movimiento, se suma el valor de la posición X a la velocidad, suma el valor de la velocidad al de la gravedad y se establecen las condiciones para cambiar el sentido de la velocidad cuando la coordenada X llega a los límites de la ventana. En el método `:draw()` se sitúa la funciones que dibuja el objeto.

Después de definir la clase se crean el objeto u objetos concretos, dando valores específicos a los parámetros determinados en el **constructor**. En este caso, los parámetros (`x, d, s, g`) para el objeto `foco1` son `(80, 80, 3.5, 0.00184562)`, es decir, el objeto aparece en la coordenada `x = 80`, tiene un diámetro de `80` píxeles, se mueve a una velocidad de `3.5` y con una gravedad de `0.00184562`. En los bloques habituales de un programa en GAmuza, el código llama a los métodos de la clase.

Es importante observar las diferencias entre clases y tablas. En una tabla las propiedades pueden definirse por el Id `[i]`, pero la tabla no tiene métodos propios, por lo que normalmente actualiza el comportamiento en el bloque `update()` y dibuja las formas en el bloque `draw()` del programa. En el siguiente ejemplo volvemos a las tablas para ver cómo se identifica el id de cada objeto, y se diferencia el color del texto que marca el ID en uno de ellos. [Para incorporar texto Véase 6.1. Textos y tipografía]

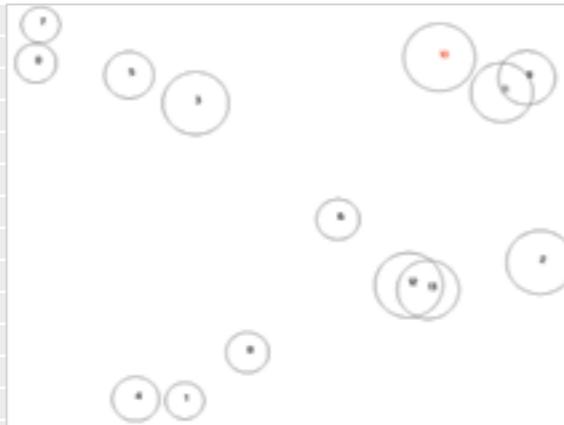
```
/*
GAmuza 043           E-4-10
-----
Tabla id No clase
creado por n3m3da | www.d3cod3.org
*/
circulo = []
tex = ofTrueTypeFont()

function setup()
    for i = 0, 13 do
        circulo[i] = {}
        circulo[i].radio = ofRandom(40, 80)
        circulo[i].x = ofRandom(circulo[i].radio, OUTPUT_W- circulo[i].radio)
        circulo[i].y = ofRandom(circulo[i].radio, OUTPUT_H- circulo[i].radio)
        circulo[i].ID = i
    end
    tex.loadFont(gaImportFile("Kautiva_Pro.otf"),20, true, true)
end

function update()
end

function draw()
    gaBackground(1.0,1.0)
    rojo = 0
    for i = 0, 13 do
        ofSetColor(0)
        ofNoFill()
        ofCircle(circulo[i].x, circulo[i].y, circulo[i].radio)
        if i == 10 then // el comportamiento, color, se genera en el bloque draw
            rojo = 255
        else
            rojo = 0
        end
        ofSetColor(rojo, 0, 0)
        tex.drawString(toString(circulo[i].ID),circulo[i].x,circulo[i].y)
    end
end

```



Cuando se quiere trabajar con muchos objetos controlando las propiedades de cada uno de ellos de forma compleja, se combina clase y tabla para asignar un id a cada objeto y poder programarle comportamientos específicos.

En el siguiente ejemplo se crea la clase ‘modulo’ y su objeto se introduce en la tabla mod. En la tabla, todos los objetos módulo se organizan siguiendo una estructura 2D para configurar una retícula, pero para determinar el número de objetos que contiene la tabla se quiere asignar el id de cada módulo con un sólo número, y no 2, para ello se despliega un index continuo a partir del index 2D. Esto permite recorrer todos los objetos con un doble `for`, manteniendo la condición 2D, pero trabajar con la tabla como si fuera de una dimensión.

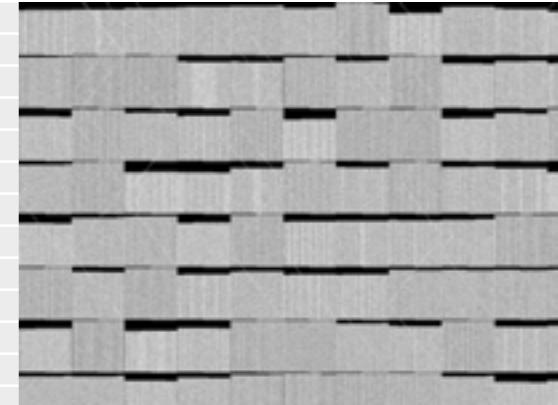
```
/*
GAmuza 043           E-4-11
-----
clase y tabla 2D
creado por mj
*/
unit = 100
ancho = math.ceil(OUTPUT_H/unit)
alto = math.ceil(OUTPUT_W/unit)
contador = ancho * alto
mod = {}

// ----- class modulo

class 'modulo' // declara la clase
function modulo:_init(mx, my, x, y, speed) // constructor
    self.mx = mx //incremento posición x módulo
    self.my = my //incremento posición y módulo
    self.x = x //posición inicial x
    self.y = y //posición inicial y
    self.speedx = speed // velocidad x
    self.ydir= 1 // dirección y
end

function modulo:update()
    self.x += self.speedx // suma velocidad a la posición
    if self.x >= unit or self.x <= 0 then // si x choca con los bordes del módulo
        self.speedx *= -1 // invierte la velocidad
        self.y += self.ydir // y suma su dirección
    end
    if self.y >= unit or self.y <= 0 then // si y choca con los bordes del módulo
        self.ydir = -1 // genera líneas diagonales al final
        self.y += self.ydir
    end
end

```



```

function modulo:draw()
    ofSetColor(ofRandom(180,255), 90) // color random casi blanco transparente
    ofCircle(self.mx + self.x, self.my + self.y, 1) //dibuja puntos
end

////////// FIN DE LA CLASE /////////////

function setup()
    ofEnableSmoothing()
    index = 0
    for i = 0, ancho-1 do //recorre objetos de la tabla 2D con un doble for
        for j = 0, alto-1 do // asocia la tabla a la clase
            mod[index] = modulo(j*unit, i*unit, unit/2, unit, ofRandom (1.9, 4.8))
                //asigna valores a los parámetros del constructor
            index = index + 1 // designa n° de objetos = ancho * alto (contador)
    end
end

function update()
    for i = 0, contador-1 do
        mod[i]:update() //aplica el update() de la clase a la tabla
    end
end

function draw()
    //gaBackground(0.0, 1.0) fondo anulado
    for i = 0, contador-1 do
        mod[i]:draw() //aplica el draw() de la clase a la tabla
    end
end

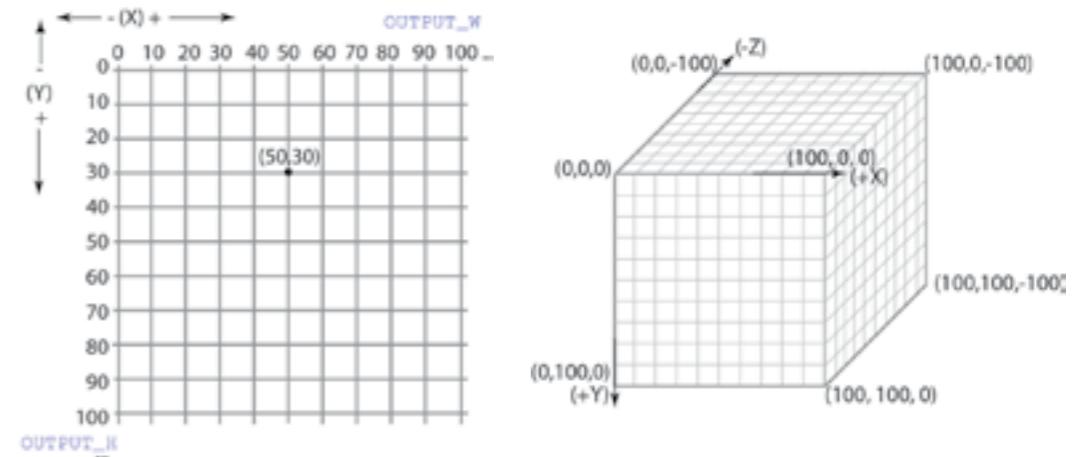
```

En lugar de nombrar uno por uno todos los objetos de la clase, asignándole manualmente el valor de sus parámetros, los despliega en una tabla 2D con un doble `for` para que se configure la cuadricula. En el ejemplo, para establecer el ancho y alto de las cuadriculas se utiliza la función matemática `math.floor()` que convierte el valor `float` en el número entero inferior. Así `math.floor(5.4)` devuelve 5.

5. Aplicaciones gráficas

Zenón no sólo mide longitudes, es también un geómetra de superficies. ¿Cuál es pues la superficie del Peloponeso? La misma razón recomienza. Supongamos un paso al cuadrado. La superficie de la península es el producto del número de tales cuadrados por su superficie media. Pero este cuadrado bien plano, bien llano, raras veces se pone de plano sobre el suelo. Si es grande, las montañas le hacen obstáculo; si es más pequeño, las colinas; si es aún más reducido, las rocas; y las motas de tierra, justamente, a escala del paso. Y en lo muy pequeño, las partículas de polvo, los cristales, y así sucesivamente. Idéntico resultado pues: ¿acaso tendría el mundo un volumen finito para una superficie infinita? La cosa resulta paradójica para su representación, como se suele decir, en el mapa.³⁹

La programación creativa, por su vinculación a las artes visuales y el diseño, contemplan una respuesta amplia a cuestiones visuales y espaciales. En este contexto, el primer elemento a comprender es el propio espacio de representación que queda definido por la ventana de salida. Sea cual sea su soporte, -tipo de monitor o proyector de vídeo- debemos entenderlo como una matriz de puntos luminosos (pixeles) que, según los casos, albergará una representación 2D ó 3D de las formas visuales, y cuyo origen o punto cero se expresa mediante las coordenadas $(0,0)$ ó $(0,0,0)$ y está situado en el vértice superior izquierdo de la ventana.



Centrándonos de momento en las formas 2D, para dibujarlas dentro de esa matriz es necesario pensar cuál será su ubicación, expresada en coordenadas (x, y) , y las cualidades que permiten definirla. Hay funciones internas que facilitan el dibujo de figuras geométricas e incluyen en sus parámetros las coordenadas y cualidades de la forma.

³⁹ Michel Serres (1991), *El paso del Noroeste*. Hermes V. Madrid: Debate, pág. 95.

5.1. Formas geométricas básicas

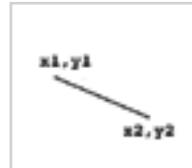
La siguiente tabla muestra las funciones de openFrameworks más comunes para dibujar figuras geométricas planas, lo que en programación se denomina Primitivas.



Punto

`ofCircle(x, y, 1)`

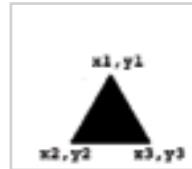
círculo de 1 pixel de diámetro
(coordenadas del centro, diámetro)



Línea

`ofLine(x1, y1, x2, y2)`

(coordenadas punto inicio, punto final)



Triángulo

`ofTriangle(x1, y1, x2, y2, x3, y3)`

(coordenadas punto 1, punto 2, punto 3)

el triángulo equilátero puede hacerse también con `ofCircle()`

Rectángulo

`ofRect(x, y, ancho, alto)`

(coordenadas vértice superior izq., lado, lado)

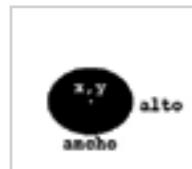
Puede cambiarse la asignación del vértice, ver RECT_MODE, página 72.



Círculo

`ofSetCircleResolution(valor)``ofCircle(x, y, radio)`

para un círculo el valor debe ser alto, si no dibuja polígonos



Elipse

`ofEllipse(x, y, ancho, alto)`

Polígono

`ofSetCircleResolution(valor)``ofCircle(x, y, radio)`

el valor define el nº de lados del polígono, en el ejemplo (5)

Antes de dibujar hay que tener en cuenta las características del fondo y particularidades de las formas: color de fondo, así como el color y ancho del borde y si ese color las rellena o no.

Funciones básicas, atributos de las formas

Color del fondo

`gaBackground(escalagris, alpha)`
`gaBackground(R,G,B,alpha)`

Esta función permite ajustar el color y transparencia del fondo, en escala de grises más transparencia con dos parámetros (gris, alfa) o en color con cuatro parámetros (R, G, B, A).

Color de relleno

Todos los valores van de **0.0 a 1.0**
A diferencia que en Processing, en GAMUZA tiene que estar en el `draw()``ofSetColor(R,G,B, alpha)`

Siempre hay que usarlo para definir el color cuando se va a generar algún gráfico, imagen o video. Si no se pone, no dibuja nada.

Si sólo tiene un parámetro funciona en escala de grises. 2 parámetros: escala de grises + alfa. 3 parámetros: color RGB y 4 parámetros: RGB + alfa. Los parámetros van de **0 a 255**.

Rellenar formas

`ofFill()`

No lleva parámetros

Se utiliza si previamente hay un `ofNoFill()` porque GAMUZA siempre rellena las formas por defecto.

No llenar

`ofNoFill()`

No lleva parámetros

Dibujar borde

`ofNoFill()`

Por defecto no los dibuja

No dibujar borde

Ancho de la línea

`ofSetLineWidth(valor)`

Valor = grosor línea en píxeles

Suavizar bordes

`ofEnableSmoothing()`

No lleva parámetros

No suavizar

`ofDisableSmoothing()`Se utiliza si previamente hay `ofEnableSmoothing()`. No lleva parámetros

Los siguientes ejemplos relacionan las funciones de formas geométricas básicas con sus atributos y estructuras de programación.

Los rectángulos y cuadrados tienen diferentes opciones para determinar el punto de referencia de las coordenadas que controlan su posición, para ello se utiliza la función: `ofSetRectMode()`. Algunos ejemplos permitirán revisar mejor lo hasta ahora visto.

```
/*
GAmuza 043      E-5-1
-----
OF_RECT_MODE
Cambia el punto de referencia
en rectángulos y cuadrados
*/
posXY = 300 // variables
lado = 400

function setup()
end

function update()
end

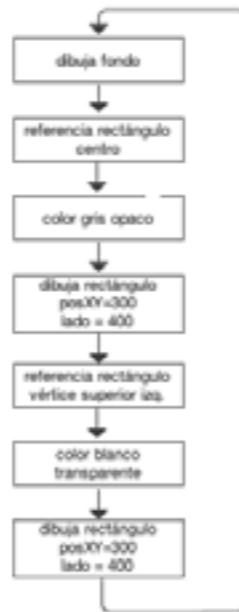
function draw()
    gaBackground(1.0, 1.0) // fondo blanco sin transparencia
    ofSetRectMode(OF_RECTMODE_CENTER) // referencia: centro
    ofSetColor(126)
    ofRect(posXY, posXY, lado, lado) // cuadrado gris
    ofSetRectMode(OF_RECTMODE_CORNER) // referencia: vértice superior izq.
    ofSetColor(0, 100)
    ofRect(posXY, posXY, lado, lado) // cuadrado negro transparente
end
```



El primer bloque de comentario da datos de lo que plantea el ejemplo y la versión del software que se ha utilizado. Cuando pasa el tiempo este dato cobra importancia porque el software suele actualizarse y con ello puede cambiar el código, por lo que nos da una referencia.

Hay dos variables globales, la primera actúa por dos para situar la coordenada de las formas, y es así porque en ambos cuadrados el valor de la coordenada *x* e *y* va a ser el mismo (300, 300), siguiendo una diagonal de 45°. La segunda variable indica el lado del cuadrado. Si queremos modificar el tamaño, manteniendo las proporciones de la imagen, simplemente se cambia el valor de esas variables.

El programa tiene datos solo en el bloque `draw()`. La primera línea de código determina el color del fondo; los parámetros (1.0, 1.0) indican blanco y opaco, si fueran (0.0, 0.5) significaría negro con 50% de transparencia. Los parámetros de la función `gaBackground()` siempre oscilan de 0.0 a 1.0.



La primera función `ofSetColor()` tiene solo un parámetro, lo que indica que actúa en escala de grises, la segunda tiene dos parámetros, escala de grises (negro) con una transparencia próxima al 40%. Los valores están compensados por lo que aparentemente dibujan el mismo color. Los parámetros de `ofSetColor()` oscilan entre 0 y 255

`ofSetRectMode()` es una función interna de openFrameworks que permite modificar las coordenadas de referencia para la posición de rectángulos y cuadrados. Por defecto las coordenadas de la función `ofRect()` sitúan el vértice superior izquierdo, que corresponde al parámetro `OF_RECTMODE_CORNER`, la variable `OF_RECTMODE_CENTER` cambia esa referencia al centro del cuadrado, por eso aunque los parámetros de las dos funciones `ofRect()` son iguales, los cuadrados se posicionan en lugares diferentes.

Cuando se utiliza una función que define atributos de las formas, se aplica a todas formas (funciones de las formas) que están situadas debajo y en las que puede incidir, si el color va a ser el mismo en todas, solo se pone una vez al principio. Aunque una función de dibujo sólo se utilice una vez, debe ponerse en el bloque `draw()`, en otros software, como Processing, pueden ponerse en el `setup()` pero en GAmuza las funciones de dibujo, como `ofNoFill()`, `ofFill()`, u `ofEnableLineWidth()`, deben ponerse en el `draw()`.

El sistema lee el código de arriba a abajo, esto afecta al orden de superposición de las formas: lo último escrito es lo que se sitúa al frente y las formas de las primeras líneas de código quedan más al fondo. En el ejemplo, el código del cuadrado negro está escrito después que el del gris, y por tanto se le superpone, aunque la transparencia disuelve el gradiente de profundidad por superposición y crea visualmente otro cuadrado.

En el siguiente ejemplo, once capas de círculos se entremezclan visualmente por transparencia disolviendo el gradiente de profundidad con las superposiciones. Todos tienen el mismo color `ofSetColor(0, 40)` definido desde el principio, color negro con una opacidad próxima al 15% porque los parámetros de esta función van de 0 a 255.

```
/* GAMuza 043           E-5-2
-----
Estructura repetición
*/
inc = 30
radioDec = 50

function setup()
    ofEnableSmoothing()
end

function draw()
    gaBackground(1.0, 1.0)
    ofSetCircleResolution(60)
    ofSetColor(0, 50)
    radio = 300          // variables locales
    x1 = OUTPUT_W/2      // vuelven a tener su valor inicial cada frame
    x2 = OUTPUT_W/2
    y = OUTPUT_H/2
    for i= 0, 5 do       // hace 6 repeticiones, de 0 a 5
        x1 = x1 + inc   // incrementa los valores en un mismo frame
        x2 = x2 - inc
        y = y + inc
        radio = radio - radioDec
        ofCircle(x1, y, radio) // dibuja 6 pares de círculos con los datos
        ofCircle(x2, y, radio)
    end
end
```



Las coordenadas del centro del primer círculo son (`OUTPUT_W/2`, `OUTPUT_H/2`), estos parámetros corresponden a las coordenadas del centro de la ventana de salida. Los centros de los otros círculos mantienen relaciones con ese punto, desplazándose 45° hacia abajo a ambos lados, porque sus dos coordenadas varían de forma regular y uniforme: los que se desplazan hacia la derecha suman 30 px en dirección X y 30 px en dirección Y, y los que lo hacen hacia la izquierda -30 px en la X y 30 px en la Y.

El siguiente ejemplo utiliza la función `ofSetCircleResolution()` para generar polígonos concéntricos mediante una estructura `for`, para el color utiliza el modelo HSB que se analiza en el capítulo 5.2. Para facilitar el cálculo de los parámetros de color se puede utilizar la herramienta **Color Selector**, en el menú **Tools** del IDE.

```
/*
GAMuza 043           E-5-3
-----
estructura for y color HSB
creado por mj
*/

radio = 100
radioMax = 250
radioInc= 40
posX = OUTPUT_W/2
posY = OUTPUT_H/2
grosor = 10
lados = 3
c = ofColor() // clase color de openFrameworks

function setup()
    ofEnableSmoothing()
end

function draw()
    gaBackground(1.0,1.0)
    ofNoFill()
    ofSetLineWidth(grosor)
    lados = 3
    j = 0           // variable local para dividir la escala cromática en 8 pasos

    for i=0, 6 do // el for da 7 vueltas
        ofSetCircleResolution(lados)
        c:setHsb((j/8)*255,255,255) // aplica el método de cambiar el modo RGB a HSB
        ofSetColor(c.r,c.g,c.b)     // parametros para pasar de RGB a HSB
        ofCircle(posX, posY, radio+(i*radioInc))
        j = j+1                  // incrementa j para el siguiente color
        c:setHsb((j/8)*255,255,255)
        ofSetColor(c.r,c.g,c.b)
        ofCircle(posX, posY, radio+(i*radioInc) + (grosor-1))
        lados = lados + 1         // incrementa cada vuelta un lado más al polígono
    end
end
```



5.1.1. Formas irregulares

También se pueden generar formas irregulares lineales o curvas. Las formas lineales requieren las funciones:

```
ofBeginShape()           // sin parámetros, indica inicio forma
ofVertex(x, y)          // coordenadas x, y del vértice
ofEndShape(true o false) // booleano para cerrar, o no, la forma
```

`ofBeginShape()` se utiliza como aviso de que se va a iniciar una nueva forma, esta forma estará definida por una serie de vértices con `ofVertex(x,y)` y tras ellos debe terminar con la función `ofEndShape()`. Su variable booleana indica si la forma se cierra, es decir si el último vertex se une al primero, cuando es `true`, o si queda abierta cuando es `false`.

```
/*
GAMuza 043           E-5-4
-----
Formas irregulares lineales

*/
function setup()
    ofEnableSmoothing()
end

function draw()
    ofBackground(1.0,1.0)      // fondo blanco opaco
    ofNoFill()                // sin relleno
    ofSetLineWidth(3)          // grosor línea
    ofSetColor(0)              // color negro
    ofBeginShape()             // aviso empieza forma
        ofVertex(146, 75)      // coordenadas de los vértices
        ofVertex(210, 117)
        ofVertex(210, 589)
        ofVertex(226, 589)
        ofVertex(190, 217)
        ofVertex(254, 132)
        ofVertex(310, 117)
    ofEndShape(false)          // fin forma: abierta
end
```

Para las formas no lineales hay dos opciones: curvas o bezier, y en ambos tipos se da la posibilidad de construcción con vértices:

```
ofCurve(cx1, cy1, x1, y1, x2, y2, cx2, cy2)
ofCurveVertex(x, y)
ofBezier(x1, y1, cx1, cy1, cx2, cy2, x2, y2)
ofBezierVertex(cx1, cy1, cx2, cy2, x2, y2)
```

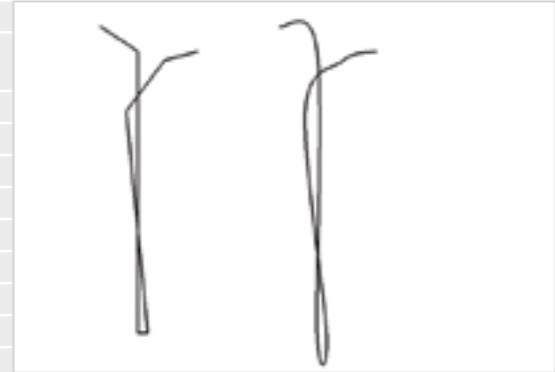
La función `ofCurveVertex(x,y)` también necesita señalar el principio y fin de la forma, y lo hace con las mismas funciones: `ofBeginShape()` y `ofEndShape()`. Deben utilizarse al menos cuatro funciones `ofCurveVertex(x,y)` porque en la primera y la última, los parámetros `(x,y)` corresponden a los puntos de control, siendo los parámetros de las otras funciones los que definen los vértices de la curva,

El siguiente ejemplo compara la función `ofCurveVertex()` con el anterior de formas lineales realizado con `ofVertex()`, desplazando el valor de las coordenadas x, además se repiten la primera y última función por ser las de control.

```
/*
GAMuza 043 ejemplos   E-5-5
-----
Compara ofVertex y ofCurveVertex

*/
// pegad aquí ejemplo anterior
// sin el end final

ofBeginShape()
    ofCurveVertex(446, 75)      // define punto de control 1: no se dibuja
    ofCurveVertex(446, 75)      // inicia la curva
    ofCurveVertex(510, 117)
    ofCurveVertex(510, 589)
    ofCurveVertex(490, 217)
    ofCurveVertex(554, 132)
    ofCurveVertex(610, 117)      // termina la curva
    ofCurveVertex(610, 117)      // define punto de control 2: no se dibuja
    ofEndShape(false)           // fin forma: no cierra
end
```



Las funciones `ofCurve()` y `ofBezier()` no requieren señalar el inicio y final de una forma. En ambas, sus 8 parámetros indican dos coordenadas de principio y fin de las curvas (`x1, y1, x2, y2`) y dos coordenadas de puntos de control (`cx1, cy1, cx2, cy2`), desde los que se genera la curva o el trazado de la bezier. El orden de estos parámetros no es el mismo en las dos funciones. Veamos un ejemplo con las mismas coordenadas en ambas funciones para observar sus diferencias:

```

/*
GAMUZA 043          E-5-6
-----
comparación ofCurve y ofBezier

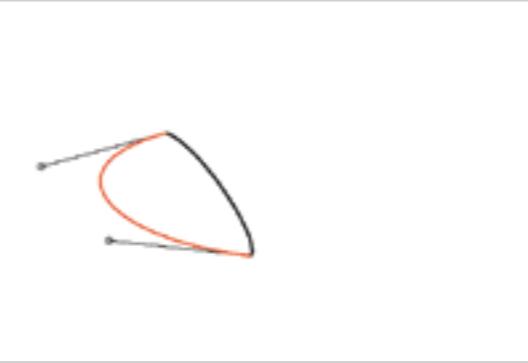
*/
function setup()
end

function draw()
gaBackground(1.0,1.0)
ofSetColor(0)
ofNoFill()

    // representan los puntos de control
    ofCircle(25, 254, 4) // para ver dónde va el punto control 1 (cx1, cy1)
    ofCircle(115, 350, 4) // y dónde el punto de control 2 (cx2, cy2)
    ofLine(25, 254, 190, 210) // representa la linea de control 1, inicio curva
    ofLine(115, 350, 300, 370) // representa la linea de control 2, fin curva

    ofSetLineWidth(3) // grosor línea
    //ofCurve(cx1, cy1, x1, y1, x2, y2, cx2, cy2) para ver orden coordenadas
    ofCurve(25, 254, 190, 210, 300, 370, 115, 350) // curva color negro
    ofSetColor(255, 0, 0)
    //ofBezier(x1, y1, cx1, cy1, cx2, cy2, x2, y2) para ver orden coordenadas
    ofBezier(190, 210, 25, 254, 115, 350, 300, 370) // bezier color rojo
end

```



The diagram illustrates the decomposition of a Bezier curve. It shows a red Bezier curve segment with its start and end points. Two black line segments, representing control lines, extend from these points to two small circles representing control points. These control points are labeled with their coordinates: (25, 254) and (115, 350).

```

/*
GAMUZA 043          E-5-7
-----
Curvas bezier encadenadas

*/
function setup()
end

function draw()
gaBackground(1.0,1.0)
ofNoFill()
ofSetCurveResolution(60) // para mejorar el trazado de la curva
ofSetLineWidth(3) // grosor línea
ofSetColor(0) // color negro
ofBeginShape() // aviso empieza forma
ofVertex(60, 120) //inicia curva 1
ofBezierVertex(80, -20, 280, 20, 160, 140)
    // control 1, control 2, fin curva 1 que es inicio de la 2
ofBezierVertex(10, 280, 180, 420, 280, 280) // curva 2
ofBezierVertex(350, 175, 440, 230, 390, 330) // curva 3
ofEndShape(false) // fin forma: abierta
end

```



The diagram shows three Bezier curves connected sequentially. The first curve starts at (60, 120) and ends at (80, -20). The second curve starts at (80, -20) and ends at (10, 280). The third curve starts at (10, 280) and ends at (350, 175). All curves are drawn in black.

Por último, la función `ofBezierVertex(cx1, cy1, cx2, cy2, x2, y2)` como tiene sólo 6 parámetros, debe estar precedida siempre por `ofVertex(x,y)` cuyas coordenadas indican el inicio de la curva. Los parámetros (`cx1, cy1, cx2, cy2,`) de `ofBezierVertex()` posicionan los dos puntos de control, y (`x2, y2`) el final de la curva. Si se encadenan varias funciones seguidas, esta última coordenada indica dónde empezará la siguiente. También deben situarse entre `ofBeginShape()` y `ofEndShape()`.

5.2. Color

¿Porqué no se puede imaginar un vidrio blanco transparente —inclusive si en la realidad no lo hay?
 ¿En dónde se pierde la analogía con el vidrio transparente coloreado? ⁴⁰

El color es una sensación perceptiva cuyo estudio recae en muy distintas disciplinas: psicología de la percepción, física-óptica, fisiología, neurología, filosofía....

Según el enfoque de la disciplina, se le considera como un fenómeno visual vinculado a la luz y localizado físicamente en una porción del espectro electromagnético (espectro lumínico), o como sinestesia entre el sujeto que percibe y el objeto percibido dentro de la psicología, o como un proceso neuro-fisiológico complejo.

Como fenómeno que depende de la luz, el color recoge de ella sus paradojas, llegando a ser un claro ejemplo para mostrar la contradicción entre el mundo tangible y el intangible. Así, todas las relaciones que se establecen entre los componentes del color pigmento (materia tangible) y el color-luz (materia intangible) son contrarias.

Color-pigmento

Mezcla sustractiva

La mezcla de sus primarios es negro

Colores primarios

Magenta, Cian, Amarillo

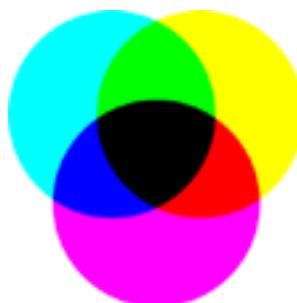
Colores secundarios

Azul, Verde, Rojo

Magenta + Cian = Azul

Cian + Amarillo = Verde

Amarillo + Magenta = Rojo



Color-luz

Mezcla aditiva

La mezcla de sus primarios es blanco

Colores primarios

Azul, Verde, Rojo

Colores secundarios

Magenta, Cian, Amarillo

Rojo + Azul = Magenta

Azul + Verde = Cian

Verde + Rojo = Amarillo



Solo se ponen de acuerdo en la relación entre complementarios u opuestos

Complementarios

Cian Rojo

Magenta Verde

Amarillo Azul

Los ordenadores utilizan color-luz porque los materiales de la pantalla o video proyector fosforecen con la mezcla aditiva. Pero existen distintos modelos o espacios de color; que son el resultado de sistematizar las relaciones que se producen entre los componentes o parámetros del color bajo un determinado criterio.

Los tres parámetros del color: tono, saturación y luminosidad, están relacionados de manera distinta en cada modelo, incluso utilizan nombres distintos para esos parámetros. El tono a veces lo encontraremos bajo la denominación de color o matiz; y la luminosidad como valor, brillo o intensidad. El término saturación suele permanecer bastante estable en los distintos modelos. Los modelos más habituales en programación son: RGB, HSB (o HSV) y hexagesimal (principalmente vinculado a la web).

MODELO RGB

El modelo RGB se basa en los colores-luz primarios (Red, Green, Blue - Rojo, Verde, Azul) más otro canal denominado alpha que se ocupa de la transparencia. Cada dato de color RGB+alpha tiene 32 bits, 8 bits por canal y cada uno de estos canales tiene un rango de valores que va de 0 a 255, siendo el valor (0,0,0) el negro, y el valor (255, 255, 255) el blanco.

MODELO HSB

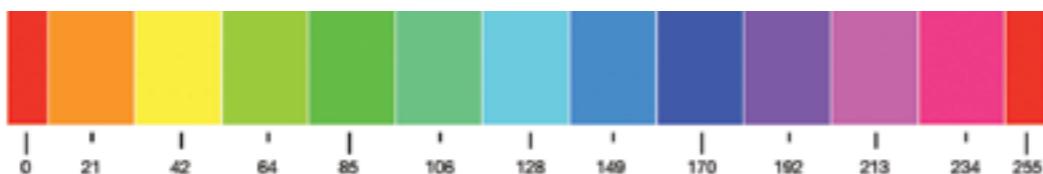
El modelo HSB hace referencia a los parámetros (Hue, Saturation, Brightness – Matiz, Saturación, Brillo), es muy semejante al modelo HSV, donde V significa Value (valor) pero en ambos casos, brillo y valor es lo mismo que hemos denominado luminosidad. El tono o matiz recorre el círculo cromático graduándose de 0 a 360°, mientras que los valores de saturación y luminosidad se escalan entre 0 y 100%.

La clase `ofColor()` de openFrameworks puede trabajar con el modelo HSB a través del método `setHsb()`, pero el rango de sus valores es el mismo que utiliza el modelo RGB (todos los parámetros

⁴⁰ Ludwig Wittgenstein (1994) *Observación sobre los colores*. Barcelona:Paidós, pág. 175.

van de 0 a 255) y el valor de estos parámetros es el que se utiliza en GAmuza al incorporar esa clase. Este cambio de modo de color tiene la ventaja de mantener una lógica que se adapta mejor en trabajos con graduaciones de escalas tímbricas de color.

En el primer parámetro, tono (Hue), hay que escalar los 360° a 256 valores. La siguiente imagen clarifica este cambio, desplegado el círculo linealmente y con la conversión de valores que se utilizan en GAmuza para los colores-luz primarios, secundarios e intermedios.



El valor 255 y 0 corresponden al mismo color rojo, punto en el que se cerraría el círculo cromático.

La herramienta Color Selector de GAmuza, ayuda a calcular los valores y ver las relaciones entre los dos modelos, porque al mover los sliders de uno se ajustan también los del otro.

Los colores luz primarios en RGB tienen el máximo valor en su parámetro correspondiente y 0 en los otros dos.

R=0, G=0, B=255 Azul con máxima luminosidad y saturación.

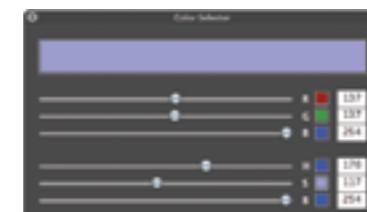
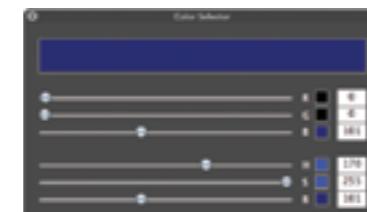
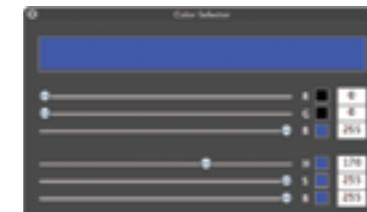
En HSB, simplemente desplazar el primer slider (Hue) hasta percibir el color (170), y mantener los otros dos con el valor máximo, 255.

Pérdida de luminosidad en los colores primarios

Si los valores del ejemplo anterior se modifican a R = 0, G=0, B=128, veremos un azul oscuro por la pérdida de luminosidad al 50%. Según va bajando el valor del azul hacia 0, y los otros permanecen igual, se llega al negro. En HSB, ir bajando el slider B hacia el 0

Pérdida de saturación en los colores primarios

Volvemos al valor B = 255, conforme G y B van incrementando sus valores de forma equivalente, el azul se irá desaturando. Cuando todos alcancen el valor 255 llegarán al blanco. En HSB, ir bajando el slider S hacia el 0



Colores-luz secundarios

En RGB se obtienen incrementando valores iguales en solo dos componentes y dejando el tercero a 0. En HSB, buscar el tono con el slider H (hue) y mantener los otros dos con el valor máximo, 255.

R=255, G=0, B= 255 Magenta máxima luminosidad y saturación

Pérdida de luminosidad en color secundario

Si los valores son R=100, G=0, B=100 veremos un magenta oscurecido, según vayan bajando los valores irá tendiendo a negro. En HSB, bajar hacia el valor 0 el slider B.



Pérdida de saturación en color secundario

Se desaturan incrementando el valor de ese tercer componente. En el ejemplo anterior, R = 255, G=170, B= 255, es un magenta desaturado, si los tres alcanzan 255, llegan a blanco. En HSB, bajar hacia el valor 0 el slider S.

Los **colores terciarios** surgen al mezclar en la misma proporción un primario y un secundario que no lo contiene, es decir, con su color complementario al 50%. En este nivel surgen los marrones, o tierras, dejando las escalas tímbricas de colores más puros para entrar en las escalas tonales⁴¹.

Para traducir estas relaciones a valores RGB, los colores terciarios con máxima saturación y luminosidad se obtienen combinando en proporciones distintas dos componentes del color, dejando el tercero con un valor 0, o muy diferenciado de los otros. En HSB son tonos que siempre tienen un descenso de luminosidad.

Para desaturarlos en RGB se incrementa proporcionalmente el valor de los tres componentes. En HSB se baja el valor de S.

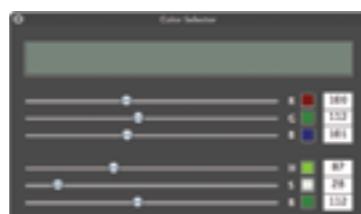


⁴¹ Gillo Dorfles, (1963) *El devenir de las artes*, México: Fondo de Cultura Económica, pág. 90
Con esta distinción Dorfles aproximó los términos del color a los de la música..

Para reducir la luminosidad se disminuye proporcionalmente el valor de los dos componentes iniciales. En HSB, bajar el valor de B.



Si en RGB se mantienen los tres valores próximos pero no iguales, se obtienen colores grisáceos con dominante del que tiene, o los dos que tienen, el valor superior, al incrementarlos proporcionalmente se desaturan y al reducirlos van perdiendo luminosidad.



Como se ha mencionado antes, en GAmuza el modelo de color HSB se utiliza a través de la clase de openFrameworks `ofColor()`. El proceso de uso en el código es el siguiente:

Se vincula una variable global con la clase:

```
nombreVariable = ofColor()
```

Y en el bloque `draw()` esa variable establece el puente o traducción de un sistema a otro a través del método `setHsb()`. Todos los métodos de las clases van precedidos de dos puntos ":"

```
nombreVariable:setHsb(valor1, valor2, valor3) // violeta sería (192,255,255)
```

Después se pone la función habitual para definir el color, pero en sus parámetros se vincula la variable

```
ofSetColor(nombreVariable.r, nombreVariable.g, nombreVariable.b)
```

Las clases tienen determinados métodos asignados, los vinculados a `ofColor()` permiten una gran variedad funciones, además de pasar el modelo de color a HSB, puede hacer una interpolación lineal entre colores, con el método `lerp()`, o invertir los colores con `invert()`⁴².

En el siguiente ejemplo utiliza la función básica `ofSetColor()` para asignar color a una forma, pero este color varía según su posición vertical, este valor RGB se mapea con la función `ofMap()`. Esta función tiene 5 parámetros: `ofMap(valor, inputMin, inputMax, outputMin, outputMax)`. El primero es el dato que se utiliza para mapear, en este caso será la posición Y. El segundo y tercer parámetro corresponde a los valores mínimo y máximo que puede alcanzar ese primer dato de entrada. El cuarto y quinto parámetro son los valores mínimo y máximo que queremos obtener de salida.

⁴² Se pueden ver los métodos de esta clase en <<http://www.openframeworks.cc/documentation/types/ofColor.html>> [29.07.2012]

```
/*
GAmuza 043 E-5-8
-----
Mapear color HSB
*/

radio = 60
x = ofRandom(200, OUTPUT_W)
y = ofRandom(0, OUTPUT_H/6)
vx = 0
vy = 0
gx= ofRandom(-0.18326,0.44326)
gy = 0.0184562 * radio
c= ofColor()
tono = 0

function setup()
end

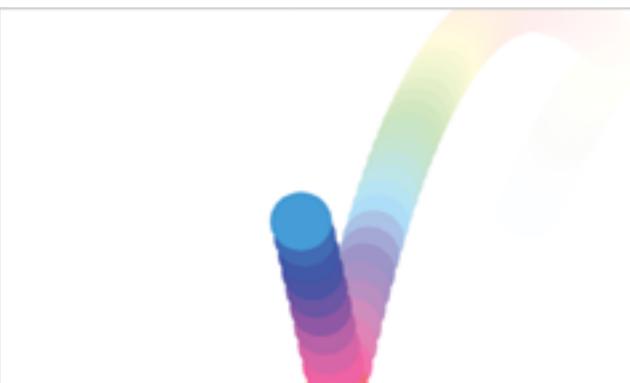
function update()
    x += vx // posición X más velocidad
    y += vy // posición Y más velocidad
    vx += gx // velocidad de X más gravedad
    vy += gy // velocidad de Y más gravedad

    if x < radio or x > OUTPUT_W-radio then
        vx *= -0.9876 // invierte velocidad al tocar los bordes en X
    end

    if y+radio > OUTPUT_H then
        vy *= -0.9876 // invierte velocidad al tocar los bordes en Y
    end

    tono = ofMap(y,0,OUTPUT_H,0,255) // mapea el valor del color según el valor de Y
end

function draw()
    gaBackground(1.0,0.1) // fondo blanco transparente
    c: setHsb(tono, 255, 255) // utiliza los valores mapeados del tono
    ofSetColor(c.r, c.g, c.b) // código para usar color HSB
    ofCircle(x, y, radio)
end
```



La función `ofSetColor()` se debe poner siempre que se dibuja algo, ya sea una forma, una imagen o un vídeo, en estos dos últimos casos se pone color blanco si queremos que esa imagen o video se reproduzcan con sus propios colores, si se utiliza otro color esta función actúa como tinte, como se muestra en el siguiente ejemplo. [Ver capítulo 6.2 archivos de imagen]

```
/*
GAMUZA 043           E-5-9
-----
Tinte imagen
creado por n3m3da | www.d3cod3.org
*/
img = ofImage()

function setup()
    img:loadImage(gaImportFile("highlands.jpg"))
end

function draw()
    gaBackground(0.0,1.0)

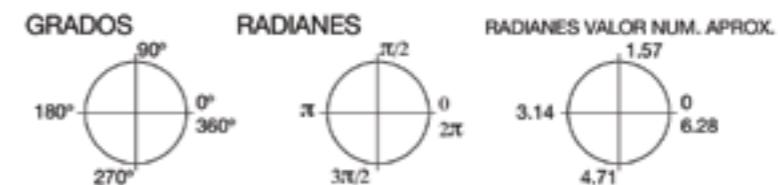
    // para centrar y escalar la imagen
    w = OUTPUT_W
    h = (w/img:getWidth())*img:getHeight()
    posX = 0
    posY = (OUTPUT_H-h)/2
    // dibuja la imagen
    ofSetColor(255, 0, 0) // tiñe la imagen de rojo
    img:draw(posX,posY,w,h)
end
```



5.3. Curvas por trigonometría y funciones matemáticas

La trigonometría es una rama de las matemáticas cuyo significado etimológico es "la medición de los triángulos" y sirve para definir relaciones entre sus lados y ángulos como extensión del Teorema de Pitágoras. Las funciones trigonométricas que más se utilizan son las del seno y coseno, con ellas se generan números repetitivos que pueden ser utilizados para dibujar ondas, círculos, arcos y espirales.

La unidad de medida angular propia de la trigonometría es el radian, en una circunferencia completa hay 2π radianes.



Esta unidad puede traducirse a Grado sexagesimal: unidad angular que divide una circunferencia en 360 grados; o al Grado centesimal: divide la circunferencia en 400 grados centesimales.

openFrameworks cuenta con las siguientes constantes para trabajar con unidades radianes:

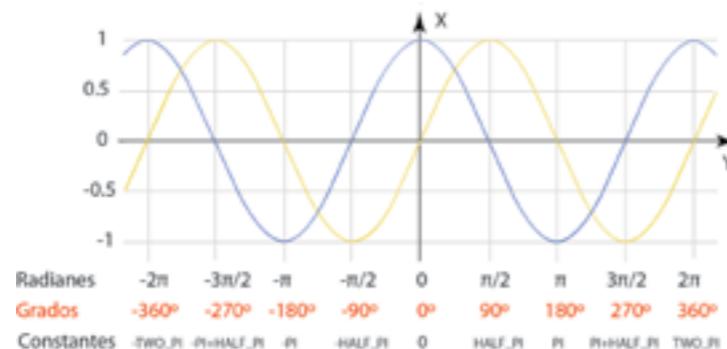
```
PI
TWO_PI
TAU // Similar a TWO_PI
FOUR_PI
HALF_PI
```

Lua tiene la constante `math.pi` para el valor π

Para convertir grados a radianes y viceversa, pueden utilizarse las funciones de Lua `math.deg()` y `math.rad()` y también las funciones de openFrameworks `ofDegToRad()` y `ofRadToDeg()`.

Las funciones `math.sin(x)` y `math.cos(x)` se utilizan para determinar el valor del seno y coseno de un ángulo en radianes. Ambas funciones requieren un parámetro, el ángulo. El siguiente gráfico orienta los valores del seno y coseno en función de los ángulos⁴³.

⁴³ Más información sobre funciones matemáticas Lua en <<http://lua-users.org/wiki/MathLibraryTutorial>> [30.07.2012]



Otras funciones matemáticas para convertir los datos obtenidos en números positivos o transformar datos con decimales en enteros (números enteros) son:

`math.abs` - Devuelve el valor absoluto (no negativo) de un dato dado

`math.abs(-100)` devuelve 100

`math.abs(25.67)` devuelve 25.67

`math.ceil` - si el parámetro es un número con decimales, devuelve el número entero superior

`math.ceil(0.5)` devuelve 1

`math.floor` - si el parámetro es un número con decimales, devuelve el número entero inferior

`math.floor(0.5)` devuelve 0

`math.modf` - A partir de un dato dado, devuelve dos valores, uno entero y otro la fracción

`math.modf(5)` devuelve 5 0

`math.modf(5.3)` devuelve 5 0.3

`math.modf(-5.3)` devuelve -5 -0.3

`math.pow` - Tiene dos parámetros y eleva el primero a la potencia del segundo

`math.pow(7, 2)` devuelve 49 [siete elevado a 2]

`math.pow(3, 2.7)` devuelve 19.419023519771 [3 elevado a 2.7]

Para ver cómo funciona gráficamente la función trigonométrica `math.sin()`, retomamos un ejemplo de Ben Fry y Casey Reas⁴⁴, traducido al lenguaje de programación de GAMUZA, porque permite observar qué parámetros controlan la amplitud y frecuencia de las curvas utilizando la función `math.sin()` en una estructura `for`.



```
/*
GAMUZA 043      E-5-10
-----
```

Onda sinusoidal

*/

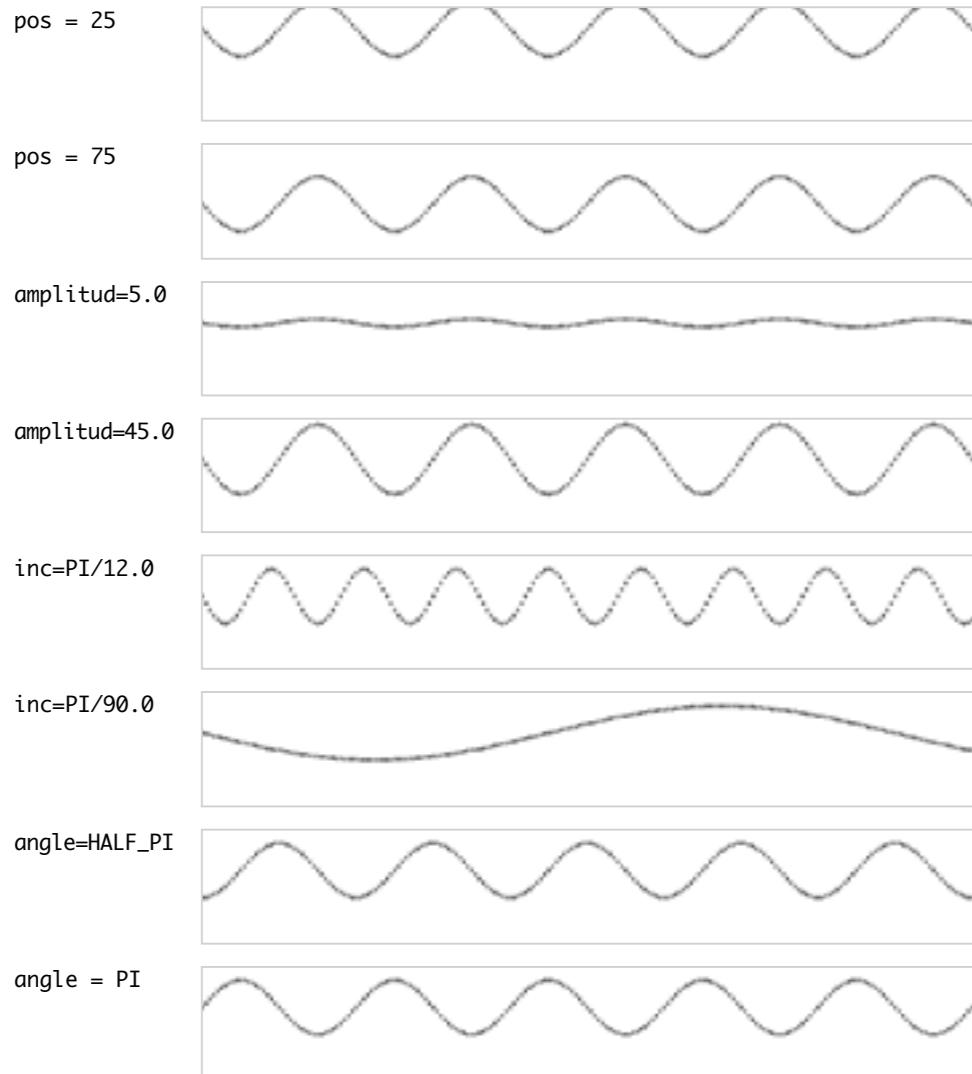
```
_angle = 0.0
pos = OUTPUT_H/2                                // posición Y
amplitude = 40.0                                 // altura de la onda
inc = PI/30.0                                    // incremento del ángulo

function draw()
    gaBackground(1.0,1.0)
    ofSetColor(0)
    for x = 0, OUTPUT_W, 5 do
        y = pos + (math.sin(_angle) *amplitude)
        ofRect(x, y, 2, 4)
        _angle = _angle + inc
    end
    _angle = 0.0
end
```

Siguiendo el ejemplo de Ben Fry y Casey Reas, si se modifican los valores asignados a las variables se puede observar los cambios en la frecuencia de la onda, teniendo en cuenta que la variable `pos` define la coordenada Y de la onda, `amplitude` controla la altura, e `inc` el incremento del ángulo.

⁴⁴ Casey Reas y Ben Fry, (2007) *Processing, a Programming Handbook for Visual Designers and Artist*. Cambridge(MA): The MIT Press, pág. 119-120.

Las siguientes imágenes muestran los cambios de frecuencia al modificar esos valores y el ángulo.



En el siguiente ejemplo se utilizan los datos que devuelven las funciones `math.sin(x)` y `math.cos(x)` para calcular los centros de una espiral de círculos, haciendo una conversión de grados a radianes con la función `math.rad()`

```
/*
```

GAMUZA 043 E-5-11

Trigonometría espiral

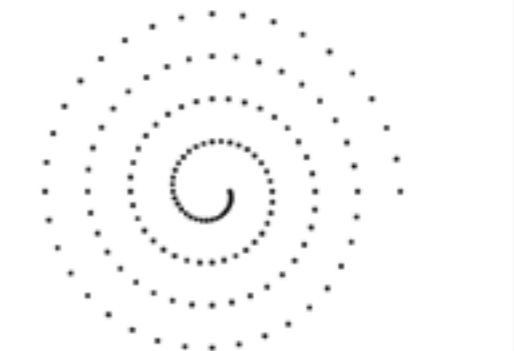
*/

radio = 30 //variable global

```
function setup()
    ofEnableSmoothing()
end
```

```
function draw()
    ofBackground(1.0, 1.0)
    ofSetColor(0)
```

```
radio = 30
for grade = 0, 360*4, 10 do
    _angle = math.rad(grade) //variables locales
    x = OUTPUT_W/2+ (math.cos(_angle) * radio)
    y = OUTPUT_H/2+ (math.sin(_angle) * radio)
    ofCircle(x, y, 4)
    radio = radio + 1 // incremento del radio
end
end
```



La espiral se genera por pequeños círculos cuyo centro se va alejando progresivamente del centro de la pantalla a la vez que giran alrededor de él.

Para calcular la posición de esos centros se utiliza una estructura `for` que recorre cuatro veces los 360° de un círculo de 10 en 10 grados. El resultado de los grados que se obtienen (10, 20, 30, 40,...) se pasa a radianes mediante la función de Lua `math.rad()`, por ejemplo `math.rad(180) = 3.1415926535898`, es decir `PI`.

El alejamiento progresivo del centro se produce al incrementar la variable `radio`, si se comentara `radio=radio+1`, el código dibujaría un círculo de círculos.

La repetición del valor inicial de la variable, `radio = 30`, en el `draw()` y fuera del `for` , sirve para detener el crecimiento en loop de su valor en cada frame.

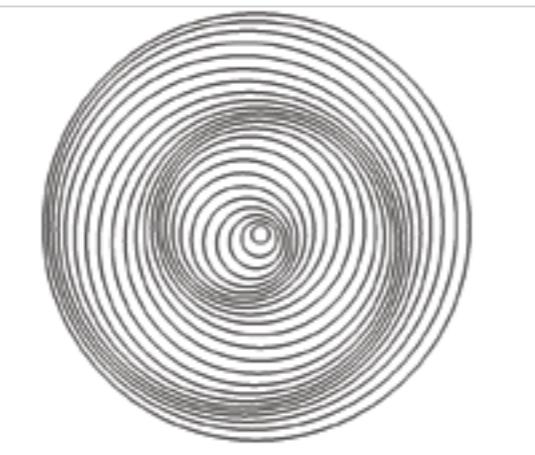
Una variación del este ejemplo nos lleva a la configuración de obra *Rotating Disc* (1925) de Marcel Duchamp. En el apartado 3.10 volveremos a ella para aplicarle la rotación, de momento analizamos el código para generar la imagen fija.

```
/*
GAMUZA 043           E-5-12
-----
Artists/Duchamp - Rotoreliefs
creado por n3m3da & mj

*/
radio = 15
spiralFactor = 15
numCircles = 23

function setup()
    ofEnableSmoothing()
    ofSetCircleResolution(50)
end

function draw()
    gaBackground(1.0, 1.0)
    ofSetColor(0)
    ofNoFill()
    ofSetLineWidth(3)
    radio = 15
    for i = 0, numCircles do
        _angle = i*TWO_PI/((numCircles/2) +1)
        x = (OUTPUT_W/2) + (math.cos(_angle) * spiralFactor)
        y = (OUTPUT_H/2) + (math.sin(_angle) * spiralFactor)
        ofCircle(x, y, radio)
        radio = radio + spiralFactor
    end
end
```



En este caso no se utilizan los grados del círculo en la estructura `for`, sino el número de repeticiones (círculos a dibujar).

El centro de todos los círculos está situado en el perímetro de un círculo no dibujado, cada uno de ellos separado por 45° . La diferencia de sus radios, `spiralFactor`, se corresponde al valor de ese mismo círculo invisible que distribuye la tangencia de los otros, es esa tangencia lo que produce la percepción de una espiral.

También es usual utilizar las funciones de generación de formas irregulares para dibujar curvas con trigonometría. El siguiente ejemplo muestra cómo dibujar un arco de circunferencia con `ofVertex()` más las funciones de seno y coseno.

```
/*
GAMUZA 0.4.3           E-5-13
-----
Arco de circunferencia

*/
resolution = 20
radio = 200
x = OUTPUT_W/2-radio/2
y = OUTPUT_H/2

function draw()
    gaBackground(1.0,1.0)
    ofSetColor(0)
    ofNoFill()
    ofSetLineWidth(3)
    ofBeginShape()
        for i = 0, resolution do
            _angle = i*PI /resolution
            ofVertex(x + (math.sin(_angle) * radio), y + (math.cos(_angle) * radio))
        end
    ofEndShape(false)
end
```



Si se reduce el valor de la variable `resolution` se dibujan arcos poligonales (de manera semejante a reducir el parámetro de `ofSetCircleResolution()`). Si se utiliza el seno y coseno de `_angle/2` dibujará el arco de un cuarto de circunferencia.

En el siguiente ejemplo, las funciones trigonométricas participan de manera indirecta en la generación de la forma, porque se utilizan para trasladar y girar una línea generando, por transparencia, una forma de revolución con velocidades angulares. Enlazamos así con el siguiente capítulo en el que se tratan los sistemas de transformación para trasladar, rotar y escalar formas.

```
/*
GAMUZA 043x           E-5-14
-----
```

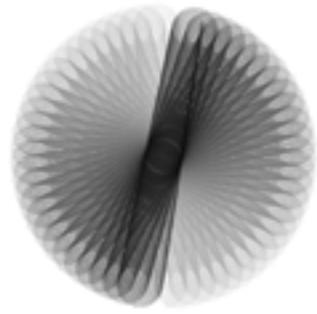
Trigonometría: trasladar y girar

```
/*
x = OUTPUT_W/2
y = OUTPUT_H/2
_angle = 0
range = 0
speed = 2.5

function setup()
    range = ofRandom(-0.6, 0.6)
end

function update()
    _angle += range
    x += math.cos(_angle)*speed
    y += math.sin(_angle)*speed
end

function draw()
    gaBackground(1.0,0.001)
    ofSetColor(0, 100)
    ofTranslate(x, y, 0)      // Traslada el punto (0, 0) al que definen x e y
    ofRotate(_angle)
    ofLine(0, -180, 0, 180)
end
```



5.4. Transformar: traslación, rotación, escalar

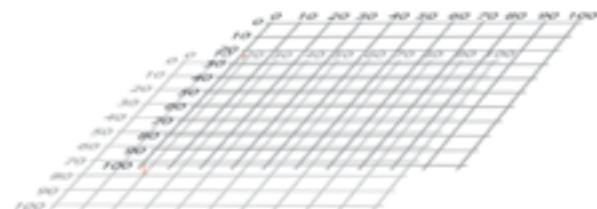
Ahora bien, sabemos, por un conocido pasaje de la República, que el espín de la peonza le había planteado a Platón el difícil problema del reposo y el movimiento. Porque dando vueltas es estable, contradicción. Platón se las ingenia para afirmar que ésta está en reposo en relación con lo recto, y en movimiento en relación con lo redondo, lo cual es verdadero sólo a condición de ignorar que su eje es tanto más estable cuanto más rápido gira. Los mecánicos griegos no conocían, por cierto, el teorema, pero el hecho, supongo, jamás fue ignorado por los niños. Éstos juegan con la contradicción que retrasa al filósofo. Gozan del reposo en y por el movimiento circular. De ahí que se pueda gozar de lo que da miedo, para volver al texto de Platón. La peonza no es un mal fármaco, veneno y remedio.⁴⁵

Sintaxis:

ofPushMatrix(), **ofPopMatrix()**
ofTranslate(float, float, float), **ofRotate(float)** , **ofScale(float, float, float)**

La matriz de píxeles en la que se representan las formas mediante coordenadas, puede transformarse trasladándola, girándola, o escalándola, es decir, no giramos y trasladamos los objetos sino la textura de salida en su conjunto.

Por esta particularidad antes de hacer estas transformaciones se debe (se recomienda) utilizar las funciones **ofPushMatrix()** y **ofPopMatrix()** para controlar ese desplazamiento. Estas funciones siempre van juntas marcando el inicio y el final del proceso. Es como si se generara una textura paralela sobre la ventana de salida donde se aplican las funciones.



La función **ofTranslate(x, y, z)** mueve la textura de salida desde su coordenada $(0, 0, 0)$ hasta el punto que se indiquen los parámetros, si se está trabajando en 2D la coordenada z es 0 . Solo las figuras situadas después de la función se verán afectadas por la translación.

En las transformaciones de rotación suelen utilizarse tanto las funciones **ofPushMatrix()** y **ofPopMatrix()**, como **ofTranslate()**, que traslada el punto $(0, 0)$ de la pantalla al eje de rotación. Se pueden apilar tantas superposiciones de la textura de salida como funciones se pongan en el código. La función **ofRotate()** puede tener uno o cuatro parámetros **float**, con uno se especifican los grados a girar, cuando tiene cuatro parámetros el primero son los grados y los tres últimos las coordenadas

⁴⁵ Michel Serres (1991). *El paso del Noroeste*. Hermes V. Madrid: Debate, pág. 166.

X, Y, Z del vector de rotación. También existen las funciones `ofRotateX(float)` `ofRotateY(float)` y `ofRotateZ(float)`, cuyo parámetro es los grados de giro en la coordenada que indica cada función.

Retomamos el ejemplo *Rotating Disc* de Duchamp para ver estas funciones con la espiral en movimiento.

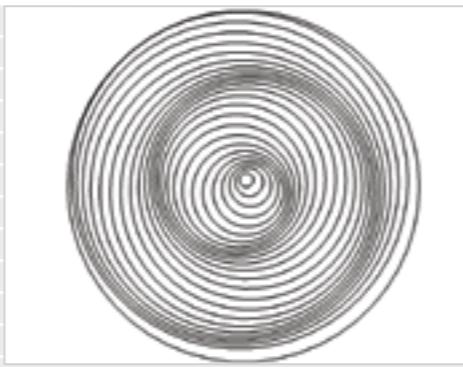
```
/*
GAMUZA 043 E-5-15
-----
Duchamp Rotoreliefs en movimiento
creado por mj & n3m3da
*/

radio = 15
spiralFactor = 15
numCircles = 23
inc = 15
counter = 0
radioMax = (radio*(numCircles+1)) + (spiralFactor+1)

function setup()
    ofEnableSmoothing()
end

function draw()
    gaBackground(1.0, 1.0)
    ofSetCircleResolution(50)
    ofSetColor(0)
    ofNoFill()
    ofSetLineWidth(3)
    ofCircle(OUTPUT_W/2, OUTPUT_H/2, radioMax)
    radio = 15

    ofPushMatrix()
    ofTranslate(OUTPUT_W/2, OUTPUT_H/2, 0.0)
    ofRotate(counter)
    for i = 0, numCircles do
        angulo = i*TWO_PI/((numCircles/2) +1)
        x = (math.cos(angulo) * spiralFactor)
        y = (math.sin(angulo) * spiralFactor)
        ofCircle(x, y, radio)
        radio = radio + spiralFactor
    end
    ofPopMatrix()
    counter = counter + 1
end
```



La estructura `for` que dibuja los círculos está después de las funciones `ofPushMatrix()`, `ofTranslate()`, `ofRotate()` que, traducido de forma simplificada es, emula una nueva capa de textura de salida, traslada su punto cero al centro y gira un número de veces (`counter`) que se va incrementando cada vez que el `for` ha terminado (`counter +1`), después se resitúa la textura de salida `ofPopMatrix()` y vuelve a empezar el loop que hace el bloque `draw()`.

Algunas veces las funciones `ofTranslate()` o `ofRotate()` pueden ir sin `ofPushMatrix()` y `ofPopMatrix()`, e incluso situarse dentro de un `for`, de modo que la translación se repita en cada paso del loop. En el siguiente ejemplo esa translación da la sensación de 3D, siendo una imagen 2D.

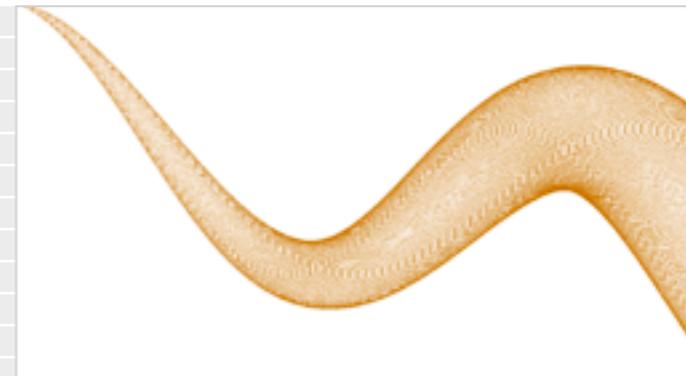
```
/*
GAMUZA 043 E-5-16
-----
Repetición de translación
*/

angulo = 0
posY = 0
amplitud = 5

function setup()
    ofEnableSmoothing()
    ofSetCircleResolution(50)
end

function draw()
    gaBackground(1.0, 1.0)

    ofSetColor (204, 116, 5)
    ofNoFill()
    radio = 0
    radioInc = 0.5
    for i = 0, OUTPUT_W do
        angulo = math.rad(i)
        posY = (math.sin(angulo) * amplitud) + radioInc
        ofTranslate(4, posY, 0.0)
        ofCircle(radio, radio, radio)
        radio += radioInc
    end
end
```



La función `ofScale(float, float, float)` se utiliza para escalar el tamaño, sus 3 parámetros indican el factor de escala en los ejes X,Y, Z (si se trabaja en 2D el parámetro de Z es 1), lo que permite un escalado no uniforme.

Otra forma para escalar valores individuales (también llamado mapear) lo proporciona la función:

```
ofMap (valor, inputMin, inputMax, outputMin, outputMax, bool)
```

El primer parámetro es el valor de entrada que vamos a utilizar para mapear, el segundo y tercero corresponden al rango actual de valores mínimo y máximo de esa entrada de datos. Los dos siguientes parámetros son el rango al que deseamos escalar los valores de entrada, y el último parámetro permite fijar los resultados o no, mediante la opción booleana `true` o `false`.

5.5. Azar y probabilidad

Una tirada de dados jamás abolirá el azar

Sthéphane Mallarmé

El paso de la física determinista a la física de probabilidades implicó un cambio de pensamiento que ha repercutido más allá de la ciencia. Las relaciones entre orden, caos y aleatoriedad influyeron fuertemente en el mundo del arte, llegando a ser un motor de creación artística para las propuestas experimentales de los años 60 y 70, como las composiciones musicales de John Cage, que definió el proceso de creación de la pieza *William Mix* (1953) del siguiente modo:

A través de medios elaborados al azar, determiné previamente cuáles de aquellas categorías tenían que agruparse, y qué parámetros había que modificar. Se llamó a las categorías A, B, C, D, F; y la agrupación de estos elementos venía determinada por operaciones al azar del I Ching⁴⁶.

Esta descripción bien parece un esquema de programación; como todos los procesos aleatorios, tiene como particularidad producir formas o sonidos que no pueden conocerse de antemano, pero el proceso no está dejado al azar en su totalidad, sino que marca un esquema dentro del cual el azar establece las relaciones.

Azar y aleatoriedad permiten observar cómo la incertidumbre entra en el mundo de las matemáticas, dado que la aleatoriedad se define como todo aquello que "bajo el mismo conjunto aparente de condiciones iniciales, puede presentar resultados diferentes"⁴⁷, rompiendo el patrón causa-efecto. En el campo de las matemáticas se han desarrollado la teoría de la probabilidad y los procesos estocásticos para caracterizar las situaciones aleatorias.

En programación, todos los lenguajes contemplan funciones random -aleatoriedad- para representar estas situaciones. En GAmuza, se puede usar la función `ofRandom(float, float)` cuyos parámetros definen el rango de dos valores, entre los que el sistema elegirá uno al azar, cuando sólo tiene un parámetro el rango va de 0 a ese valor. Esta función suele utilizarse para generar formas o sonidos menos definidos y que presentan una configuración distinta cada vez que se activan, pudiendo reflejar mejor un carácter orgánico. En el siguiente ejemplo, un número determinado de puntos se distribuyen siguiendo una diagonal de 45° pero que fluctúan a un lado y otro con una doble aleatoriedad que va incrementando sucesivamente el rango de fluctuación.

⁴⁶ Richard Kostelanetz (1973), *Entrevista a John Cage*, Barcelona: Anagrama, pág. 36.

⁴⁷ Definición de aleatoriedad [enciclopedia on-line] <<http://es.wikipedia.org/wiki/Aleatoriedad>> [30.07.2012]

```

/*
GAMUZA 043 E-5-17
-----
Random Regulado

*/
totalPuntos = 800
rango = 0

function setup()
end

function draw()
  gaBackground(1.0,1.0)
  ofSetColor(0)
  rango = 0
  for i = 1, totalPuntos do
    for j = 1, totalPuntos do
      if i == j then
        ofCircle(i + ofRandom(-rango, rango), j+ ofRandom(-rango, rango), 2)
        rango = rango + ofRandom(-1,1.5)
      end
    end
  end
end

```



La función `for` tiene sólo dos declaraciones: valor inicial de la variable y test (hasta que `i` sea menor o igual a `totalPuntos`); al no tener la tercera se entiende que es 1, por lo que su acción se reduce a recorrer todos los valores entre 1 y 800. Cada vez que incrementa 1, el otro `for` recorre sus 800 pasos y vuelve el primer `for` a sumar un punto más...

De todos esos valores, la condicional `if` desestima los valores del primer y segundo `for` que no son iguales, por lo que al aplicar los datos que pasan ese filtro a las coordenadas del centro de los círculos, estos se situarían siguiendo una línea de 45°.

En la función `ofCircle()`, esas coordenadas `x` e `y` alteran esa distribución uniforme al fluctuar un número indeterminado de píxeles a un lado y otro de esa dirección como consecuencia de la función `ofRandom()`, cuyos valores están a su vez sujetos a la indeterminación de otra función `ofRandom()` que tiende a incrementarlos.

Aunque hagamos un random de random recursivo, la función `ofRandom()` produce lo que se conoce como "una distribución «uniforme» de números".

Los números aleatorios que recibimos de la función `random()` no son verdaderamente aleatorios y por lo tanto, se conoce como "pseudo-aleatorios". Son el resultado de una función matemática que simula el azar. Esta función daría lugar a un patrón durante un tiempo, pero ese período de tiempo es tan largo que para nosotros, parece tan bueno como el azar puro!⁴⁸

La conjunción de azar y probabilidad busca producir una distribución «no uniforme» de números aleatorios, reflejando que en ningún sistema todos los elementos son iguales, ni tienen las mismas condiciones.

Pensemos de nuevo en la descripción de John Cage. Para trabajar con el azar Cage establece 6 categorías para los sonidos: A (sonidos de la ciudad), B (sonidos del campo), C (sonidos electrónicos), D (sonidos producidos de forma manual), E (sonidos producidos por el viento) y F ("pequeños" sonidos que deben ser amplificados), y además establece en todos ellos otros parámetros vinculados al tono, timbre e intensidad. Después tira las monedas del I Ching para que el azar decida la composición a partir de ese diagrama de relaciones. No se trata de una aleatoriedad uniforme entre todos los sonidos, sino que opera articulándose con un contexto de relaciones condicionales.

Incluso las decisiones que se puedan tomar siguiendo el ejemplo más sencillo del azar, tirar una moneda al aire, tiene condiciones. En principio existe una equiprobabilidad, es igual de probable que salga cara o cruz: $1/2 = 50\%$

Lanzar la moneda una sola vez, nos sitúa ante eventos independientes, cuyo resultado no está condicionado ni tiene efecto en la probabilidad de que ocurra cualquier otro evento. Pero si lanzamos la moneda dos veces, la probabilidad de que salga cara en ambos sucesos cambia. Según los principios básicos de probabilidad, que 2 o más eventos independientes ocurran juntos o en sucesión, es igual al producto de sus probabilidades marginales: $1/2 * 1/2 = 1/4 \text{ el } 25\%$

En el caso de tres tiradas (... el I Ching implica 6 tiradas de 3 monedas cada vez): $1/2 * 1/2 * 1/2 = 1/6 \text{ el } 16'6\%$.

Así llegamos a que la probabilidad de que suceda un evento está condicionada a que haya ocurrido, o no, otro evento.

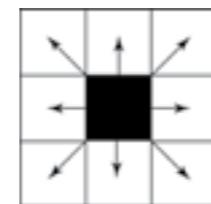
Daniel Shiffman propone otro ejemplo particular dentro del campo del azar, el *Camino aleatorio* (Random Walks), que de algún modo mantiene también relación con las acciones artísticas psicogeográficas de los Situacionistas.

Camino aleatorio es una formalización matemática de la trayectoria que resulta de hacer sucesivos pasos aleatorios. Por ejemplo, la ruta trazada por una molécula mientras viaja por un líquido o un gas, el camino que sigue un animal en su búsqueda de comida, el precio de una acción fluctuante y

⁴⁸ Daniel Shiffman (2012), *Nature of code*, autopublicado, pág. 7.

la situación financiera de un jugador pueden tratarse como un camino aleatorio. El término Camino aleatorio fue introducido por Karl Pearson en 1905. Los resultados del análisis del paseo aleatorio han sido aplicados a muchos campos como la computación, la física, la química, la ecología, la biología, la psicología o la economía⁴⁹.

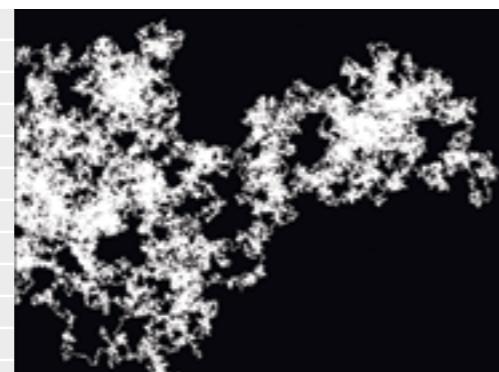
El siguiente ejemplo muestra la programación de un Camino aleatorio tradicional que se inicia en el centro de la pantalla y a cada paso elige al azar una entre las 8 opciones, que en coordenadas oscilará entre +1 y -1 tanto para x como para y. Por eso, en lugar de utilizar la función `ofRandom(-1, 1)` se usa `ofRandom()`, sin parámetros, porque son esos los que tiene establecidos por defecto.



```
/*
GAmuza 043      E-5-18
-----
Random Walker tradicional
*/
x0 = OUTPUT_W/2
y0 = OUTPUT_H/2
step = 4

function draw()
    // gaBackground(0.0,1.0)
    ofSetColor(255)
    xsig = x0 + (step * ofRandom())
    ysig = y0 + (step * ofRandom())

    ofLine(x0, y0, xsig, ysig)
    x0 = xsig           // el valor de las dos últimas coordenadas de la línea
    y0 = ysig           // pasa a las primeras para unir las líneas en recorrido
end
```



Este caminante recorre 4 px en cada paso, lo que transforma la opción de puntos en pequeñas líneas encadenadas. El segundo par de coordenadas (`xsig, ysig`) de cada línea será el primero de la siguiente, trazando así el recorrido de su deriva. Como el fondo en GAmuza se dibuja de nuevo en cada frame, para visualizar el recorrido hay que quitar o comentar la función `gaBackground()`.

En esta opción tradicional existe $1/8 = 12,5\%$ de probabilidades de que el punto se traslade a una de las 8 posiciones posibles cada vez que se efectúa el random, indistintamente de cuantas veces lo haga.

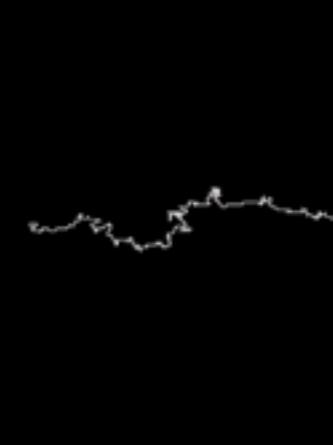
⁴⁹ Definición Camino aleatorio [enciclopedia on-line] <http://es.wikipedia.org/wiki/Camino_aleatorio> [31.07.2012]

Daniel Shiffman⁵⁰ plantea un Random Walker que tiende a moverse hacia una dirección porque, como mencionaba la definición de este algoritmo, su planteamiento sirve para representar trayectorias, especialmente en medios fluidos, como el vuelo de las aves, o el movimiento en líquidos, donde la rigidez de la geometría euclíadiana queda relegada por lo fluido. Estas trayectorias orgánicas no están totalmente abiertas sino que marcan una tendencia de dirección. Shiffman regula esta dirección de trayectoria por medio de porcentajes, con un Random Walker que tiende a la derecha según las siguientes probabilidades: Moverse hacia arriba 20%, abajo 20%, izquierda 20%, derecha 40%. Quedando el código de programación sujeto a condicionales `if` de modo que, sin hacer nunca el mismo camino, siempre que activemos el programa, el recorrido va a la derecha.

```
/*
GAmuza 043      E-5-19
-----
Random Walker condicionado
*/
x0 = OUTPUT_W/2
y0 = OUTPUT_H/2
step = 4
xsig = x0-step
ysig = y0-step

function update()
    x0 = xsig
    y0 = ysig
end

function draw()
    //gaBackground(0.0,1.0)           //el fondo se ha anulado
    ofSetColor(255)
    r = ofRandom(1)                 // si solo hay un parámetro es entre 0 y ese valor
    if r < 0.4 then                // condición 40% de probabilidad hacia derecha
        xsig = x0 + step
    elseif r < 0.6 then            // no es 60%, elseif solo actúa si no cabe en if
        xsig = x0 - step
    elseif r < 0.8 then
        ysig = y0 + step
    else
        ysig = y0 - step
    end
    ofLine(x0, y0, xsig, ysig)
end
```



⁵⁰ Daniel Shiffman (2012), *Nature of code*, autopublicado, pág. 9.

5.5.1. Noise

Con la función `ofRandom()` se pueden generar series de valores inesperados, con una distribución uniforme, no uniforme o personalizada, pero son valores que no guardan relación entre sí. La función `ofNoise()` genera también valores inesperados pero de una forma más controlable. Su nombre proviene de la función matemática Noise Perlin desarrollada por Ken Perlin para generar las texturas de la película *Tron*.

El Ruido Perlin es una función matemática que utiliza interpolación entre un gran número de gradientes precalculados de vectores que construyen un valor que varía pseudo-aleatoriamente en el espacio o tiempo. Se parece al ruido blanco, y es frecuentemente utilizado en imágenes generadas por computadora para simular variabilidad en todo tipo de fenómenos⁵¹.

La función `ofNoise()` puede tener de 1 a cuatro parámetros para calcular valores Noise Perlin de una a cuatro dimensiones, expresado en valores entre `0.0` y `1.0`. Por ejemplo `ofNoise(float x, float y)` puede crear una textura bidimensional, y con 4 parámetros `ofNoise(float x, float y, float z, float w)` calcula el valor de Noise Perlin de cuatro dimensiones entre `0.0 ... 1.0`. Los siguientes ejemplos muestran el uso de noise para una y dos dimensiones.

```
/*
GÂmuza 043      E-5-20
-----
Basics/noise_map
Modulación Noise 1D
*/
v = 0.0
inc = 0.01

function update()
    v = v + inc           // el valor de noise se incrementa cada frame
end

function draw()
    gaBackground(0.0, 0.1)
    ofSetColor(255)        // X depende del valor de noise mapeado
    x = ofMap(ofNoise(v),0,1,0,OUTPUT_W, false)
    y = OUTPUT_H/2 + ofRandom(-4, 4) // Y depende de un random
    ofCircle(x,y,16,16)
end
```

51 Definición de Ruido Perlin [enciclopedia on-line] <http://es.wikipedia.org/wiki/Ruido_Perlin> [01.08.2012]

La función `ofNoise()` no genera realmente números más relacionados entre sí que la función `ofRandom()`, funciona de manera diferente. El rango de los datos de salida es fijo, siempre el mismo valor. Por eso, el cálculo se realiza sobre una variable que va incrementando su valor paso a paso en el bloque `update()`, generando una imagen en movimiento que serpentea ondulatoriamente de forma continua e irregular. La oscilación en el eje Y se realiza con un `random`, pero también puede hacerse con un `noise 2D`

```
/*
GÂmuza 043      E-5-21
-----
Modulación Noise 2D
origen: Noise Wave de Daniel Shiffman
processing.org/ejemplos/noisewave.html
*/

yoff = 0.0          // Perlin noise 2D
xoff = 0.0

function update()
    yoff += 0.01      // incremento valor de noise
end

function draw()
    gaBackground(0.0,1.0)
    ofSetColor(255)
    ofBeginShape()      // inicio forma irregular para los puntos de la onda
    xoff = 0.0
    for x = 0, OUTPUT_W, 10 do // para y se mapean los valores de noise
        y = ofMap(ofNoise(xoff, yoff), 0, 1, 300,350) // 2D noise
        ofVertex(x, y)        // primer punto del vertex
        xoff += 0.05          // incrementa valor x para noise dentro del for
    end
    ofVertex(OUTPUT_W, OUTPUT_H)
    ofVertex(0, OUTPUT_H)
    ofEndShape(true)
end
```



5.6. Gráficos vinculados a eventos de ratón

Hasta ahora se han visto pocas funciones propias del framework de GAmuza, dado que su desarrollo se dirige más a la interactividad, a partir de ahora iremos revisándolas, empezando por las funciones que transmiten al sistema la posición del ratón en la ventana de salida, es decir, los datos relativos a las coordenadas X e Y de su posición, si se ha movido, si algún botón está presionado, ha sido soltado o es arrastrado. Las funciones para la posición del ratón son:

```
gaMouseX(), gaMouseY() // coordenadas x e y de la posición del ratón
```

Cuando el ratón está en el lado izquierdo de la pantalla su coordenada X tiene como valor 0, y se incrementa conforme se desplaza hacia la derecha. Cuando el ratón está en el lado superior de la pantalla, el valor de la coordenada Y es 0 y se incrementa conforme va bajando. Estos datos NO los da el sistema si las funciones `gaMouseX()` y `gaMouseY()` se sitúan en el bloque `setup()` porque cuando el programa se pone en marcha la posición del ratón es `(0,0)` y el código escrito en el bloque `setup()` solo se lee una vez. Por ejemplo, para establecer relaciones de interactividad entre posición del ratón y de las formas, las funciones se sitúan en el `update()`:

```
function update()
    posX = gaMouseX() // actualiza cada frame la posición del ratón
    posY = gaMouseY()
end

function draw()
    gaBackground(0.0, 0.5)
    ofSetColor(255)
    ofCircle(posX, posY, 50) // el centro vinculado a la posición del ratón
end
```

Se pueden establecer relaciones menos evidentes si se combinan operadores matemáticos que aumenten o disminuyan determinados valores, por ejemplo:

```
function draw()
    gaBackground(0.0, 0.5)
    ofSetColor(255)
    ofCircle(OUTPUT_W - gaMouseX(), OUTPUT_H - gaMouseY(), 50)
end
```

La relación inversa entre la dimensión de la ventana de salida respecto a la posición del ratón hace que el movimiento responda a direcciones con sentido contrario.

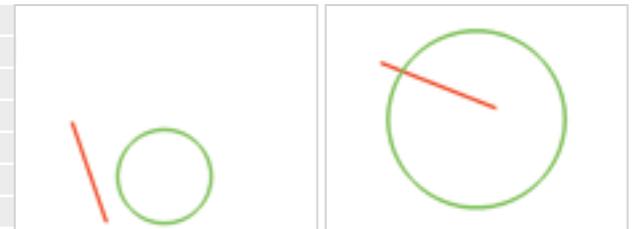
También puede aplicarse el valor de las coordenadas X,Y del ratón a las dimensiones de las formas que cambiarán de tamaño según el movimiento del ratón.

```
/*
GAmuza 043 E-5-22
-----
Interactividad Ratón básica
*/
posX=0
posY=0

function setup()
    ofEnableSmoothing()
end

function update()
    posX= gaMouseX()
    posY= gaMouseY()
end

function draw()
    gaBackground(1.0,1.0)
    ofSetCircleResolution(50)
    ofSetColor(255,0,0) // color rojo
    ofSetLineWidth(10) // ancho línea 10px
    ofLine(posX, posY, 200, 200) // línea un punto fijo y el otro en posición del ratón
    ofSetColor(0,255,0) // color verde
    ofNoFill() // no rellenar
    ofCircle(OUTPUT_W/2, OUTPUT_H/2, posX/2) //radio vinculado a posición X ratón
end
```



Las relaciones de tamaño entre la línea y el círculo, vinculadas a la posición del ratón, establecen una sensación como de bombeo flexible entre las formas, al estirar la línea el círculo crece.

GAmuza articula los eventos de ratón a través de bloques de función que permiten programar acciones que serán efectuadas cuando esos eventos sucedan.

```
function mouseDragged() // Arrastrar el ratón con el botón apretado
end
function mouseMoved() // Movimiento del ratón
end
function mousePressed() // Botón del ratón apretado
end
function mouseReleased() // Botón del ratón soltado
end
```

El siguiente ejemplo contiene acciones vinculadas al movimiento del ratón en el bloque `mouseMoved()`, y otras al clicar el botón, situadas en el bloque `mousePressed()`.

```

/*
GAMUZA 043 E-5-23
-----
mouseDragged y mouseMoved
Basado en un ejemplo de
Processing de Ira Greenberg
*/
x = OUTPUT_W/2
y = OUTPUT_H/2
r = 2
_angle = 0.0
fluctuate = 20
disperse = 5

function setup()
  ofEnableSmoothing()
end

function draw()
  gaBackground(1.0, 0.06)
  ofSetColor(0)
  for i = 0, fluctuate do
    _angle = ofRandom(TWO_PI)
    dispX = x + math.cos(_angle)*disperse
    dispY = y + math.sin(_angle)*disperse
    ofCircle(dispX, dispY, r*2)
  end
end

function mousePressed()
  x = gaMouseX()
  y = gaMouseY()
end

function mouseMoved()
  disperse = gaMouseX()*0.5 //posición X del ratón define diámetro anillo
  fluctuate = gaMouseY()*0.5 //posición Y del ratón define nº círculos en anillo
end

```



En este caso la función `mouseMoved()` modifica el valor de las variables: `fluctuate` y `disperse`. La variable `fluctuate` controla el límite de repeticiones de una estructura `for` (el número de círculos pequeños que construyen el anillo) y la variable `disperse` controla el radio del anillo de círculos generado por el `for`. Con la función `mousePressed()` se desplaza el centro del anillo a la zona de la ventana de salida donde clique el ratón. Tras ello, si se mueve el ratón diagonalmente se verán los cambios. La transparencia del fondo permite ver el rastro de la transformación.

En el siguiente ejemplo se utiliza `mouseMoved()`, `mouseDragged()` y `mouseReleased()`, para modificar el número de lados de un polígono y su posición, generando un efecto elástico, como de cuerdas de tensión, con las líneas que van de los vértices de la ventana de salida al centro del polígono.

```

/*
GAMUZA 043 E-5-24
-----
mouseDragged, mouseReleased, mouseMoved
creado por Paco Fuentes
*/

x = OUTPUT_W/2
y = OUTPUT_H/2
mod = 0.1
geom = 3

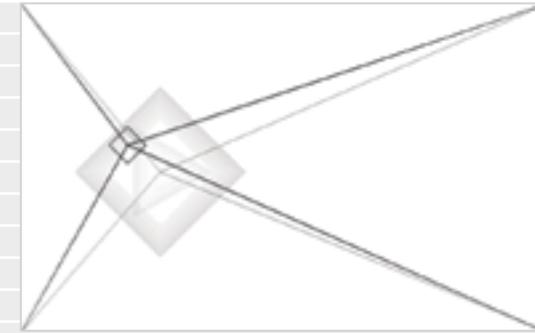
function draw()
  gaBackground(1.0, 0.1)
  ofSetColor(0)
  ofNoFill()
  ofSetCircleResolution(geom) // nºlados polígono variable según arrastre ratón
  ofCircle(x, y, mod) // tamaño polígono también variable
  ofSetLineWidth(ofRandom(4))
  ofLine(x, y, 0, 0)
  ofLine(OUTPUT_W, 0 , x, y)
  ofLine(0, OUTPUT_H, x, y)
  ofLine(OUTPUT_W, OUTPUT_H, x, y)
end

function mouseDragged()
  mod = mod + 10
  geom = geom + 0.1
end

function mouseReleased()
  mod = 50
end

function mouseMoved()
  x = gaMouseX()
  y = gaMouseY()
  geom = geom - 0.1
  if geom < 3 then // si el nº de lados es menor que 3
    geom = 3 // vuelven a ser 3
  end
end

```



5.7. Gráficos vinculados a eventos de teclado

El teclado es el dispositivo habitual para escribir texto en el ordenador. Cada tecla presionada es una señal de entrada al sistema que tiene como respuesta (salida) la impresión de caracteres en el monitor. Esa señal de entrada (input) puede utilizarse para interactuar con las acciones programadas en el código.

La función de GAMUZA para enviar al sistema los datos de las teclas es `gaKey()` y para que el sistema sepa cuál es esa tecla, se utilizan estructuras condicionales, como:

```
if gaKey() == string.byte('g') then
    haz tal cosa
end
```

La función de Lua `string.byte` devuelve los códigos numéricos internos de los caracteres⁵². Al igual que en los eventos de ratón, GAMUZA articula los eventos de teclado a través de bloques de función que permiten programar acciones que serán efectuadas cuando esos eventos sucedan.

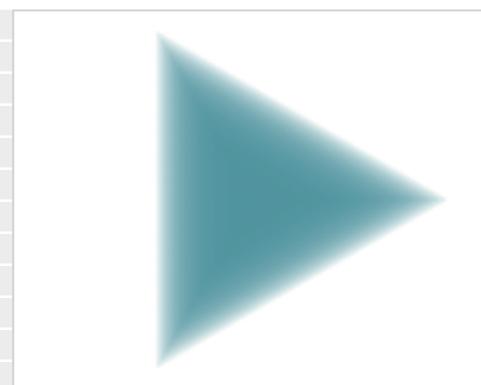
```
function keyPressed() //el código actúa cuando se presiona una tecla
end

function keyReleased() //el código actúa cuando se suelta una tecla
end
```

En el siguiente ejemplo se establecen distintas composiciones en función de la tecla presionada.

```
/*
GAMUZA 0.4.1 E-5-25
-----
interactividad con teclado
creado por Luis Miguel Jaramillo
*/
evento = 1

function setup()
    ofEnableSmoothing ()
end
```



```
function draw()
    gaBackground(1.0,1.0)
    if evento == 1 then
        for a = 0, OUTPUT_H/2, 20 do
            ofSetColor(78, 147, 158, 50)
            ofSetCircleResolution(3) //triángulos
            ofCircle(OUTPUT_W/2, OUTPUT_H/2, a+2)
        end
    elseif evento == 2 then
        for a = 0, OUTPUT_H/2, 20 do
            ofSetColor(0, 110, 156, 50)
            ofSetCircleResolution(6) //hexágonos
            ofCircle(OUTPUT_W/2, OUTPUT_H/2, a+3)
        end
    elseif evento == 3 then //dodecágonos
        for a = 0, OUTPUT_H/2, 20 do
            ofSetColor(0, 31, 64, 50)
            ofSetCircleResolution(12)
            ofCircle(OUTPUT_W/2, OUTPUT_H/2, a+1)
        end
    end
end

function keyReleased()
    if gaKey() == string.byte('h') then
        evento = 2
    end
    if gaKey() == string.byte('p') then
        evento = 3
    end
    if gaKey() == string.byte('t') then
        evento = 1
    end
end
```

⁵² Hay que tener en cuenta que los códigos numéricos no son siempre iguales en las distintas plataformas.

6. Trabajar con archivos: texto, imagen, video y audio

6.1. Textos y tipografía

Entre los tipos de datos que almacena y con los que opera el software, están también los textos, no como datos o nombres de variables sino como caracteres (**char**), y/o cadena de caracteres (**string**), palabras, textos que se pueden incorporar como imagen, pues las fuentes de texto (tipografías) se convierten en texturas que se dibujan en la pantalla.

Para ello se pueden utilizar la clase de openFrameworks **ofTrueTypeFont()** para ASCII extendido, o el addon **ofxTrueTypeFontUC()** que contempla todos los caracteres Unicode. En ambos casos es necesario cargar el archivo del tipo de fuente previamente en la carpeta **data** del archivo del script. La estructura de código básico para trabajar con textos se inicia asociando una variable global a la clase de openFrameworks o de ofx.

```
nombreVariable = ofTrueTypeFont()  
nombreVariable = ofxTrueTypeFontUC()
```

Cuando se trabaja con **ofTrueTypeFont()**, en el bloque del **setup()** se asocia esa misma variable al método **loadFont()**. Los métodos se vinculan a las variables con el operador ":" , y sus parámetros son como mínimo los dos primeros, pueden ser 4 y en casos especiales hasta 6.

```
nombreVariable:loadFont(string,int,bool,bool)
```

Estos parámetros corresponden a:

- **string**: Tipo de fuente, el archivo debe estar guardado en la carpeta **data** y para cargarlo se utiliza la función **gaImportFile("nombre archivo fuente")**
- **int**: Tamaño
- **bool**: Si tiene anti-aliasing (**true** o **false**)
- **bool**: Si la fuente contiene el conjunto completo de caracteres o un subconjunto, **ofTrueTypeFont()** solo incluye ASCII extendido, **ofxTrueTypeFontUC()** incluye Unicode que contempla los acentos, también booleano (**true** o **false**)

El contenido del texto se dibuja en el bloque **draw()** con el método **drawString()** que tiene 3 parámetros: el texto entre comillas y las coordenadas que definen la posición de inicio del texto.

```
nombreVariable: drawString("texto entrecomillado", x, y)
```

Si se va a modificar el tamaño mientras el programa está activo, el método **loadFont()** se pone en el bloque **update()** o **draw()** y el valor del tamaño vinculado a una variable que se actualiza en el bloque **update()**.

El mundo de la programación es anglosajón, por lo que para trabajar con acentos, ñ..., hay que utilizar la clase **ofxTrueTypeFontUC()** para texto Unicode.

```
/*
GAMuza 043           E-6-1
-----
Font Unicode Random
creado por n3m3da | www.d3cod3.org
*/

texto = ofxTrueTypeFontUC()
t = 30               // tamaño fuente
tx = "Randonée"      // contenido texto

function update()
    t = ofRandom(30, 100)          // actualiza el valor del tamaño aleatoriamente
end

function draw()
    gaBackground(1.0,0.5)         // fondo blanco 50% transparencia
    ofSetColor(0, 255, 0)          // color verde
    posX = ofRandom(50,OUTPUT_W-50) // variable local
    posY = ofRandom(50, OUTPUT_H-50) // tamaño variable
    texto:loadFont(gaImportFile("Anonymous_Pro_B.ttf"), t, true, true)
    texto:setLetterSpacing(t/50)   // kerning vinculado a tamaño
    texto:drawString(tx, posX ,posY)
end
```



```
/*
GAMuza 043           E-6-2
-----
Font curveText
texto curvo con ofxArcText()
creado por n3m3da | www.d3cod3.org
*/

font = ofxArcText()
text = "GAMuza - Hybrid Live Coding"

function setup()
    ofSetCircleResolution(50)
    font:loadFont(gaImportFile("D3Litebitmapism.ttf"),58)
    font:setLetterSpacing(1.3)
end

function draw()
    gaBackground(1.0,1.0)

    ofSetColor(0)
    ofPushMatrix()
        ofTranslate(OUTPUT_W/2,OUTPUT_H/2,0)
        ofRotateZ(ofGetElapsedTimef())
        font:drawString(text,0,0,OUTPUT_H/2)
    ofPopMatrix()
end
```



Además de los métodos `loadFont()` y `drawString()`, en el ejemplo se ha utilizado `setLetterSpacing()`, que regula el kerning (espaciado entre letras). Otros métodos posibles son⁵³: `setLineHeight()` que regula la altura de la línea de texto, o `setEspaceSize()` que ajusta el espacio entre palabras.

Existen también otras clases vinculadas a textos y tipografía como `ofxArcText()`⁵⁴ para textos con posiciones curvas.

El siguiente ejemplo utiliza la clase `ofxArcText()`, con funciones de traslación, rotación, y control del tiempo usando `ofGetElapsedTimef()` que devuelve el tiempo transcurrido desde que se activa el código en valores `float`, siendo esa la velocidad de giro en el parámetro de la función `ofRotateZ()`.

⁵³ Más información "ofTrueFont" [texto on-line] <<http://www.openframeworks.cc/documentation/graphics/ofTrueTypeFont.html>> [03.08.2012]

⁵⁴ Addon source <<https://github.com/companje/ofxArcText>> [01.09.2012]

El siguiente ejemplo dibuja el texto como si los caracteres fueran formas, con el método `drawStringAsShapes()` de la clase `ofTrueTypeFont()`, por ello el método `loadFont()` tiene un parámetro más de los vistos inicialmente que corresponde a la construcción de los contornos (`makeContours bool`) y además utiliza las siguientes clases, métodos y funciones:

- La clase de openFrameworks `ofPath()`⁵⁵, crea uno o múltiples path a través de puntos, generando un vector. Está vinculada con los métodos `clear()` que borra los puntos y `setFilled()` cuyos parámetros, `true` o `false`, establecen si el path es wireframe o relleno.
- Como se vio anteriormente, la función de Lua `string.byte()`, devuelve el código numérico interno del carácter que se ponga como parámetro.
- El método `getCharacterAsPoints()`, obtiene los puntos que definen el carácter y que serán usados por `ofPath()` para el trazado.

⁵⁵ ofPath Reference <<http://www.openframeworks.cc/documentation/graphics/ofPath.html>> [01.09.2012]

```
/*
GAmuza 043 E-6-3
-----
Font: texto como formas
creado por n3m3da | www.d3cod3.org
*/
font1 = ofTrueTypeFont()
font2 = ofTrueTypeFont()
character = ofPath()
letter = string.byte('&')

function setup()
    font1:loadFont("Anonymous_Pro_B.ttf",160,true,true,true)
    font2:loadFont("D3Litebitmapism.ttf"),52,true,true,true
    character = font1:getCharacterAsPoints(letter)
end

function draw()
    gaBackground(1.0,1.0)
    ofSetColor(255,0,100) // dibuja el texto como bitmap
    font2:drawString("hello - I am a bitmap",50,400)
    font1:drawString("&", 50, 250) // dibuja el carácter como bitmap
    character:setFilled(true) // dibuja carácter como forma rellena
    character:draw(200,250)
    character:setFilled(false) // o solo el contorno sin relleno
    character:draw(350,250)
    character:clear()
    ofFill() // dibuja el texto como forma rellena
    font2:drawStringAsShapes("Hello - I am a vector",50,480)
    ofNoFill() // o solo el contorno sin relleno
    font2:drawStringAsShapes("Hello - I am a vector",50,550)
end
```

hello - I am a bitmap
Hello - I am a vector
Hello - I am a vector

También puede aparecer texto en la pantalla de salida sin cargar fuentes para visualizar datos de la programación o simplemente caracteres. Para ello se utiliza la función `ofDrawBitmapString()`. Ejemplos de código.

```
text = "texto a dibujar"
ofDrawBitmapString(text, x, y) // dibuja en la ventana de salida el contenido
// de text en la posición x,y
text = string.format("Position X: %f", gaMouseX())
ofDrawBitmapString(text, x, y) // dibuja en la ventana de salida el texto
// "Position X:" más el valor X del ratón
// y lo coloca en la coordenada x,y
```

6.2. Archivos de imagen

Sintaxis

<code>ofImage()</code>	Clase de openFrameworks
<code>loadImage()</code>	Método de esta clase
<code>gaImportFile()</code>	Función de GAmuza

La clase `ofImage()` de openFrameworks permite incorporar fácilmente archivos de imagen a la ventana de salida. Al cargar el archivo se asigna automáticamente un objeto `ofPixels()` y crea una textura de OpenGL con `ofTexture()` para mostrar los píxeles de la imagen. Los archivos de imagen deben estar guardados en la carpeta `data` situada junto al archivo del script. GAmuza puede incorporar archivos de imagen del tipo jpg, gif, tga, tiff, bmp y png.

Para cargar y mostrar la imagen, se vincula la clase `ofImage()` a una variable global. Después, en el bloque `setup()` se carga el archivo con el método, `loadImage(string)`, vinculado al nombre de la variable por medio del operador `:`, su parámetro es la función de GAmuza `gaImportFile("nombreArchivo.jpg")` que importa ese archivo desde la carpeta `data`. Por último, en el bloque `draw()` se vincula la variable al método `draw()` cuyos 4 parámetros señalan la posición `x, y, más el ancho y alto de la imagen`. En el siguiente ejemplo se establece una fórmula para escalar el tamaño de la imagen a las dimensiones de la ventana de salida.

```
/*
GAmuza 043 E-6-4
-----
Cargar imagen - loadImage
creado por n3m3da | www.d3cod3.org
*/
img = ofImage() // variable global vinculada a la clase
function setup() // carga la imagen
    img:loadImage(gaImportFile("highlands.jpg"))
end

function draw()
    gaBackground(0.0, 1.0)
    // para centrar y escalar la imagen ajustada al ancho de la ventana de salida
    w = OUTPUT_W
    h = (w/img:getWidth())*img:getHeight()
    posX = 0
    posY = (OUTPUT_H-h)/2
    ofSetColor(255) // siempre hay que poner el color para dibujar la imagen
    img:draw(posX,posY,w,h) // dibuja la imagen
end
```

6.2.1. Animación con archivos de imagen y control del tiempo (fps)

También se pueden realizar animaciones de imágenes a modo de stopmotion o pixelación manejando una carpeta en la que las imágenes estén ordenadas con un dígito que se inicia con el 0, por ejemplo: imag0, imag1, imag2..., organizando las imágenes en una tabla. En el siguiente ejemplo se muestra cómo en el bloque del `setup()` se puede asignar a todos los elementos de la tabla `images` la clase `ofImage()` y pre-cargarlas con la línea de código `images[i]:loadImage(gaDataPath(string.format("imag%i.png", i)))` dentro de una estructura `for`.

```
/*
GAmuza 0434      E-6-5
-----
Tabla imágenes - animación
*/
images = {}          // declara tabla de imágenes
currentImage = 0     // imagen actual
numImages = 180       // número total imágenes
                    // TIEMPO
wait = 40            // incrementar o reducir valor para mayor o menor velocidad

function setup()
    for i=0,numImages-1 do           // carga todas las imágenes
        images[i] = ofImage()
        images[i]:loadImage(gaDataPath(string.format("imag%i.png", i)))
    end
end

function update()
    if ofGetElapsedTimeMillis() > wait then // Para la velocidad de reproducción
        currentImage = (currentImage + 1) % numImages
        ofResetElapsedTimeCounter()           // resetea el contador de tiempo
    end
end

function draw()
    gaBackground(0.0,1.0)
    ofSetColor(255)

    w = OUTPUT_W //para centrar y escalar las imágenes al ancho de la ventana de salida
    h = (w/images[currentImage]:getWidth())*images[currentImage]:getHeight()
    posX = 0
    posY = (OUTPUT_H-h)/2
    images[currentImage]:draw(posX,posY,w,h) // dibuja la imagen
end
```

GAmuza no tiene una función para regular el número de frames por segundo, dado que esa regulación depende más de la capacidad de procesamiento que tenga el ordenador, por eso se utiliza un cálculo entre el tiempo transcurrido en milisegundos desde que la aplicación se activa, que se obtiene con la función `ofGetElapsedTimeMillis()`, vinculado a la variable `wait`, cuando ha transcurrido el tiempo de espera estipulado, la función `ofResetElapsedTimeCounter()` resetea la diferencia y pone el sistema a la espera de que transcurra de nuevo el tiempo marcado por `wait`. Cuando este cálculo debe ser más preciso o irregular que el utilizado en el ejemplo, se suele usar el sistema denominado en programación "semáforo"⁵⁶.

Otras funciones de openFrameworks para obtener el tiempo transcurrido son:

`ofGetElapsedTimeMicros()` tiempo transcurrido en microsegundos (1000000 microsecs = 1 second)

`ofGetElapsedTimef()` tiempo transcurrido en segundos como `float`.

6.2.2. Efectos imagen

Además de los efectos de ajuste de color que se pueden hacer desde la herramienta Color Correction de la Barra de menú, también se pueden utilizar algunas clases del addon ofx para efectos de desenfoque gaussiano o tipo bokeh, iluminación bloom o glow, utilizando `ofxGaussianBlur()`, `ofxBokeh()`, `ofxBloom()` u `ofxGlow()`, entre otros. Este tipo de efectos utilizan la técnica Shader de openGL, que actúa directamente sobre la Ram de la tarjeta gráfica por lo que el renderizado es más rápido y no resta capacidad al procesador del ordenador. En el siguiente ejemplo el nivel de desenfoque gaussiano de una imagen está relacionado con la posición del ratón.

```
/*
GAmuza 043           E-6-6
-----
SHADERS ejemplo - desenfoque
creado por n3m3da | www.d3cod3.org
*/

imag = ofImage()          // se vinculan las variables a las clases
.blur = ofxGaussianBlur()

posX = 0                  // para tamaño y posición de la imagen y textura
posY = 0
w = 0
h = 0

function setup()
    imag:loadImage(gaImportFile ("nombreArchivo.jpg"))      // Cargar archivo
    _blur:allocate(OUTPUT_W,OUTPUT_H) // Reservar memoria para pantalla completa
    _blur:setPasses(6)           // desenfoca en 6 pasos

    w = OUTPUT_W             // ajustar tamaño a pantalla completa
    h = (w/imag:getWidth())*imag:getHeight()
    posX = 0
    posY = (OUTPUT_H-h)/2
    imag:resize(w,h)          //reasignar ese tamaño a la imagen
end

function update()
    _blur:setTexture(imag:getTextureReference(), 0) // blur toma imagen como textura
    _blur:setRadius(gaMouseX()/ OUTPUT_W*10)        // valor según la posición ratón
    _blur:update()                                // actualiza cada frame el efecto
end
```

```
function draw()
    gaBackground(0.0,1.0)
    ofSetColor(255)           // siempre definir el color
    _blur:draw(posX,posY, w, h) // dibuja la imagen con el efecto
end
```

Además de vincular la clase `ofImage()` a una variable global, se declara otra para asociarla a la clase `ofxGaussianBlur()`.

En el bloque `setup()`, después de cargar la imagen, se inicializa el efecto aplicando los métodos con el operador `:`. Los parámetros del método `allocate()` especifican `anchura` y `altura` de la textura a aplicar en el FBO (Framebuffer object), es decir, el sistema reserva memoria para una textura de ese tamaño. El método `setPasses()` realiza el filtro de desenfoque tantas veces como valor se indique en su parámetro. Las siguientes líneas de código corresponden al ajuste del tamaño, en este caso no se hace directamente en el bloque `draw()`, con variables locales, porque va a ser necesario utilizar los datos del nuevo tamaño en el `update()` para situar la imagen con ese tamaño como textura del desenfoque y de nuevo en el `draw()` para dibujarla.

En el bloque `update()`, la línea de código `_blur:setTexture(imag:getTextureReference(), 0)` vincula la textura del efecto a la de la imagen. El método `setRadius()` describe el radio del kernel de desenfoque, cuanto mayor sea el radio más desenfoca, en este caso el valor está vinculado a la posición X del ratón. Por último, actualiza en cada frame los valores del desenfoque.

En el bloque `draw()` simplemente se definen el fondo y el color para dibujar el efecto que lleva ya integrada la imagen.⁵⁷

⁵⁷ Más datos sobre ofxFX en <<http://www.patriciogonzalezvivo.com/blog/?p=488>> [12.08.2013]

6.3. Archivos de video

Sintaxis

<code>ofVideoPlayer()</code>	Clase de openFrameworks
<code>loadMovie()</code>	Método de esta clase
<code>gaImportFile()</code>	Función de GAmuza

La clase `ofVideoPlayer()` permite incorporar fácilmente archivos de video a la ventana de salida, estos archivos deben estar guardados en la carpeta `data` situada junto al archivo del script.

El código básico para reproducir o poner en pausa un archivo de video se muestra en el siguiente ejemplo. Se vincula la clase `ofVideoPlayer()` a una variable global. Después, de forma similar a los archivos de imagen, en el bloque `setup()` se localiza el archivo con el método, `loadMovie()` cuyo parámetro es una función de GAmuza `gaImportFile("nombreArchivo.mov")` que carga ese archivo desde la carpeta `data`, y según las características de reproducción que se deseen: en loop, una sola vez, o que se repita en loop invertido, se utiliza el método `setLoopState()` cuyo parámetro puede ser: `OF_LOOP_NONE`, para una sola reproducción, `OF_LOOP_NORMAL`, para reproducción en loop y `OF_LOOP_PALINDROME` para reproducción en ida y vuelta.

En el bloque `function update()` se actualiza la información del video cada frame, mediante el método `update()`. En el bloque `function draw()` se vincula esa misma variable al método `draw()` cuyos 4 parámetros señalan su posición `x`, `y`, y las dimensiones `ancho` y `alto` del video. En el ejemplo se establece la fórmula usada anteriormente para ajustar el tamaño a las dimensiones de la ventana de salida, utilizando los métodos `getWidth()` y `getHeight()` para obtener el ancho y alto del video original. El método `setPaused()` en los bloques `mousePressed()` y `mouseReleased()`, permite detener la reproducción del video cuando su parámetro es (`true`), o continuar reproduciéndolo cuando el parámetro es (`false`).

```
/*
GAmuza 043           E-6-7
-----
Video Player
creado por n3m3da | www.d3cod3.org
*/
myVideo = ofVideoPlayer()

function setup()
    myVideo.loadMovie(gaImportFile("video.mov")) // carga archivo de video
    myVideo.play() // reproduce el video
    myVideo.setLoopState(OF_LOOP_PALINDROME) // tipo loop ida y vuelta
end
```

```
function update()
    myVideo:update()
end

function draw()
    gaBackground(0.0,1.0)
    ofSetColor(255)
    scaleH = (OUTPUT_W/myVideo:getWidth())*myVideo:getHeight()
    myVideo:draw(0,OUTPUT_H/2 - scaleH/2,OUTPUT_W,scaleH)
end

function mousePressed()
    myVideo:setPaused(true) // detiene la reproducción
end

function mouseReleased()
    myVideo:setPaused(false) // reanuda la reproducción
end
```

Las recomendaciones para tamaño y códec de exportación del archivo de video con QuickTime o FFmpeg son las siguientes, si bien pueden utilizarse otros tamaños teniendo en cuenta los recursos que consumen y las posibilidades del ordenador.

QUICKTIME Encoding

Tipo de archivo Quicktime (.mov), códec: Photo-JPEG
 JPEG quality 49% - 61%
 Tamaños: 800x600 - 720x576 - 640x480 - 320x240
 Frames x segundo: 25 - 15 FPS

FFMPEG Encoding

```
-pix_fmt yuv420p
-vcodec mjpeg
-s hd480(852x480) - svga(800x600) - vga(640x480) - qvga(320x240)
-aspect 16:9 - 16:10 - 4:3 or any
-r 25 - 15
-qscale 12.0 - 6.0
Ejemplo:
ffmpeg -i inputFile.ext -pix_fmt yuv420p -s hd480 -aspect 16:9 -r 25 -qscale 9.5 -vcodec mjpeg -f mov -acodec copy -y outputFile.mov
```

En el siguiente ejemplo, en lugar de cargar un archivo de video, se utiliza la clase `ofVideoGrabber()` para trabajar con video en directo de una cámara conectada al ordenador, y las clases `ofColor()` y `ofPixel()` para manipular la información de sus frames. Después, algunos frames se guardan como archivos de imagen, con la función de GAmuza `gaSaveFrame()` cada vez que se presione la tecla espacio.

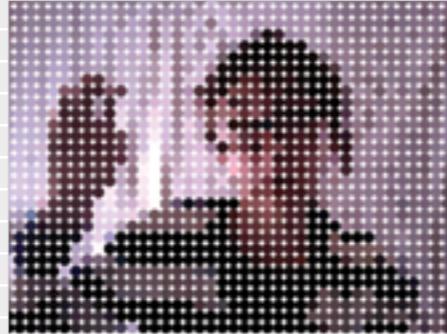
```
/*
GAmuza 043      E-6-8
-----
Imagen webCam + ofPixel()
*/
grabber = ofVideoGrabber()
c = ofColor()
pixels = ofPixels()

captureW = 320
captureH = 240
radio = 4
numImag = 0
_path = ""

function setup()
    grabber:setDeviceID(0)           //identifica ID de la cámara conectada
    grabber:initGrabber(captureW,captureH) // tamaño de la captura de la cámara
end

function update()
    grabber:update()               // actualiza frame de la imagen capturada
    pixels = grabber:getPixelsRef() //lee color y posición de cada pixel del frame
end

function draw()
    gaBackground(1.0,1.0)
    ofSetCircleResolution(60)
    ofPushMatrix()
    ofScale(OUTPUT_W/captureW,OUTPUT_H/captureH,1) //escala a pantalla completa
    for i=0, captureW-1,radio*2 do
        for j=0, captureH-1,radio*2 do
            c = pixels:getColor(i,j) // obtiene color de los pixel del frame video
            ofSetColor(c.r,c.g,c.b) // asigna ese color
            ofCircle(i,j,radio)   // dibuja los círculos
        end
    end
    ofPopMatrix()
end
```



`function keyReleased()`

```
if gaKey() == string.byte(' ') then      //ruta para guardar la imagen
    _path = gaImportFile(string.format("export%i.png", numImag))
    gaSaveFrame(_path)
    numImag = numImag + 1
end
end
```

El sistema asigna a la cámara de video un ID, si hay varias cámaras, la primera será `ID=0`, la segunda `ID=1`, etc., puede utilizarse la herramienta `tools/logger` para ver el ID de cada dispositivo conectado al ordenador.

La variable global `grabber` se vincula a la clase `ofVideoGrabber()` que gestiona la captura de secuencias de video QuickTime, después en el bloque `setup()`, a esa misma variable, el método `setDeviceID()` asigna el ID de la cámara, y el método `initGrabber()` el tamaño de captura.

En el bloque `update()`, se actualiza la información de captura de cada frame con `grabber:update()`. Despues se relacionan métodos entre las dos clases `ofVideoGrabber()` y `ofPixels()` al identificar el valor de la variable `pixels` con el método `getPixelsRef()` de la clase `ofVideoGrabber()` para obtener los datos de color y posición de los píxeles de cada frame.

En el bloque `draw()`, para escalar la imagen a pantalla completa se utiliza la función `ofScale()` cuyos parámetros son: el factor de escala del eje X que se calcula dividiendo el ancho de la ventana de salida (`OUTPUT_W`) por el ancho de la imagen captura (variable `captureW`); el factor de escala del eje Y, es lo mismo pero respecto a las alturas de pantalla y captura, y, como se está trabajando en 2D, el factor de escala del eje Z es 1. Antes de declarar la escala se debe poner `ofPushMatrix()` y después de los elementos que recogen ese escalado `ofPopMatrix()`, como se mencionó en el capítulo anterior.

La retícula de círculos se distribuye con un doble `for`, y para definir los parámetros del color, se llama a la variable global `c` vinculada a la clase `ofColor()`, y se le asigna como parámetro la variable global `pixeles`, que en el bloque `update()` va leyendo los valores de color y posición de cada pixel de la imagen, y aquí, mediante el método `getColor()` va cogiendo de esos valores leídos solo los que corresponden al color en las posiciones que van adoptando `(i,j)` en el doble `for`. En cada vuelta `i,j` suman a su valor el del diámetro de los círculos, que se van dibujando en esas coordenadas y se rellenan con el color que tiene la imagen-video en esas mismas posiciones.

Para guardar algunas de las imágenes generadas, en el bloque `keyReleased()` se establece una estructura condicional: cuando se suelta tecla espacio, se define la ruta donde se guardará la imagen asignándole a la variable `_path`, primero la función `gaImportFile()` que indica la carpeta `data` que hay junto al archivo del script, y como parámetros de ella, la función de Lua `string.format` define el nombre y tipo de archivo como string, y la expresión `%i` va incorporando al final del nombre el valor de `numImag`, sumando 1 cada vez que se suelte la tecla espacio. [Ver capítulo 5.7 Eventos de teclado]

6.4. Archivos de audio

Sintaxis

<code>ofSoundPlayer()</code>	Clase de openFrameworks
<code>loadSound()</code>	Método de esta clase
<code>gaImportFile()</code>	Función de GAmuza

La clase `ofSoundPlayer()` permite reproducir un archivo de sonido. Este archivo debe estar guardado en la carpeta `data` situada en el ordenador junto al script.

En su programación básica, se vincula la clase `ofSoundPlayer()` a una variable global. Después, al igual que con los archivos de imagen, en el bloque `setup()` se carga el archivo con el método `loadSound()`, cuyo parámetro es una función de GAmuza `gaImportFile("nombreArchivo.tipo")` que carga ese archivo desde la carpeta `data` y opcionalmente puede llevar otro parámetro booleano para el stream, si se pone `true` el archivo se transmite desde el disco en lugar de estar completamente cargado en memoria, y si es `false`, lo contrario. Según las características de reproducción que se deseen, en loop o una sola vez, se utiliza el método `setLoop()` cuyo parámetro puede ser `true` o `false`. El método `setVolume()` permite regular el nivel de volumen, su parámetro puede ir de `0.0` a `1.0`. El método `setPan()` designa el pan de reproducción, su parámetro oscila entre `-1.0` y `1.0` siendo `0.0` el centro, reproduce por los dos canales, `-1.0` es totalmente a la izquierda y `1.0` totalmente a la derecha. El método `setSpeed()` regula la velocidad de reproducción, `1.0` es la velocidad normal, `2.0` es el doble, etc⁵⁸. Se pueden reproducir archivos de sonido del tipo wav, aif, y mp3.

En el bloque `function update()` se puede actualizar la información del archivo con la función `ofSoundUpdate()`. Como el sonido no se dibuja no requiere ninguna función en el bloque `function draw()` a no ser que se quiera generar gráficos o visualizar datos con los valores del sonido.

El siguiente ejemplo muestra cómo cargar un archivo de audio, modificar su velocidad de reproducción (cambia el pitch del sonido) con el método `setSpeed()`, vinculado a la posición Y del ratón arrastrado dentro de un área dibujada, y modificar el pan `setPan()`, según la posición del ratón respecto al eje X, para que se escuche por el altavoz izquierdo, derecho o ambos.

La modificación de estos datos se monitorizan en la ventana de salida utilizando las funciones `ofDrawBitmapString()` y `string.format()` vinculadas a los métodos, `getPosition()`, `getPan()` y `getSpeed()` que van mostrando el cambio de valores según los movimientos del ratón, [ver capítulo 5.6 Gráficos vinculados a eventos de ratón].

⁵⁸ Ver los métodos de `ofSoundPlayer()` en <<http://www.openframeworks.cc/documentation/sound/ofSoundPlayer.html>> [30.03.2013]

```
/*
GAMUZA 043          E-6-9
-----
Audio/soundPlayer
Ejemplo original de openFrameworks:
of_folder/ejemplos/sound/soundPlayerExample
creado por n3m3da | www.d3cod3.org
*/

mySound = ofSoundPlayer()
pos = 0.0
widthStep = OUTPUT_W/3

function setup()
    mySound:loadSound(ofImportFile("hypno00.wav"),true) // cargar archivo
    mySound:setVolume(1.0)
    mySound:setLoop(true)
    mySound:play()
end

function update()
    pos = mySound:getPosition() // obtiene posición de lectura del archivo
    mySound:setPan(-1 + gaMouseX()/OUTPUT_W*2) // relación Pan con posición X ratón
end

function draw()
    gaBackground(1.0,1.0)
    ofSetColor(211,21,51)
    text = "Drag over Y in this area to change pitch" // dibuja el texto
    ofDrawBitmapString(text, widthStep + 10, 30)
    text=string.format("Position: %f",mySound:getPosition()) // datos posición
    ofDrawBitmapString(text,widthStep + 10,OUTPUT_H-100)
    text = string.format("Speed: %f",mySound:getSpeed()) // velocidad
    ofDrawBitmapString(text,widthStep + 10,OUTPUT_H-80)
    text = string.format("Pan: %f",mySound:getPan()) // pan
    ofDrawBitmapString(text,widthStep + 10,OUTPUT_H-60)
    ofSetLineWidth(3) // dibuja línea a modo de scroll de reproducción
    ofLine(pos*OUTPUT_W,0,pos*OUTPUT_W,OUTPUT_H)
    ofEnableAlphaBlending()
    ofSetColor(255,20)
    ofFill() // dibuja el área donde arrastrar el ratón
    ofRect(widthStep,0, widthStep,OUTPUT_H)
    ofDisableAlphaBlending()
end
```



```
function mouseDragged()
    if gaMouseX() >= widthStep and gaMouseX() < widthStep*2 then
        mySound:setSpeed(0.5 + (OUTPUT_H-gaMouseY()) / OUTPUT_H)
    end // relaciona velocidad de reproducción con posición Y ratón
```

6.4.1. Espectro sonoro de archivos de audio

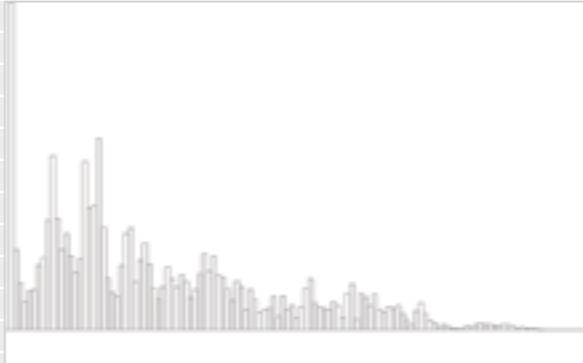
La función `gaGetSoundSpectrum()`, permite trabajar con la FFT (Transformada rápida de Fourier, ver capítulo 8. Sonido) para obtener los valores del espectro sonoro de ficheros de audio guardados en disco, técnicamente es un contenedor de la función de openFrameworks `ofGetSoundSpectrum()` basada en la librería FMOD.

```
/*
  GAMUZA 043          E-6-10
  -----
  Análisis simple FFT de archivo audio
  creado por n3m3da | www.d3cod3.org
*/

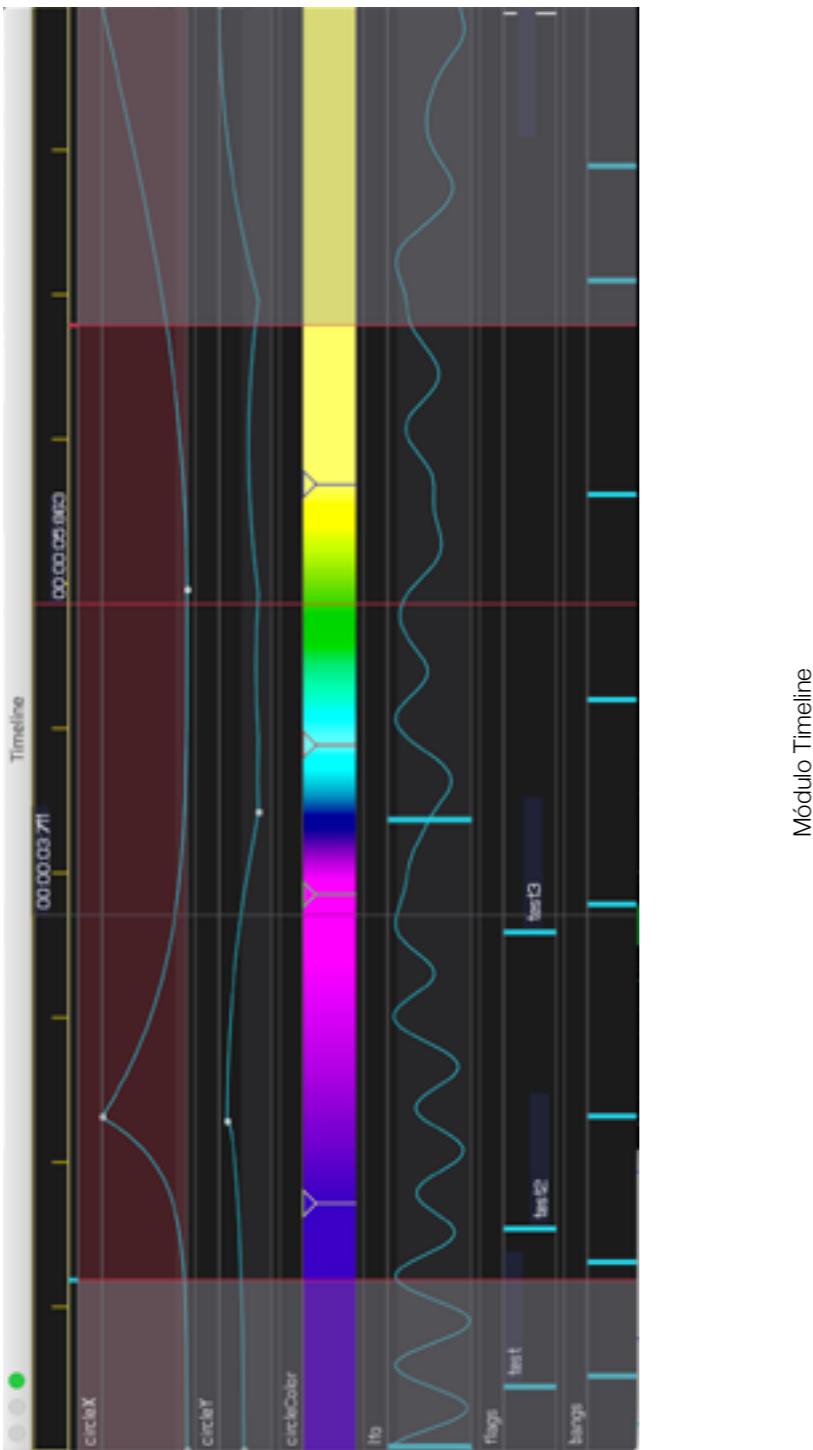
nBandsToGet = 128
audio = ofSoundPlayer()
val = memarray('float',nBandsToGet)
fftSmoothed = memarray('float', nBandsToGet)

function setup()
  audio:loadSound(gaImportFile("0406_01_RS.mp3"),true)
  audio:setVolume(1.0)
  audio:setLoop(true)
  audio:play()
  for i=0,nBandsToGet-1 do    // inicializa el memarray
    val[i] = 0
    fftSmoothed[i] = 0
  end
end

function update()
  for i=0,nBandsToGet-1 do
    val[i] = gaGetSoundSpectrum(i,nBandsToGet)
    if fftSmoothed[i] < val[i] then
      fftSmoothed[i] = val[i]
    end
    fftSmoothed[i] = fftSmoothed[i]*0.9
  end
end
```



```
function draw()
  gaBackground(0.0,1.0)
  ofSetColor(255,130)
  ofNoFill()
  w = OUTPUT_W/nBandsToGet
  for i=0,nBandsToGet-1 do
    ofRect(i*w,OUTPUT_H-100,w,-(fftSmoothed[i] * 400))
  end
end
```



7. Módulo Timeline

Como vimos en la descripción de la Barra de Menú, GAmuza incorpora Módulos de interfaz gráfica. Uno de ellos es el Timeline que facilita mucho la programación al permitir visualizar el tiempo y poner keyframes (fotogramas-clave) para trabajar con color o curvas de transformación que interpolan sus valores entre los keyframes, así como utilizar flags, bangs o switches para activar cambios a lo largo del Timeline⁵⁹.

Los datos de estos cambios se almacenan en archivos de configuración .xml que se van almacenando en la carpeta data del script. Por eso la primera función para generar un Timeline es:

```
gaTimelineSetup(string, string)
```

Su primer parámetro indica el directorio donde se guardarán esos archivos de configuración, y para que se sitúen en la carpeta data se utiliza la función de GAmuza `gaDataPath(string)`, siendo ese string el nombre de la carpeta donde se van a guardar los archivos del timeline, en el caso de que no sea la propia carpeta data ha de generarse manualmente la carpeta dentro de ella. El segundo parámetro es el nombre que tendrá el timeline para esos archivos. En el ejemplo que viene a continuación estos datos son:

```
gaTimelineSetup(gaDataPath("timeline/"), "timelineTest")
```

Vamos a analizar la incorporación de pistas al timeline y los archivos que genera, a través del ejemplo.

```
/*
GAmuza 043           E-7-1
-----
Módulo Timeline
presionar barra espacio para
reproducir/parar el timeline
creado por n3m3da | www.d3cod3.org
*/

circleX = 0
circleY = 0

function setup()
  ofSetCircleResolution(50)
    // indica la carpeta donde guardar los archivos XML del timeline
  gaTimelineSetup(gaDataPath("timeline/"), "timelineTest")
  gaTimelineSetLoopType(OF_LOOP_NORMAL)
  gaTimelineSetFrameRate(25)      // frames por segundo
```

⁵⁹ La base del Timeline procede de James George, ofxTimeline <<http://jamesgeorge.org/opensource.html>> original source code: <<https://github.com/YCAMInterlab/ofxTimeline>> [12.08.2013]

```

gaTimelineDurationInSeconds(10)          // duración en segundos del timeline
gaTimelineAddCurves("circleX",0,OUTPUT_W) // genera pista curvas y su archivo
gaTimelineAddCurves("circleY",0,OUTPUT_H)
gaTimelineAddColors("circleColor")        // genera pista tipo color y su archivo
gaTimelineAddLFO("lfo")                  // genera pista tipo LFO y su archivo
gaTimelineAddFlags("flags")              // genera pista tipo flag y su archivo
gaTimelineAddBangs("bangs")              // genera pista tipo bang y su archivo
gaTimelineAddSwitches("switches")        // genera pista tipo switch y su archivo
gaTimelinePlay()                        // activa la reproducción del timeline
end

function update()
    circleX = gaTimelineGetValue("circleX") // actualiza valores de pistas de curvas
    circleY = gaTimelineGetValue("circleY")
end

function draw()
    gaBackground(1.0,0.03)
    ofSetColor(gaTimelineGetColor("circleColor")) // asigna datos de la pista color
    ofNoFill()
    ofCircle(circleX,circleY,gaTimelineGetValue("lfo")*300) //radio valor pista LFO

    ofSetColor(0)                      // escribe en pantalla datos de bang y switch
    ofDrawBitmapString(tostring(gaTimelineGetBang()),200,200)
    ofDrawBitmapString(tostring(gaTimelineGetSwitch("switches")), 200, 250)
end

```

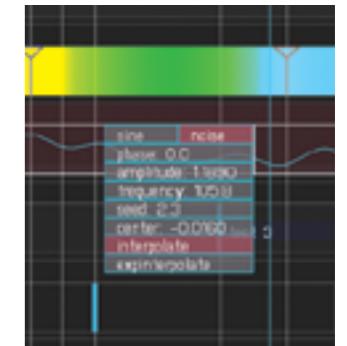
Las primeras cuatro líneas de código vinculadas al timeline en el bloque `setup()`, configuran la base: la ruta de los archivos, el tipo de loop que sigue la reproducción del Timeline, el frame rate y su duración en segundos. Después, `gaTimelineAddCurves("circleX",0,OUTPUT_W)` genera un track (o pista) llamado `circleX` que envía sus valores según la posición vertical de los keyframes, si está en su parte inferior envía el valor 0 y en su parte superior el valor `OUTPUT_W`, interpolando entre uno y otro los valores intermedios. Lo mismo para el track `circleY`.

Si después de poner estas líneas de código abrimos la carpeta `data/timeline` veremos que se han generado los archivos `timeline0_Page_One_trackPositions.xml` y `timeline0_timelineTest_trackPositions.xml`, en este último se irán almacenando los datos de todos los tracks, y al abrir el Módulo del Timeline, en cuanto empecemos a clicar keyframes en esos tracks, se crean los archivos `timeline0_circleX.xml` y `timeline0_circleY.xml`, de manera que el código genera el track y al señalar keyframes en esos tracks se van creando archivos `.xml` que guardan su configuración, lo mismo para la pista de color, flags, bangs, switches y lfos, un archivo para cada uno de ellos.

Al clicar en la pista `circleColor`, se sitúa un keyframe y aparece una representación del sistema de color, al clicar sobre un determinado color queda asignado ese valor para ese keyframe. Si se establece otro keyframe con otro color, se despliega una escala cromática entre ambos tonos.

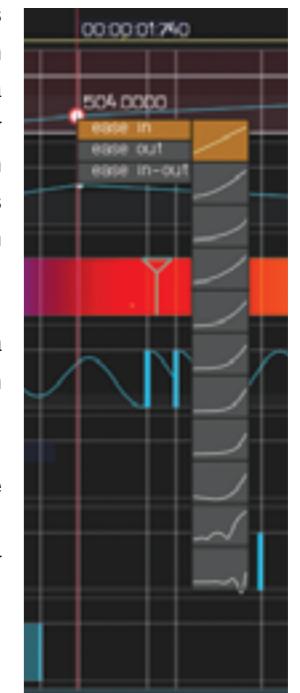
Para recoger estos valores de color se utiliza la función `gaTimelineGetColor("circleColor")`, que en el ejemplo es el parámetro de `ofSetColor()` para los círculos.

La función `gaTimelineAddLFO("lfo")` genera un tipo de pista que simula una oscilación de baja frecuencia (Low Frequency Oscillation), al poner en ella un keyframe y clicar sobre él con el botón derecho aparece una tabla de configuración, se puede seleccionar curva tipo seno o noise, y las características de la curva que genera se regulan arrastrando con el ratón a derecha o izquierda de cada uno de los valores.



En el ejemplo, los valores de estas curvas se recogen con la función `gaTimelineGetValue("lfo")` para aplicarlos al diámetro del círculo.

La función `gaTimelineAddFlags("flags")` genera una pista cuyos keyframes pueden llevar etiquetas, mientras que la función `gaTimelineAddBangs("bangs")` genera una pista en la que cada keyframe señala un bang que puede actuar como disparador para iniciar eventos. Tanto los flags como los bangs pueden leerse con la función `gaTimelineGetBang()`, con la diferencia de que en los primeros podemos diferenciar el `string` de la etiqueta concreta cuando la línea de reproducción del Timeline alcanza su posición.



La función `gaTimelineAddSwitches("switches")` genera una pista donde situar zonas más o menos amplias de interruptores que devuelven al sistema un valor booleano, `true` o `false`.

Por último señalar que la interpolación entre los valores de los keyframe de las primeras pistas generadas con la función `gaTimelineAddCurves()` pueden seguir diferentes tipos de curva. Estas opciones aparecen al clicar con el botón derecho sobre el keyframe tal como muestra la imagen lateral.

8. Interactividad

...I do think I can interact with a cat. That's a feeling I have. But not with an ant, for example...

Jim Campbell⁶⁰

Toda percepción implica un grado de interactividad con los estímulos que la generan y su entorno. A través de los sentidos recogemos datos (inputs), nuestra mente los codifica y/o reflexiona sobre ellos (procesa) y como resultado pueden modificar nuestro conocimiento, comportamiento, estado de salud, ánimo, etc., (output), es decir, lo cognitivo, cultural y psicológico se incorpora a la percepción, al tiempo que lo percibido los puede transformar si ese encuentro ha suscitado un verdadero efecto en nosotros. Umberto Eco señala respecto a la lectura: "Todo texto es una máquina perezosa que le pide al lector que le haga parte de su trabajo"⁶¹. Marcel Duchamp, respecto a las artes visuales dice: "el acto creativo no es desempeñado por el artista solamente; el espectador lleva la obra al contacto con el mundo exterior por medio del desciframiento y la interpretación de sus cualidades internas y así agrega su contribución al acto creativo"⁶².

Es el espectador quien completa el texto y el acto creativo, con una interactividad que se mueve a nivel cognitivo, cultural y psicológico. En el contexto de los medios digitales la idea de interactividad crece sobretodo en los factores técnicos, transformando el espectador con su interactividad el proceso de desarrollo de la obra. Es de esperar que este nivel se sume a los otros, para que esa experiencia de interactividad humano-máquina, espectador-obra, no quede restringida.

Con la siguiente selección de definiciones y referencias sobre la interactividad en el arte digital, queremos aportar un panorama amplio de las distintas posiciones que artistas y teóricos han adoptado ante este debate. Algunas son muy extensas, sí, pero dibujan bien el cambio que se ha producido desde los pioneros hasta la actualidad.

60 Richard Whittaker (1999) "Jim Campbell: Frames of Reference" [texto on-line] <<http://www.conversations.org/story.php?sid=30>> [28.08.2012]

61 Umberto Eco (1996), *Seis paseos por los bosques narrativos*. Barcelona: Lumen, pág. 11.

62 Marcel Duchamp (1998) "El proceso creativo" en *Notas de Marcel Duchamp*. Madrid: Tecnos,

8.1. Definiciones y referencias de Interactividad

Myron W. Krueger:

[...] observando a los participantes que interactuaban con las piezas, he desarrollado una fuerte intuición sobre lo que la gente puede llegar a entender de lo que está haciendo y lo que quieren que suceda. Mi idea de interacción surgió de esta observación de los participantes, más que de ideas preconcebidas, evitando la creación de un arte que suena muy impresionante cuando se describe por escrito, pero que no tiene éxito con su público. Esto fue poco después de las instalaciones originales, tras dieciséis años de desarrollar las capacidades de la visión por ordenador necesarias para que el trabajo empezara a ser autónomo. Todo el trabajo posterior se ha basado en una estética de los interactivos aprendida en las primeras piezas:

1. El arte debe ser divertido y accesible.
2. La interacción física es una nueva dimensión que debe ser explorada. Aunque nuestra cultura sólo permite a los adultos modos limitados de movimiento físico, el arte interactivo puede inducir a moverse en nuevas formas.
3. Las pantallas deben dominar el campo de visión de los participantes para que se sumerjan en la experiencia interactiva.
4. La imagen de los participantes es un ingrediente útil en la pantalla visual. La gente considera su imagen como una extensión de su identidad. ¿Qué pasa, que les pasa. Qué tocan, ellos sienten?.
5. Las respuestas del ordenador deben ser obvias. Siempre debe haber un nivel superficial, que la gente pueda entender en pocos segundos. En un mundo en el que las experiencias interactivas compiten con muchas otras alternativas, una pieza que no es aprehendida casi de inmediato se abandona rápidamente.
6. Todos los estímulos generados por el ordenador deben ser una respuesta a las acciones de los participantes. Cuando el equipo inicia estímulos gratuitos, la capacidad de los participantes para comprender lo que está pasando se ve amenazada.
7. No es necesario el realismo gráfico para una interacción convincente. De hecho, los entornos gráficos realistas suelen aportar confusión y son menos claros que las posibilidades interactivas que el mundo virtual ofrece.
8. La interacción entre el participante y el mundo virtual debe ser fluida y sin problemas. Esto significa que las respuestas deben ser instantáneas, tal como lo serían en el mundo real. Así como un taxi que corre por una pista de despegue, pero no se mueven lo suficientemente rápido como para despegar no es un avión, una experiencia interactiva en la que el participante es consciente de un desfase entre su acción y la respuesta de la pieza no es interactiva.

9. Dando a la audiencia muchas interacciones diferentes para elegir los mantienes interesados por más tiempo.

10. Si se ofrece un lienzo, pincel y pinturas de óleo en un lugar público, muy poca gente tratará de crear arte, pero cuando se enfrentan a la posibilidad de utilizar un medio en el que las reglas son desconocidas y con el que las posibilidades de éxito estético son Altas, la mayoría de la gente moverán sus cuerpos para crear un resultado que les agrade. De hecho, la exploración estética ofrece una nueva razón para mover el cuerpo.

11. Idealmente, una pieza debería hacer algo que tenga sentido en todas las condiciones de exposición: lleno frente vacío, ruidosa frente silenciosa, individuos aislados frente pequeños grupos.

12. Mientras que los videojuegos se basan en los resultados para motivar la participación en experiencias muy estructuradas, la participación física permite desarrollar otros estilos de interacción.

13. Cargar con dispositivos como head-mounted displays distancia al participante de la experiencia virtual y aún no son suficientemente buenos para utilizarlos.⁶³

Stephen Wilson

La interactividad es a menudo considerada el rasgo distintivo de los medios basados en el ordenador. El público está invitado a actuar para influir en el curso de los acontecimientos, o para navegar por el hiperespacio de datos. Al principio, esta relación entre público y obra se consideró una aportación bastante radical, que transformaba a artista y público en co-creadores lo que incrementaba las probabilidades de un compromiso intelectual, espiritual y estético. A medida que el campo ha madurado, tanto artistas como teóricos han tratado de deconstruir la interactividad.⁶⁴

Joshua Noble

En una obra de arte interactivo, el objeto de arte es realmente la interacción entre el observador y el sistema que el artista ha creado. Ese sistema puede ser muy complejo tecnológicamente, puede tener un único elemento técnicamente simple, o puede no tener nada.

[...] La interactividad es un concepto que tiende a utilizarse para evadir las descripciones de los mecanismos tecnológicos de los medios, y como resultado, con demasiada frecuencia elude una atención analítica y crítica sostenida.⁶⁵

⁶³ Myron W. Krueger, 'Towards Interactive Aesthetics', en *Ars Electronica Katalogartikel* <http://90.146.8.18/en/archives/festival-archive/festival_catalogs/festival_artikel.asp?ProjectID=12978> [20.02.2012].

⁶⁴ Stephen Wilson, (2002) *Information Arts: Intersections of Art, Science, and Technology*, Cambridge (MA): MIT Press, pág. 653.

⁶⁵ Joshua Noble (2009), *Programming Interactivity. A Designer's Guide to Processing, Arduino, and openFrameworks*. Sebastopol (CA): O'Reilly, pág. 14.

Erkki Huhtamo,

[...] la tecnología interactiva soporta una gran promesa. Bien podría subvertir las prácticas de *media art* predominantes y sustituirlas con formas más versátiles, amigables y "democráticas". Sin embargo, dirigirse solo al desarrollo de hardware, diseño de interfaz y curvas de ventas no será suficiente para lograr este objetivo. La tecnología digital ha de ser valorada en un contexto más amplio que abarca no sólo el espectro completo de las prácticas contemporáneas sociales e ideológicas, sino la historia también. Este artículo ha tratado de demostrar -aunque a través de muestreo muy selectivo- que las obras de arte interactivas pueden tener un papel fundamental en este proceso, manteniendo un metadiálogo continuo sobre la interactividad. Esto no es una tarea fácil. Como Andy Darley ha dicho de manera sucinta: «Las posibilidades de formas constructivas igualitarias, más democráticas, que ofrecen nuevas formas de interacción, conocimiento y comprensión puede ser mejorada por la capacidad novedosa de las nuevas tecnologías. Y más que nunca, tienen que esforzarse en ello».⁶⁶

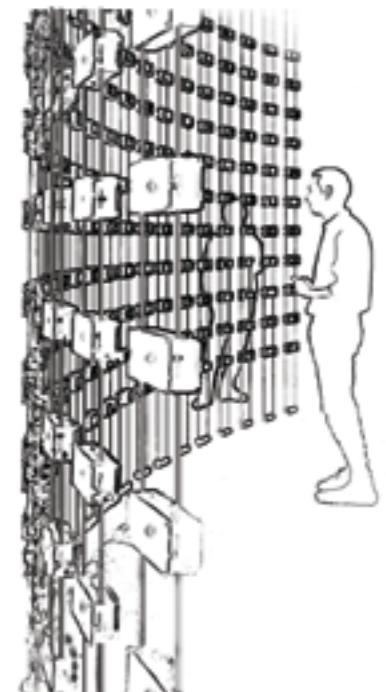
Estas definiciones señalan algunas características del arte interactivo, y su transformación desde la interactividad directa y casi mimética de los pioneros, muy vinculada a la relación humano-máquina (*human computer interaction*⁶⁷), la interactividad indirecta, hasta el metalenguaje interactivo que señala Huhtamo o su deconstrucción según Wilson.

En la interactividad directa, la experiencia del espectador habitualmente se debate ante el desafío de encontrar los mecanismos de interacción (su funcionamiento técnico) y lo que la obra transmite, llegando a tener una experiencia satisfactoria semejante a la que se alcanza cuando consigues superar el nivel en un videojuego, pero a veces distante al tipo de fruición y reflexión propios del arte.

A finales de los 90's algunas propuestas empezaron a cuestionar el sentido de la interactividad en el arte digital. Para mostrarlas de una forma concisa reduciremos el rango a dos posiciones extremas, si bien entre ambas se sitúa gradualmente ese desplazamiento: por una parte, la interactividad directa entre el espectador y la pieza, en la que el rol del espectador se aproxima al performer; por otra parte, el contenido de la obra se configura a través de la interactividad de los mecanismos técnicos de la pieza con datos obtenidos del entorno físico o del flujo de datos de la red. Con ello se desplazaba la noción de interactividad desde la relación espectador/obra [humano-máquina] hacia entorno/obra pudiendo el espectador participar en los datos que proporciona el entorno, o no.

En el año 2002, la obra *Listening Post*⁶⁸, de Ben Rubin y Mark Hansen, subrayó ese desplazamiento del concepto de interactividad hacia otro tipo de relación entre inputs -- procesamiento -- output. La pieza combina de forma sugerente la entrada dinámica de datos en tiempo real con el debate sobre

espacio privado/publico en la red. Esa entrada dinámica está constituida por la selección de fragmentos de texto de miles de salas de chat de Internet (no restringidas), tablones de anuncios y otros foros públicos en la red. El procesamiento de esos datos tiene como resultado que los textos seleccionados son cantados por un sintetizador de voz, y al mismo tiempo se muestran a través de más de doscientas pequeñas pantallas electrónicas que construyen una membrana reticular curva. El flujo de esa información, textos y sonidos, va generando una composición musical y visual, articulada en seis movimientos, cada uno de los cuales mantiene una dinámica cíclica, generando un ambiente casi ceremonial y ritual. La participación de los espectadores es principalmente contemplativa, interpretar la instalación, si bien podemos pensar que pueden introducirse en las redes sociales para incrementar ese flujo de datos, a la espera de que el buscador de la pieza seleccione su aportación. Aunque tal vez es más fascinante sentir que personas distantes están generando este trabajo, sin saberlo.



Basado en *Listening Post*, (2002) Ben Rubin y Mark Hansen

Respecto a la transformación del rol del espectador en performer, el análisis de piezas como, *Very Nervous System* (1982-1991) de David Rokeby, o *Videoplase* (1989) de Myron W. Krueger, entre otras muchas, pueden ilustrarlo.

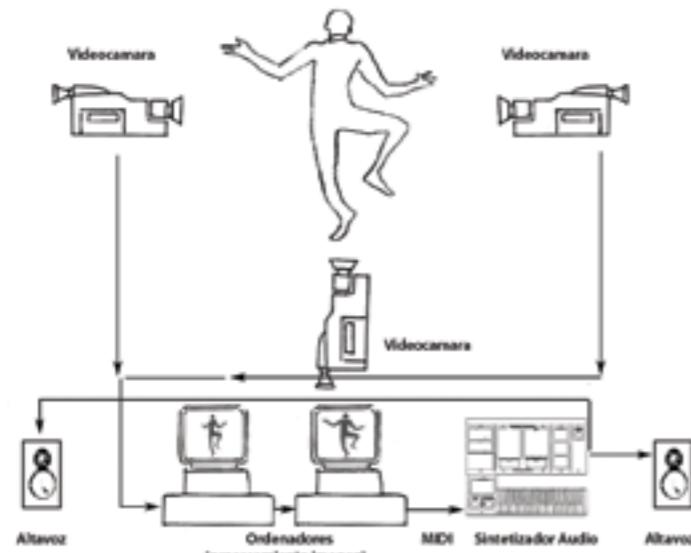
"La primera gran obra interactiva que he creado se llama "Very Nervous System" y se inició en 1982. Utilicé cámaras de video, procesadores de imagen, ordenadores, sintetizadores y un sistema de sonido para crear un espacio en el que los movimientos de un cuerpo crearan sonido y/o música. El ordenador como medio es muy tendencioso por lo que mi impulso mientras utilizaba el equipo era trabajar firmemente en contra de estos sesgos. Debido a que el equipo es puramente lógico, el lenguaje de la interacción debe esforzarse en ser intuitivo. Debido a que el equipo te saca de tu cuerpo, el cuerpo debe estar fuertemente implicado. Dado que la actividad del ordenador tiene lugar en la escala microscópica de discos de silicio, el encuentro con el equipo debe tener lugar en la escala humana del espacio físico. Y como el equipo es objetivo y desinteresado, la experiencia debería ser íntima.

El resultado es un espacio interactivo en el que el público utiliza sus cuerpos como el elemento activo de la interfaz. El movimiento corporal es rico, complejo y lleno de sutileza y ambigüedad. El Arte digital pionero utilizaba generadores de números aleatorios para proporcionar variedad y complejidad. He sustituido el generador de números aleatorios con la complejidad de la respuesta humana consciente"⁶⁹.

66 Erkki Huhtamo, (1992) 'Seeking Deeper Contact. Interactive Art as Metacommentary', [texto on-line] <<http://www.kenfeingold.com/seekingdeeper.html>> [17.06.2012].

67 Interacción persona-ordenador: Es la disciplina que estudia el intercambio de información mediante software entre las personas y ordenadores. Ésta se encarga del diseño, evaluación e implementación de los aparatos tecnológicos interactivos, estudiando el mayor número de casos que les pueda llegar a afectar. El objetivo es que el intercambio sea más eficiente: minimizar errores, incrementar la satisfacción, disminuir la frustración y, en definitiva, hacer más productivas las tareas que rodean a las personas y los computadores. [Enciclopedia on-line] <http://es.wikipedia.org/wiki/Interaccion_persona-computador> [05.07.2012]

68 Listening Post - Ear Studio <<http://earstudio.com/2010/09/29/listening-post/>> [04.09.2012]

Interpretación esquema técnico de *Very Nervous System* (1982-1991) de David Rokeby

Tanto esta obra de David Rokeby, como *Videoplase* (1989) de Myron W. Krueger, se consideran pioneras de la técnica Tracking Video y no es casual que ambos sean también desarrolladores de software para tracking. David Rokeby programó softVNN, un conjunto de objetos externos para Max MSP con los que se podía hacer tracking y procesamiento de vídeo en tiempo real, si bien la programación de la obra *Very Nervous System* fue muy anterior y programada en C. En la obra, los datos procesados relacionan las capturas de las cámaras con la generación de sonido, mientras que la pieza de Myron Krueger lo hace con la representación de formas, pero en ambas piezas los sonidos o las formas generadas, en tiempo real, guardan relación con las acciones, gestos o movimientos que el espectador realiza ante la cámara.

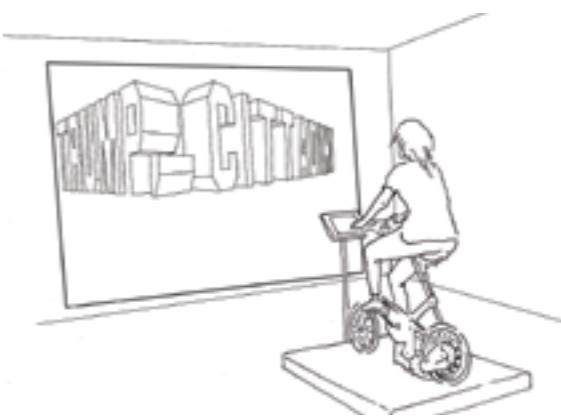
En las instalaciones de tracking video, la visibilidad de los dispositivos técnicos tienden a minimizarse, solo las cámaras, un foco de luz y los altavoces son visibles en *Very Nervous System*; la pantalla, un foco de luz, el video proyector y la cámara de video en *Videoplase*, mientras que los dispositivos técnicos que operan esa transformación con las acciones de los espectadores en imagen o sonido, permanecen ocultos. Esta no presencia de aquello que genera lo que percibe el espectador incrementa su carácter de espectáculo, y la identificación del espectador con la ilusión proyectiva próxima al "estadio del espejo" lacaniano. Esta situación especular puede propiciar una reflexión sobre la forma en que los espectadores ven, entienden y actúan sobre el mundo, alcanzando la interactividad esa intensidad demandada por muchos teóricos, pero también puede plantear un simple juego o divertimento.

Otros artistas mantienen una posición contraria, prefieren que los mecanismos estén a la vista del espectador, para que sean más conscientes de los dispositivos que accionan y del proceso de transformación de la obra. Esta conexión genera un doble efecto, sintiendo la experiencia y viendo así mismo cómo se produce, manteniendo una situación más distante a la ilusión. Es el caso de *Laser Tag* (2006) de Graffiti Research Lab⁷⁰, que utilizan también técnicas de tracking video, con una aplicación desarrollada con openFrameworks. La obra, pide al espectador un carácter activo y actitud diferente: transformarse por un momento en un grafitero que deja mensajes efímeros y luminosos en la calle, utilizando un láser para dibujar que el software transforma según los parámetros seleccionados.

Basado en *Laser Tag* (2006) de Graffiti Research Lab

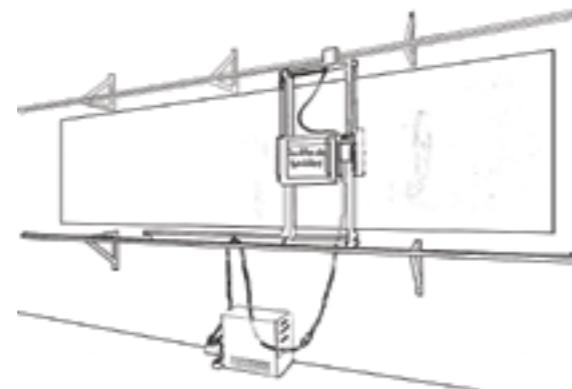
Otra tipología de arte interactivo se basa en la construcción de interfaces físicos que el espectador manipula para ir desplegando los contenidos de la obra, ejemplos pioneros son *The Legible City* (1988-1991) de Jeffrey Shaw, o *Blackboard* (1993) de Frank Fietzek, ambas recurren a un planteamiento metafórico que guía la construcción de sentido, asociando la interfaz a objetos cotidianos cuyo uso conoce bien el espectador.

En *The Legible City* la presencia de una bicicleta estática invita al espectador a recorrer una ciudad virtual, urbanizada por grandes bloques de textos tridimensionales. Pedaleando y girando el manillar, el espectador controla la dirección y velocidad de su recorrido por la ciudad legible. Un pequeño monitor situado junto a la bicicleta muestra el mapa de la ciudad, indicando la posición actual del ciclista.

Basado en *The Legible City* (1988-1991) de Jeffrey Shaw

⁷⁰ Documentación en su web <<http://www.graffitiresearchlab.com/blog/projects/laser-tag/>> [26.02.2013]

Existen 3 versiones de esta pieza, la versión de Manhattan (1989) consta de ocho hilos narrativos distintos a modo de monólogos del ex-alcalde Koch, Frank Lloyd Wright, Donald Trump, un guía, un estafador, un embajador y un taxista. Cada hilo de la historia tiene un color de letra distinto para que el ciclista pueda elegir uno u otro, si desea seguir el camino de una narración particular. En las versiones de Amsterdam (1990) y Karlsruhe (1991) todas las letras se escalan para que tengan la misma proporción y ubicación de los edificios reales a los que sustituyen, resultando una representación transformada pero exacta de la apariencia arquitectónica de estas ciudades. Los textos de estas dos ciudades proceden de documentos de archivo que describen hechos históricos.



Basado en *Blackboard* (1993) de Frank Fietzek

En *Blackboard*, de Frank Fietzek el espectador ve un pequeño monitor móvil montado con rieles sobre una vieja pizarra de colegio, en la que se perciben todavía ligeros rastros de tiza mal borrados. El monitor está conectado a un ordenador visible, situado debajo de la pizarra. Si el espectador desplaza el monitor a lo largo de la pizarra, horizontalmente y verticalmente, aparecen en él la imagen de palabras y frases escritas con tiza como recuerdo desvelado de lo que la pizarra contuvo, a nivel simbólico.

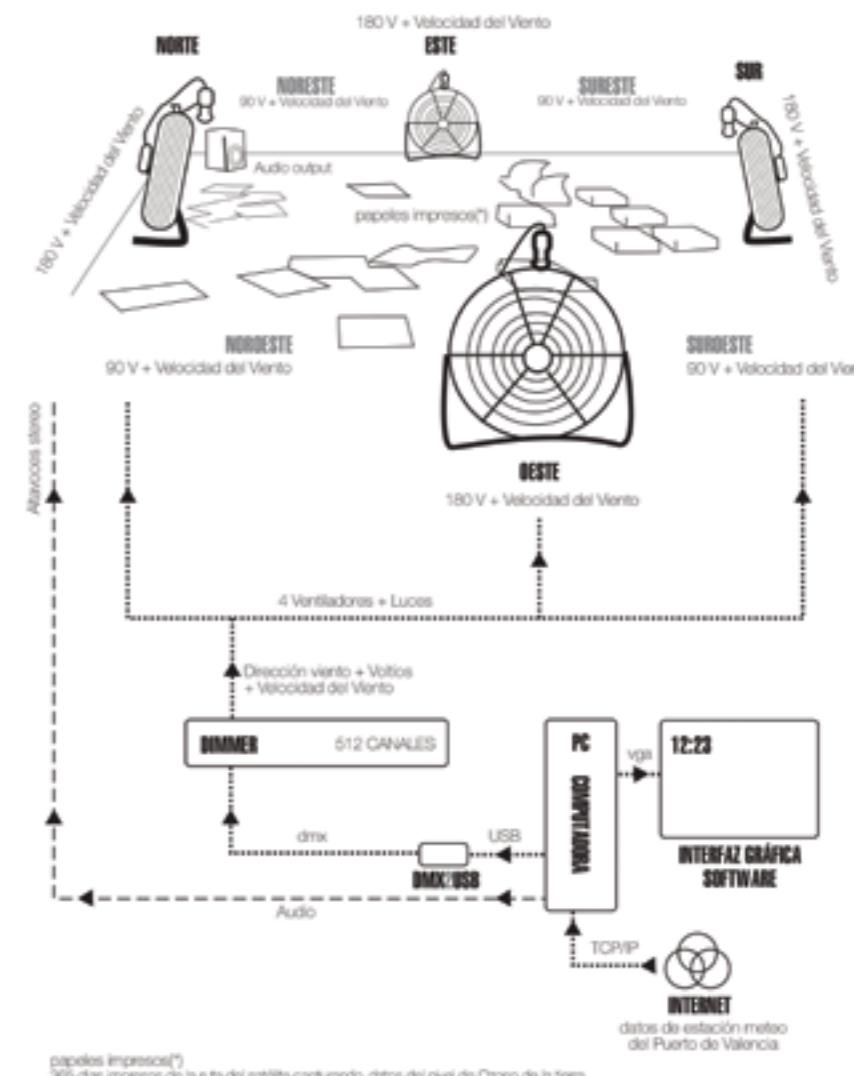
Estos textos no se ubican de forma concreta en lugares específicos sino que aparecen aleatoriamente pero manteniendo una continuidad con los movimientos. Ante cada nuevo movimiento de la pantalla una serie de sensores indican al sistema las posiciones que va tomando el monitor y aplica esos cambios a las coordenadas de la imagen.

Como se mencionó antes, algunas piezas no plantean una interactividad con el espectador sino con los datos del entorno, es el caso de *WIN-D [World · In · Now] Data* (2007), de Moisés Mañas que se genera dinámicamente al recoger los datos relativos al viento de la web de una estación meteorológica.

Instalación sonora controlada mediante los datos variables capturados en tiempo real de una estación meteorológica de Valencia.

A partir de esta información, se pone en funcionamiento una serie de parámetros claves para construir un descriptivo paisaje sonoro de síntesis FM (creado en tiempo real por la aplicación) y una particular deriva a partir de la dirección, grados y velocidad del viento, datos con los que se

dirigen cuatro ventiladores industriales situados en los cuatro puntos cardinales de la sala y que provocan con sus variaciones una corriente y una marea de papeles impresos con la información y las coordenadas de la ruta diaria del satélite Meteosat a lo largo del año 2005. Este juego entre el dato, su significado técnico-conceptual y el poder de control, produce una experiencia de feedback telecontrolada entre el espacio exterior de la Galería, la naturaleza del propio dato traducido a voltajes, notas, tonos y matices, y el interior de la Galería, que se configura como una maqueta cartográfica de una situación de protesta⁷¹.



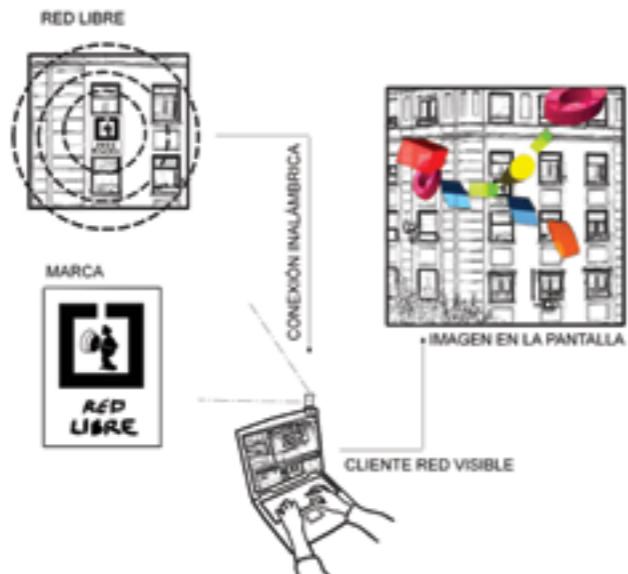
Esquema técnico de *WIN-D [World · In · Now] Data* (2007), de Moisés Mañas (esquema realizado por el propio autor)

⁷¹ Moisés Mañas, *WIN-D [World · In · Now] Data*, [texto on-line] <<http://www.hibye.org/wind/>> [06.03.2013]

Como describe el autor, la instalación más que reflejar una visualización de datos, retoma ese principio para insertar al espectador en una experiencia visual y sonora que metafóricamente alude a los sistemas de control, a la capacidad que tienen para que un implacable y objetivo flujo de datos repercuta en acontecimientos locales cargados de subjetividad. A diferencia de otros sistemas de interactividad en los que los datos existen previamente y el usuario va accediendo a ellos; el procesado de datos dinámicos, obtenidos en tiempo real, genera situaciones que no se puedan predecir con antelación con exactitud, sólo plantean un entorno de relaciones cambiantes, en este caso reguladas por la dirección e intensidad del viento.

Otra obra cuyo planteamiento reside en los datos que circulan por internet es *Red Libre Red Visible* (2004) de Diego Díaz y Clara Boj, pero en este caso se trata de una intervención colaborativa en el espacio público urbano que no busca reinterpretar el contenido de esos datos sino visualizar, mediante técnicas de Realidad Aumentada, el flujo y dirección que siguen los paquetes de información digital por las redes inalámbricas abiertas. La parte colaborativa del proyecto es que son los propietarios de esos nodos de red los que identifican el punto de acceso libre y gratuito mediante unas marcas, al tiempo que reivindican un acceso libre de los ciudadanos a dispositivos públicos de wifi en las ciudades.

El proyecto *Red Libre Red Visible* propone la visualización de los flujos de información intercambiados mediante redes inalámbricas como medio para apoyar a las comunidades wireless en su tarea de construir redes de conectividad alternativas para el intercambio de información de manera libre y gratuita. Hacer visible lo invisible nos permitirá intervenir en el entorno urbano y aportar nuevos significados al espacio colectivo mediante la construcción de arquitecturas que, superpuestas al espacio real, generen un lugar híbrido, mezcla del espacio físico y el espacio digital, producto de la incorporación de las tecnologías de la comunicación a la vida cotidiana.⁷²



Basado en *Red Libre Red Visible* (2004) de
Diego Díaz y Clara Boj

⁷² Clara Boj y Diego Díaz. (2004). "Red Libre Red Visible" en *a mínima* nº 7, Barcelona.

Por último, retomamos otro fragmento de la entrevista a Jim Campbell que inicia este apartado:

Siempre he pensado que el sentido que se le da al término interactividad en el trabajo con ordenadores estaba equivocado. Me refiero a que el sentido histórico de la "interactividad" siempre ha estado vinculado a un suceso de comunicación mutua, recíproco. Tal como lo veo, lo que realmente se establece con los ordenadores es control. Se les controla para hacer algo [o se controla al espectador para hacer algo]. Las piezas dedicadas a Heisenberg sirven para mostrar estos problemas de una forma directa: cuanto más se quiere ver algo, más se intenta controlar la situación, pero entonces se ve menos y cuando menos lo controlas, más posibilidades tienes de ver. Por ejemplo, en la pieza "Sin título para Heisenberg" cuando se camina hacia la cama, tienes la imagen cada vez más cerca, pero nunca se verán los poros de la piel, lo que se ve son píxeles. Cuando estás más lejos se puede ver a los dos amantes, pero por el hecho de aproximarte no puedes ver más. La pieza del Buda responde de forma similar. Desde la distancia se puede ver la estatua de Buda, pero a medida que te acercas no ves más que su sombra. ¿Qué estás viendo? Ambas piezas hacen referencia al principio de Heisenberg. ¿Lo conoces?⁷³

Las definiciones y piezas revisadas incitan a plantear sistemas interactivos complejos, con una base conceptual y menos directos, pero para hacer esos desarrollos es necesario conocer la base técnica de la interactividad. Por eso en los siguientes capítulos se irá analizando el código para interactividad articulado en distintos niveles. Por ejemplo, veremos la interactividad entre formas y sonidos, una suerte de sinestesia visual que recoge datos del análisis de la entrada de audio y los vincula a los parámetros de generación de formas o color. También las técnicas de tracking video (sensor visual) para recoger datos de la posición, movimiento o gestos del espectador y aplicarlos a sonidos, formas, selección de archivos, velocidad de reproducción de video, etc. Despues se verá la comunicación de datos procedentes de sensores físicos a través del microcontrolador Arduino y por último la comunicación de datos por red.

Es en estos niveles donde GAMUZA cobra más sentido, favoreciendo con los módulos de aplicación un ajuste de los datos de una forma mucho más sencilla que con otros software, lo que permite a las personas que se están iniciando en este campo, centrarse más en el planteamiento de los proyectos artísticos cuando no se tiene una base de programación alta.

⁷³ Richard Whittaker (1999) "Jim Campbell: Frames of Reference" [texto on-line] <<http://www.conversations.org/story.php?sid=30>> [28.08.2012]

8.2. Interactividad con el sonido

El sonido es un medio pasajero que se mueve en el tiempo, pero la incorporación del audio en los medios digitales ha modificado nuestra relación con él, cambiando los hábitos de escucha y percepción del sonido. Sin perder su calidad temporal se ha incrementado su capacidad espacial, reflejando estos cambios en el campo del arte a través de las instalaciones de arte sonoro o en las actuaciones de música con visuales; eventos que en muchos casos han ido borrando la distancia que los separaba, al incorporar sonido espacializado o propuestas sinestésicas entre audio, vídeo, luz y/o gráficos que establecen conexiones directas o indirectas entre la materialidad de lo sonoro y la sonoridad de lo visual. El propio sistema Live coding en que se basó GAmuza es reflejo de estos cambios.

Muchas de estas propuestas artísticas requieren el análisis de los parámetros del sonido. Para el sistema estos datos suponen solo valores numéricos que pueden aplicarse para producir otras situaciones sonoras, visuales o físicas, y es la casi simultaneidad de los cambios que se producen en el sonido analizado y en la respuesta generada, lo que establece un efecto empático entre ambos.

Métodos de medición se convierten en observaciones físicas. Los oídos se vuelven instrumentos ópticos; el movimiento de los ojos, una manera de entender el funcionamiento del sonido.⁷⁴



Basado en la obra *test pattern [nº3] (2010)* de Rioji Ikeda⁷⁵

La entrada de audio puede plantearse como un recurso de interactividad directa con los espectadores a través de micrófonos dispuestos en la sala. Es el caso de la obra *Modulador de Luz 3.0* (2008) del Laboratorio de Luz.

⁷⁴ Magnus Haglund (2005) "The air between the planets, on the audio and visual works of Carsten Nicolai", en *Carsten Nicolai Anti Réflex*, Frankfurt: Max Hollein, pág. 27.

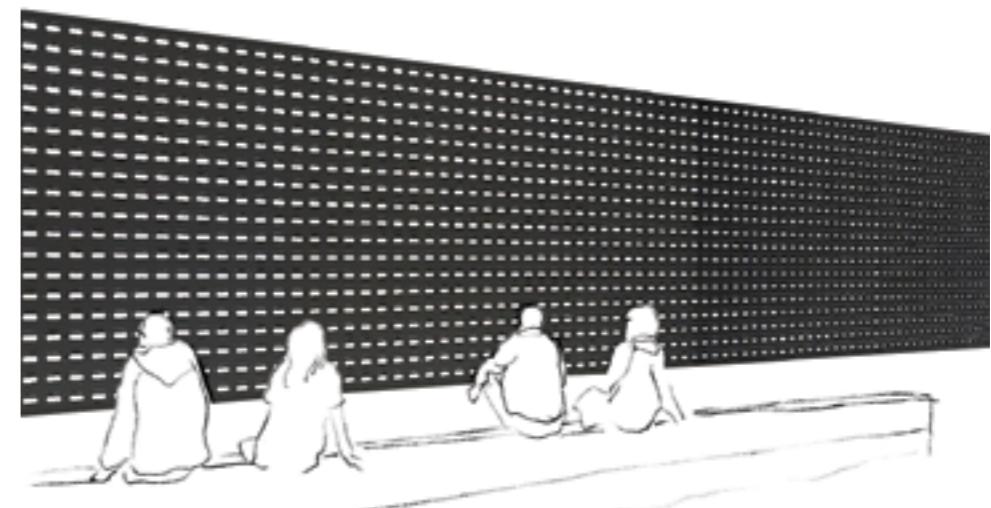
⁷⁵ Rioji Ikeda <<http://www.ryojiikeda.com/project/testpattern/>> [08.09.2012]

La instalación se presenta como un espacio escénico vacío, a la espera del acto del habla; hablar por los micrófonos para atraer la luz teatral. Pero el comportamiento de las luces en relación a lo parlante, lo sonoro y lo musical propicia juegos imprevistos; distintos roles y actitudes de la luz: a veces de forma rápida e intensa, otras dubitativa o tímidamente se dirigen hacia un micrófono u otro. Una física social de acciones y reacciones en la que no siempre se alcanza la iluminación deseada para todos los espectadores-actores⁷⁶.



Basado en la instalación *Modulador de Luz 3.0* (2008), Laboratoriode Luz

También es un recurso habitual en la realización de visuales. El artista Alva Noto (Carsten Nicolai) ha participado en el desarrollo del software Derivative de TouchDesigner⁷⁷ para ajustar las posibilidades del software a las necesidades en la visualización de datos de audio en tiempo real de su pieza *unitxt*, presentada en el Festival Ars Electronica de Linz en 2009.

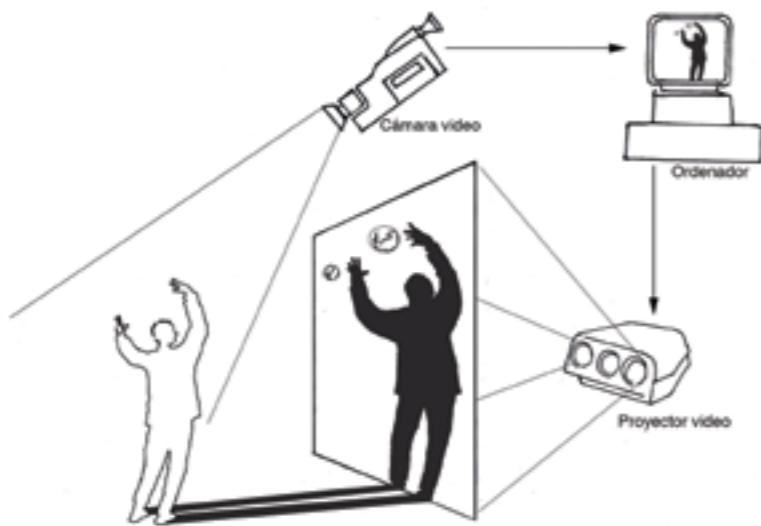


Basado en *unitxt*, (2012), Carsten Nicolai.

⁷⁶ Laboratorio de Luz, "Modulador de luz 2.0/3.0" [texto on-line] <http://www.laboluz.org/pages/pro_modula.html> [21.02.2012]

⁷⁷ Más información en <<http://www.derivative.ca/Events/RasterNoton/>> [07.03.2013]

8.3. Instalaciones interactivas por Tracking Video



Basado en *Bubbles*, (2002) W. Muench y K. Furukawa

Los sistemas de tracking video están vinculados al campo de computer vision, su aplicación en el ámbito del arte proporciona una información importante para explorar nuevos modos de conexión e interactividad entre el cuerpo físico del espectador y la representación del mundo digital. A través del ojo de la cámara, el sistema puede reconocer posiciones, movimientos, colores, gestos, rostros, dentro de su campo visual, siempre que las condiciones de iluminación sean favorables.

Conociendo esos datos, se pueden programar aplicaciones ante las que el espectador puede interactuar de forma más intuitiva, denominándose por ello interfaces gestuales, es decir, se interactúa con la pieza sin tocar botones u otros interfaces físicos. En el arte interactivo y electrónico son numerosos los ejemplos de este tipo de aplicaciones, que también ha incrementado su presencia en las artes escénicas. Referenciamos solo dos obras que mantienen una relación inversa entre obra y el espectador, *Bubbles*, (2002) de W. Muench y K. Furukawa; y *Body Navigation* (2008) O. Kristensen y J. Jongejan .

En *Bubbles*⁷⁸, una de las piezas pioneras de arte interactivo con tracking video, el espectador juega con las formas proyectadas, modificando su movimiento al colisionar con su sombra.



Basado en *Body Navigation* (2008) O. Kristensen y J. Jongejan

En *Body Navigation*⁷⁹, es el cuerpo de los bailarines lo que genera las formas. Aunque en ambos casos, es el sistema el que las modifica o produce al conocer con precisión la posición del cuerpo del espectador o de los bailarines.

El tracking de movimiento, colores o formas puede utilizarse también para generar o modificar sonidos, activar o desactivar dispositivos, simplemente hay que comprender la entrada de datos que genera el sistema, y codificar su procesamiento en función de la respuesta de salida deseada.

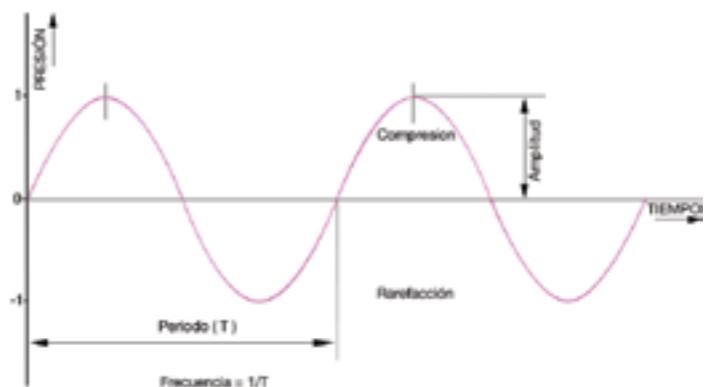
⁷⁸ *Bubbles*, (2002) W. Muench y K. Furukawa <<http://at.zkm.de/node/193>> [10.07.2012]

⁷⁹ *Body Navigation* (2008) O. Kristensen y J. Jongejan <<https://vimeo.com/1362832>> [10.07.2012]

9. Sonido

El sonido es una sensación que se produce en el oído debido a las oscilaciones de la presión del aire, o de la densidad de un medio, provocadas por la propagación de ondas. Estas ondas hacen que el tímpano del oído vibre, y este movimiento se traduce en otros tipos de energía dentro del oído, que finalmente se envían al cerebro en forma de impulsos eléctricos para su análisis.

A nivel físico, dado que las ondas sonoras se dan en el tiempo y en el espacio, hay dos tipos de representaciones. La primera, como una variación de voltaje o corriente de datos en el tiempo, y permite visualizar las siguientes características físicas del sonido: Periodo, Frecuencia y Amplitud



Periodo, es el tiempo que tarda en finalizar una onda u oscilación, se mide en unidades de tiempo (segundos) y se representa por la letra T.

Frecuencia, viene definida por el número de ondas en un segundo. La unidad de medida es el Hertz (Hz). El rango de frecuencias audibles es de 20 a 20.000 Hz. Se representa por la letra f y es la inversa del periodo. $f = 1/T$

Amplitud, indica el nivel de potencia en que se han producido las oscilaciones. Mayor amplitud implica sonido más fuerte y menor amplitud sonido débil.

En el otro tipo de representación el eje vertical corresponde al espacio, en lugar de tiempo, y lo que en el anterior gráfico representa el periodo, pasa a representar la longitud de onda (λ), que es el espacio recorrido en un periodo de tiempo, dando lugar a una nueva fórmula para definir la frecuencia:

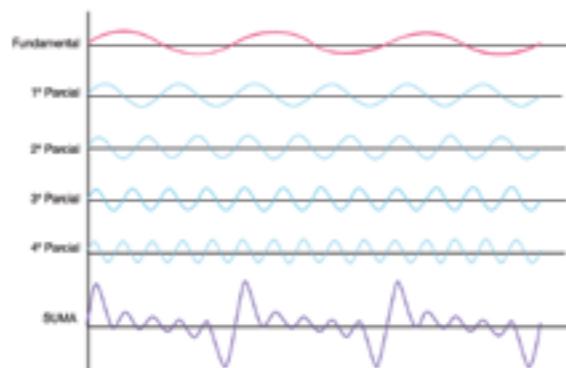
$$\text{Frecuencia} = \text{velocidad} (v) / \text{longitud de onda} (\lambda)$$

Desde el punto de las propiedades perceptivas básicas, los parámetros del sonido son: **Tono** (o altura), **Volumen** (o intensidad), **Timbre** (o color) y Duración. Estas propiedades mantienen relación con las características físicas:

Propiedades perceptuales	Características físicas	Rango
Tono o Altura (Pitch)	Frecuencia de onda	Agudo, medio, grave
Intensidad o Volumen	Amplitud de onda	Alto o fuerte, bajo, débil o suave
Timbre o Color	Series armónicas de la onda (o parciales).	Según las características de la fuente sonora (ej: cada instrumento tiene un timbre distinto al tocar la misma nota)
Duración	Tiempo de vibración	Largo o corto

Otro concepto importante para trabajar con análisis de audio es la Transformada rápida de Fourier (FFT, Fast Fourier Transform), que podemos definir de forma simplificada como un algoritmo que permite analizar las frecuencias parciales que se dan en una señal discreta. Acabamos de ver que la propiedad perceptual del timbre se corresponde físicamente con las series armónicas de una onda sonora. A esta onda sonora se le denomina fundamental y, en función de la fuente que produzca el sonido, se generan una serie de frecuencias armónicas, o parciales, que tienen distinta amplitud, frecuencia y periodo, pero que están armónicamente relacionadas entre sí.

En la naturaleza no se dan ondas sinusoidales puras (solo la fundamental), como la dibujada en el gráfico anterior, sino que percibimos, junto a la onda fundamental, una serie de frecuencias parciales que producen una sensación de sonido determinada.



Por eso es muy frecuente el uso de la FFT en dos procesos distintos del análisis de audio: 1) en su forma más simplificada se utiliza para descomponer una señal en ondas parciales para analizarlas más fácilmente; 2) para analizar el espectro o amplitud de cada frecuencia de un buffer de audio.

En GAmuza se puede trabajar con sonido para analizar algunos valores de entrada de audio (obtener los valores del pitch, volumen y FFT), para trabajar con samples, o para generación básica de sonido. En cualquier caso hay que activar el módulo **Audio Analysis** y configurarlo en **Preferences / Audio Streaming** según las características del hardware del ordenador, GAmuza generará tantas interfaces del módulo **Audio Analysis** como canales de entrada de audio tenga el dispositivo.

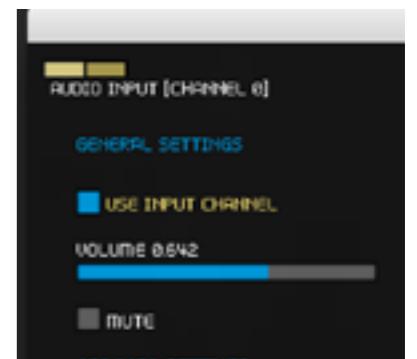
9.1. Módulo Audio Analysis

El módulo de Audio se visualiza seleccionando en el menú superior **Modules/Audio Analysis**, o tecleando **(F9)**. Antes de describir las opciones de la interface, señalar que este módulo GUI es solo para ajustar el análisis de entrada de sonido (**Input Channel**), no la salida. Si el dispositivo seleccionado tiene varios canales de entrada, veremos varias interfaces de **Audio Input** iguales que permiten ajustes independientes, a cada uno de ellos se accede clicando en una de las pestanas amarillas. Debajo de las pestanas se indica el canal que estamos visualizando [**CHANNEL 0**], [1], [2]..., que será necesario poner como parámetro en muchas de las funciones de programación de audio.



Si sólo se va a analizar la entrada de un canal, se pueden desactivar los otros con el botón **Use Input Channel**.

Debajo hay un slider para regular el nivel de volumen y un botón para silenciarlo.



OSC Data Settings. El slider Smoothing factor suaviza la relación de los datos a enviar por OSC cuando presentan grandes cambios, utilizando un filtro de Kalman.

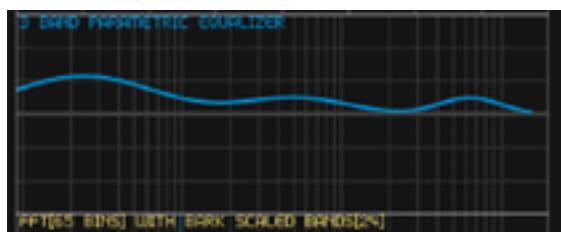
Volume Chart muestra el gráfico y valores del volumen. Su rango de valores va de **0.0** a **1.0**. La visualización de estos datos permite calcular mejor si deben escalarse, y por cuánto, para su uso en programación. La función de GAmuza que recoge el volumen es `gaGetVolume()`.

Pitch Chart muestra igualmente el gráfico y valores del tono (pitch), su rango oscila entre 20Hz to 20000Hz, pero cuando los recoge la función de GAmuza, `gaGetPitch()`, los normaliza entre **0.0** y **1.0**.

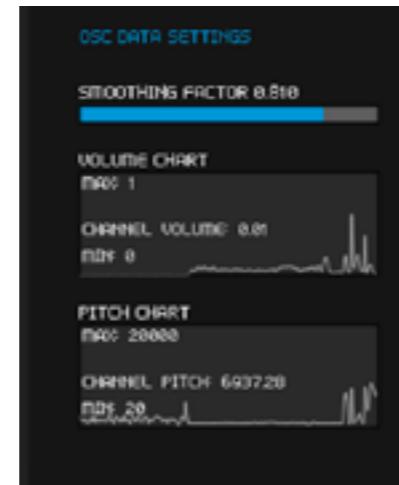
El siguiente bloque de ajustes, **Parametric Equalization**, prepara la señal para la transmisión de datos, ajustando el control individual de tres parámetros, en tres bandas: frecuencia central, ganancia y ancho de banda.

- **Center [BIN]**, selecciona la frecuencia central en Hercios. Dependiendo del valor que se ha asignado al BufferSize en el panel de preferencias, el número de BINs que tendrá la FFT será de $(\text{BufferSize}/2) + 1$. Por ejemplo con un BufferSize de 512 se realizará un análisis FFT sobre 257 BINs, es decir 257 grupos de frecuencias equidistribuidas entre 20Hz y 20000Hz.
- **Amplitude [DB]**, controla la ganancia, determinando cuánto se amplifican o recortan esas frecuencias, por eso puede tener valores negativos, como muestra la imagen y el gráfico en la segunda banda.
- **Width [Q]**, el factor Q determina la nitidez o calidad de la anchura de banda.

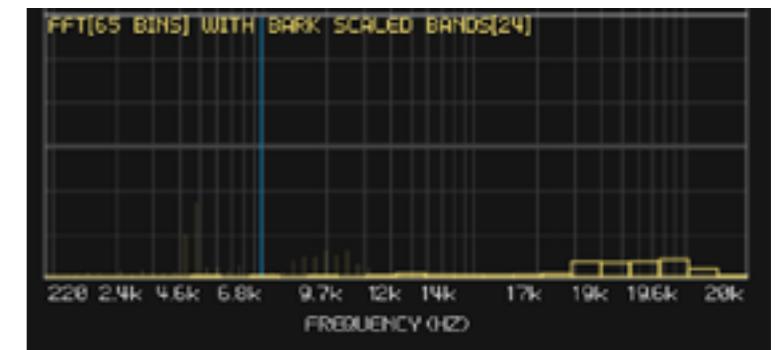
Los ajustes se visualizan en el gráfico Parametric EQ.



Este gráfico se corresponde a los valores ajustados con los sliders de la imagen anterior. Primero se busca la posición de la frecuencia central y se ajustan la ganancia y el factor Q.



Hay otros dos datos que GAmuza obtiene del análisis FFT: Bark Scaled Bands y Standard FFT. La escala de Bark⁸⁰, es una escala psicoacústica propuesta por Eberhard Zwicker en 1961, que corresponde a la subdivisión de las primeras 24 bandas críticas del oído. Los márgenes de las bandas en Hercios son 0, 100, 200, 300, 400, 510, 630, 770, 920, 1080, 1270, 1480, 1720, 2000, 2320, 2700, 3150, 3700, 4400, 5300, 6400, 7700, 9500, 12000, 15500.



GAmuza obtiene el valor de la escala de Bark desde el análisis del FFT. Para trabajar con estos datos se utiliza la función `gaGetBin(int, int)` y la variable interna `FFT_BANDS`.

La transformada rápida de Fourier, FFT (Fast Fourier Transform), es un algoritmo muy utilizado para el tratamiento de la señal de audio. En GAmuza opera sobre las muestras de sonido almacenadas en el BufferSize, por lo que su rango de valores depende del tamaño de buffer que se haya asignado en Preferences.

Si hay un tamaño de buffer 512, el gráfico FFT tendrá un tamaño [257 BINS], si el buffer es 1024, ese valor cambia a [513], por lo que el algoritmo establece que el tamaño del FFT es el del BufferSize /2 +1, que serán los puntos de análisis de la frecuencia.

En programación, estos datos se utilizan con la función `gaGetFFT(int, int)` que tiene dos parámetros: ID del canal de entrada de audio: `[CHANNEL num]`, y la posición que ocupa la muestra (sample) en el Buffer, recorriéndolo desde **0** al valor de la variable global interna `BUFFER_SIZE`.

En el capítulo siguiente se revisan todas las funciones de GAmuza vinculadas a este módulo, con aplicaciones y ejemplos.

⁸⁰ Definición Bark scale [enciclopedia on-line] <http://en.wikipedia.org/wiki/Bark_scale> [25.08.2012]

9.2. Audio input

Las funciones de GAmuza para enviar al sistema datos del análisis de la entrada de audio son:

`gaGetPitch(int)`, `gaGetVolume(int)`, `gaGetFFT(int,int)`, `gaGetBin(int,int)`,
`gaGetSoundSpectrum(int,int)` y `gaGetInputBuffer(int,int)`

`gaGetPitch(nºcanal)` Devuelve el valor del pitch detectado en el canal de entrada de audio especificado como parámetro. Como se ha mencionado antes, el valor del pitch oscila entre 20 y 20.000 Hz, tal como aparece en la interface, pero esta función de GAmuza lo escala entre 0 y 1. En el siguiente ejemplo, los datos que recoge la función de análisis del pitch se actualizan en el bloque `update()`, además utiliza la función `ofNoise()` cuyo parámetro es una función que regula el tiempo, `ofGetElapsedTimef()`, y controla el ritmo del movimiento:

```
/*
GAmuza 043           E-9-1
-----
Audio input - Pitch
Creado por Inma Alabajos Moreno
*/
inputPitch = 0.0
xPos = 0
yPos = 0

function update()
    inputPitch = gaGetPitch(0) // obtiene valor pitch (entre 0.0 y 1.0)
    if xPos < OUTPUT_W then
        xPos = xPos + (inputPitch*1000) // escala el valor vinculado a la posición X
    else
        // y limita el movimiento de X al tamaño de la pantalla
        xPos = 0
    end
    if yPos < OUTPUT_H then
        yPos = yPos + ofNoise(ofGetElapsedTimef())
    else
        // Y se incrementa con un noise vinculado al tiempo
        yPos = 0
    end
end

function draw()
    gaBackground(1.0, 0.1)
    ofSetColor(0, 85, 255)
    ofCircle(xPos, yPos, 20)
end
```



`gaGetVolume(nºcanal)` Devuelve el valor del volumen detectado en el canal de entrada de audio, especificado como parámetro. Estos valores oscilan entre 0.0 y 1.0.

Ejemplo:

```
/*
GAmuza 043           E-9-2
-----
Audio input - Volumen
creado por Paco Fuentes
*/

inputVol = 0.0 // volumen
mod = 0 // modificación
red = mod/2 // modificación reducida

function setup()
    ofEnableSmoothing()
end

function update()
    inputVol = gaGetVolume(0) // obtiene volumen del canal 0
    mod = mod+ofRandom(inputVol*1000)
end

function draw()
    gaBackground(1.0,ofRandom(0.01, 0.07)) //transparencia fondo aleatoria
    ofSetLineWidth(ofRandom(1, 7)) //ancho línea aleatorio
    ofNoFill()
    ofSetCircleResolution(ofRandom (3, 10)) //lados polígono aleatorios

    ofSetColor(0, 40+(inputVol*10)) //transparencia vinculada al volumen
    ofCircle(OUTPUT_W/2+ofRandom(red,-red),OUTPUT_H/2+ofRandom(red,-red),inputVol*1000)
    // tamaño radio vinculado al volumen
end
```



En el siguiente ejemplo, el nivel del volumen en la entrada de sonido activa la reproducción de un archivo de video. Se retoman así algunos métodos de la clase `ofVideoPlayer()` vistos anteriormente, junto con otros nuevos como `isPlaying()` y `getPosition()`, estos métodos abren las posibilidades para plantear instalaciones de vídeo interactivo. Recordad que el archivo `video.mov` debe estar en la carpeta `data` del script.

```
/*
GAmuza 043           E-9-3
-----
Audio/audioTrigger
creado por n3m3da | www.d3cod3.org
*/

vol = 0.0
video = ofVideoPlayer()
posY = 0
scaleH = 0

function setup()
    video:loadMovie("video.mov")
        // para escalar video a pantalla completa y mantener la proporción
    scaleH = (OUTPUT_W/video:getWidth())*video:getHeight()
    posY = OUTPUT_H/2 - scaleH/2
end

function update()
    vol = gaGetVolume(0)
    if video.isPlaying() then
        video:update()
    end
    if vol > 0.5 and not video.isPlaying() then
        video:play()          // si el volumen es mayor a 0.5 video se reproduciría
    end
    if video:getPosition() > 0.999 and video.isPlaying() then
        video:stop()         // si el video llega al final se detiene
    end
end

function draw()
    gaBackground(0.0,1.0)
    ofSetColor(255)
    if video.isPlaying() then    // muestra el video
        video:draw(0,posY,OUTPUT_W,scaleH)
    end
end

```

La segunda estructura condicional del bloque `update()` activa la reproducción del video cuando el volumen registrado por el micrófono supera el valor `0.5` y no se está reproduciendo en ese momento el vídeo. La siguiente condicional, detiene la reproducción cuando está llegando al final, porque los valores que devuelve el método `getPosition()` van de `0.0` a `1.0`.

`gaGetFFT(canal, posición)`, devuelve el valor del análisis FFT en el canal de entrada de audio y en la posición del buffer especificados. El parámetro `nº posición` depende del tamaño de buffer que se haya elegido en la pestaña **Audio Streaming** de Preferences. La variable global interna de GAmuza `BUFFER_SIZE` acoge ese tamaño del buffer y es utilizada comúnmente con esta función.

```
/*
GAmuza 043           E-9-4
-----
Audio input - FFT
creado por n3m3da | www.d3cod3.org
*/

rotationX = 0.0
rotationY = 0.0
thetaX = 0.0
thetaY = 0.0

function setup()
    ofEnableSmoothing()
end

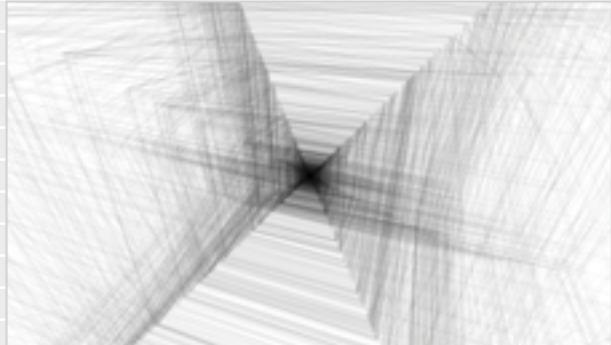
function update()
    thetaX = gaGetFFT(0,20)*10
    thetaY = gaGetFFT(0,200)*10
    rotationX += thetaX
    rotationY += thetaY
end

function draw()
    gaBackground(1.0,0.005)
    ofPushMatrix()
    ofTranslate(OUTPUT_W/2,OUTPUT_H/2,0.0)
    ofRotateX(rotationX)
    ofRotateY(rotationY)
    ofSetColor(0,40)
    ofNoFill()

    for i=0,BUFFER_SIZE-1 do
        ofDrawBox(gaGetFFT(0,i)*200)
    end

    ofPopMatrix()
end

```



`gaGetBin(nº canal, nº bin)`, devuelve el valor de los bins del análisis FFT en el canal de entrada de audio y bin especificado en sus parámetros. El nº bin puede ir de 0 a 23, la variable de sistema `FFT_BANDS` devuelve el nº de FFT bins. Los bins vienen de la escala de Bark, una escala psicoacústica propuesta por Eberhard Zwicker en 1961, que corresponde a la subdivisión de las primeras 24 bandas críticas del oído. Ejemplo:

```
/*
GAmuza 043           E-9-5
-----
Audio input Análisis escala Bin
creado por n3m3da | www.d3cod3.org
*/
function draw()
    gaBackground(1.0,0.03)
    ofEnableAlphaBlending()
    ofSetColor(20,20,20,80)
    ofNoFill()
        // dibuja fft bins
    for i=0,FFT_BANDS do
        ofRect(i*(OUTPUT_W/FFT_BANDS),OUTPUT_H,OUTPUT_W/FFT_BANDS,-OUTPUT_H*gaGetBin(0,i))
    end
    ofDisableAlphaBlending()
end
```

9.3. Audio Sampling

Las funciones de GAmuza para Audio Sampling utilizan los dispositivos para entrada y salida de audio simultáneamente. Las funciones son:

`gaDrawRecHead()`, `gaRecSampleLooping()`, `gaRecSamplePaused()`, `gaRecSamplePlay()`, `gaRecSampleSpeed()`, `gaRecSampleStop()`, `gaRecSampleVolume()`, `gaStartRec()`, y `gaStopRec()`.

`gaDrawRecHead(int,int,int,int,int)`, representa el cabezal de grabación de un sample previamente grabado. Los samples se indexan automáticamente con un id que empieza por el valor **0**. Si se han creado 100 samples el primero es el **0** y el último el **99**. Los parámetros de esta función son: `gaDrawRecHead(index,x,y,w,h)`, donde `index` es el número id del sample grabado, `x` e `y` son las coordenadas del rectángulo del cabezal, `w` y `h` son la anchura y altura de ese rectángulo.

`gaRecSampleLooping(int,bool)` asigna el estado del loop de un sample previamente grabado. Sus parámetros son `gaRecSampleLooping(index,loopState)`, donde `index` es el número id del sample grabado, y `loopState` es booleano y puede ser `true` o `false`

`gaRecSamplePaused(index,pauseState)` activa o desactiva la pausa de un sample previamente grabado con las mismas características de su id explicadas antes, `index` es el número id del sample grabado, y `pauseState` es booleano: `true` o `false`

`gaRecSamplePlay()`, reproduce un sample previamente grabado con las mismas características de su id explicadas antes. Su parámetro es: `gaRecSamplePlay(index)`

`gaRecSampleSpeed(index,speed)`, asigna la velocidad de reproducción de un sample previamente grabado, `index` es el número id del sample grabado, y `speed` es un número cuyo valor puede oscilar entre **0.0** y **1.0**.

`gaRecSampleStop(index)` para la reproducción de un sample previamente grabado con las mismas características de su id explicadas antes.

`gaRecSampleVolume(index,vol)` asigna el nivel de volumen en la reproducción de un sample previamente grabado, `index` es el número id del sample grabado, y `vol` es un número cuyo valor puede oscilar entre **0.0** y **1.0**.

`gaStartRec(channel)` activa el buffer de grabación de audio en un canal seleccionado, `channel` es el canal de entrada de audio

`gaStopRec()` detiene el buffer de grabación de audio previamente activado con `gaStartRec()`. No tiene parámetros. Después de parar la grabación, la posibilidad de grabación de un nuevo sample de audio estará automáticamente disponible.

```
/*
GAmuza 043           E-9-6
-----
Grabación - reproducción sample - creado por n3m3da | www.d3cod3.org
*/
record = false
numRecordings = 0

function draw()
    gaBackground(0.0,1.0)
    ofSetCircleResolution(50)
    ofSetColor(255,125,55)
    ofDrawBitmapString(string.format("Samples recorded: %i",numRecordings),200,200)
    if record then
        ofSetColor(255,0,0)
        ofCircle(OUTPUT_W/2,OUTPUT_H/2,100)
    end
    if numRecordings>0 then
        for i=0, numRecordings-1 do
            gaDrawRecHead(i,0,0,OUTPUT_W,OUTPUT_H)
        end
    end
end

function mouseReleased()
    if numRecordings>0 then
        for i=0, numRecordings-1 do
            gaRecSampleVolume(i,1.0)
            gaRecSampleLooping(i,true)
            gaRecSamplePlay(i)
            gaRecSampleSpeed(i,1.0)
        end
    end
end

function keyReleased()
    if gaKey() == string.byte(' ') then
        record = not record
        if record then
            gaStartRec(0)
        else
            gaStopRec()
        numRecordings = numRecordings + 1
        end
    end
end
```

9.4. Síntesis básica de audio

GAmuza no está especializado en la generación de audio ya que existen otros software *open source* que cubren mejor este campo como Pure Data⁸¹ o CSound⁸², con los que GAmuza se puede comunicar por OSC, si bien como veremos en el apartado 9.6 hay también un wrapper de funciones específicas para comunicar con Pure Data directamente. Pero para proyectos cuyo planteamiento no contempla una generación de audio compleja GAmuza incorpora una serie de funciones que permiten un nivel básico de síntesis de sonido. Para ello tiene que estar seleccionado un dispositivo de audio con capacidad de salida en **Preferences**.

Las funciones de GAmuza para generación de audio son: **gaWave()**, **gaMonoWave()**, **gaWaveVolume()**, **gaWaveFrequency()**, **gaWaveTuning()**, **gaNToF()**, **gaWaveType()**,

La función **gaWave()** actúa como un oscilador de onda con canal múltiple (que puede reproducirse en todos los canales de salida de audio). Sus parámetros son **gaWave(tipo onda, frecuencia)**. Los tipos de onda disponibles son:

GA_SINE y **GA_COSINE**, generan una onda sinusoidal y cosinusoidal. Este tipo de ondas solo contienen la onda fundamental sin parciales o armónicos (series armónicas) y sólo se pueden producir electrónicamente.



GA_TRI, genera una onda triangular, que contiene la fundamental y las series armónicas (o parciales) impares, cuyas amplitudes mantienen una relación de $1/n^2$, siendo n el número que corresponde a la serie de armónicos.



GA_PULSE genera una onda de pulso. La forma más común de pulso es la onda cuadrada que se genera con **GA_RECT**, esta tiene la particularidad de que su relación entre compresión y rarefacción es 2:1, pero en las ondas de pulso esta relación se puede cambiar. Las ondas de pulso y las cuadradas contienen la onda fundamental y las series armónicas impares, pero en este caso con una relación entre su amplitud de $1/n$, por lo que su sonoridad es muy distinta a la onda triangular.



GA_SAW genera ondas de diente de sierra, contiene junto a la onda fundamental todas las series armónicas, pares e impares, y su relación de amplitud es $1/n$



81 Ver <<http://puredata.info>> [13.02.2014]

82 Ver <<http://www.csounds.com/tutorials>> [13.02.2014]

Además de los tipos de onda, la función `gaWave()` puede tener en su primer parámetro la generación de ruido, expresado con la variable de sistema: `GA_NOISE` para ruido blanco; `GA_PINK` ruido rosa; `GA_BROWN` ruido marrón; y por último el fasor: `GA_PHASOR`

La segunda coordenada es la frecuencia. El oído es sensible a las frecuencias entre 20 a 20.000 Hz. Los tonos graves van de 20 a 300 Hz, los medios de 300 a 2.000 Hz y los agudos de 2.000 a 20.000 Hz.

La función `gaMonoWave()` es semejante a la anterior pero sólo para un canal de salida, por lo que incorpora un parámetro más, el número del canal por el que se desee escuchar el sonido `gaMonoWave(tipo de onda, frecuencia, nº canal)`

La función `gaWaveFrequency()` ajusta la frecuencia de un oscilador de onda que ha sido previamente creado. Cuando se crean distintos pasos en un oscilador, cada uno se indexa automáticamente con un número de identificación a partir de 0, por lo que si se ha creado un oscilador con 100 pasos, el primero tendrá un id 0 y el último un id 99, si el id seleccionado no existe, nada va a suceder. Sus parámetros son: el id del oscilador de onda y el nuevo valor de frecuencia a aplicar.

En el siguiente ejemplo, un oscilador `gaWave()` regula una onda de diente de sierra, `GA_SAW`, variando su frecuencia con los valores que le va asignando una estructura repetitiva `for` en 101 pasos, de 0 a 100, al mismo tiempo asigna valores a dos tablas que se utilizarán en el bloque `update()` para actualizar el volumen de cada uno de los pasos y la frecuencia relacionada con la posición del ratón.

```
/*
GAmuza 043           E-9-7
-----
Generación básica síntesis audio
creado por n3m3da | www.d3cod3.org
*/
randFreq = {}
randVol = {}
function setup()
  for i=0, 100 do
    gaWave(GA_SAW,4*i)
    randVol[i] = ofRandom(0.6,1.0)
    randFreq[i]= ofRandom(0.1,10.0)
  end
end
function update()
  for i=0, 100 do
    gaWaveFrequency(i,randFreq[i]*gaMouseX())
    gaWaveVolume(i,randVol[i])
  end
end
function draw()
  gaBackground(0.0,1.0)
end
```

Como hemos visto en este ejemplo, la función `gaWaveVolume()` ajusta el volumen de un oscilador previamente creado, por lo que hay que tener en cuenta el id del oscilador igual que en la función anterior. Sus parámetros son: el id del oscilador de onda y el nuevo valor de volumen, que va de `0.0` (mínimo) a `1.0` (máximo)

La función `gaNTOF()`, convierte una nota musical en su frecuencia armónica, usando el "Just Scale" (a veces referido como "afinación armónica" o "escala de Helmholtz"). Su parámetro es la nota; las notas disponibles son `DO_N`, `DOB_N`, `RE_N`, `REB_N`, `MI_N`, `FA_N`, `FAB_N`, `SOL_N`, `SOLB_N`, `LA_N`, `LAB_N`, `SI_N`; siendo N un número desde 0 a 8, por ejemplo `LA_4` que se corresponde 440 Hz. La siguiente tabla muestra el valor en frecuencia de las notas LA en las distintas octavas:

Nombre alemán octavas	Notación científica	Notación GAmuza	Frecuencia (Hz).
Subcontraoctava	A0	LA_0	27.50
Contraoctava	A1	LA_1	55
Gran octava	A2	LA_2	110
Pequeña octava	A3	LA_3	220
Octava prima	A4	LA_4	440
Octava segunda	A5	LA_5	880
Octava tercera	A6	LA_6	1.760
Octava cuarta	A7	LA_7	3.520
Octava quinta	A8	LA_8	7.040

Ejemplo:

```
/*
GAmuza 043           E-9-8
-----
Síntesis audio - Notas musicales
*/
nota = {DO_5, RE_4, MI_3, FA_2, SOL_3, LA_4, SI_5, DO_6, RE_5, MI_4, FA_3, SOL_4,
LA_5, SI_6, DO_7, RE_6, MI_5, FA_4, SOL_3, LA_2, SI_1, DO_8, RE_7, MI_6, FA_5, SOL_4,
LA_3, SI_2 }
totalNotas = 27
actualNota = 1
wait = 100

function setup()
  gaWave(GA_TRI,100.0)
  gaWaveVolume(0,0.5)
end
```

```

function update()
    if ofGetElapsedTimeMillis() > wait then
        actualNota += 1
        if actualNota > totalNotas then
            actualNota = 0
            actualNota += 1
        end
        ofResetElapsedTimeCounter()
        gaWaveFrequency(0, gaNtOf(nota[actualNota]))
    end
end

function draw()
    gaBackground(0.0,1.0)
end

```

En este ejemplo, hay un solo oscilador, por lo que su id es `0`, con una forma de onda triangular a `100` hertzios de frecuencia; la función `gaWaveVolume(0, 0.5)` le asigna un nivel de volumen medio y la función `gaWaveFrequency()` le ajusta secuencialmente la frecuencia asignándole cada una de las notas almacenadas en la tabla, el tiempo se controla de forma similar al utilizado en el apartado 6.2.1. Animación con archivos de imagen y control del tiempo (fps).

La función `gaWaveType()` asigna un tipo de onda a un oscilador previamente creado, como en otras funciones vistas anteriormente, el primer parámetro indica el id del oscilador, el segundo parámetro es el nuevo tipo de onda. Los tipos de onda disponibles son los señalados al principio de este capítulo.

Ejemplo:

```

/*
GAmuza 043           E-9-9
-----
Síntesis audio - Cambio tipo de onda
creado por n3m3da | www.d3cod3.org
*/

r = 0

function setup()
    for i=0, 100 do
        gaWave(GA_SINE,6*i)
        gaWaveVolume(i,ofRandom(0.6,1.0))
    end
end

```

```

function update()
    for i=0, 100 do
        r = ofRandomf()
        if r < 0.92 then
            gaWaveType(i, GA_SINE)
        else
            gaWaveType(i, GA_RECT)
        end
    end
end

function draw()
    gaBackground(0.0,1.0)
end

```

El siguiente ejemplo plantea un nivel superior de generación de audio y gráficos combinado el uso de las funciones descritas y con un bloque de función propia denominado `freqPassFilter()` que regula los parámetros de las funciones `gaWaveFrequency()`

```

/*
GAmuza 1.0.1           E-9-10
-----
Audio/generativeAudio.ga
generative audio synth/graphics
nivel intermedio
creado por n3m3da | www.d3cod3.org
*/

```

```

f = 440
vol = 0.8

noiseTime = ofRandom(0,10000)
step = ofRandom(0.01,0.03)

```

```

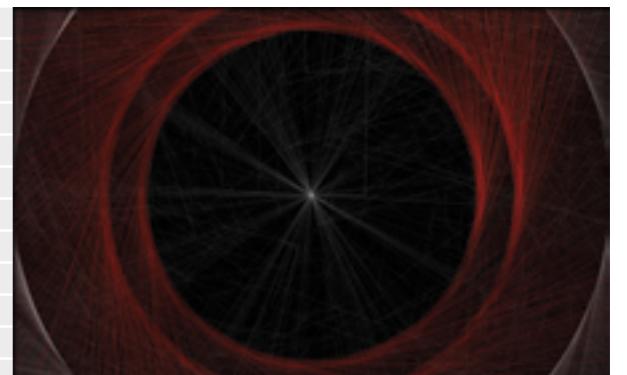
phase = 0

```

```

function setup()
    gaWave(GA_RECT,f)
    gaWaveVolume(0,0.9)
    gaWave(GA_PULSE,f/200)
    gaWaveVolume(1,0.1)
end

```



```

function update()
    mapFreq = ofMap(ofNoise(noiseTime),0.15,0.85,100,700)
    gaWaveFrequency(0,freqPassFilter(mapFreq,f,f/2))
    gaWaveFrequency(1,freqPassFilter(mapFreq,f/200,f/2))
    noiseTime += step
    phase += ofNoise(noiseTime)
end

function draw()
    gaBackground(0.0,0.001)
    ofPushMatrix()
    ofTranslate(OUTPUT_W/2,OUTPUT_H/2,0)
    ofRotateZ(phase)

    ofSetColor(freqPassFilter(mapFreq,f,f/2),20)
    ofNoFill()
    ofRect(OUTPUT_W/2,0,freqPassFilter(mapFreq,f,f/2),OUTPUT_H)
    ofRect(OUTPUT_W/2,0,-freqPassFilter(mapFreq,f,f/2),OUTPUT_H)

    ofFill()
    ofSetColor(255,0,0,30)
    posY = (OUTPUT_H/2)+freqPassFilter(mapFreq,f/200,f/2)
    ofLine(0,posY,OUTPUT_W,posY)
    posY = (OUTPUT_H/2)-freqPassFilter(mapFreq,f/200,f/2)
    ofLine(0,posY,OUTPUT_W,posY)

    ofPopMatrix()
end

function freqPassFilter(_input,freq,range)
    if _input < freq+(range/2) and _input > freq-(range/2) then
        return gaNToF(math.ceil(ofRandom(_input/6,_input/12)))
    else
        return gaNToF(math.ceil(ofRandom(16,24)))
    end
end

```

9.5. Audio Unit Plugins

El conjunto de funciones de GAMUZA de Audio Unit (au) está vinculado a la arquitectura de plugins que reconoce el sistema operativo de Mac OS X, tanto los que el propio sistema incorpora como otros instalados, por ejemplo, Simbiosis o FXpansion VST-AU Adapter, para procesar o manipular la señal de audio en tiempo real con una latencia mínima.

Estos plugins aparecen en la ventana del logger.

The screenshot shows a terminal window titled "GA logger" with a list of audio unit plugins. The log entries are as follows:

- 22-08-2003 20:14:23 -> AUDIO STREAMING STARTED
- 22-08-2003 20:14:23 -> PURE DATA SYNTHESIZER ENGINE STARTED
- 22-08-2003 20:14:23 AUDEO UNIT PLUGINS AVAILABLE
- 22-08-2003 20:14:23 0 - Apple: AUReverb
- 22-08-2003 20:14:23 1 - Apple: AUdynamicsProcessor
- 22-08-2003 20:14:23 2 - Apple: AUDelay
- 22-08-2003 20:14:23 3 - Apple: AUDistortion
- 22-08-2003 20:14:23 4 - Apple: AUDFilter
- 22-08-2003 20:14:23 5 - Apple: AUGraphEQ
- 22-08-2003 20:14:23 6 - Apple: AUHighPass
- 22-08-2003 20:14:23 7 - Apple: AUHighShelfFilter
- 22-08-2003 20:14:23 8 - Apple: AUPeakDetector
- 22-08-2003 20:14:23 9 - Apple: AUReverb
- 22-08-2003 20:14:23 10 - Apple: AULowShelfFilter
- 22-08-2003 20:14:23 11 - Apple: AUMultiBandCompressor
- 22-08-2003 20:14:23 12 - Apple: AUMatrixReverb
- 22-08-2003 20:14:23 13 - Apple: AUReverb
- 22-08-2003 20:14:23 14 - Apple: AUParametricEQ
- 22-08-2003 20:14:23 15 - Apple: AUReverb
- 22-08-2003 20:14:23 16 - Apple: AUSequencer
- 22-08-2003 20:14:23 17 - Apple: AUWTF

Ejemplo:

```

/*
GAMUZA 043           E-9-11
-----
Audio unites plugins
creado por n3m3da | www.d3cod3.org
*/

dim= BUFFER_SIZE * AUDIO_OUTPUT_CHANNELS
outputBuffer = memarray('float', dim)
stretch = OUTPUT_W/(dim-1)
zeroOffset = OUTPUT_H /4

function setup()
    gaWave(GA_SINE, 220)
    auAddPlugin("Apple: AUMatrixReverb") //inicializa un plugin de audio
end

function update()
    gaWaveVolume(0, (gaMouseY()/OUTPUT_H)/4)

    for i=0, dim-1 do
        outputBuffer[i] = gaGetOutputBuffer(i)
    end
end

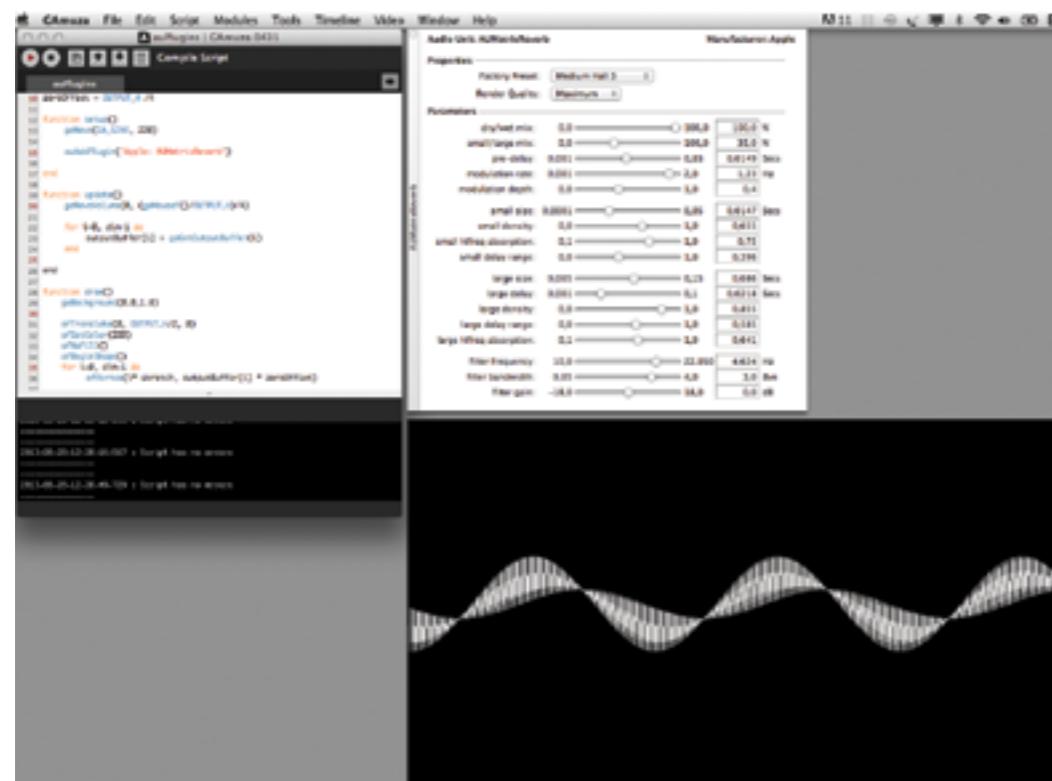
```

```

function draw()
gaBackground(0.0,1.0)

ofTranslate(0, OUTPUT_H/2, 0)
ofSetColor(255)
ofNoFill()
ofBeginShape()
for i=0, dim-1 do
    ofVertex(i* stretch, outputBuffer[i] * zeroOffset)
end
ofEndShape(false)
end

```



Al activar el script, se abre la interfaz del plugin pudiendo ajustar sus valores con el ratón.

9.6. Audio Pure Data Synthesis Engine

Como se ha señalado antes, hay un wrapper de funciones específicas para comunicar con Pure Data sin necesidad de utilizar el protocolo OSC, a continuación se listan estas funciones indicando sus parámetros y una breve descripción.

pdAddToSearchPath(string "folder")

Indica la carpeta en la que están los archivos de pd. Esta carpeta debe estar dentro de data.

pdOpenPatch(string gaDataPath("folderName/fileName.pd"))

Abre el archivo de Pure data con la ruta/nombre especificados.

pdClosePatch(string gaDataPath("folderName/fileName.pd"))

Cierra el archivo de Pure data con la ruta/nombre especificados.

pdStart()

Inicia el archivo de Pure data.

pdStop()

Detiene el archivo de Pure data.

pdComputeAudio(bool true or false)

Activa o desactiva la opción Compute Audio de Pure data.

pdSendBang(string "labelName")

Envía un bang al objeto de pure data con la etiqueta especificada.

pdSendFloat(string "labelName", float value)

Envía un valor float al objeto de pure data con la etiqueta especificada.

pdSendSymbol(string "labelName", string "symbolName")

Envía un símbolo al objeto de pure data con la etiqueta especificada.

pdStartMessage()

Inicializa el sistema de envío de mensajes a Pure data.

pdAddFloat(float value)

Añade el valor float especificado en su parámetro.

pdAddSymbol(string "symbolName")

Añade el símbolo especificado en su parámetro.

pdFinishList(string "labelName")

Finaliza el envío de la lista de mensajes a la etiqueta especificada.

pdFinishMessage(string "labelName", string "messageName")

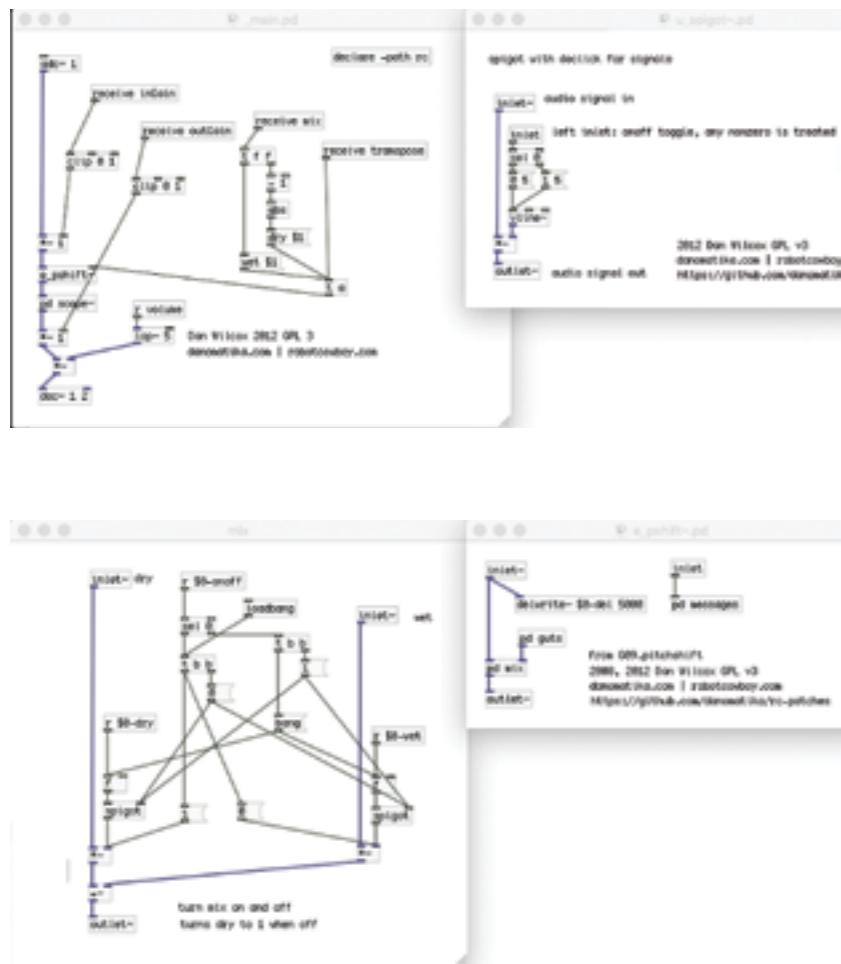
Finaliza el mensaje indicado de la etiqueta especificada en sus parámetros.

pdSendNoteOn(int channel, int pitch, int velocity)

Envía una nota con el pitch, velocidad y por el canal especificados.

```
pdSendControlChange(int channel, int controller, int value)
    Envía un nuevo valor al controlador y por el canal especificados.
pdSendProgramChange(int channel, int value)
pdSendPitchBend(int channel, int value)
pdSendAftertouch(int channel, int value)
pdSendPolyAftertouch(int channel, int pitch, int value)
```

Un ejemplo del uso de algunas de estas funciones, en el que los patch de Pure data configuran un PitchShifter:



En GAMUZA, además del código para trabajar con los plugins, se le incorpora una interface gráfica con la clase del addon de openFrameworks `ofxControlPanel()`, el ejemplo está basado en un trabajo de Golan Levin y el CMU Studio for Creative Inquiry⁸³

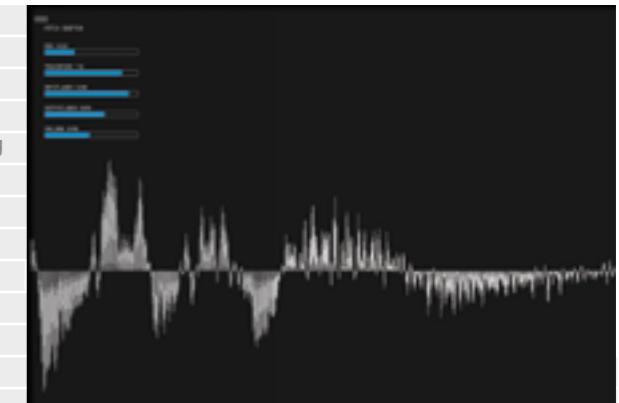
```
/*
GAMUZA 0432           E-9-12
-----
PD Pitch Shifter
creado por n3m3da | www.d3cod3.org
*/
gui = ofxControlPanel()
viewGui = true
mix = 0
_transpose = 0
inGain = 0
outGain = 0
volume = 0

dim = BUFFER_SIZE*AUDIO_OUTPUT_CHANNELS
outputBuffer = memarray('float', dim)
stretch = OUTPUT_W/(dim-1)
zeroOffset = OUTPUT_H/4

function setup()
    gui:setup(0,10,480,OUTPUT_H)      // GUI setup
    gui:addPanel(1)
    gui:setWhichPanel(0)
    gui:setWhichColumn(0)
    gui:addLabel("PITCH SHIFTER")
    gui:addSlider("MIX", mix, 0.0, 1.0, false)
    gui:addSlider("TRANSPOSE", _transpose, -12, 12, false)
    gui:addSlider("INPUT_GAIN", inGain, 0.0, 1.0, false)
    gui:addSlider("OUTPUT_GAIN", outGain, 0.0, 1.0, false)
    gui:addSlider("VOLUME", volume, 0.0, 1.0, false)
    gui:loadSettings(guiImportFile("guiSettings.xml"))

    for i = 0, dim - 1 do      // inicializar memarray outputBuffer
        outputBuffer[i] = 0.0
    end

    pdStop()
    pdOpenPatch(gaDataPath("pd/_main.pd"))
}
```



⁸³ Ver <<http://flong.com/>> y <<http://studiofrocreativeinquiry.com/>> [20.03.2016]

```

pdStart()
pdComputeAudio(true)
end

function update()
    // actualizar GUI
    gui:update()
    mix = gui:getValueF("MIX")
    _transpose = gui:getValueF("TRANSPOSE")
    inGain = gui:getValueF("INPUT_GAIN")
    outGain = gui:getValueF("OUTPUT_GAIN")
    volume = gui:getValueF("VOLUME")
    // enviar mensaje al patch PD
    sendPDMessag("mix", mix)           // 0 - 1
    sendPDMessag("transpose", _transpose) // -12 - 12
    sendPDMessag("inGain", inGain)      // 0 - 1
    sendPDMessag("outGain", outGain)     // 0 - 1
    pdSendFloat("volume", volume)

    for i = 0, dim - 1 do             // actualizar buffer de salida de audio
        outputBuffer[i] = gaGetOutputBuffer(i)
    end
end

function draw()
    gaBackground(0.1,1.0)
    ofPushMatrix()
    ofTranslate(0,OUTPUT_H/2,0)
    // Dibujar niveles del audio
    ofSetColor(255)
    ofNoFill()
    ofBeginShape()

    for i = 0, dim - 1 do
        ofVertex(i* stretch, outputBuffer[i] * zeroOffset)
    end

    ofEndShape(false)
    ofPopMatrix()

    if viewGui then
        gui:draw()
    end
end

```

```

function sendPDMessag(m,v)
    pdStartMessage()
    pdSendFloat(m,v)
    pdFinishList("TO_PD")
end

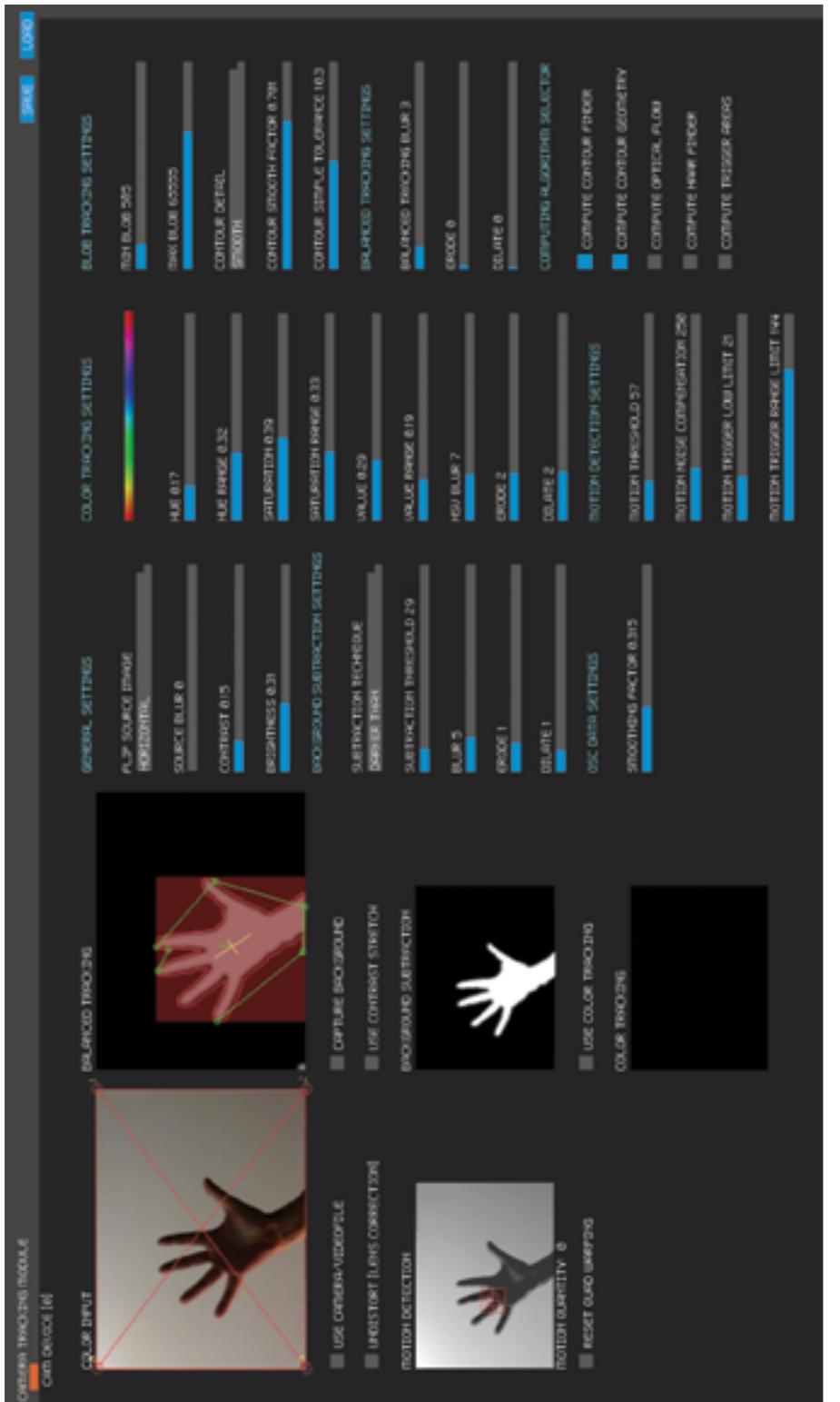
function mouseDragged()
    gui:mouseDragged()
end

function mousePressed()
    gui:mousePressed()
end

function mouseReleased()
    gui:mouseReleased()
end

function keyReleased()
    if gaKey() == string.byte('s') then
        gui:saveSettings(gaImportFile("guiSettings.xml"))
    elseif gaKey() == string.byte('g') then
        viewGui = not viewGui
    end
end

```



Panel Tracking Video

10. Tracking video

En GAmuza se pueden hacer scripts para tracking video utilizando las clases y métodos propios de openFrameworks: `ofVideoGrabber()` u `ofxKinect()` para detectar la fuente y los addons de openCV `ofxCvColorImage()`, `ofxCvContourFinder()`, `ofxCvFloatImage()`, `ofxCvGrayscaleImage()`, `ofxCvShortImage()`, `ofxCvHaarFinder()`, `ofxOpticalFlowLK()`, o utilizar el panel de Computer visión con interfaz GUI que ha sido preconfigurado principalmente para facilitar este proceso a aquellos que no sean expertos en programación.

A diferencia de los módulos, los paneles se generan por programación en la ventana de salida usando las clases de GAmuza `gaCameraTracking()` o `gaKinectTracking()`, se pueden activar desde los ejemplos Computer Vision/camTrackingPanel para cámaras de video o Computer Vision/kinectTrackingPanel cuando se trabaja con el sensor kinect.

10.1. Panel Tracking: por cámara video

En este apartado se describe la interface del panel de Tracking para cámaras de video y los métodos de la clase de GAmuza `gaCameraTracking()`, explicándolo a partir del código que genera su carga.

```
/*
GAmuza 043          E-10-1
-----
Panel de Tracking video
creado por n3m3da | www.d3cod3.org
*/
camPanel = gaCameraTracking() // asocia variable global a la clase de GAmuza
drawGUI = true
camID = 0

captureWidth = 320           // tamaño captura de video
captureHeight = 240          // textura para acoger la imagen video
cam = ofTexture()

function setup()             // para guardar el archivo de configuración XML
    camPanel:setGuiSettingsFile(gaDataPath("camTrackingSettings.xml"))
    camPanel:setup(camID,captureWidth,captureHeight) // inicializa panel
                                                // inicializa textura openGL para imagen video
    cam:allocate(captureWidth,captureHeight,GL_RGB)
end
```

```

function update()
    cam = camPanel:getCameraTextureMod() // la textura actualiza los frames de video
    camPanel:update() // actualiza el panel de tracking
end

function draw()
    gaBackground(0.1,1.0)
    // para mostrar la imagen en directo de la cámara a pantalla completa
    ofSetColor(255)
    scaleH = OUTPUT_H
    scaleW = scaleH* captureWidth / captureHeight
    cam:draw(OUTPUT_W/2 - scaleW/2,0, scaleW,scaleH)

    ofSetColor(255)
    if drawGUI then
        camPanel:draw() // muestra el panel si drawGUI está activo
    end
end

function keyReleased()
    if gaKey() == string.byte('g') then
        drawGUI = not drawGUI // con la tecla g se muestra/oculta el panel
    end
end

function mouseDragged() // para ajustar las opciones del panel con el ratón
    camPanel:mouseDragged(gaMouseX(),gaMouseY())
end

function mousePressed()
    camPanel:mousePressed(gaMouseX(),gaMouseY())
end

function mouseReleased()
    camPanel:mouseReleased(gaMouseX(),gaMouseY())
end

```

Como en otros trabajos con clases, inicialmente se asocia una variable global, `camPanel`, a la clase `gaCameraTracking()`; la segunda variable, `drawGUI`, es booleana y permitirá visualizar/ocultar el panel en la ventana de salida; `camID` asigna el ID de la cámara y las dos siguientes variables, `captureWidth` y `captureHeight`, definen las dimensiones del video capturado; la última variable, `cam`, está vinculada a la clase `ofTexture()` como soporte de la imagen capturada por la cámara de video

En el bloque `setup()` se inicializa el panel con el método `setGuiSettingsFile` cuyo parámetro indica la ruta y nombre del archivo XML que guardará la configuración del panel. Al activar el script aparece el panel en la ventana de salida, tras realizar los ajustes que requiera el proyecto, debe guardarse esa configuración clicando el botón "save" situado en el vértice superior derecho del panel. Al guardarla se genera el archivo `camTrackingSettings.xml` en la carpeta `data` del script. El método `setup` tiene como parámetros el ID de la cámara y el tamaño del frame. Después, con el método `allocate` de la clase `ofTexture()` se asignan el tamaño y tipo de color de openGL del frame de vídeo.

En el bloque `update()`, se identifica la textura con el panel a través del método `getCameraTextureMod()` [devuelve la captura de la cámara con el eventual mirror aplicado en horizontal/vertical] o con `getCameraTexture()` [devuelve la captura desde la cámara en "raw"], con cualquiera de ellos se relaciona la clase de openFrameworks con la de GAmuza, y después con el método `update` se actualiza, para cada frame, la información de la imagen capturada.

En el bloque `draw()`, se dibuja la imagen en directo de la cámara adaptada a pantalla completa, y después, para que quede encima de la imagen, se dibuja el panel cuando la variable booleana `drawGUI` es verdadera.

En el bloque `keyReleased()`, se establece una condicional para que al teclear la letra "g" la variable `drawGUI` pase de verdadera a falsa y viceversa, construyendo un switch para ver u ocultar el panel.

Los bloques `mouseDragged()`, `mousePressed()` y `mouseReleased()` permiten manipular con el ratón los sliders y botones de la interface del panel.

10.1.1. Descripción panel

El panel Tracking video permite analizar características visuales del campo que registra la cámara, recoger determinados datos como la luminosidad, color o movimiento, seleccionar y/o ajustar esos datos, para después generar con ellos otra situación que responda en tiempo real a sus cambios, facilitando la aplicación de los algoritmos para tracking video al regular con sliders, menús y botones las condiciones básicas de entrada de datos al sistema.

GAmuza recoge el reconocimiento que el ordenador ha hecho de la cámara o cámaras conectadas, asignando un ID a cada una de ellas, empezando por el 0, es decir, si solo hay una cámara conectada su id es 0, si hay dos la primera es 0 y la segunda 1 y así sucesivamente.

Las interface del panel está distribuida en función de las distintas técnicas básicas para tracking video:

Blob detection, reconocimiento de regiones o áreas. Puede detectar los blobs por **Background subtraction** o por **Color tracking**. Para el reconocimiento del contorno se debe activar el algoritmo **Compute Contour Finder** y para la geometría del contorno hay que añadir el algoritmo **Compute Contour geometry**

Motion detection, reconoce la cantidad de movimiento y localiza el punto medio de la masa, siempre está activo aunque ningún algoritmo haya sido seleccionado

Haar Finder, para reconocimiento de partes del cuerpo humano, el algoritmo **Compute Haar Finder** debe estar seleccionado

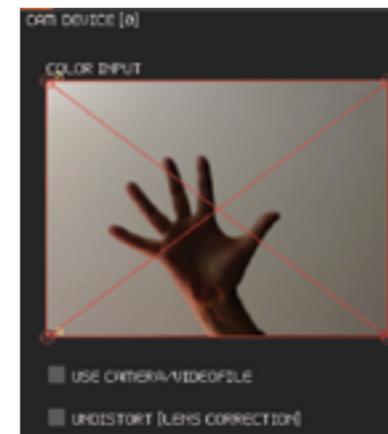
Optical Flow, dirección del movimiento representado vectorialmente, debe estar seleccionado el algoritmo **Compute Optical Flow**

Trigger Areas, son nueve áreas de activación ajustables en tamaño y posición desde el módulo **Computer vision**. Se debe activar el algoritmo **Compute Trigger Areas**

Análisis detallado de la interface

El monitor de entrada de la señal video, **Color input**, permite reducir el área a trackear, este *crop* se ajusta arrastrando con el ratón los puntos resaltados de los vértices. Los demás monitores muestran solo la zona seleccionada. El crop se resetea con **Reset Quad Warping**.

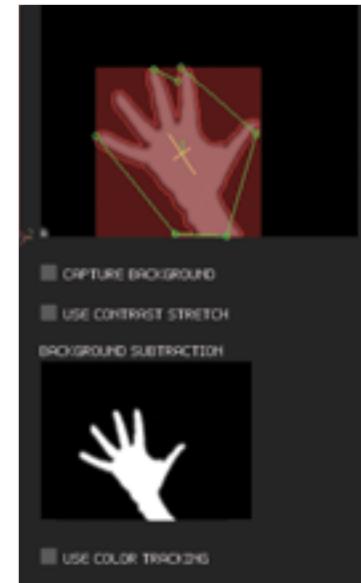
USE CAMERA/VIDEOFILE. Activa/desactiva la posibilidad de trabajar con un archivo de video pregrabado (video file) o con la imagen capturada por la cámara en directo.



El monitor **Balanced Tracking** muestra el resultado de todos los algoritmos de tracking seleccionados y ajustados con los sliders, menús y botones del bloque derecho.

Capture Background, al clicar el botón se captura un frame de la imagen video que debe corresponder con el fondo. Esta imagen se toma como referencia para comparar los cambios que se producen entre el valor de sus píxeles y el de los frames de la imagen a trackear, aplicando el algoritmo **Background Subtraction**. Es el primer nivel para el cálculo de **Tracking**, y siempre está activo. Según las condiciones de iluminación, puede mejorarse la imagen activando **Use Contrast Stretch**: incrementa el rango de los valores más intensos de la imagen en escala de grises.

El resultado de **Background subtraction** se ve en el monitor inferior.



GENERAL SETTINGS

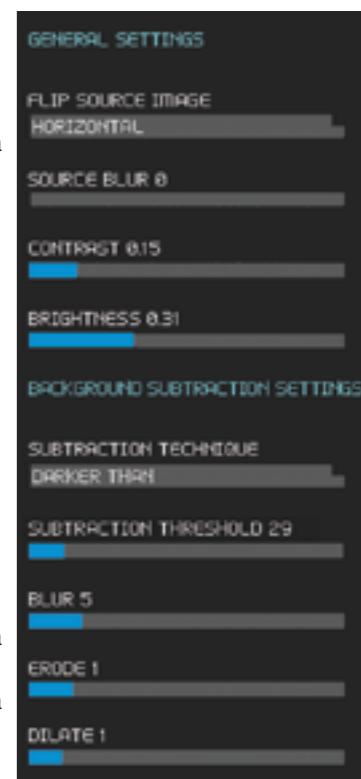
Flip Source Image volteá la imagen-video que proviene de la cámara, su ajuste depende de las condiciones de exhibición:

- Off: la imagen se percibe sin ningún cambio
- Horizontal: se selecciona cuando los movimientos ante la imagen responden al efecto de estar frente a un espejo.
- Vertical: imagen video cabeza abajo.
- Horizontal + Vertical: volteá en los dos sentidos.

Source Blur: desenfoque de la imagen de entrada de video.

Contrast: contraste.

Brightness: brillo.



BACKGROUND SUBTRACTION SETTINGS

Subtraction Technique, seleccionar entre:

- Color ABS: color absoluto
- B&W ABS: blanco y negro absoluto
- Lighter than. Cuando la figura a trackear es más clara que el fondo.
- Darken than. Cuando La figura a trackear es más oscura que el fondo.

Subtraction Threshold, ajusta el umbral de detección. Cuanto menor es el valor más sensible. Si se trabaja con Tracking Color se pone el valor máximo para anular los datos de substracción del fondo
Blur, desenfoca la imagen procesada.

Erode, reduce las superficies detectadas.

Dilate, expande las superficies detectadas

OSC DATA SETTINGS

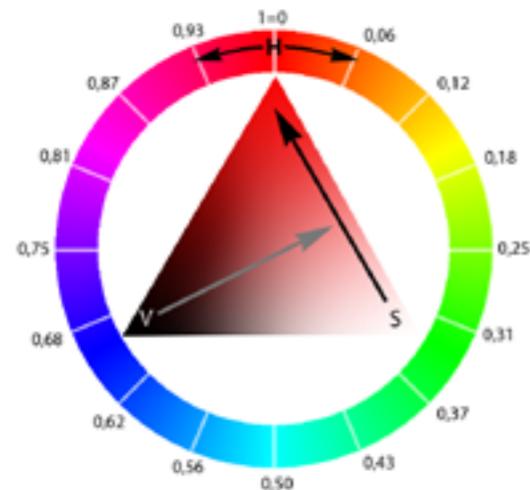
Smoothing factor: ajusta el rango de suavizado.

USE COLOR TRACKING

Color Tracking es un algoritmo de reconocimiento del color. Requiere condiciones de iluminación estables y la elección de un color de seguimiento muy diferenciado del contexto. **Use Color Tracking**, activa/desactiva su uso y los resultados se ven en el monitor inferior. Como se ha mencionado antes, cuando se trabaja con **COLOR TRACKING** debe subirse al máximo el valor del slider **Subtraction Threshold** para anular los datos del **BACKGROUND SUBTRACTION**.

COLOR TRACKING SETTINGS

La configuración de este bloque solo se ajusta si está activado **Use Color Tracking**. El Tracking Color transforma el espacio de color RGB al sistema HSV, porque asigna los valores del tono del color de una forma más estable. HSV identifica cada tono con un número que va de 0° a 360° según su posición en el círculo cromático, pero en GAmuza el rango va de 0.0 a 1.0, quedando su equivalencia tal como muestra la imagen.



Las opciones de ajuste son:

Hue, selecciona el tono a trackear según la escala de valores de GAmuza entre 0.0 a 1.0. La franja de color sirve de orientación.

Hue Range, amplía el rango del tono seleccionado.

Saturation, asigna el nivel de saturación del tono (hue) seleccionado. El valor 0.0 es igual a blanco, 1.0 es el tono puro, dependiendo del nivel de luminosidad (value).

Saturation Range, amplia el rango de saturación seleccionado.

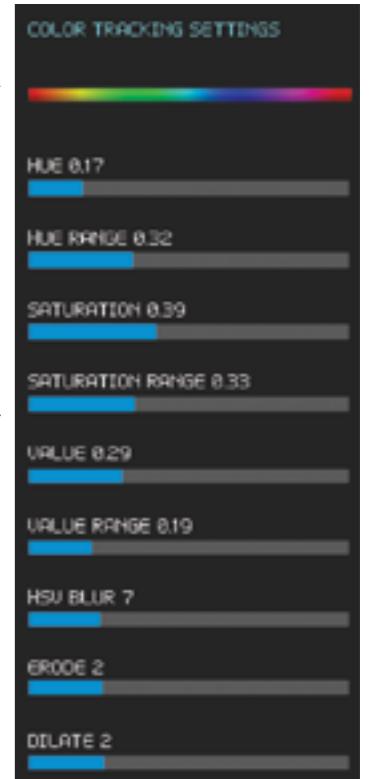
Value, regula el nivel de luminosidad (value) del tono. El valor 0.0 es igual a negro, 1.0 es igual al tono puro, dependiendo del valor de saturación.

Value Range, amplía el rango de luminosidad seleccionado.

HSV Blur, filtro de desenfoque de la imagen en HSV.

Erode, reduce las superficies detectadas.

Dilate, expande las superficies detectadas.



BLOB TRACKING SETTINGS

Los blob son las zonas independientes que el tracking detecta según los algoritmos seleccionados, sus opciones de configuración son:

Min Blob, tamaño mínimo de los blobs en píxeles.

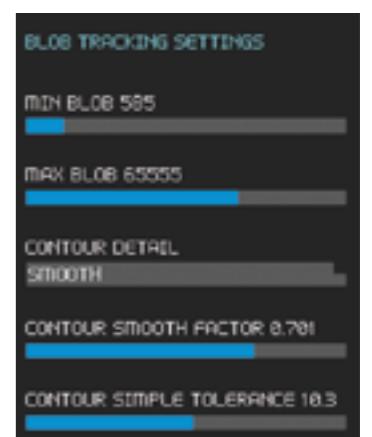
Max Blob, tamaño máximo.

Contour Detail, el detalle del contorno se puede ajustar en:

- **Raw**, todo el perímetro tal cual es
- **Smooth**, suavizado
- **Simple**, adaptado a formas geométricas simples

CONTOUR SMOOTH FACTOR, si se ha seleccionado smooth, ajusta el nivel de suavizado

CONTOUR SIMPLE TOLERANCE, si se ha seleccionado simple, ajusta el rango de vértices que articulan el contorno.

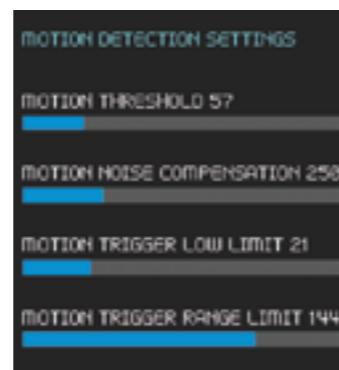


MOTION DETECTION SETTINGS

El algoritmo para detección de movimiento está siempre activo.

El monitor **MOTION DETECTION** muestra el cálculo de la cantidad de movimiento que se registra en la imagen y señala la posición media de ese movimiento. Los ajustes para su configuración son:

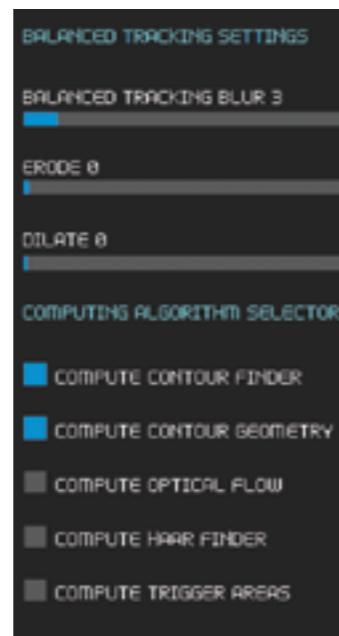
- **Motion Threshold**, regula la sensibilidad del sistema para activar la detección, si es poco sensible se debe ampliar el umbral (Threshold)
- **Motion Noise Compensation**, compensa el ruido de la señal de imagen video
- **Motion Trigger Low Limit**, nivel más bajo para activar la detección de movimiento
- **Motion Trigger Range Limit**, limita los niveles de movimiento, como si bajara la sensibilidad de detección.



BALANCED TRACKING SETTINGS

Vuelve a ajustar, en conjunto, los valores dados en cada sección.

- **Balanced Tracking Blur**, ajusta nivel de desenfoque global
- **Erode**, reduce las superficies detectadas
- **Dilate**, expande las superficies detectadas



COMPUTING ALGORITHM SELECTOR

En este bloque se seleccionan otros algoritmos específicos que se pueden aplicar, además de los mencionados hasta ahora.

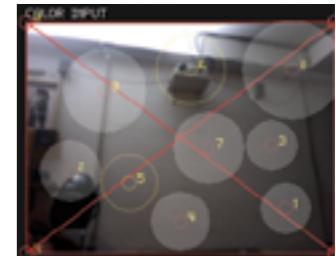
Compute Contour Finder, para procesar el contorno de las zonas detectadas. Sus valores se ajustan con Blob Tracking Settings.

Compute Contour geometry, para procesar el contorno de las zonas detectadas con formas geométricas.

Compute Optical Flow, calcula el flujo óptico del movimiento y lo describe con vectores que indican la dirección e intensidad.

Compute Haar Finder, activa el algoritmo para detectar partes del cuerpo humano. Hay distintos algoritmos posibles. Por defecto, está seleccionado `frontalface_alt` (Ver página 198).

Compute Trigger Areas. Activa la posibilidad de utilizar 9 zonas circulares que tienen asignado un ID. Su posición y tamaño pueden ajustarse con el ratón; con ellas se pueden determinar las áreas en las que debe responder el tracking utilizando funciones específicas de GAmuza (ver página 229)

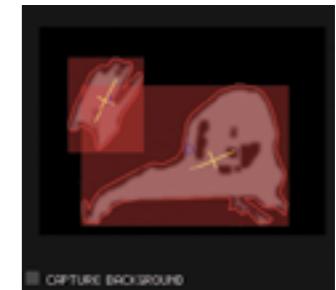


Si hay varias cámaras conectadas al ordenador, se pueden abrir distintas interfaces del panel para cada una de ellas que pueden ajustarse independientemente, por eso en las funciones de programación hay que señalar el id de la cámara.

10.1.2. Blob detection

El reconocimiento de **blobs** se realiza por métodos matemáticos que pueden detectar en la imagen zonas que tienen determinadas características constantes, o que varían dentro de un rango de valores predeterminado (tolerancia), tales como el nivel de luminosidad o color, en comparación con las áreas que les rodean, de manera que el sistema los considera similares entre sí. En GAmuza hay dos técnicas para la detección de **blobs**: **Background subtraction** y **Color Tracking**.

En el primero, sustracción del fondo, el sistema guarda una imagen tomada del fondo antes de que los objetos o personas a detectar entren en el campo visual de la cámara, para ir después comparando los cambios que se producen en el valor de cada uno de los píxeles de los frames de video, de modo que si detecta un cambio significativo en una región de la imagen, respecto del modelo de fondo, representa ese área delimitando la zona o el perímetro. El panel **Computer vision** tiene un botón para capturar el fondo situado bajo del monitor **Balanced Tracking**.



Para trabajar con la técnica **COLOR TRACKING**, se debe activar el botón **Use Color Tracking** y subir al máximo el valor del slider **Subtraction Threshold** en el módulo **Computer vision**, para anular los datos de **BACKGROUND SUBTRACTION**. El área a detectar debe tener un color muy diferente al resto de colores que entran en el campo visual de la cámara y mantener un nivel de iluminación lo más uniforme posible. El Color Tracking utiliza el espacio de color HSV, porque asigna los valores del tono del

color de una forma más estable, ver en la página 182 los parámetros de color para seleccionar sus ajustes en la interface del panel.

La clase de GAmuza `gaCameraTracking()` tiene una serie de métodos para trabajar con blobs que se comunican con el panel de computer vision, recogiendo en tiempo real el ajuste que hagamos en su menú, por ejemplo, mediante los sliders del menú **BLOB TRACKING SETTINGS** podemos desestimar los blobs menores o mayores a un determinado tamaño, en **BACKGROUND SUBTRACTION SETTINGS** podemos seleccionar si los blobs buscados son más claros o más oscuros que el fondo, también se puede reducir o ampliar la dimensión de los blobs detectados, etc. Pasamos a describir cada uno de los métodos y sus parámetros, para analizarlos en conjunto posteriormente a través de algunos ejemplos. Recordad que para que funcionen debe estar seleccionado, al menos, el algoritmo **Compute Contour Finder**:

Con el método `getNumBlobs()`, se obtiene el número de blobs que detecta una cámara específica en cada frame de GAmuza.

`getBlobX(blobID)` y `getBlobY(blobID)`, devuelven la coordenada x e y, respectivamente, de un determinado blob detectado desde una cámara específica. GAmuza identifica los blobs por orden de aparición, el blob con id 0 será el que entró primero en el encuadre de la cámara y mantendrá ese id hasta que salga. Si dos áreas inicialmente identificadas con dos id se juntan, se convierten en un solo blob y el id de todos los blobs restantes puede cambiar. El parámetro de estas funciones es: id del blob.

`getBlobW(blobID)` y `getBlobH(blobID)`, devuelven la anchura y altura, en píxeles, de un determinado blob detectado desde una cámara específica.

`getBlobContourSize(blobID)`, devuelve la posición de todos los puntos del contorno de un blob detectado por una cámara específica.

`getBlobCPointX(blobID,x)` y `getBlobCPointY(blobID,y)`, devuelven las coordenadas x e y, respectivamente, de un punto determinado del contorno.

`getBlobGeometrySize(blobID)`, devuelve las líneas geométricas que devienen del contorno de un blob desde una cámara específica. Para que funcione debe estar seleccionado el algoritmo **Compute Contour geometry**.

`getBlobGLineX1(blobID,x1)`, `getBlobGLineY1(blobID,y1)`, `getBlobGLineX2(blobID,x2)` y `getBlobGLineY2(blobID,y2)`, devuelven las coordenadas x1, y1, x2, y2 de una línea determinada del contorno geométrico de un blob. Para que funcione debe estar seleccionado el algoritmo **Compute Contour geometry**.

El siguiente ejemplo básico muestra cómo utilizar los datos obtenidos por los métodos `getNumBlobs()`, `getBlobX()`, `getBlobY()`, `getBlobW()` y `getBlobH()` desde el algoritmo **Compute Contour Finder** para detección de blobs y remarcarlos con formas rectangulares.

```
/*
GAmuza 043           E-10-2
-----
ComputerVision - Blobs
creado por n3m3da | www.d3cod3.org
*/
camPanel = gaCameraTracking() // asocia variable global a la clase de GAmuza
drawGUI = true // booleano para activar/desactivar el panel
camID = 0 // id de la cámara
runningBlobs = 0 // variable para el nº de blobs activos en cada frame
cam = ofTexture() // textura para mostrar imagen de la cámara
captureWidth = 320 // anchura y altura frame captura video
captureHeight = 240

function setup() // para guardar el archivo de configuración XML
    camPanel:setGuiSettingsFile(gaDataPath("camTrackingSettings.xml"))
    camPanel:setup(camID,captureWidth,captureHeight)
    cam:allocate(captureWidth,captureHeight,GL_RGB)
end

function update()
    cam = camPanel:getCameraTextureMod() // la textura actualiza los frames de video
    runningBlobs = camPanel:getNumBlobs() // devuelve el nº de blobs detectados
    camPanel:update() // actualiza los datos del panel
end

function draw()
    gaBackground(0.0,0.8)
    ofSetColor(255)
    scaleH = OUTPUT_H //para dibujar imagen cámara a pantalla completa
    scaleW = scaleH* captureWidth/captureHeight
    cam:draw(OUTPUT_W/2 - scaleW/2,0,scaleW,scaleH)
    ofSetLineWidth(3) // inicio dibujo de blobs, grosor líneas
    ofNoFill() // no rellenar
    ofSetColor(31,165,210) // color líneas rectángulos
    ofPushMatrix() // activa la opción escalar
    ofScale(OUTPUT_W,OUTPUT_H,1.0) // para escalar la imagen a pantalla completa
    for j=0, runningBlobs-1 do // para reconocer todos los blobs
        x = camPanel:getBlobX(j) // coordenada X de cada blob
        y = camPanel:getBlobY(j) // coordenada X
        w = camPanel:getBlobW(j) // anchura de cada blob
        h = camPanel:getBlobH(j) // altura
        ofRect(x,y,w,h) // dibuja rectángulos con esos datos
    end
    ofPopMatrix() // desactiva la opción escalar

```

```

ofSetColor(255)
if drawGUI then
    camPanel:draw() // muestra el panel si drawGUI está activo
end

function keyReleased()
    if gaKey() == string.byte('g') then
        drawGUI = not drawGUI // con la tecla g se muestra/oculta el panel
    end
end

function mouseDragged() // para ajustar las opciones del panel con el ratón
    camPanel:mouseDragged(gaMouseX(),gaMouseY())
end

function mousePressed()
    camPanel:mousePressed(gaMouseX(),gaMouseY())
end

function mouseReleased()
    camPanel:mouseReleased(gaMouseX(),gaMouseY())
end

```

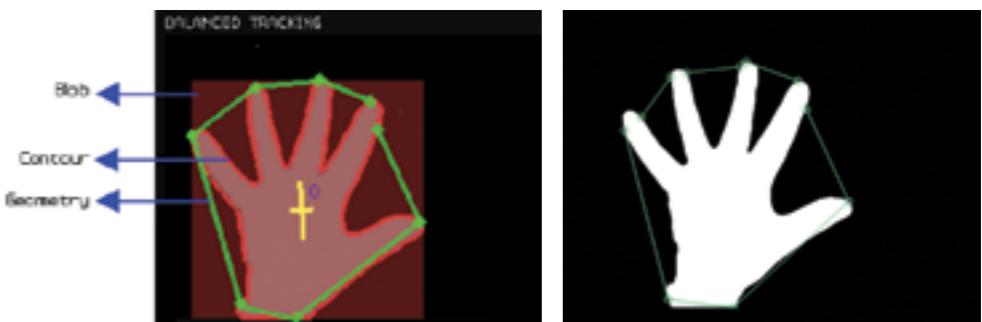
Analizaremos las diferencias que aporta este código, respecto al visto anteriormente para cargar el panel de tracking video. La variable global `runningBlobs` va a gestionar el número de blobs que detecta el sistema, como es un dato que puede cambiar según lo que acontezca delante de la cámara, en el bloque `update()` se iguala el valor de esa variable al que devuelve el método `getNumBlobs()`, así en cada frame se actualiza su valor al número de blobs detectados.

En el bloque `draw()`, se vincula la variable `cam` con el método `:draw()` de la clase, cuyos parámetros señalan las coordenadas x,y, y la anchura y altura de la imagen, en este caso, pantalla completa. Si no queremos que la ventana de salida muestre la imagen de la cámara, no se pone este método. Por ejemplo, si comentamos la siguiente línea de código del bloque `draw()`: `// cam:draw(OUTPUT_W/2 - scaleW/2,0,scaleW,scaleH)`, para desactivar su acción, se seguirán viendo los rectángulos superpuestos a los blob sin visualizar la imagen de la cámara.

Después, la función `ofScale()` tiene como parámetros el incremento en los ejes X, Y y Z, los valores utilizados son pantalla completa para X e Y, y 1 para Z porque no se está trabajando en 3D. Después una estructura `for` asigna valores a unas variables vinculadas con las funciones `getBlobX()`, `getBlobY()`, `getBlobW()` y `getBlobH()` tantas veces en cada frame como número de blobs se hayan detectado, menos 1, porque el primero tiene un id 0, y asigna los valores obtenidos por esas funciones a los

parámetros de tantos rectángulos como blobs detectados, los parámetros son: coordenadas x, y, anchura y altura de cada blob.

En el siguiente ejemplo, sin que se muestre la imagen de la cámara, vamos a analizar el uso del método `getBlobContourSize()` relacionado con las funciones de detección de los puntos del contorno, y el método `getBlobGeometrySize()` relacionada con las de detección de las líneas de geometría, parecido tiene que estar activado el algoritmo `Compute Contour geometry`.



```

/*
GAmuza 043 ejemplos      E-10-3
-----
ComputerVision - contorno y geometría
creado por n3m3da | www.d3cod3.org
*/
camPanel = gaCameraTracking() // asocia variable global a la clase de GAmuza
drawGUI = true // booleano para activar/desactivar el panel
camID = 0 // id de la cámara
runningBlobs = 0 // nº de blobs activos en cada frame
cam = ofTexture() // textura para mostrar imagen de la cámara
captureWidth = 320 // anchura y altura frame captura video
captureHeight = 240

function setup() // para guardar el archivo de configuración XML
    camPanel:setGuiSettingsFile(gaDataPath("camTrackingSettings.xml"))
    camPanel:setup(camID,captureWidth,captureHeight) // inicializa el panel
    cam:allocate(captureWidth,captureHeight, GL_RGB) // inicializa textura
end

```

```

function update()
    cam = camPanel:getCameraTextureMod() // la textura actualiza los frames de video
    runningBlobs = camPanel:getNumBlobs() //devuelve el nº de blobs detectados
    camPanel:update() //actualiza los datos del panel
end

function draw()
    gaBackground(0.0,1.0)

    scaleH = OUTPUT_H // para escalar imagen video a pantalla completa
    scaleW = scaleH* captureWidth / captureHeight
    ofPushMatrix()
    ofTranslate(OUTPUT_W/2 - scaleW/2,0,0)
    ofScale(scaleW/captureWidth,scaleH/captureHeight,1.0)

    for j=0, runningBlobs-1 do // recorre todos los blobs
        ofSetLineWidth(1)
        ofSetColor(255)
        ofFill()
        ofBeginShape() // inicia dibujo contorno de cada blob detectado
        // recorre todos los puntos del contorno
        for i=0, camPanel:getBlobContourSize(j)-1 do
            x = camPanel:getBlobCPointX(j,i) // coordenadas X de cada punto
            y = camPanel:getBlobCPointY(j,i) // coordenadas Y
            ofVertex(x,y) // dibuja el contorno
        end
        ofEndShape(false) // finaliza el dibujo sin cerrar las formas

        ofSetLineWidth(3) // atributos del dibujo de la geometría
        ofSetColor(9,245,160)
        ofNoFill()
        // recorre todos los puntos de la geometría
        for z=0, camPanel:getBlobGeometrySize(j)-1 do
            x1 = camPanel:getBlobGLineX1(j,z) // coordenadas X 1er punto de la línea
            y1 = camPanel:getBlobGLineY1(j,z) // coordenadas Y 1er punto
            x2 = camPanel:getBlobGLineX2(j,z) // coordenadas X 2º punto
            y2 = camPanel:getBlobGLineY2(j,z) // coordenadas Y 2º punto
            ofLine(x1,y1,x2,y2) //dibuja las líneas geométricas de cada blob
            ofCircle(x1,y1,3) //dibuja círculos en cada extremo de las líneas
            ofCircle(x2,y2,3)
        end
    end
    ofPopMatrix() // cierra la opción escalar

```

```

ofSetColor(255)
if drawGUI then
    camPanel:draw() // muestra el panel si drawGUI está activo
end

function keyReleased()
    if gaKey() == string.byte('g') then
        drawGUI = not drawGUI // con la tecla g se muestra/oculta el panel
    end
end

function mouseDragged() // para ajustar las opciones del panel con el ratón
    camPanel:mouseDragged(gaMouseX(),gaMouseY())
end

function mousePressed()
    camPanel:mousePressed(gaMouseX(),gaMouseY())
end

function mouseReleased()
    camPanel:mouseReleased(gaMouseX(),gaMouseY())
end

```

En este ejemplo, vemos que en el bloque del `update()` el valor de la variable global `runningBlobs` está de nuevo vinculado al que devuelve el método `getNumBlobs()`, para saber el número de blobs que se detectan en cada frame.

En el bloque `draw()`, las funciones `ofPushMatrix()` y `ofPopMatrix()` establecen el inicio y fin del código afectado por `ofScale()` que, como en el ejemplo anterior, escala los valores obtenidos de la cámara a pantalla completa. Después, para dibujar el contorno de los blobs establece un doble `for`, el primero se repite en cada frame tantas veces como blobs se hayan detectado, obteniendo el valor de la variable `runningBlobs-1`, mientras que el segundo `for` se repite tantas veces como número de puntos tenga el contorno de cada blob, obtenido por el valor de la función `getBlobContourSize(j)`, cuyo parámetro es el id del blob, que se lo asigna el primer `for`.

Para realizar el contorno utiliza el código de dibujo de formas irregulares por vértices `ofVertex(x,y)` que precisa tener antes y después, las funciones `ofBeginShape()` y `ofEndShape(false)` respectivamente. El valor de las coordenadas de los distintos vértices lo obtiene de las funciones `getBlobCPointX(j,i)` y `getBlobCPointY(j,i)` cuyos parámetros son el id del blob, que se obtiene del primer `for` y el id del punto que se obtiene del segundo `for`.

El código siguiente del bloque `draw()` inicia el dibujo de la geometría de cada blob, con un tercer `for` que se repetirá tantas veces como líneas tenga esa geometría, ese valor se obtiene de la función `getBlobGeometrySize(j)` cuyo parámetro, id del blob, viene dado por el primer `for` (`runningBlobs-1`). Los parámetros de las líneas se obtienen de las cuatro funciones que nos dan las coordenadas de los puntos de inicio y fin de cada una de ellas, `getBlobGLineX1(j,z)`, `getBlobGLineY1(j,z)`, `getBlobGLineX2(j,z)`, `getBlobGLineY2(j,z)`, cuyo segundo parámetro es el id de la línea que es asignado por este tercer `for`, con estos datos se dibujan también pequeños círculos en los extremos de las líneas para visualizar mejor las articulaciones y aproximarla a la representación que tiene GAMUZA en el panel Computer vision.

El siguiente ejemplo vinculado con la detección blobs utiliza Color tracking, y está planteado para la detección de un sólo blob, una forma de color diferenciado que el espectador mueve delante de la cámara para modificar el volumen y velocidad de reproducción (pitch) de un archivo de audio. Para ajustar el reconocimiento del color, en el bloque `color tracking settings` del panel computer vision, el color seleccionado es `hue: 0.28`, `Saturation: 0.75` y `Value: 0.45`, y el slider `Subtraction Threshold` se ha desplazado a su máximo valor para anular la aparición de otros blobs detectados por BACKGROUND SUBTRACTION. En el bloque de algoritmos, debe mantenerse seleccionado Compute Contour Finder para la detección del blob.

```
/*
GAMUZA 043      E-10-4
-----
ComputerVision - Color tracking
creado por mj
*/
//Variables tracking
camPanel = gaCameraTracking()
drawGUI = true
captureWidth = 320
captureHeight = 240
cam = ofTexture()
// Variables sonido
mySound = ofSoundPlayer()
posX = OUTPUT_W/2
posY = OUTPUT_H/2
posxVol= 0.0
posySpeed= 0.0

function setup() // inicializa panel de tracking
    camPanel:setGuiSettingsFile(gaDataPath("camTrackingSettings.xml"))
    camPanel:setup(0,captureWidth,captureHeight)
    cam:allocate(captureWidth,captureHeight,GL_RGB)
```

```
//sonido
mySound:loadSound(gaImportFile("hypno00.wav"),true)
mySound:setLoop(true)
mySound:play()
end

function update()
    cam = camPanel:getCameraTextureMod() // la textura actualiza los frames de video
    camPanel:update() // actualiza los datos del panel
    ofSoundUpdate() // actualiza sonido
end

function draw()
    gaBackground(0.0,1.0)
    posX = OUTPUT_W * camPanel:getBlobX(0) //otro modo de escalar a pantalla completa
    posY = OUTPUT_H * camPanel:getBlobY(0) // solo se busca posición del blob 0
    ofSetColor(0,255,0) // color verde
    ofCircle(posX, posY, 50) // dibuja círculo en la posición del blob
    posxVol= ofMap(posX, 0, OUTPUT_W, 0.1, 1.0, true) // mapea el valor de posición
    posySpeed= ofMap(posY, 0, OUTPUT_H, 0.1, 2.0, true) // al del volumen y speed
    mySound:setVolume(posxVol) // volumen según posición X del blob
    mySound:setSpeed(posySpeed) // velocidad según posición Y del blob
    ofSetColor(255)
    if drawGUI then
        camPanel:draw() // muestra panel si drawGUI está activo
    end
end

function keyReleased()
    if gaKey() == string.byte('g') then
        drawGUI = not drawGUI // con tecla g se muestra/oculta el panel
    end
end

function mouseDragged() // para ajustar las opciones del panel con el ratón
    camPanel:mouseDragged(gaMouseX(),gaMouseY())
end

function mousePressed()
    camPanel:mousePressed(gaMouseX(),gaMouseY())
end

function mouseReleased()
    camPanel:mouseReleased(gaMouseX(),gaMouseY())
end
```

Los bloques `setup()` y `update()` contienen código destinado a la reproducción del sonido, e inicialización y actualización del panel tracking video, pero nada para la detección de blobs porque no se requiere saber el número de blobs, ni actualizar ese valor, dado que solo se debe reconocer uno.

En el bloque `draw()`, sólo hay que escalar dos coordenadas, se simplifica multiplicando el valor de la coordenada que devuelven las funciones `getBlobX(0)` y `getBlobY(0)` por el ancho y alto de la ventana de salida, el parámetro de estas funciones es el id del único blob buscado, el `0`. El volumen del archivo de audio se regula mapeando los valores de la posición X entre `0.1` y `1.0`; de forma similar, la velocidad de reproducción se mapea respecto a la posición Y, entre `0.1` y `2.0`. Y en función de los movimientos que la forma de ese color detectado realice ante la cámara de vídeo, el sonido se va modulando al acelerar o ralentizar la velocidad de reproducción y variar la intensidad de volumen.

10.1.3. Motion

Otro tipo de datos que obtiene la clase `gaCameraTracking()` de GAmuza es la cantidad de movimiento y localización media de la masa de píxeles que reflejan ese movimiento. Para este tipo de datos no es necesario activar ningún algoritmo en el panel `computer vision`.

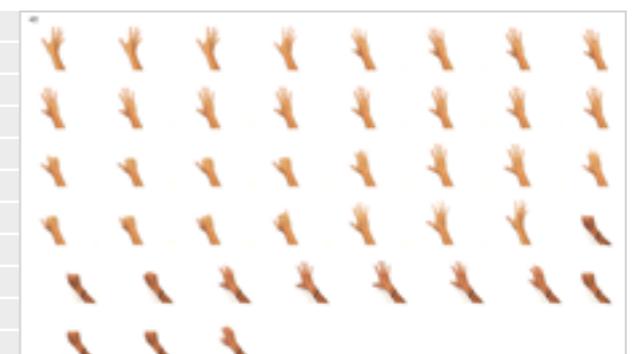
Los métodos vinculados a Motion son: `getMotionQ()`, `getMotionX()` y `getMotionY()`, y la sensibilidad del sistema para los datos que devuelven se regula en la interface del panel con los sliders del grupo **MOTION DETECTION SETTINGS**.

El método `getMotionQ()`, devuelve la cantidad de movimiento que se produce ante la cámara. Su valor refleja la cantidad de píxeles que se han modificado en cada frame respecto a la imagen de fondo que tiene almacenada, realmente ese valor oscila entre `0` y el ancho por alto de la imagen capturada, si bien GAmuza lo escala entre `0.0` y `1.0`. Los métodos `getMotionX()` y `getMotionY()`, devuelven el valor de las coordenadas x e y, respectivamente, del punto medio de la masa de movimiento, el desplazamiento de la representación de ese punto puede indicar hacia donde se traslada mayoritariamente el conjunto de píxeles que se han movido en el frame de la imagen video, respecto al modelo de fondo que ha guardado el sistema.

El siguiente ejemplo muestra la utilización del método `gaMotionQ()` en un proyecto básico de imágenes interactivas. Según la cantidad de movimiento registrada por la cámara de video, el sistema selecciona y reproduce un número mayor o menor de archivos de imagen que se despliegan por la pantalla siguiendo una retícula cuadrangular.

Para trabajar con la base de datos de imágenes se utiliza la clase de openFrameworks `ofDirectory()` que gestiona el directorio de los archivos guardados en una carpeta llamada `imag`, situada dentro de la carpeta `data` del script.

```
/*
GAmuza 043           E-10-5
-----
Interactividad motion tracking
creado por mj
*/
// variables panel tracking video
camPanel = gaCameraTracking()
drawGUI = true
captureWidth = 320
```



```

captureHeight = 240
    // variables imágenes y movimiento
dir= ofDirectory()
numImagenes = 0
imagesCount = 0
imagenBase = ofImage()
imagenes = {}          // tabla archivos de imagen
motion = {}            // tabla valores de movimiento
motionScaler = 0.133132 //ajustar según lecturas de movimiento en cada caso

function setup()           //inicializa panel tracking
    camPanel:setGuiSettingsFile(gaDataPath("camTrackingSettings.xml"))
    camPanel:setup(0,captureWidth,captureHeight)

    motion[0]={}          //inicializa tabla movimiento
    motion[0].q = 0         // valores cantidad de movimiento
    motion[0].factor = 0    // factor a aplicar para el número de imágenes
    imagenBase:loadImage(gaImportFile("mano0.png")) // carga imagen base
    dir:listDir(gaImportFile("imag/"))             // carga directorio de imágenes
    numImagenes = dir:size()                      // calcula nº archivos

    if dir:size()> 0 then
        for i=0,numImagenes-1 do      // inicializa la tabla de archivos de imagen
            imagenes[i] = ofImage()   // asigna la clase a cada objeto de la tabla
            imagenes[i]:setPath(dir.getPath(i)) //carga todas las imágenes
        end
    end
end

function update()
    camPanel:update()           // actuaiza los datos del panel
    motion[0].q = camPanel:getMotionQ() // detecta cantidad de movimiento
    motion[0].factor= math.floor(ofMap(motion[0].q,0.0,motionScaler,0,numImagenes,true))
        // escala la cantidad de movimiento al número de imágenes
    //gaLog(string.format("motion:%f",motion[0].q))
    //descomentar gaLog para ver en la consola el valor de movimiento máximo = motionScaler
end

function draw()
    gaBackground(1.0,1.0)
    ofSetColor(255)
    ancho = imagenBasegetWidth()/4 // calcula el ancho de la imagen base
    alto = imagenBasegetHeight()/4 // calcula el alto de la imagen base
    cols = math.floor(OUTPUT_W/ancho) // número de columnas de imágenes
    c=0                            // para hacer una retícula 2D con un solo for

```

```

r=0
for i=0, numImagenes-1 do
    if i < motion[0].factor then
        imagenes[i]:draw(c*ancho,r*alto,ancho,alto)
        c += 1
    end
    if c == cols then           // si las imágenes llegan al ancho de pantalla
        c= 0                     // se ubican en una fila inferior
        r +=1
    end
end

ofSetColor(0)
    //dibuja en pantalla valor actual de numImagenes
ofDrawBitmapString(tostring(motion[0].factor),20,20)

ofSetColor(255)
    if drawGUI then
        camPanel:draw()           // muestra el panel si drawGUI está activo
    end
end

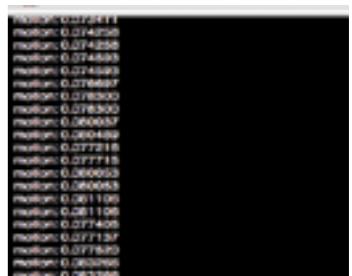
function keyReleased()
    if gaKey() == string.byte('g') then
        drawGUI = not drawGUI     // con la tecla g se muestra/oculta el panel
    end
end

function mouseDragged()           // para ajustar las opciones del panel con el ratón
    camPanel:mouseDragged(gaMouseX(),gaMouseY())
end
function mousePressed()
    camPanel:mousePressed(gaMouseX(),gaMouseY())
end

function mouseReleased()
    camPanel:mouseReleased(gaMouseX(),gaMouseY())
end

```

El ejemplo plantea que cuanto mayor sea el movimiento que se registra delante de la cámara, mayor será el número de imágenes



reproducidas. Sabemos que los valores del método `getMotion()` oscilan entre `0.0` y `1.0`, pero ese valor máximo depende de las condiciones de presentación del proyecto por lo que para ajustarlo se utiliza el siguiente código en el bloque `update()` para monitorizar los valores:

```
gaLog(string.format("motion: %f", motion[0].q))
```

La consola del IDE los muestra, y el mayor de esos valores sirve para delimitar cuál va a ser el de la variable `motionScaler`.

Para gestionar los archivos de imagen, se vincula la variable global `dir` a la clase de openFrameworks `ofDirectory()`, se declara una tabla de imágenes y otra para los datos de la cantidad de movimiento.

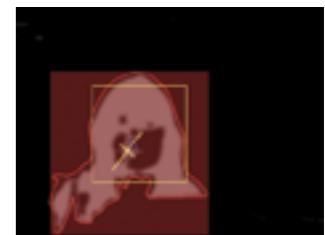
En el bloque `setup()`, después de inicializar el panel de tracking, se crea una nueva tabla en `motion[0]` con dos elementos: `motion[0].q` albergará los datos de lectura del movimiento y `motion[0].factor` se utilizará para mapear el número de imágenes en función del valor de movimiento. Dado que tablas solo guardarán un valor que va cambiando, su `index` es `0` e inicialmente se les asigna un valor `0` que se actualizará en el bloque `update()`. La siguiente línea de código con la variable `imagenBase` es para cargar la imagen que se toma como referencia. Después, viene el código dirigido a la gestión de los archivos de imagen, aplicándole a la variable `dir` el método `listDir()` cuyo parámetro es la función de GAmuza `gaImportFile()` que indica la carpeta en la que están almacenados. A la variable global `numImagenes` se le asigna el valor que devuelve el método `size()` de la clase `ofDirectory()`, así se obtiene el número de imágenes guardadas en el directorio antes indicado. Y mediante un `for` que se repetirá tantas veces como imágenes hayan, se le asignan a todas la clase `ofImage()`, y se cargan utilizando los métodos `loadImage` de la clase `ofImage()` y `getPath` de la clase `ofDirectory()`.

En el bloque `update()`, se le asigna a la tabla `motion[0].q` el valor que devuelve el método `getMotion()` y para el valor de `motion[0].factor` se establece el siguiente cálculo: la función `math.floor()` devuelve un número entero redondeando hacia abajo, si el resultado de los datos que tiene entre paréntesis fuera `0.5`, devuelve `0`, y con la función `ofMap()`, los valores de la cantidad de movimiento que vayan entre `0.0` y lo que se le asignó a la variable `motionScaler` se mapean entre `0` y el número de imágenes almacenadas, devolviendo así este factor el número de imágenes que van a mostrarse en función de la cantidad de movimiento.

En el bloque `draw()`, un `for` se repite tantas veces como el valor que `motion[0].factor` ha asignado a `numImagenes` en el `update()`. La línea de código `ofDrawBitmapString(tostring(motion[0].factor), 20, 20)`, muestra en el borde superior izquierdo de la ventana de salida cuál es ese valor.

10.1.4. Haar

La clase `gaCameraTracking()` tiene una serie de métodos para el reconocimiento de partes del cuerpo humano que vienen preconfigurados por archivos Haar cascade de OpenCV. Por defecto, tiene cargado el archivo "haarcascade_frontalface_alt.xml". Si se quiere reconocer otra parte del cuerpo o utilizar otra configuración para la detección de rostros frontales hay que cargar el archivo con el método `setHaarFile(gaImportFile("ArchivoHaar.xml"))` y guardar previamente ese archivo Haar en la carpeta data.



Debe estar activo el algoritmo `compute haar finder` en el panel. Otros métodos vinculados a este tipo de tracking son: `getNumHaars()`, `getHaarX()`, `getHaarY()`, `getHaarW()`, `getHaarH()`,

`getNumHaars()`, devuelve el número de Haar detectados en cada frame.

`getHaarX(haarID)` y `getHaarY(haarID)`, devuelven las coordenadas X e Y, respectivamente, del haar especificado en su parámetro.

`getHaarW(haarID)` y `getHaarH(haarID)`, devuelven la anchura y altura, respectivamente, del haar especificado en su parámetro.

Ejemplo:

```
/*
GAmuza 043           E-10-6
-----
ComputerVision - Haar
creado por n3m3da | www.d3cod3.org
*/
// variables panel tracking video
camPanel = gaCameraTracking()
drawGUI = true
captureWidth = 320
captureHeight = 240
runningHaars = 0

function setup()
  //inicializa panel tracking
  camPanel:setGuiSettingsFile(gaDataPath("camTrackingSettings.xml"))
  camPanel:setup(0,captureWidth,captureHeight)
end
```

```

function update()
    camPanel:update()
    runningHaars = camPanel:getNumHaars() // n° de haar activos
end

function draw()
    gaBackground(0.0,1.0)
    ofSetColor(255,0,0)
    ofFill()
    for i=0, runningHaars do
        posX = OUTPUT_W * camPanel:getHaarX(i) // escala X de haars a ancho pantalla
        posY = OUTPUT_H * camPanel:getHaarY(i) // escala Y
        ancho = camPanel:getHaarW(i) * OUTPUT_W //escala ancho haars a ancho pantalla
        alto = camPanel:getHaarH(i) * OUTPUT_H // escala alto
        ofRect(posX, posY, ancho, alto) // dibuja rectángulos en cada haar
    end

    ofSetColor(255)
    if drawGUI then
        camPanel:draw() // muestra el panel si drawGUI está activo
    end
end

function keyReleased()
    if gaKey() == string.byte('g') then
        drawGUI = not drawGUI // con la tecla g se muestra/oculta el panel
    end
end

function mouseDragged() // para ajustar las opciones del panel con el ratón
    camPanel:mouseDragged(gaMouseX(),gaMouseY())
end

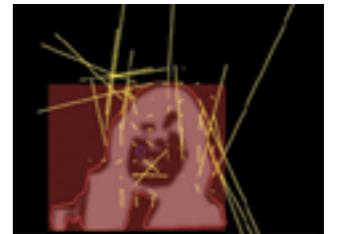
function mousePressed()
    camPanel:mousePressed(gaMouseX(),gaMouseY())
end

function mouseReleased()
    camPanel:mouseReleased(gaMouseX(),gaMouseY())
end

```

10.1.5. Optical Flow

GAmuza establece una rejilla a partir de la captura de la cámara de video, vinculada a las constantes `OPTICAL_FLOW_GRID_X`, número de columnas de la rejilla y `OPTICAL_FLOW_GRID_Y`, número de filas de la rejilla, para representar vectorialmente la dirección del movimiento que se produce en cada una de las posiciones que marca esa rejilla. Los métodos vinculados a este tipo de tracking son:



`getOpticalFlowX(index)` y `getOpticalFlowY(device,index)`, devuelven las coordenadas x e y, respectivamente, de una determinada posición en la rejilla. Su parámetro es el index de la posición en la rejilla.

`getOpticalFlowVX(index)` y `getOpticalFlowVY(device)`, devuelven la velocidad de movimiento de las coordenadas x e y, respectivamente, de una determinada posición en la rejilla del optical flow. Su parámetro es el index de la posición en la rejilla.

En el panel Computer Vision debe estar activo el algoritmo Compute Optical Flow

```

/*
GAmuza 043           E-10-7
-----
ComputerVision - Optical Flow
creado por n3m3da | www.d3cod3.org
*/

// variables panel tracking video
camPanel = gaCameraTracking()
camID = 0
cam = ofTexture()
drawGUI = true

captureWidth = 320
captureHeight = 240
opticalFlowCols = math.ceil(captureWidth/OPTICAL_FLOW_COLS_STEP)
opticalFlowRows = math.ceil(captureHeight/OPTICAL_FLOW_ROWS_STEP)

function setup() //inicializa panel tracking
    camPanel:setGuiSettingsFile(gaDataPath("camTrackingSettings.xml"))
    camPanel:setup(camID,captureWidth,captureHeight)

    cam:allocate(captureWidth,captureHeight,GL_RGB)
end

```

```

function update()
    cam = camPanel:getCameraTextureMod()
    camPanel:update()
end

function draw()
    gaBackground(0.0,1.0)
    ofSetColor(255)
    scaleH = OUTPUT_H // dibuja imagen video a pantalla completa
    scaleW = scaleH* captureWidth / captureHeight
    cam:draw(OUTPUT_W/2 - scaleW/2,0,scaleW,scaleH)
    ofSetColor(31,165,210) // dibuja representación de optical flow
    ofSetLineWidth(3)
    ofPushMatrix()
    ofTranslate(OUTPUT_W/2 - scaleW/2, 0, 0)
    ofScale(scaleW/captureWidth,scaleH/captureHeight,1)
    for i=0, opticalFlowCols*opticalFlowRows-1 do
        x = camPanel:getOpticalFlowX(i)
        y = camPanel:getOpticalFlowY(i)
        vX = camPanel:getOpticalFlowVX(i)
        vY = camPanel:getOpticalFlowVY(i)
        ofLine(x, y, x + vX, y + vY)
    end
    ofPopMatrix()

    ofSetColor(255)
    if drawGUI then
        camPanel:draw() // muestra el panel si drawGUI está activo
    end
end

function keyReleased()
    if gaKey() == string.byte('g') then
        drawGUI = not drawGUI // con la tecla g se muestra/oculta el panel
    end
end

function mouseDragged() //para ajustar opciones del panel con el ratón
    camPanel:mouseDragged(gaMouseX(),gaMouseY())
end
function mousePressed()
    camPanel:mousePressed(gaMouseX(),gaMouseY())
end
function mouseReleased()
    camPanel:mouseReleased(gaMouseX(),gaMouseY())
end

```

10.1.6. Trigger Areas

Desde el panel de Camera Tracking, en la imagen que captura la cámara se pueden activar nueve áreas circulares, ajustables en tamaño y posición para que las acciones programadas para el tracking sólo se apliquen si el punto central del blob entra en el área especificada. Clicando en el centro de un área y arrastrando el ratón se puede desplazar su posición. Clicando en cualquier otra zona del área y arrastrando el ratón, se puede ampliar o reducir su tamaño.



```

/*
GAmuza 043           E-10-8
-----
ComputerVision - triggerAreas
uso opción trigger area con panel Camera Tracking.
creado por n3m3da | www.d3cod3.org
*/

camPanel = gaCameraTracking()
camID = 0
cam = ofTexture()
drawGUI = true
captureWidth = 320
captureHeight = 240

function setup()
    gaWave(GA_BROWN,1) // inicializa un oscilador con ruido marrón
    // inicializa panel tracking
    camPanel:setGuiSettingsFile(gaDataPath("camTrackingSettings.xml"))
    camPanel:setup(camID,captureWidth,captureHeight)
    cam:allocate(captureWidth,captureHeight,GL_RGB)
end

function update()
    cam = camPanel:getCameraTextureMod()
    camPanel:update()
    // activa el volumen del ruido marrón si el trigger area 0 está activada
    if camPanel:getTrigger(0) then
        gaWaveVolume(0,0.8)
    else
        gaWaveVolume(0,0.0)
    end
end

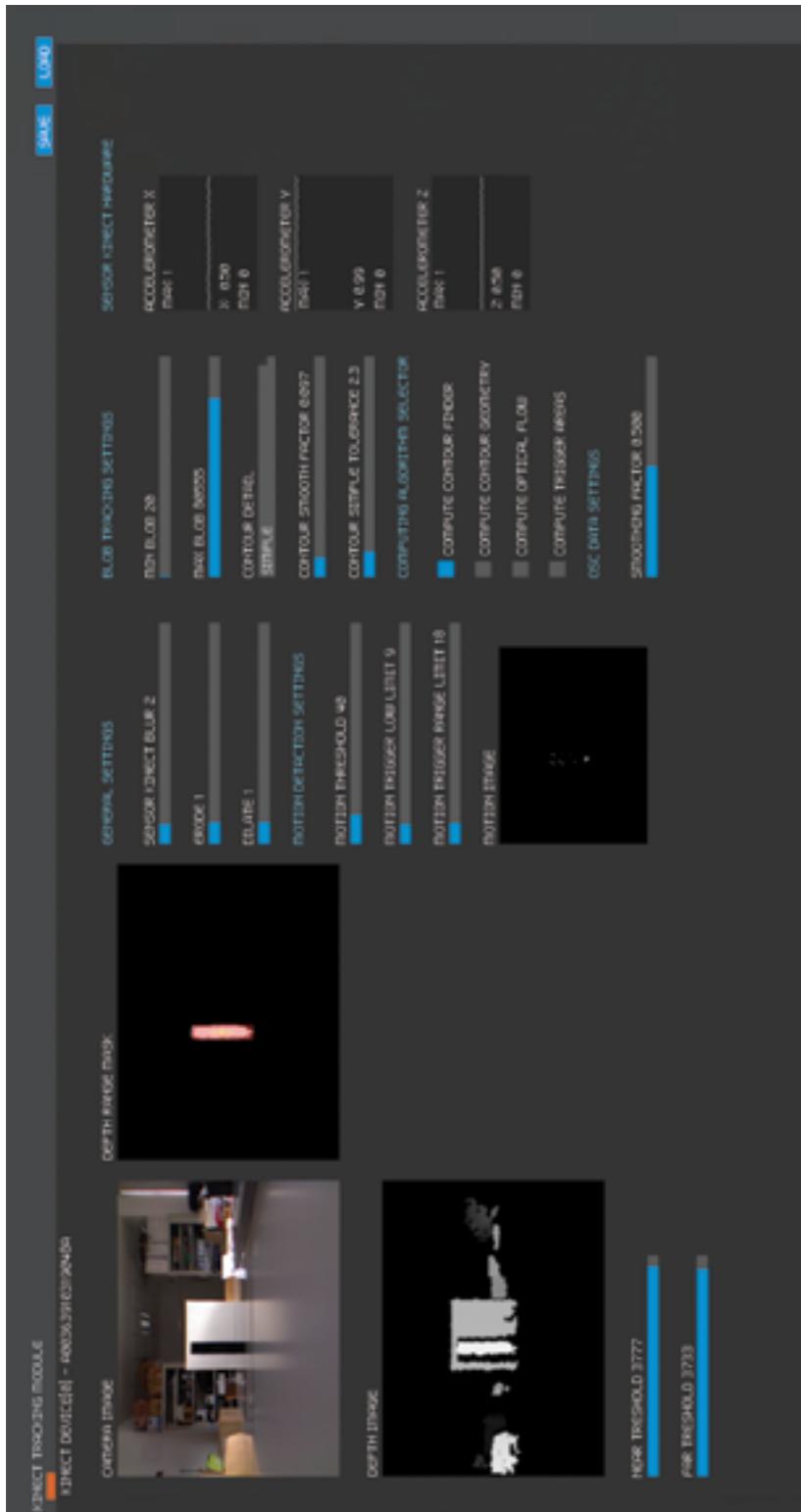
```

```
function draw()
    gaBackground(0.0,0.8)
        // dibuja la imagen capturada por la cámara
    ofSetColor(255)
    scaleH = OUTPUT_H
    scaleW = scaleH* captureWidth / captureHeight
    cam:draw(OUTPUT_W/2 - scaleW/2,0,scaleW,scaleH)

    ofSetColor(255)
    if drawGUI then
        camPanel:draw() // muestra el panel si drawGUI está activo
    end
end

function keyReleased()
    if gaKey() == string.byte('g') then
        drawGUI = not drawGUI // con la tecla g se muestra/oculta el panel
    end
end

function mouseDragged() // para ajustar las opciones del panel con el ratón
    camPanel:mouseDragged(gaMouseX(),gaMouseY())
end
function mousePressed()
    camPanel:mousePressed(gaMouseX(),gaMouseY())
end
function mouseReleased()
    camPanel:mouseReleased(gaMouseX(),gaMouseY())
end
```



Panel Tracking por sensor Kinect

10.2. Panel Tracking: por sensor kinect

A continuación se describe la interfaz del panel de Tracking para el sensor Kinect y los métodos de la clase de GAmuza `gaKinectTracking()`, deteniéndonos especialmente en las diferencias que hay respecto al tracking por cámara de video. Al igual que en el capítulo anterior, se explica la interfaz a partir del código que genera su carga.

```
/*
GAmuza 0428 ejemplos      E-10-9
-----
Panel Tracking kinect
creado por n3m3da | www.d3cod3.org
*/

kinectPanel = gaKinectTracking()
drawGUI = true
kinectDevID = 0
kinectImage = ofTexture()
function setup()
    kinectPanel:setGuiSettingsFile(goDataPath("kinectTrackingSettings.xml"))
    // parámetros setup: id,IR,videoImage,LED MODE
    kinectPanel:setup(kinectDevID,true,false,1)
end

function update()
    kinectImage = kinectPanel:getCameraTexture()
    kinectPanel:update()
end

function draw()
    gaBackground(0.1,1.0)
    // muestra la imagen desde el dispositivo escalada a pantalla completa
    ofSetColor(255)
    scaleH = OUTPUT_H
    scaleW = scaleH* 640 / 480
    kinectImage:draw(OUTPUT_W/2 - scaleW/2,0, scaleW,scaleH)

    ofSetColor(255)
    if drawGUI then
        kinectPanel:draw() // muestra el panel si drawGUI está activo
    end
end
```

```

function keyReleased()
    if gaKey() == string.byte('g') then
        drawGUI = not drawGUI // con la tecla g se muestra/oculta el panel
    end
end // para ajustar las opciones del panel con el ratón

function mouseDragged()
    kinectPanel:mouseDragged(gaMouseX(),gaMouseY())
end

function mousePressed()
    kinectPanel:mousePressed(gaMouseX(),gaMouseY())
end

function mouseReleased()
    kinectPanel:mouseReleased(gaMouseX(),gaMouseY())
end

function exit()
    kinectPanel:close() // siempre hay que cerrar el dispositivo
end

```

Al igual que para el tracking con cámara de video, se inicia asociando una variable a la clase de GAmuza que gestiona, en este caso, al sensor Kinect, `gaKinectTracking()`, se establece un switch booleano para mostrar u ocultar el panel en la ventana de salida; y la variable `kinectImage` se vincula a la clase de openFrameworks `ofTexture()` para acoger la imagen que viene de la Kinect.

La interfaz del panel del sensor Kinect también está regulada por un archivo .xml que se guarda en la carpeta data junto al archivo del script utilizando el método `setGuiSettingsFile()` cuando se salvan los ajustes del panel, y se inicializa con el método `setup(int,bool,bool,int)`, cuyos parámetros son: el ID del sensor, el uso o no de infrarrojos, el uso o no de color en la imagen y el último parámetro `int` es Led Mode cuyos valores posibles son: 0 -> LED_OFF, 1 -> LED_GREEN, 2 -> LED_RED, 3 -> LED_YELLOW, 4 -> LED_BLINK_GREEN y 6 -> LED_BLINK_YELLOW_RED.

En el bloque del `update()`, se asigna la imagen obtenida por el sensor Kinect a la textura: `kinectImage = kinectPanel:getCameraTexture();` y se actualiza la información del panel con el método `update`.

En el bloque del `draw()`, se puede dibujar la imagen capturada por el sensor y dibujar o no el panel en la ventana de salida. El resto de bloques de código funcionan igual que en el panel de tracking con cámara de video.

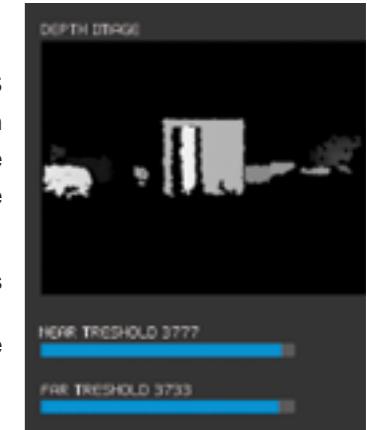
10.2.1. Descripción panel

Las diferencias fundamentales entre el sensor Kinect y una cámara de video en su utilización para tracking reside en la capacidad para detectar profundidad y el mecanismo de inclinación motorizado.

DEPTH IMAGE

El sensor de profundidad combina infrarrojos con un sensor CMOS monocromo lo que le permite a Kinect ver/calcular el espacio en 3D sin que las condiciones de luz ambiental influyan; el rango de detección de profundidad va 70 cm a 6 metros y en el panel puede ajustarse con los sliders:

- **NEAR TRESHOLD:** determina el plano de los elementos más próximos, como mínimo desde 70 cm del sensor
- **FAR TRESHOLD:** hasta qué plano de profundidad se quiere detectar, como máximo 6 m

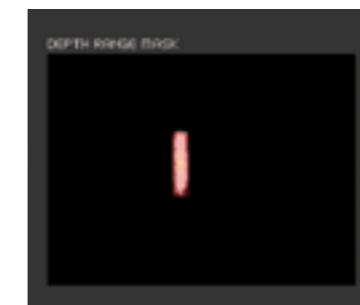


SENSOR KINECT HARDWARE

El mecanismo de inclinación es un acelerómetro para determinar la orientación de la Kinect. La interfaz del panel refleja estos datos en los 3 monitores de la columna derecha y en los scripts se pueden conocer con los métodos: `getAccelX()`, `getAccelY()` y `getAccelZ()` que devuelven un valor `float` entre 0.0 y 1.0

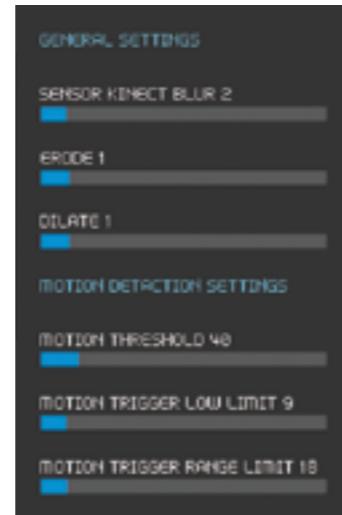
Los otros componentes de la interfaz actúan de forma similar a los panel de tracking para cámaras de video.

DEPTH RANGE MASK: monitor donde se muestra el resultado de los algoritmos aplicados.



GENERAL SETTINGS

- **SENSORKINECTBLUR:** grado de desenfoque de la señal de entrada
- **ERODE:** reducir el área de las superficies detectadas
- **DILATE:** ampliar el área de las superficies detectadas

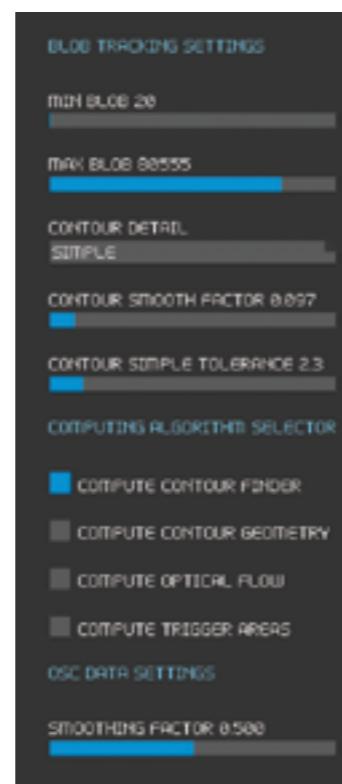


MOTION DETECTION SETTINGS. Regula la detección de movimiento según niveles de profundidad.

- **MOTION TRESHOLD:** regula la sensibilidad del sistema para activar la detección, si es poco sensible se debe ampliar el umbral (Threshold)
- **MOTION TRIGGER LOW LIMIT:** nivel más bajo para activar la detección de movimiento
- **MOTION TRIGGER RANGE LIMIT:** límite del rango para la detección de movimiento.

BLOB TRACKING SETTINGS

- **MIN BLOB:** tamaño mínimo de los blobs
- **MAX BLOB:** tamaño máximo de los blobs
- **CONTOUR DETAIL:** menú con tres opciones, raw, smooth o simple
- **CONTOUR SMOOTH FACTOR:** si se ha seleccionado smooth, ajusta el nivel de suavizado
- **CONTOUR SIMPLE TOLERANCE:** si se ha seleccionado simple, ajusta el rango de vértices que articulan el contorno.



COMPUTING ALGORITHM SELECTOR

- **COMPUTE CONTOUR FINDER:** para procesar el contorno de los blobs detectados.
- **COMPUTE CONTOUR GEOMETRY:** Determina formas geométricas a partir de los contornos. Al activarlo se activa automáticamente el anterior si no lo estaba.
- **COMPUTE OPTICAL FLOW:** Calcula el flujo óptico del movimiento y lo describe con vectores que indican la dirección e intensidad.
- **COMPUTE TRIGGER AREAS:** Activa la posibilidad de utilizar 8 zonas circulares, como en el módulo Computer Vision.

10.2.2. Métodos y funciones para el sensor Kinect

Además de los métodos mencionados hasta ahora la clase `gaKinectTracking()` tiene también los siguientes:

Clase	Método	Devuelve
	<code>gaKinectTracking:getDepthTexture()</code>	<code>ofTexture</code>
	<code>gaKinectTracking:getCameraPixels()</code>	<code>ofPixelsRef</code>
Blobs	<code>gaKinectTracking:getNumBlobs()</code>	<code>int num Blobs</code>
	<code>gaKinectTracking:getBlobX(int blobID)</code>	<code>float posX del blob</code>
	<code>gaKinectTracking:getBlobY(int blobID)</code>	<code>float posY del blob</code>
	<code>gaKinectTracking:getBlobW(int blobID)</code>	<code>float ancho del blob</code>
	<code>gaKinectTracking:getBlobH(int blobID)</code>	<code>float alto del blob</code>
	<code>gaKinectTracking:getBlobAngle(int blobID)</code>	<code>float ángulo del blob</code>
Contorno	<code>gaKinectTracking:getBlobContourSize(int blobID)</code>	<code>int num puntos contorno</code>
	<code>gaKinectTracking:getBlobCPointX(int blobID,int pointContID)</code>	<code>float posX del punto</code>
	<code>gaKinectTracking:getBlobCPointY(int blobID,int pointContID)</code>	<code>float posY del punto</code>
Geometría (líneas)	<code>gaKinectTracking:getBlobGeometrySize(int blobID)</code>	<code>int tamaño geometría</code>
	<code>gaKinectTracking:getBlobGLineX1(int blobID,int lineID)</code>	<code>float posX1 lineaGeom</code>
	<code>gaKinectTracking:getBlobGLineY1(int blobID,int lineID)</code>	<code>float posY1 lineaGeom</code>
	<code>gaKinectTracking:getBlobGLineX2(int blobID,int lineID)</code>	<code>float posX2 lineaGeom</code>
	<code>gaKinectTracking:getBlobGLineY2(int blobID,int lineID)</code>	<code>float posY2 lineaGeom</code>
Optical Flow	<code>gaKinectTracking:getOpticalFlowX(int indexGridPosition)</code>	<code>float posX grid</code>
	<code>gaKinectTracking:getOpticalFlowY(int indexGridPosition)</code>	<code>float posY grid</code>
	<code>gaKinectTracking:getOpticalFlowVX(int indexGridPosition)</code>	<code>float posVX grid</code>
	<code>gaKinectTracking:getOpticalFlowVY(int indexGridPosition)</code>	<code>float posVY grid</code>
Trigger Areas	<code>gaKinectTracking:getTrigger(int triggerID)</code>	<code>bool true or false</code>

En el siguiente ejemplo se muestra el uso de algunos de estos métodos para dibujar el contorno de la imagen reconocida a través de los blobs.

```
/*
GAMUZA 043           E-10-10
-----
Sensor Kinect - Kinect contorno

creado por n3m3da | www.d3cod3.org
*/
// Kinect Variables
kinectPanel = gaKinectTracking()
drawGUI = true
kinectDevID = 0
kinectImage = ofTexture()
runningBlobs = 0

function setup()
    // guarda archivo configuración XML e inicializa Kinect
    kinectPanel:setGuiSettingsFile(gaDataPath("kinectTrackingSettings.xml"))
    kinectPanel:setup(kinectDevID, false, true, 1)
end

function update()
    kinectImage = kinectPanel:getCameraTexture() // asigna imagen kinect a la textura
    kinectPanel:update() // actualiza datos del panel
    runningBlobs = kinectPanel:getNumBlobs() // número de blobs detectados
end

function draw()
    gaBackground(0.1,1.0)
    ofSetColor(255) // dibuja imagen capturada por kinect a pantalla completa
    scaleH = OUTPUT_H
    scaleW = ((scaleH* 640) / 480) - 240
    kinectImage:draw(OUTPUT_W/2 - scaleW/2,0, scaleW, scaleH)

    ofPushMatrix() //inicializa opción escalar
    ofTranslate(OUTPUT_W/2 - scaleH/2,0,0) // traslada y escala las posiciones
    ofScale(scaleW/640, scaleH/480,1.0)
    for j=0, runningBlobs-1 do // recorre los blobs detectados
        ofSetLineWidth(3)
        ofSetColor(255)
        ofNoFill()
        ofBeginShape() //inicializa el dibujo del contorno de cada blob
```

```
for i=0, kinectPanel:getBlobContourSize(j)-1 do // recorre los puntos contorno
    x = kinectPanel:getBlobCPointX(j,i) // posición X de cada punto
    y = kinectPanel:getBlobCPointY(j,i) // posición Y de cada punto
    ofVertex(x,y) // dibuja los contornos

end
ofEndShape(false) //finaliza sin cerrar las formas
end
ofPopMatrix() // finaliza opción escalar

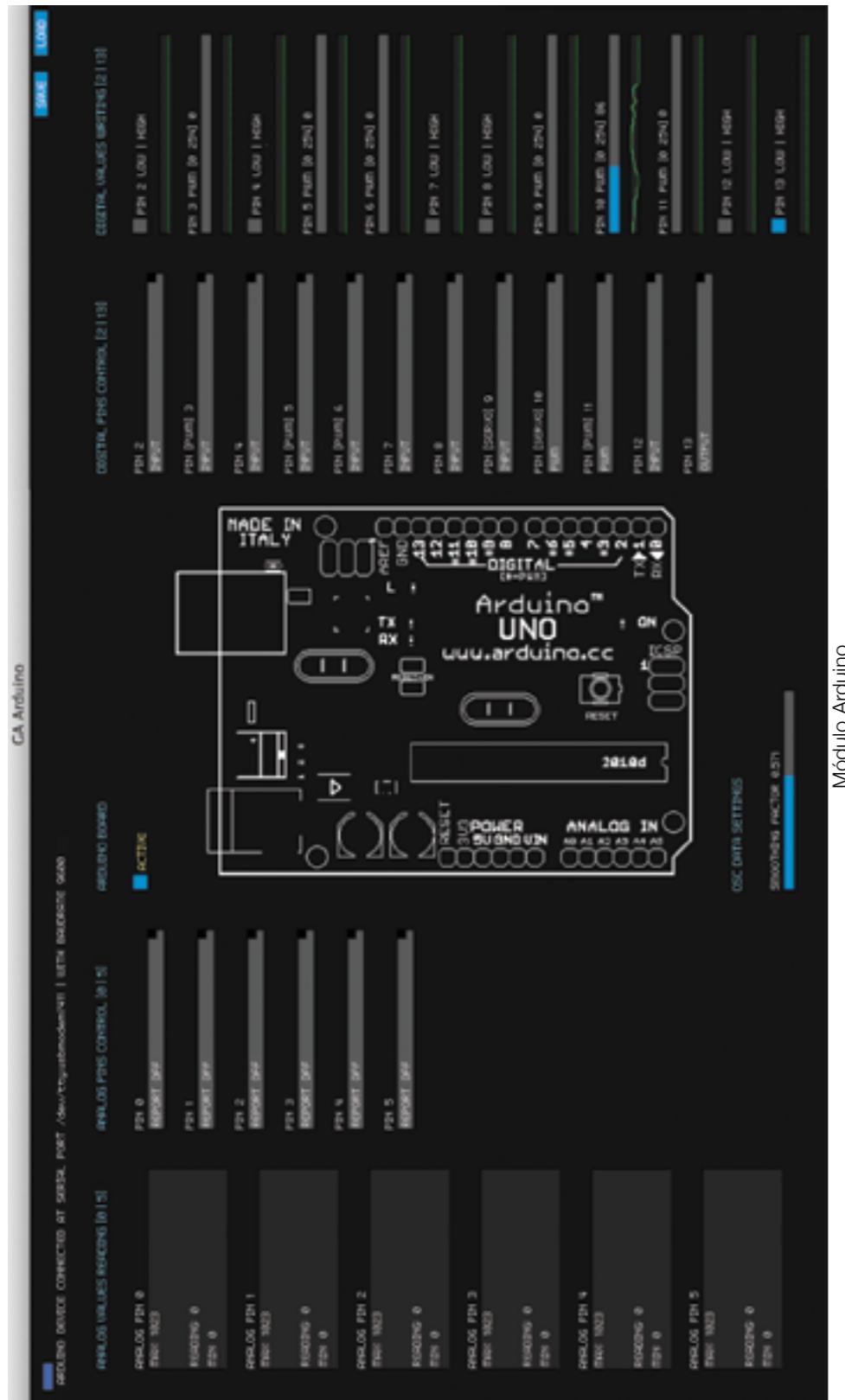
ofSetColor(255)
if drawGUI then
    kinectPanel:draw() // muestra el panel si drawGUI está activo
end

function keyReleased()
    if gaKey() == string.byte('g') then
        drawGUI = not drawGUI // con la tecla g se muestra/oculta el panel
    end
end
// para ajustar las opciones del panel con el ratón
function mouseDragged()
    kinectPanel:mouseDragged(gaMouseX(),gaMouseY())
end

function mousePressed()
    kinectPanel:mousePressed(gaMouseX(),gaMouseY())
end

function mouseReleased()
    kinectPanel:mouseReleased(gaMouseX(),gaMouseY())
end

function exit()
    kinectPanel:close() // siempre hay que cerrar el dispositivo
end
```



Módulo Arduino

11. Interface Módulo Arduino

El módulo Arduino se visualiza seleccionando en el menú superior Modules/Arduino, o tecleando (↑⌘5) y establece la comunicación entre las funciones de GAmuza y el micro-controlador Arduino⁸⁴. En el panel de Preferences, pestaña Arduino, hay que seleccionar el puerto y el baud rate. Las opciones de configuración son:

ANALOG VALUES RECORDING. Lectura de valores analógicos, del PIN 0 al 5. Cada monitor muestra el gráfico de la señal y su valor entre 0 y 1023. La función de GAmuza, `gaAnalogRead()` normaliza ese rango de valores de **0.0** a **1.0**.

En la selección **ANALOGPINCONTROL**, hay que seleccionar **ReportON** en los pin que se deseen leer y **Report OFF** en los no conectados.

Arduino tiene 14 pin digitales, de los cuales 6 son PWM (Pulse-width modulation) de 8 bits, pudiendo transmitir un rango de valores de 0 a 254. Si se selecciona la opción **Input** u **Output** funcionan como los demás, solo envían o reciben dos datos: HIGH o LOW.

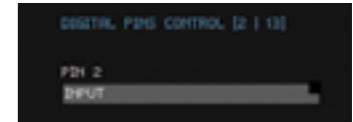
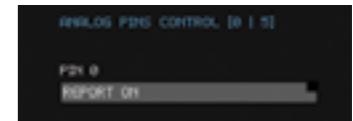
Digital values writing: los ajustes de los valores de escritura de los pin digitales y PWM, están vinculados a los datos programados con la función de GAmuza `gaDigitalWrite(pwmPin,valor)`, el pin al que haga referencia la función debe estar activado en la interfaz.

Las funciones de GAmuza para comunicarse con Arduino son:

gaAnalogRead(int) su parámetro es el pin analógico que se quiere leer y devuelve en tiempo real el valor **float** del pin especificado, normalizado en un rango de **0.0** a **1.0**.

gaDigitalRead(int) su parámetro es el pin digital que se quiere leer y devuelve el valor **int** del pin especificado como **0** o **1**.

gaDigitalWrite(int,int) escribe en el pin digital especificado, como primer parámetro, el valor indicado (**0** o **1**), o bien si el pin es PWM un valor dentro del rango 0-255, ese valor es el segundo parámetro. No olvidar activar previamente los pines en el módulo Arduino de GAmuza, disponible en el menú principal, Modules/Arduino; por ejemplo, si se está utilizando



el pin digital 10 como PWM, hay que seleccionar esa opción en el menú desplegable que hay junto a ese pin en el módulo Arduino.

gaSerialDevice() Es una referencia del dispositivo serial que esté conectado, puede ser un Arduino u otro dispositivo, como por ejemplo, el controlador DMX USB pro. Devuelve un **string** con el nombre del puerto serial del dispositivo.

gaServoWrite(int,int) sus parámetros son el número del pin servo de Arduino (en Arduino Uno el 9 o el 10) y el valor que se le quiere asignar, de ahí que la función escriba en el pin digital servo especificado el valor indicado. Para el pin servo usar un rango de valores entre 0 y 255. No olvidar activar previamente los pines de Arduino en el módulo Arduino de GAmuza, disponible en el menú principal, Modules/Arduino.

gaSetServoPin(int,bool) esta función activa/desactiva el pin de Arduino especificado como pin de control de un servomotor. Asegurarse de que se ha elegido un pin de Arduino preparado para trabajar como control de servo, por ejemplo, en el Arduino UNO son los pin digital 9 y 10.

El siguiente ejemplo muestra el uso de estas funciones.

```
/*
GAmuza 0435          E-11-1
-----
Arduino - Servo Motor
Ejemplo para conectar y controlar un servo motor con arduino y usarlo
con GAmuza.
*/
// El pin del motor debe ser un pin servo digital
// En Arduino UNO son los pins 9 y 10
servoPin = 9
servoValue = 0

function setup()
    // activar pin 9 como pin servo
    gaSetServoPin(servoPin,true)
end

function update()
    gaServoWrite(servoPin,servoValue)
end
```

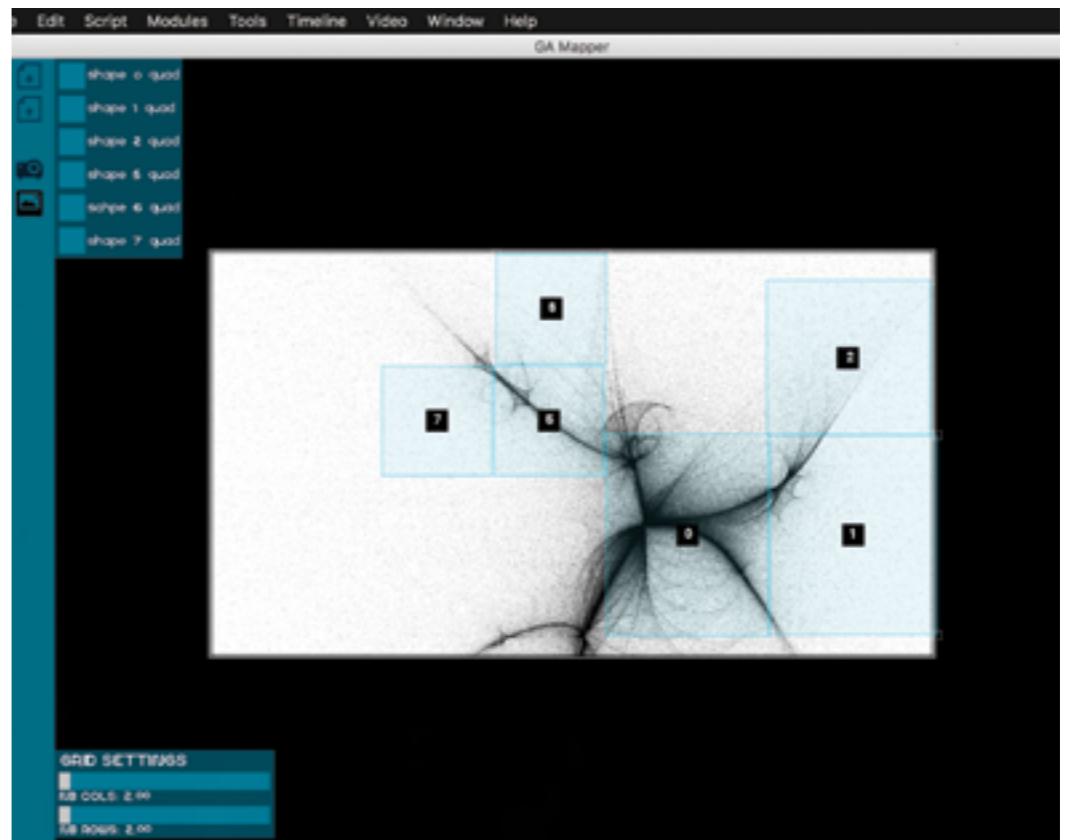
```
function draw()
    gaBackground(0.0,1.0)

    ofSetColor(255)
    ofCircle(servоВalue/255*OUTPUT_W,OUTPUT_H/2,50)
end

function keyPressed()
    if gaKey() == string.byte('r') then
        servoValue = math.floor(ofRandom(255))
    end
end
```

12. Módulo Mapper

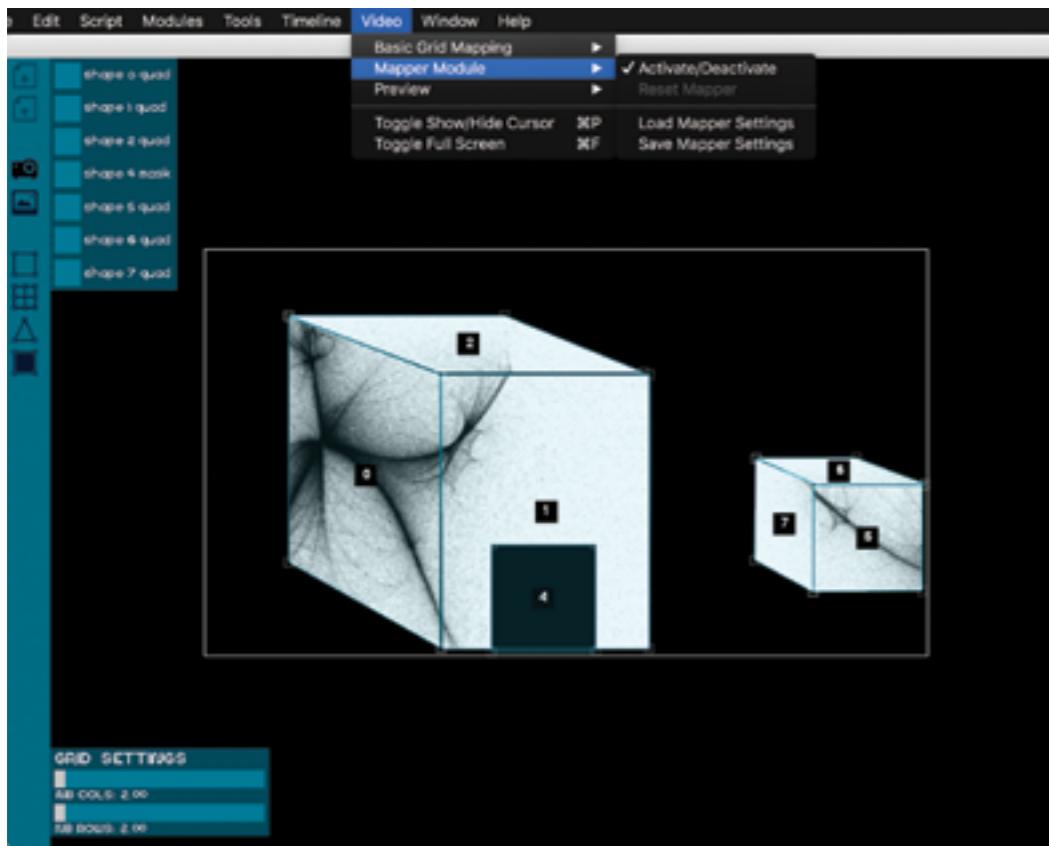
El módulo Mapper se visualiza seleccionando en el menú superior **Modules/Mapper** o tecleando (**↑⌘6**) y proporciona una interface sencilla de mapping con recursos más flexibles que la rejilla de mapping básica. Se articula en 2 interfaces, la primera relativa a la imagen para seleccionar los fragmentos a proyectar y la segunda, vinculada al proyector de video, permite el ajuste de visualización de esos fragmentos, pudiendo también incorporar máscaras. Este módulo está completamente diseñado como GUI y no tiene funciones de GAMUZA vinculadas a él.



Módulo Mapper, opción interface imagen



Los dos iconos laterales permiten cambiar de la interface imagen a la de proyector. Primero, en la opción proyector se pueden elegir las formas rectangulares, triangulares, o con rejilla, después, en la opción imagen, seleccionar determinados fragmentos o su totalidad, y en tercer lugar, en la opción del proyector, definir el ajuste de la composición moviendo con el ratón los puntos activos de los vértices.



Módulo Mapper, opción interface proyector

En la interface de proyección se pueden elegir también máscaras ajustables igualmente con el ratón en sus vértices para eliminar determinados fragmentos en la proyección de la imagen. Inicialmente las máscaras aparecen con forma triangular y se le pueden añadir vértices presionando la tecla control y clicando el ratón en el borde de la máscara donde se quiera incluir el vértice.

Los ajustes del Módulo Mapper se activan/desactivan en la ventana de salida a través del menú principal **Video / Mapper Module / Activate-Deactivate**. También se puede guardar la configuración del mapper con **Save Mapper Settings**, aparecerá una ventana de diálogo para elegir dónde se quiere guardar (se recomienda la carpeta **data** del script) el archivo del tipo **.xml**. Igualmente se pueden cargar configuraciones previamente guardadas con la opción del menú **Load Mapper Settings**.

13. Protocolos de Comunicación

La arquitectura de la red contempla un conjunto de sistemas estándar de comunicación que determinan cómo se pueden intercambiar mensajes entre distintos dispositivos, estos sistemas se denominan protocolos y, aunque pueden estar implementados como hardware, software, o una combinación de ambos, solo vamos a remitirnos a los algoritmos que permiten incorporar en GAmuza algunos de estos protocolos.

13.1. Comunicación por OSC

Como se mostró en el apartado 2.1 Configuración Preferencias, hay una pestaña para asignar los puertos por los que GAmuza envía y/o recibe información por medio del protocolo OSC (Open Sound Control), este protocolo permite intercambiar datos con otros software dentro del mismo ordenador, siempre que sean compatibles con OSC, o entre ordenadores distintos.

Open Sound Control (OSC) es un protocolo de comunicación entre ordenadores, sintetizadores de sonido y otros dispositivos multimedia que está optimizado para la tecnología de redes actuales, entre sus ventajas destaca su facilidad de implementación y su capacidad de comunicar en tiempo real con precisión y flexibilidad, además cuenta con una mayor organización y documentación que otros protocolos como el MIDI. Fue desarrollado por el Center for New Music and Audio Technology (CNDMAT) de la Universidad UC Berkeley.⁸⁵

Para que esta comunicación se produzca, en la ventana **OSC**, de Configuración Preferencias, hay que asignar la dirección IP del ordenador al que se envían los datos (**SENDING TO IP**), si se envían internamente (entre dos software del mismo ordenador) se puede usar la IP local: "127.0.0.1". También hay que asignar el puerto UDP por el que se envían (**SENDING TO PORT**) y el puerto UDP por el que se reciben (**RECEIVING AT PORT**).

Antes de describir las funciones que facilitan esa comunicación, hay que señalar dos cuestiones importantes: 1) Recordad que en GAmuza no se diferencian explícitamente en el código el tipo de variables que se utilizan, pero si debe tenerse en cuenta de qué tipo son cuando se van a enviar a otros software. 2) GAmuza sólo envía datos del tipo string, que luego hay que reconvertir al tipo necesario, por eso vamos a volver a señalar cuáles son las principales funciones que en GAmuza permiten transformar el tipo de datos y posteriormente analizaremos, mediante ejemplos concretos, cómo se señala el tipo en los software de destino para intercambiar mensajes con Processing y Pure data.

⁸⁵ Más información en <<http://opensoundcontrol.org/>> [25.05.2013]

Principales funciones de Lua para manipulación de strings:

toString() el argumento que acoge entre los paréntesis puede ser de cualquier tipo y la función lo convierte en un string con un formato razonable.

Para un mayor control en la conversión de números a string es aconsejable usar **string.format()** dado que incorpora códigos para especificar el tipo de dato, los principales códigos son:

```
string.format("%s", 100)           // %s devuelve un string
100
string.format("%c", 76)           // %c devuelve un carácter, char
L
string.format("%f", math.pi)       // %f devuelve un float como string
3.141593
string.format("%i,%i",-100,100)    // %i número entero con su signo como string
-100, 100
string.byte('L')                 // código numérico de caracteres
76
```

Funciones de openFrameworks útiles para convertir un string en números:

```
ofToFloat("3.14")                // Convierte string(float-string) en valor float
3.14
ofToInt("100")                   // Convierte string (int-string) en valor int
100
```

Las funciones internas de GAMUZA que implementan el protocolo OSC para enviar datos a otros software o dispositivos son: **gaSendingOscTo()**, **gaSetOSCMessage()** y **gaSetOSCValue()**, más la variable interna **OSC_SENDING_PORT**

gaSendingOscTo(), devuelve el número IP del ordenador al que GAMUZA está enviando los datos, se utiliza habitualmente para chequear si la ruta de envío es correcta, junto con la variable **OSC_SENDING_PORT** para ver el número de puerto, por ejemplo con el siguiente código:

```
text=string.format("Sending OSC data IP: %s at port: %s", gaSendingOscTo(), OSC_SENDING_PORT)
ofDrawBitmapString(text,20,200)
```

La función **gaSetOSCMessage()** inicializa, en el bloque **setup()**, un nuevo contenedor de envío de mensajes OSC. Sus parámetros son: una etiqueta y el tipo de dato a enviar. Las opciones de tipo son **OSC_INT**, **OSC_FLOAT** y **OSC_STRING**. La etiqueta se debe definir como un string entre comillas y empezar con barra inclinada, quedando la función: **gaSetOSCMessage("/etiqueta", OSC_FLOAT)**, en el caso de enviar un dato tipo float.

Con la función **gaSetOSCValue()** se actualiza, en el bloque **update()**, el valor del mensaje enviado por OSC, asociado a la etiqueta especificada previamente. Sus parámetros son: la etiqueta y el valor o valores a enviar que deben ser también del tipo tipo string, para lo que se utiliza la función de Lua **toString()**. Si por ejemplo se van a enviar valores aleatorios tipo **float** entre -1 y 1 la función sería: **gaSetOSCValue("/etiqueta", toString(ofRandom()))**

13.1.1. GAmuza <→> Processing

En Processing se debe incorporar la librería externa oscP5 desarrollada por Andreas Schlegel para que pueda realizar envíos o recepción de mensajes por OSC.

El siguiente ejemplo muestra el código del envío de los datos del pitch y volumen de la entrada de audio desde GAmuza a Processing donde se modifica el tamaño de una forma circular.

```
/*
GAmuza 043           E-13-1
-----
OSC - Envío datos GAmuza y Processing
creado por n3m3da | www.d3cod3.org
*/
inputVol = 0.0          //audio input
inputPitch = 0.0
// etiquetas a enviar
sendingInputVol = "/volumen"
sendingInputPitch = "/pitch"

function setup()
    // inicializa las etiquetas a enviar por OSC
    gaSetOSCMessage(sendingInputVol, OSC_FLOAT)
    gaSetOSCMessage(sendingInputPitch, OSC_FLOAT)

end

function update()
    inputVol = gaGetVolume(0)
    inputPitch = gaGetPitch(0)
    // enviar mensajes OSC
    gaSetOSCValue(sendingInputVol,tostring(inputVol))
    gaSetOSCValue(sendingInputPitch,tostring(inputPitch))
end

function draw()
    gaBackgroundColor(0.0,1.0)
    ofSetColor(200)
    // para monitorizar el valor de la IP, el puerto y los datos que se envían
    text =string.format("Send OSC data IP: %s at port: %s",gaSendingOscTo(),OSC_SENDING_PORT)
    ofDrawBitmapString(text,20,200)
```

```
text = string.format("Send OSC data Volume: %s", inputVol)
ofDrawBitmapString(text,20,240)
text = string.format("Send OSC data Pitch: %s", inputPitch)
ofDrawBitmapString(text,20,280)
end
```

El código en Processing para recibir los datos es:

```
/**
 * oscP5sendreceive by andreas schlegel
 * el ejemplo muestra cómo recibir mensajes osc.
 * oscP5 website http://www.sojamo.de/oscP5
 */
//importar librería
import oscP5.*;
import netP5.*;
// Motor OSC
OscP5 oscP5;
// el objeto para enviar datos
NetAddress gamuzalLocation;
// recibir variables
float volumen = 0.0;
float pitch = 0.0;

void setup() {
    size(400,400);
    // inicializar oscP5, a la escucha para recibir mensajes por puerto 9666
    // datos enviados desde GAmuza
    oscP5 = new OscP5(this,9666);
}

void draw() {
    background(0);
    fill(255);
    ellipse(width/2,height/2,pitch*3000,volumen*200);

    println("receiving volumen: " + volumen);
}

// esta función se encarga de estar a la escucha de cualquier mensaje en entrada
void oscEvent(OscMessage theOscMessage) {
    // filtrado de etiquetas
```

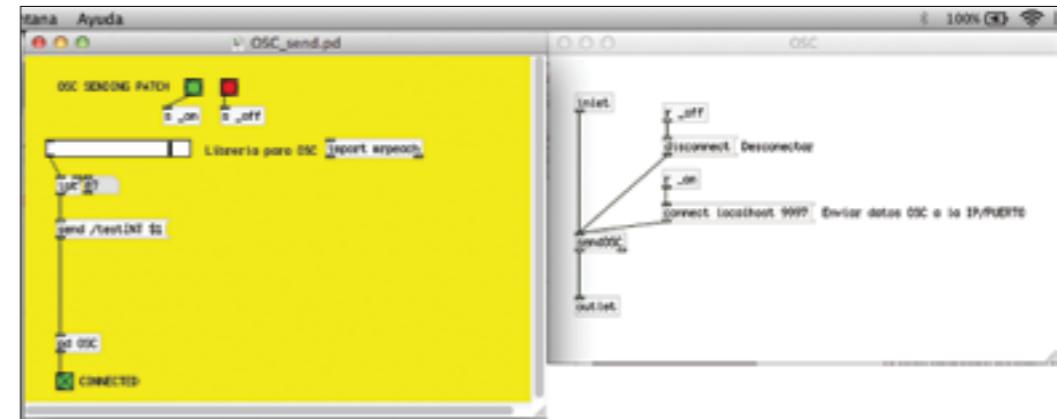
```

if(theOscMessage.checkAddrPattern("/volumen")==true) {
    // filtrado de tipo de datos (int float o string, i - f - s)
    // para comunicar con GAmuza hay que usar datos de tipo String
    if(theOscMessage.checkTypeTag("f")) {
        volumen = theOscMessage.get(0).floatValue();
        return;
    }
} else if(theOscMessage.checkAddrPattern("/pitch")==true){
    if(theOscMessage.checkTypeTag("f")) {
        pitch = theOscMessage.get(0).floatValue();
        return;
    }
}

```

13.1.2. GAmuza <→ Pure Data

En el siguiente ejemplo se establece comunicación enviando datos desde Pure Data y los recibe GAmuza. El patch de pure data utiliza la librería `mrpeach` y el objeto `sendOSC` que está en el subPatch `pd OSC`, quedando:



En GAmuza se recibe el valor entero del slider y el código es:

```

/*
GAmuza 043           E-13-2
-----
OSC - Recibir datos de Pure data
creado por n3m3da | www.d3cod3.org
*/

font = ofTrueTypeFont()

////////// OSC
dataValueFromPD = ""
normalizedValue = 0.0

function setup()
    font.loadFont(gaImportFile("Anonymous_Pro_B.ttf"),24, true, false)
    gaGetOSCMessage("/testINT",1) // inicia recibir etiquetas OSC
end

```

Receiving OSC data at Port: 9997
dato_recibido: 87

```

function update()
    // captura valores OSC
    dataValueFromPD = gaGetOSCValue("/testINT",0)
    normalizedValue = ofToInt(dataValueFromPD)
end

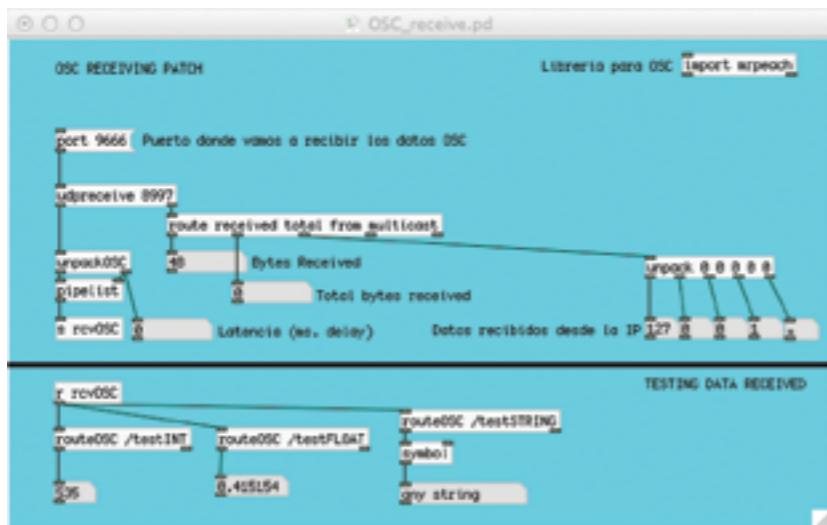
function draw()
    gaBackground(0.0,1.0)

    ofSetColor(221,38,76)
    text = "Receiving OSC data at Port: %s"
    font:drawString(string.format(text, OSC RECEIVING_PORT),20,100)
    font:drawString(string.format("dato_recibido: %i", normalizedValue),20,200)
    ofCircle(OUTPUT_W/2, OUTPUT_H/2, normalizedValue)
end

```

Y en sentido inverso, un ejemplo en el que los datos van desde GAmuza a Pure data por OSC.

El patch de pure data es el siguiente:



Y recibe tres tipos de datos: **int**, **float** y **string** con las etiquetas **testINT**, **testFLOAT** y **testSTRING** respectivamente.

Para ello el código en GAmuza es:

```

/*
GAmuza 043           E-13-3
-----
OSC - Enviar mensajes a Pure Data
creado por n3m3da | www.d3cod3.org
*/

intLabel = "/testINT"
floatLabel = "/testFLOAT"
stringLabel = "/testSTRING"

function setup()
    // inicia recibir etiquetas OSC
    gaSetOSCMessage(intLabel,OSC_INT)
    gaSetOSCMessage(floatLabel,OSC_FLOAT)
    gaSetOSCMessage(stringLabel,OSC_STRING)
end

function update()
    // envía mensajes OSC
    gaSetOSCValue(intLabel,tostring(gaMouseX()))
    gaSetOSCValue(floatLabel,tostring(ofRandomuf()))
    gaSetOSCValue(stringLabel,"any string")
end

function draw()
    gaBackground(0.0,1.0)
    ofSetColor(34,217,179)
    text=string.format("Send OSC data to IP: %s at port: %s",gaSendingOscTo(),OSC_SENDING_PORT)
    ofDrawBitmapString(text,20,200)
end

```

13.2. Comunicación por MIDI. Recibir datos

MIDI (Musical Instrument Digital Interface) es un protocolo estándar de la industria de la tecnología musical que permite conectar instrumentos musicales electrónicos, ordenadores, tablets y otros dispositivos relacionados para que se comuniquen entre sí. A través de la conexión MIDI se pueden transmitir hasta dieciséis canales de información⁸⁶.

Si hay algún dispositivo MIDI conectado al ordenador, hay que seleccionarlo en la pestaña MIDI del panel de Preferences. Cuando GAmuza lo reconoce, lo muestra en la herramienta Logger (menú principal Tools/Logger).

Las funciones de GAmuza para recibir datos del dispositivo MIDI son:

```
int gaGetMidiChannel()
    Devuelve la referencia al dispositivo MIDI conectado y en comunicación. Se puede cambiar
    la configuración MIDI seleccionando el dispositivo en GAmuza / Preferences / pestaña MIDI.

int gaGetMidiByteOne()
    Devuelve la referencia al index del último elemento MIDI modificado, como un slider, botón,
    mando...

int gaGetMidiByteTwo()
    Devuelve el valor del último elemento MIDI modificado, como un slider, botón, mando...

int gaGetMidiPitch()
    Devuelve el pitch de la última tecla MIDI tocada, en el rango MIDI de 0-127.

int gaGetMidiVelocity()
    Devuelve la velocidad de la última de tecla MIDI tocada, en el rango MIDI de 0-127.

float gaGetMidiValue(int deviceMIDI,int indexMIDI)
    El valor normalizado del index MIDI en el dispositivo MIDI seleccionados Por ejemplo:
    gaGetMidiValue(1,11) toma el valor de un slider con index 11 del dispositivo MIDI 1.
```

En el siguiente ejemplo se puede ver el funcionamiento de algunas de estas funciones.

```
/*
GAmuza 0.4.1 ejemplos E-13-4
-----
Uso básico de funciones MIDI
Asegurar que el dispositivo MIDI está conectado y configurado en Preferencias/MIDI.
creado por n3m3da | www.d3cod3.org
*/

font = ofTrueTypeFont()
midiIndex = 11
midiValue = 0
```

⁸⁶ Más información en <<https://www.midi.org/>> y <<https://es.wikipedia.org/wiki/MIDI>> [20.03.2015]

```
function setup()
    font:loadFont("fonts/Anonymous_Pro_B.ttf",64, true, false)
end

function update()
    if gaGetMidiByteOne() == midiIndex then
        midiValue = gaGetMidiByteTwo()
    end
end

function draw()
    gaBackground(0.0,0.8)
    ofSetColor(255)
    ofDrawBitmapString(string.format("Testing MIDI Index %i: ",midiIndex),20,OUTPUT_H/2-80)
    ofRect(20,OUTPUT_H/2 - 40, midiValue, 20)

    ofPushMatrix()
    ofTranslate(20,OUTPUT_H/2 + 80,0)

    ofSetColor(23,169,210)
    // canal MIDI (dispositivo MIDI / programa)
    font:drawString(string.format("Device: %i",gaGetMidiChannel()),0,0)

    // MIDI byte uno (MIDI index, un mando, o trigger, o pad, etc ..)
    font:drawString(string.format("Index: %i",gaGetMidiByteOne()),0,100)

    // MIDI byte dos (valor midi, 0-127)
    font:drawString(string.format("Value: %i",gaGetMidiByteTwo()),0,200)

    ofPopMatrix()
end
```

13.3. Comunicación por DMX. Enviar datos

DMX (Digital MultipleX), también llamado DMX512, es un protocolo electrónico utilizado en luminotecnia para el control de los aparatos de iluminación, permitiendo la comunicación entre los equipos de control de luces (en nuestro caso el ordenador) y las propias fuentes de luz; generalmente ofrece comunicación del tipo unidireccional y sólo transmite información desde el controlador, pero no recibe feedback desde los focos.

Para utilizar este protocolo en GAmuza es necesario el interface Enttec DMX USB Pro⁸⁷ y el rango disponible de canales va de un mínimo de 24 a un máximo de 512. Está especialmente pensado para mover focos de cabeza robótica, asignando un valor de canal a cada una de las funciones, del tipo Pan (giro horizontal), Tilt (giro vertical), Pan/Tilt Speed (velocidad de los movimientos), Color de la luz, Shutter, Dimmer...

En GAmuza funciona gracias al addon de openFrameworks ofxDmx desarrollado por Kyle McDonald⁸⁸, más la función de GAmuza `gaSerialDevice()` que devuelve la referencia del dispositivo conectado en serie, que puede ser el controlador USB DMX u otro dispositivo serie, como un por ejemplo Arduino.

El siguiente ejemplo muestra algunos métodos de la clase ofxDmx

```
/*
GAmuza 0.4.1 ejemplos          E-13-5
-----
Comunicación con dispositivo DMX, clase ofxDmx();
GAmuza puede usar solo el Enttec DMX USB Pro,
<http://www.enttec.com/>
El rango de canales disponible va de un mínimo de 24 y máximo 512.
Original addon de kylemcnold <https://github.com/kylemcnold/ofxDmx>
creado pro n3m3da | www.d3cod3.org
*/
dmx = ofxDmx()
channels = 32
level = 255 // rango señal dmx 1 - 255

function setup()
    dmx:connect(gaSerialDevice(), channels)
    dmx:clear()
    dmx:update(true)
end
```

```
function update()
    dmx:clear()

    level = math.floor(gaMouseX() / OUTPUT_W * 255)
    for i=1, channels do
        dmx:setLevel(i, level)
    end

    if dmx:isConnected() then
        dmx:update(true)
    end

function draw()
    gaBackground(0.0,1.0)

    if dmx:isConnected() then
        ofSetColor(100,78,228)
        ofDrawBitmapString(string.format("DMXdevice %s CONNECTED ",gaSerialDevice()),200,200)
    else
        ofSetColor(234,18,28)
        ofDrawBitmapString("There is no DMX device connected or recognized",200,200)
    end
end
```

⁸⁷ Para más información ver <<http://www.enttec.com/>> [15.02.2016]

⁸⁸ Para más información ver <<https://github.com/kylemcnold/ofxDmx>> [15.02.2016]

13.4. Comunicación por Puerto Serie

El Puerto Serie es una interface de comunicación de datos por el que la información es transmitida bit a bit, enviando un solo bit cada vez, siendo este un dato `int`, con un rango de valor entre 0 y 256.

Una modificación de la clase `ofSerial()` de openFrameworks permite a GAMUZA la conexión con el puerto serie para leer o escribir datos. Es necesario saber el nombre del puerto al que está conectado y la velocidad de datos (baud rate), por ejemplo, en el caso de Arduino se puede obtener estos datos en su app.

Si bien GAMUZA tiene un módulo para comunicar directamente con Arduino, en aquellos proyectos que requieren el uso de librerías específicas para Arduino o una comunicación de bytes más rápida, debe hacerse a través de esta clase.

El siguiente ejemplo muestra la comunicación de bytes con Arduino, para enviar y recibir datos por el puerto serie. En el código de GAMUZA se introduce además una pequeña interfaz gráfica para detener o reactivar la comunicación con la clase `ofxControlPanel()`.

El código de Arduino es:

```
// Serial packet
#define SERIAL_BAUD_VEL 9600

const char HEADER = 'H'; // ASCII H == DEC. 72
const char FOOTER = 'F'; // ASCII F == DEC. 70
const char RECEIVE_CHAR = 'A'; // ASCII A == DEC. 65

int sensorData1 = 0; // int value TO GAMUZA (send)
int sensorData2 = 0; // int value TO GAMUZA (send)
int sensorData3 = 0; // int value TO GAMUZA (send)
int gamuzaData = 0; // int value FROM GAMUZA (receive)

void setup() {
    // inicia el puerto serie con el baud rate definido al principio
    Serial.begin(SERIAL_BAUD_VEL);
}

void loop() {
    // lee el puerto serie
    gamuzaData = Serial.read();
```

```
// ----- TESTING
/*
 * Para enviar datos en forma de byte: 1 byte, necesitamos numeros int entre 0 y 256
 * La forma más cómoda de convertir un rango numérico a otro es la función map()
 */
// Para simular el envío de datos de un sensor se genera un numero random entre 0 y 256
sensorData1 = ceil(random(256));
// se genera un número entre 0 y 1024, y luego se mapea al rango [0-256]
sensorData2 = ceil(map(random(1024), 0, 1024, 0, 256));
// se genera un número random entre 0 y 256
sensorData3 = ceil(random(256));
// ----- TESTING END

if (gamuzaData != -1) { // si la lectura no es nula
    if (gamuzaData == RECEIVE_CHAR) { // si se recibe el valor correcto,
        // entonces envía los datos del sensor a GAMUZA
        sendSerialPacket();
        delay(6);
    }
}
}

void sendSerialPacket(){
    Serial.write(HEADER); // 1 BYTE
    Serial.write((byte)sensorData1); // 1 BYTE
    Serial.write((byte)sensorData2); // 1 BYTE
    Serial.write((byte)sensorData3); // 1 BYTE
    Serial.write(FOOTER); // 1 BYTE
    Serial.flush();
}
```



Y el código en GAMUZA es:

```
/*
GAMUZA 0436           E-13-6
-----
Comunicación con puerto serie: enviar y recibir
Este ejemplo muestra como comunicar con dispositivos vía el puerto serie a nivel de
bytes (ASCII, stuff...), puede ser usado para construir una comunicación sólida y
rápida con Arduino y circuitos customizados
creado por n3m3da | www.d3cod3.org
*/

baudRate = 9600
serial = ofSerial()
bytesReturned = ""
bytesTable = {}
bytes = {}

    // los datos enviados desde Arduino son un paquete de 5 bytes
    // 1 HEADER, 1 SENSOR1, 1 SENSOR2, 1 SENSOR3, 1 FOOTER
    // solo interesa recibir los datos de los sensores

packetLength = 5
HEADER = "H"
FOOTER = "F"
TRIGGER = "A"

// GUI
gui = ofxControlPanel()
viewGui = true
activeCommunication = true

function setup()

    serial:setup("/dev/cu.usbmodem1411",baudRate) //inicializa el puerto serie
                                                // GUI
    gui:setup(0,10,300,OUTPUT_H)
    gui:addPanel(2)
    gui:setWhichPanel(0)
    gui:setWhichColumn(0)
    gui:addLabel("GUI")
    gui:addToggle("Communicate",activeCommunication)

    gui:loadSettings(gaImportFile("simpleGui.xml"))
end
```

```
function update()
    if serial:available() then
        if activeCommunication then
            gaSerialWriteByte(serial,TRIGGER) // envía a Arduino la activación datos
        end
                                // recibe los datos desde Arduino
        bytesReturned = gaSerialReadBytes(serial,packetLength)
        count = 0
        for word in string.gmatch(bytesReturned, "[^:]+") do //patrón de búsqueda
            bytesTable[count] = word
            count = count + 1
        end

        if bytesTable[0] == HEADER then // para controlar el orden de recepción de datos
            count = 0
            for i=1, 3 do
                bytes[count] = bytesTable[i]
                count = count + 1
            end
        end
    end

    gui:update()
    activeCommunication = gui:getValueB("Communicate")
end

function draw()
    gaBackground(0.9,1.0)
    ofSetColor(0)
    for k,v in pairs(bytes) do
        ofDrawBitmapString(tostring(string.byte(bytes[k])),500 + 40*k,200)
        ofDrawBitmapString(string.format("SENSOR %i", k+1),500, 245+100*k )
        ofNoFill()
        ofRect(500, 250+100*k, string.byte(bytes[k]), 30)
    end

    if viewGui then
        gui:draw()
    end
end

function mouseDragged()
    gui:mouseDragged()
end
```

```

function mousePressed()
    gui:mousePressed()
end

function mouseReleased()
    gui:mouseReleased()
end

function keyReleased()
    if gaKey() == string.byte('s') then
        gui:saveSettings(gaImportFile("guiSettings.xml"))
    elseif gaKey() == string.byte('g') then
        viewGui = not viewGui
    elseif gaKey() == string.byte('x') then
        activeCommunication = not activeCommunication
    end
end

```

Para la recepción de datos, la función `gaSerialReadBytes()` de GAMUZA establece internamente un código de separación, introduciendo el signo de dos puntos ":" al inicio de cada dato, por eso el patrón de búsqueda utilizado en `string.gmatch` es "[^:]+", ver el apartado 14.1. Data Parsing, donde se menciona que "Si un patrón comienza con un '^", buscará coincidencias sólo al principio del string", y "Si el patrón va seguido del signo + significa que busque todos los caracteres representados en el patrón".

14. Web Data Access

Muchos artistas visuales utilizan Internet como una gran base de datos, un flujo de inputs que puede ser indexado, escaneado y analizado usando clientes web que por medio de software se automatiza el proceso de solicitud y recepción de archivos remotos en internet. Este proceso utiliza la mutabilidad de los datos (lo que los programadores llaman "casting" una variable de un tipo de datos a otro) pudiendo utilizar diversas fuentes de datos como flujos de entrada para alimentar matrices que pueden rastrear cambios o comportamientos significativos, o simplemente para generar contenidos.

En este capítulo vamos a mostrar diversos procesos que se pueden seguir para obtener y analizar esos datos de entrada ya sea de archivos guardados en disco o directamente de la web.

14.1. Data Parsing

El análisis formal de datos (data parsing) es un proceso de rastreo para encontrar determinado tipo de datos y también para transformar un tipo de datos en otro, por ejemplo, pasar una serie de datos tipo `string` o binarios, a una estructura de datos que sea válida para el programa que se esté codificando.

Si revisamos de nuevo la fórmula básica de Jim Campbell que iniciaba el capítulo 4, donde enfatiza el papel de los datos en el arte digital: DATOS (INPUT) → PROCESAMIENTO → (OUTPUT) INFORMACIÓN, podemos comprender la importancia y necesidad de profundizar en algunos sistemas de programación que utilizan grandes cantidades de datos como recurso básico para generar, con la reutilización de esos datos, un tipo de narrativa o correlación particular entre situaciones dispares. Según el propósito que se plantee, esos conjuntos de datos tendrán necesidades específicas que inciden tanto en la planificación como en la naturaleza de los propios datos.

En el lenguaje Lua hay varias funciones vinculadas a las tablas para rastrear, recoger y guardar los datos del tipo `string`, utilizando patrones de búsqueda denominados Regex (Regular Expression). Las principales funciones son:

`string.find((string, "patrón" [,inicio [,básica]]))`, busca dentro de un `string` las coincidencias con el patrón dado. La función devuelve la posición donde se encuentra el patrón, o nulo si no pudo encontrarlo. Cuando encuentra su patrón, devuelve dos valores: el índice donde comienza la coincidencia y el índice donde termina. El parámetro opcional `inicio` indica dónde empezar la búsqueda. El cuarto parámetro, `básica`, también opcional, es booleano, si su valor es `true`, desactiva las utilidades de detección de patrones, realizando entonces la función una operación de "búsqueda

básica de substring", sin caracteres "mágicos" en el patrón. Si se pone el parámetro **básica** también debe ponerse el parámetro **inicio**.

`string.gmatch(string, "patrón")`, es una función iteradora que, cada vez que se invoca, devuelve las capturas del patrón en el string. Para ello se utiliza un **for** iterativo sobre todas las palabras del string (que es diferente a la estructura **for** de repetición). Si el texto es largo, la búsqueda puede hacerse línea por línea como se muestra en el ejemplo E-14-1.

`string.sub((string, "patrón", reemplaza [, n]))`, devuelve todos los términos del string (o los n primeros, si se especifica el parámetro opcional n) reemplazándolos por el especificado en el tercer parámetro, que puede ser un string, una tabla o una función. `string.sub()` también devuelve, como segundo valor, el número total de coincidencias detectadas.

En Lua, los principales patrones⁸⁹ o Regex de búsqueda son:

- .: (un punto) representa cualquier carácter.
- %a: representa cualquier letra.
- %c: representa cualquier carácter de control.
- %d: representa cualquier dígito.
- %l: representa cualquier letra minúscula.
- %p: representa cualquier carácter de puntuación.
- %s: representa cualquier carácter de espacio.
- %u: representa cualquier letra mayúscula.
- %w: representa cualquier carácter alfanumérico.
- %x: representa cualquier dígito hexadecimal.
- %z: representa el carácter con valor interno 0 (cero).
- %x: (donde x es cualquier carácter no alfanumérico) es la manera estándar de trabajar con los denominados caracteres mágicos ^ \$ () % . [] * + - ?

También sirve para incluir las vocales acentuadas con el patrón %á %é..., cualquier carácter de puntuación precedido por un signo de porcentaje '%' se representa a sí mismo en el patrón.

- [patrones]: Se utilizan los corchetes cuadrados para señalar un conjunto de patrones. Si se inserta el signo '?' junto algún patrón del conjunto, hace que ese patrón sea opcional.
- +: Si el patrón va seguido del signo + significa que busque todos los caracteres representados en el patrón.

Por ejemplo, para buscar en un texto todos los strings que contengan letras mayúsculas, minúsculas, dígitos y comas, el patrón se especificaría "[%a%d%,]+". Si solo debe rastrear las letras mayúsculas y los dígitos "%u%d]+".

También se pueden incluir rangos de caracteres, escribiendo el primero y último carácter del rango separados por un guión. Rara vez se utiliza, ya que los rangos más útiles ya están predefinidos, por ejemplo, "[%0-9]" es más sencillo si se escribe como "%d", "[%0-9&a-f%A-F]" es lo mismo que

"%x". Sin embargo, cuando se necesita encontrar un dígito octal, puede que sea preferible utilizar "[%0-7]", en lugar de una enumeración explícita "[%0%1%2%3%4%5%6%7]".

Si un patrón comienza con un "^", buscará coincidencias sólo al principio del string. Del mismo modo, si termina con "\$", buscará la coincidencias sólo al final del string. Estas marcas se pueden utilizar tanto para restringir los patrones que se encuentran como para anclar patrones. Por ejemplo:

```
if string.find(s, "^%d") then ...
```

comprueba si el string s empieza por un dígito,

```
if string.find(s, "^[-]?(%d+)$") then ...
```

comprueba si el string s representa un número entero, sin otros caracteres iniciales o finales.

Otro tipo de patrón empieza por "%b", y se utiliza para buscar strings simétricos. Se escribe como "%bxy", donde x e y son dos caracteres distintos, la x actúa como carácter de inicio, la y como carácter de cierre. Por ejemplo, el patrón "%b()%" rastrea las partes de los strings que empiezan con "(" y terminan con ")". Este patrón suele utilizarse como "%b()", "%b[]%", "%b%{}%" o "%b<>", pero se puede utilizar cualquier carácter como delimitadores.

Situándonos en un nivel básico, empezaremos observando determinados procesos para recoger datos almacenados en archivos txt y XML guardados en el ordenador.

Estos procesos para trabajar con archivos se denominan **FileIO** [archivos input-output] y para ello GAmuza utiliza las clases de openFrameworks **ofFile()** y **ofBuffer()** para archivos txt y **ofxXmlSettings()** para archivos XML.

14.1.1. Archivos genéricos txt

Las clases y métodos necesarios para analizar los datos de un archivo txt son:

```
ofFile()
open(gaImportFile("nombreArchivo.txt"))
readToBuffer()
writeFromBuffer(ofBuffer)

ofBuffer()
readToBuffer()
getFirstLine()
getNextLine()
getText()
isLastLine()
```

En el siguiente ejemplo se van a almacenar en una tabla todos los números que hay en un archivo txt (guardado en la carpeta **data** del script), los pinta en filas y columnas, cambia a color rojo el que está leyendo en ese momento que también es pintando a mayor tamaño en la parte derecha de la pantalla.

```
/*
Gamuza 043      E-14-1
-----
Leer datos de archivo txt
*/
// Variables archivo texto
textFile = ofFile()
buffer = ofBuffer()
textLines = []
textNums = []
_line = 0
_num = 0
totalNums = 0
actualIndex = 0

font = ofTrueTypeFont() // variables fuentes
fontBig = ofTrueTypeFont()

GamuzaTime = 0 // variables tiempo
scriptTime = 0
myTime = 0
wait = 500 // medio segundo
waitToStart = 3000 // 3 segundos para empezar
lastReset = ofGetElapsedTimeMillis()
```



```
semaphore = true // variable semáforo
 posX = 0 // variables posición números
 posY = 320
 dist = 90
 dx = 60
 dy = 30

function setup()
    // cargar fuentes
    font:loadFont(gaImportFile("NewMediaFett.ttf"),14, true, false)
    fontBig:loadFont(gaImportFile("NewMediaFett.ttf"),132, true, false)

    // capturar datos desde el archivo, línea a línea, número a número
    textFile:open(gaImportFile("num_primos.txt"))
    buffer = textFile:readToBuffer()

    while not buffer:isLastLine() do
        textLines[_line] = buffer:getNextLine()
        for n in string.gmatch(textLines[_line], "%d+") do //for iterativo (%d+ = dígitos)
            table.insert(textNums,_num,string.format("%i", n))
        // insertar nº entero en la tabla
        _num += 1 // incrementar el index de la tabla textNums
    end
    _line += 1 // incrementar el index de la tabla textLines
end

totalNums = _num
scriptTime = ofGetElapsedTimeMillis()
end

function update()
    // obtener el tiempo que ha pasado desde que se ha abierto GMuza
    GMuzaTime = ofGetElapsedTimeMillis() - scriptTime

    if GMuzaTime > waitToStart then
        // crear una linea temporal paralela empezando desde lastReset
        myTime = GMuzaTime - lastReset

        if semaphore == true then
            // loop por todos los números
            if actualIndex < totalNums-1 then
                actualIndex += 1
            else
                actualIndex = 0
            end
        end
    end
end
```

```

// resetear la línea temporal paralela
lastReset = GAMuzaTime
semaphore = false
end

// establece el tiempo de espera
if GAMuzaTime-lastReset > wait then
    semaphore = true
end
end

function draw()
gaBackground(1.0,1.0)

actualNum = textNums[actualIndex]
c = 0 // para desplegar la tabla lineal en 2D
r = 0
for i=0,table.getn(textNums) do // repetición del for = longitud de la tabla

    if i == actualIndex then // si es el nº actual se pinta en rojo
        ofSetColor(255,0,0)
    else
        ofSetColor(0) // si no lo es, en negro
    end

    // dibuja todos los números encontrados
    font.drawString(tostring(textNums[i]), dist+dx*c,dist+dy*r)
    c += 1 // incrementa posición X
    if c == 10 then //si es = 10, vuelve a 0 e incrementa posición Y
        c = 0
        r += 1
    end
end

ofSetColor(255,0,0) // para el nº con fuente grande

posX = OUTPUT_W-100
numW = fontBig:stringWidth(actualNum) // calcula el ancho del nº actual y lo dibuja
fontBig.drawString(tostring(textNums[actualIndex]),posX-numW,OUTPUT_H/2)
end

```

14.1.2. Archivos XML

Las clases y métodos necesarios para analizar los datos de un archivo XML son:

```

ofxXmlSettings()
loadFile(gaImportFile("nombreArchivo.xml"))
pushTag("nombreTag", int)
popTag()
getValue("nombreTag", tipo datos -string "" double o int, int)
getNumTags("nombreTag")
getAttribute("nombreTag", "nombreAtributo", tipo datos, int)
setValue("nombreTag", valor, int)
saveFile()

```

Los documentos XML tienen una estructura jerárquica estricta. Sólo puede tener un elemento raíz, <ROOT> del que todos los demás forman parte y se cierra al final del documento </ROOT>. Su interior está organizado por cadenas de tags (etiquetas) que igualmente se abren y cierran, pudiendo contener atributos, valores o a otros tags anidados.

Cada tag tiene un nombre y además puede tener atributos y un valor, o puede ser un tag vacío. Los atributos sirven para incorporar características o propiedades a los elementos y deben ir entre comillas. En el siguiente ejemplo el tercer tag está vacío:

```

<ROOT>
<data>
    <nombreeTiqueta atributo ="caracteristica"> Valor </nombreeTiqueta>
    <nombreeTiqueta atributo2 ="caracteristica2"> Valor </nombreeTiqueta>
    <nombreeTiqueta atributo3 ="caracteristica3" />
</data>
</ROOT>

```

Además de esta estructura de contenidos, los documentos XML pueden tener un prólogo y comentarios.

GAMuza utiliza la clase de openFrameworks `ofxXmlSettings()` para leer y/o escribir datos en los archivos XML. Con el método `loadFile()` carga el archivo previamente guardado en la carpeta data del script, utilizando la función `gaImportFile("nombreArchivo.xml")` como parámetro.

Para moverse por la estructura anidada del documento utiliza `pushTag("nombreTag", int)` y `popTag()` para establecer el nivel buscado en el documento, empezando por el nivel superior <ROOT>. El segundo parámetro de `pushTag()` indica el número de orden de la etiqueta buscada dentro ese nivel, empezando por `0`. A cada `pushTag()` debe corresponderle un `popTag()` después de que finalicen las operaciones realizadas en ese nivel.

El método `getNumTags("nombreTagActual")` se utiliza para la iteración a través de una lista de tags hermanos que se encuentran dentro del nivel actual y tienen el mismo nombre. Devuelve el número de tags encontrado.

El método `getAttribute("nombreTagActual","nombreAtributo","","int")` devuelve el valor del atributo dentro del tag indicado, puede hacerlo como `string` si el tercer parámetro es "", como `int` si en ese tercer parámetro se pone `0`, o como `float` si se pone `0.0`. El cuarto parámetro indica el número de orden de ese atributo buscado, empezando a contar por `0`.

El método `getValue("nombreTagActual","","int")` devuelve el valor almacenado en el tag especificado en el primer parámetro. El segundo parámetro indica el tipo de datos que queremos que devuelva, "" para `string`, `0` para `int`, o `0.0` para `float`, el tercer parámetro indica el número de orden del valor buscado.

El método `setValue("nombreTagActual","","int")` sirve para escribir o cambiar en el documento XML el valor del tag especificado en su primer parámetro. Si ya existe ese tag, el valor que tenía se sustituye con el ahora asignado, si no existía, se crea un nuevo tag y le asigna este valor.

El método `saveFile(gaImportFile("nombreArchivo.xml"))` guarda el documento, la función de GAmuza le indica la ruta de la carpeta data del script.

El siguiente ejemplo muestra los métodos de lectura de un archivo XML comentados, los incorpora a una tabla y los dibuja.

```
/*
GAmuza 0.545      E-14-2
-----
Leer datos de archivo XML
creado por n3m3da | www.d3cod3.org
*/
file = ofxXmlSettings()
font = ofTrueTypeFont()
fontSmall = ofTrueTypeFont()
source = ""
indicator = ""
data = {}
index = 0

function setup()
    file:loadFile(gaImportFile("test.xml")) // carga archivo xml de la carpeta data
    font:loadFont(gaImportFile("NewMediaFett.ttf"),32, true, false) // carga fuentes
    fontSmall:loadFont(gaImportFile("NewMediaFett.ttf"),20, true, false)
```

Index	Value
0	204
1	202
2	201
3	200
4	199
5	198
6	196
7	194

```
file:pushTag("ROOT",0)           // entra al nivel de ROOT
source = file:getValue("source","",0) // lee datos tipo string
indicator = file:getValue("indicator","",0)
file:pushTag("data",0)           // entra al nivel del tag data
for i=0,file:getNumTags("record")-1 do // obtiene el nº de tags record
    file:pushTag("record",i)         // entra al nivel del tag record
    for j=0,file:getNumTags("field")-1 do // obtiene el nº de tags fiel
        if file:getAttribute("field","name","",j) == "Value" then
            tempValue = file:getValue("field","",j)
            table.insert(data, index, tempValue) // inserta los datos en la tabla
            index += 1                         // incrementa valor de index
        end
    end
    file:popTag()                     // baja un nivel tag
end
file:popTag()                     // baja dos niveles hasta ROOT
file:popTag()

function draw()
    gaBackground(1.0,1.0)
    ofSetColor(125,125,0)

    font:drawString(indicator, 40, 60)
    ofSetColor(125,125,145)
    fontSmall:drawString(source, 40, 110)
    for i=0, table.getn(data) do
        ofSetColor(100)
        fontSmall:drawString(tostring(data[i]),50 + 100*i,200)
        yPos = ofMap(ofToFloat(data[i]),19,21,OUTPUT_H,100,true)

        ofSetColor(125,125,145,90)
        offFill()
        ofCircle(80 + 100*i,yPos,10)
        ofSetColor(125,125,0)
        ofNoFill()
        ofCircle(80 + 100*i,yPos,10)
        ofSetLineWidth(2)
        if i > 0 then
            oldYPos = ofMap(ofToFloat(data[i-1]),19,21,OUTPUT_H,100,true)
            ofLine(80 + 100*i,yPos,80 + 100*(i-1),oldYPos)
        end
    end
end
```

La estructura del archivo XML de referencia es:

```
<ROOT>
<source>http://data.worldbank.org/</source>
<indicator>Birth rate, crude (per 1,000 people)</indicator>
<data>
  <record>
    <field name="Country or Area" key="WLD">World</field>
    <field name="Year">2004</field>
    <field name="Item" key="SP.DYN.CBRT.IN">Birth rate, crude (per 1,000 people)</field>
    <field name="Value">20.4</field>
  </record>
  <record>
    <field name="Country or Area" key="WLD">World</field>
    <field name="Year">2005</field>
    <field name="Item" key="SP.DYN.CBRT.IN">Birth rate, crude (per 1,000 people)</field>
    <field name="Value">20.2</field>
  </record>
  ...
...
```

Tras declarar las variables vinculadas a las clases de openFrameworks para gestionar el archivo XML, las fuentes y la tabla, en el bloque `setup()` se recorre el archivo. Con el método `pushTag()` al nivel de "ROOT" accede a los valores de los tags en él alojados: "`source`" e "`indicator`", tras ello, un nuevo `pushTag()` da acceso al contenido del tag "`data`" y, como el archivo XML contiene distintos tags denominados `record` en ese nivel, una estructura for repite el acceso al siguiente nivel tantas veces como valores devuelve el método `getNumTags("record")` para acceder a la información de cada uno de ellos, y lo mismo con otro for para el nivel "`field`".

Y ya en ese nivel, mediante una estructura condicional, se recogen sólo los datos del atributo "`name`" que se denomina "`Value`" y se van insertando a la tabla `data` con la función de Lua `table.insert`.

14.2. Internet Data

Los datos guardados en un archivo están fijos; podemos rastrearlos, recolectarlos y modificarlos como hemos visto en los apartados anteriores, pero cuando planteamos esas acciones directamente en páginas web, lo primero que debemos tener en cuenta es que esos datos no están fijos, internet es una base de datos de inmensa magnitud en la que todos esos datos son susceptibles de cambiar dinámicamente y el trabajo debe reflejar esos cambios en el tiempo.

¿Qué pasa cuando las cosas empiezan a moverse? ¿Cómo interactuamos con datos "en vivo"? ¿Cómo podemos desentrañar los datos cuando cambian con el tiempo? Podríamos utilizar animación para reproducir la evolución de un conjunto de datos, o la interacción para controlar el lapso de tiempo que estamos mirando. ¿Cómo podemos escribir código para estas situaciones?⁹⁰

Las técnicas que permiten ese acceso dinámico se denomina web scraping, cuyo proceso implica, especialmente, la transformación de datos que están sin estructura (como el formato HTML) en datos estructurados para poder almacenarlos y analizarlos en un archivo o en una base de datos.

Utilizando los patrones de búsqueda o Regex vistos en el apartado 14.1 Data Parsing y las funciones de GAMUZA: `gaStartHTTPService()` `gaAddHTTPUrl(string)` y `gaGetHTTPReceivedResponse()` se puede rastrear la información tipo `string` de una página web y seleccionar los datos buscados almacenándolos en una tabla. Hay que tener en cuenta que no todas las páginas web permiten el scraping por este método.

Estas funciones de GAMUZA actúan como un browser o navegador primitivo que accede a todo el código fuente de la página, por eso es importante encadenar correctamente los patrones o Regex para filtrar y limpiar el formato de esos datos.

La función `gaGetHTTPReceivedResponse()` actúa como el método GET de las peticiones a HTTP que se desarrolla con mayor profundidad en el apartado siguiente. Como avance señalar que siempre que escribimos una dirección en un navegador solicitamos un determinado fichero a un servidor, es decir, hacemos un `request` al servidor. Al teclear Enter se pone en marcha el sistema petición-respuesta entre cliente y servidor siguiendo el método GET.

En el siguiente ejemplo se muestra cómo acceder a una página web y rastrear en ella todas las direcciones web que contiene, ya sean http o https, con la extensión que tengan y el tipo de archivo que sea, lo que permite observar cómo encadenar los Regex para esta búsqueda.

⁹⁰ Ben Fry. (2008). *Visualizing Data*. Sebastopol (CA): O'Reilly, pág. 3.

```

/*
GAmuza 0.545 [442]      E-14-3
-----
Internet data/http parse data
Este ejemplo muestra cómo buscar y recopilar datos de internet utilizando Regular
Expression (regex)
creado por n3m3da | www.d3cod3.org
*/
action_url = "http://WWW.WEBPAGE.ORG/"    // Poner una página web
response = ""
loaded = false
links = {}
lindex = 0

function setup()
  gaStartHTTPService()          // inicializa la conexión
end

function update()
  response = gaGetHTTPReceivedResponse() // respuesta de la petición http
  if response != "" and not loaded then
    for u in string.gmatch(response, "https?:/+%+[a%d]+%.+[%a%d]+%.+[%a%d%/%:.%;?%&%ñ%-%_=]+")
      //gaLog(tostring(u))           // monitoriza en consola las respuestas
      table.insert(links,lindex,tostring(u))  // las inserta en la tabla
      lindex += 1                  // incrementa el index
    end
    loaded = true
  end
end

function draw()
  gaBackground(0.2,1.0)
  ofSetColor(255)
  for i=0, lindex-1 do
    ofDrawBitmapString(links[i],20,20 + (20*i))
  end
end

function keyReleased()
  if gaKey() == string.byte('s') then
    gaAddHTTPUrl(action_url)
  end
end

```

La función `gaStartHTTPService()` en el bloque `setup()` activa la capacidad de conectar GAmuza con internet.

En el bloque `update()` la línea de código: `response = gaGetHTTPReceivedResponse()` obtiene todo el código fuente de la página web si se teclea la letra 's', dado que el envío de la dirección de la página web está condicionado a esa acción en el bloque `keyReleased()`. Para no saturar el sistema, en este mismo bloque el rastreo de datos sólo se efectúa si la variable `response` no tiene ningún `string` y la variable booleana `loaded` es `false`, tras el rastreo pasará a `true`, lo que detiene la búsqueda.

Para determinar el patrón de las regular expression es necesario comprender el formato de los datos buscados, en el caso de encontrar todas las direcciones web que aparecen en una página, el inicio del patrón es `"https?://"` donde el signo ? significa que la letra s es opcional por lo que recogerá todo lo que empiece por http:/ o https:/, la segunda barra podría incorporarse también en este bloque, solo que al coincidir con el signo de comentario // modifica el color del texto, pero funcionaría igual la búsqueda, se ha optado por incorporársela con `+%`, cuando el signo + no está al final del patrón significa que el patrón de búsqueda continúa con los siguientes regex, de ahí que el patrón completo sea: `"https?://+%/+[a%d]+%.+[%a%d%/%:.%;?%&%ñ%-%_=]+"`, tras la segunda barra se añade un bloque para que recoja cualquier letra o dígito, después que busque un punto '.' luego de nuevo cualquier letra o dígito, un nuevo punto y en el último bloque cualquier letra, dígito, barra inclinada, dos puntos, punto, punto y coma, signo de interrogación, signo '&', ñ, guión, guión bajo o signo de igual que pueda tener el string; estos caracteres pueden estar en el orden que sea dentro de esa parte del string, por último, el signo + al final indica que busque todos los string que respondan a ese patrón.

Cada uno de los datos encontrados se inserta en la tabla `links` como string, y se incrementa el valor `index` que lo identifica.

En el bloque `draw()` simplemente se dibujan todos los datos encontrados y almacenados en la tabla `links`.

14.2.1. HTTP Client, enviar datos a un servidor

También se puede establecer comunicación de peticiones con un servidor a través de formularios con el Protocolo HTTP (Protocolo de transferencia de hipertexto) para introducir o modificar datos que estén en la red y más concretamente que estén en un servidor al que tengamos acceso. GAmuza incorpora la versión HTTP/1.0 que permite los métodos de petición GET y POST.

Ya hemos mencionado que cuando se introduce una dirección URL se está ejecutando un método GET. Con el método POST lo que se hace es enviar datos desde el cliente a un programa ubicado en el servidor para que sean procesados y como respuesta actualice o agregue información en ese servidor.

En el proceso de ambos métodos hay una petición (request) de algún dato que será procesado para luego devolver la respuesta (response), por lo que en realidad se podría obtener datos del servidor o subir datos con cualquiera de los dos métodos. Juan Ardissono señala la diferencia fundamental entre ambos métodos:

Las llamadas GET pueden ser cacheadas (historial del navegador), indexadas por buscadores, agregar los enlaces a nuestros favoritos o hasta pasar una url completa a otra persona para que directamente ingrese a esa página. Con el método POST sin embargo no se puede hacer esto.

Generalmente usamos links para ejecutar llamadas GET ya que la idea del link es simplemente “solicitar” una información (página) al servidor y que sea devuelta como una respuesta. Mientras que usamos formularios para actualizar datos de productos, clientes, noticias, etc, también teniendo en cuenta que por el método POST se puede enviar mucha más cantidad de datos que por GET.⁹¹

Para llevar a cabo el método POST desde GAmuza, se utiliza el addon de openFrameworks `ofxHttpForm()` para componer los formularios a través de sus métodos: `addFile(string, string)`, `addFormField(string, string)`, `clearFormFields()` y `getFieldValue(string)`

Esta clase tiene además los métodos `.action()` vinculado normalmente al `string` de la dirección web, y `.method()` que se asocia a las variables `OFX_HTTP_POST` o `OFX_HTTP_GET` según el método de petición que se utilice. La función de GAmuza `gaAddHTTPForm()` gestiona el envío de esas peticiones del formulario.

En el siguiente ejemplo se muestra el código para subir imágenes capturadas con la webCam a un servidor cuando se presiona la tecla 's'. Para ello el script de GAmuza se comunica con un archivo php ubicado en el servidor, y es este archivo el que, como respuesta a las solicitudes del formulario, permite subir las imágenes. Observad en el código del ejemplo y del archivo `update.php` la concordancia de etiquetas para que funcione la comunicación entre ellos. También se debe crear una carpeta llamada `images` que esté alojada en el mismo nivel que el archivo `update.php` dentro del servidor.

⁹¹ Juan Ardissono. 2011. Métodos GET vs POST del HTTP <<http://blog.micayael.com/2011/02/09/metodos-get-vs-post-del-http/>> [06.04.2016]

```
/*
GAmuza 0.545 [442]      E-14-4
-----
http Form Upload File
creado por n3m3da | www.d3cod3.org
*/

// HTTP Formulario
httpForm = ofxHttpForm()

// SERVER data - se necesita crear una carpeta llamada /images en el mismo nivel en el
// que esté alojado archivo upload.php
// ej. archivo upload.php en --> http://www.myserver.org/test/upload.php
// y la carpeta en           --> http://www.myserver.org/test/images/

server = "http:YOUR_SERVER_URL_HERE/" //No olvidar poner la última barra /
                                         // para componer el final de la dirección web con el archivo upload.php
action_url = string.format("%supload.php",server)
counter = 0
requestStr = ""
                                         // webcam imagen
grabber = ofVideoGrabber()
pixels = ofPixels()
image = ofImage()
                                         // tamaño imagen
captureW = 320
captureH = 240

function setup()
    // iniciar HTTP service
    gaStartHTTPService()
                                         // iniciar camara web y las texturas para la imagen
    grabber:setDeviceID(0)
    grabber:initGrabber(captureW,captureH)
    pixels:allocate(captureW,captureH,OF_PIXELS_RGB)
    image:allocate(captureW,captureH,OF_IMAGE_COLOR)
end

function update()
    grabber:update()
    if grabber:isFrameNew() then
        pixels = grabber:getPixelsRef() // coge los pixeles del frame
        image:setFromPixels(pixels) // con ellos construye la imagen
    end
end
```

```

function draw()
gaBackground(0.0,1.0)
ofSetColor(255)

    // Para monitorizar los datos
ofDrawBitmapString(requestStr,20,300) // datos del envío
    // respuesta del servidor (imagen superior)
ofDrawBitmapString(gaGetHTTPReceivedResponse(),20,400)
image:draw(0,0) // Muestra las imágenes de la webCam
end

function keyReleased()
if gaKey() == string.byte('s') then // al teclear 's'
    // compone la ruta de la imagen en el disco
compFile = gaImportFile("GAMUZA_cam")
    // guarda la imagen
image:saveImage(string.format("%s_%i.jpg",compFile,counter),OF_IMAGE_QUALITY_HIGH)

    // .action toma la dirección web del archivo
httpForm.action = action_url

    // hace el http Form con el método POST
httpForm.method = OFX_HTTP_POST

    // añade al formulario el valor de counter y lo llama number
    // esto correspondería a una variable con nombre number y con
    // su valor igual a counter: ej. number = 1
httpForm:addFormField("number", tostring(counter))
    // añade al formulario los datos de envío del archivo de imagen
    //componiendo su nombre y ruta
httpForm:addFile("screenshot",string.format("%s_%i.jpg",compFile,counter))
    // envía el formulario al archivo remoto php
gaAddHTTPForm(httpForm)
    // Compone datos del envío para monitorizarlos en la pantalla
requestStr=string.format("Image %s_%i.jpg sent to server: %s",compFile,counter,server)

    counter += 1 //incrementa el valor de counter para la siguiente imagen
end
end

```

```

<?php

if(isset($_FILES['screenshot'])){

    print("Success! \n");
    print("tmpName: " . $_FILES['screenshot']['tmp_name'] . " \n");
    print("size: " . $_FILES['screenshot']['size'] . " \n");
    print("mime: " . $_FILES['screenshot']['type'] . " \n");
    print("name: " . $_FILES['screenshot']['name'] . " \n");

move_uploaded_file($_FILES['screenshot']['tmp_name'], "images/" . $_FILES['screenshot']['name']);

} else{
    print("Error: " . $_FILES['screenshot']['error'] . " \n");
}

if(isset($_POST['number'])){
    print("Image uploaded Number:". $_POST['number'] . " \n");
}

?>

```

Síntesis remota: /web/test				
	Nombre de archivo	Tamaño de archivo	Tipo de archivo	Última modificación
+	images	Directorio		18/05/2016 13:37:51 02755 33 87286
+	upload.php	644	PHP/Hypertext	30/03/2016 20:26:41 0644 33 87286

Síntesis remota: /web/test/images				
	Nombre de archivo	Tamaño de archivo	Tipo de archivo	Última modificación
+	GAMUZA_cam_0.jpg	10050	jpg-archivo	18/05/2016 13:36:21 0644 33 87286
+	GAMUZA_cam_1.jpg	9434	jpg-archivo	18/05/2016 13:36:30 0644 33 87286
+	GAMUZA_cam_2.jpg	9806	jpg-archivo	18/05/2016 13:36:39 0644 33 87286
+	GAMUZA_cam_3.jpg	9013	jpg-archivo	18/05/2016 13:36:56 0644 33 87286
+	GAMUZA_cam_4.jpg	9416	jpg-archivo	18/05/2016 13:37:15 0644 33 87286
+	GAMUZA_cam_5.jpg	9481	jpg-archivo	18/05/2016 13:37:40 0644 33 87286
+	GAMUZA_cam_6.jpg	9453	jpg-archivo	18/05/2016 13:37:51 0644 33 87286

Y el código del formulario en el archivo update.php es:

15. Aplicaciones: proyectos artísticos

En este apartado se van a mostrar algunos proyectos artísticos desarrollados con GAmuza o traducidos a su lenguaje, describiendo brevemente su planteamiento, los recursos técnicos utilizados y el código de programación.

15.1. Binary Ring, Jared Tarbel

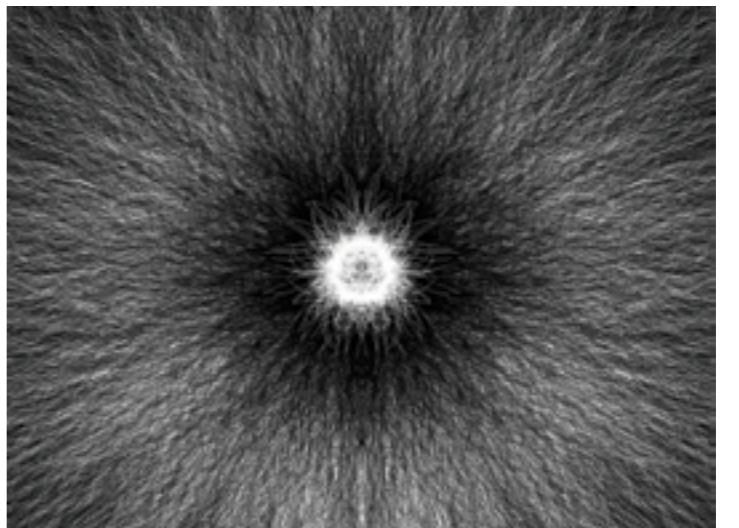
Las premisas del proyecto son la generación de partículas, el Movimiento Browniano y el concepto de *momentum* (*p*), utilizado por Newton para describir un cuerpo en movimiento vinculado al impulso inicial y su masa. $p = m * v$; donde *m* = masa y *v* = velocidad. El proyecto está escrito originariamente en Processing.

Según Tarbell su planteamiento es:

Un sistema de partículas evoluciona continuamente a partir de su creación inicial mostrando el trazo de su recorrido. Cada partícula comienza su vida discretamente quieta, sin experiencia o momentum. Eso cambia rápidamente cuando fuerzas brownianas se aplican al momentum de la partícula en dos dimensiones, y a medida que la partícula se mueve, deja tras de sí el rastro de su paso. Con el tiempo la partícula envejece y muere, dejando atrás su historia de vida como un débil rastro de aventura y descubrimiento⁹².

La composición se articula entre estados de oscuridad que juegan arbitrariamente con estados de luz, este cambio también se puede provocar al clicar sobre la imagen.

Binary Ring, Jared Tarbel, 2003.



92 Jared Tarbell, "Binary Ring". <<http://www.complexification.net/gallery/machines/binaryRing/>> [23.02.2016]

```
/*
GAMUZA 0.4.1 ejemplos E-15-1
-----
Binary Ring de Jared Tarbel
creado pro n3m3da | www.d3cod3.org
*/
c = ofColor()
blackout = false
kaons = {}
numKaons = 1000
///////////
// particle Class (puede estar en otro tab)

class 'Particle'

function Particle:_init(Dx,Dy,r)
    self.Dx = Dx
    self.Dy = Dy
    self.r = r
    self.ox = OUTPUT_W/2
    self.oy = OUTPUT_H/2
    self.x = math.ceil(self.ox-self.Dx)
    self.y = math.ceil(self.oy-self.Dy)
    self.xx = 0
    self.yy = 0
    self.vx = 2*math.cos(self.r)
    self.vy = 2*math.sin(self.r)
    self.age = ofRandom(0,200)

    if blackout then
        c:set(0,0,0,24)
    else
        c:set(255,255,255,24)
    end
end

function Particle:update()
    self.xx = self.x
    self.yy = self.y

    self.x = self.x + self.vx
    self.y = self.y + self.vy

```

```
self.vx = self.vx + (ofRandom()/-2)
self.vy = self.vy + (ofRandom()/-2)
// grow old
self.age = self.age + 1
if self.age > 200 then
    local t = ofRandom(TWO_PI)
    self.x = 3*math.sin(t)
    self.y = 3*math.cos(t)
    self.xx = 0
    self.yy = 0
    self.vx = 0
    self.vy = 0
    self.age = 0

    if blackout then
        c:set(0,0,0,24)
    else
        c:set(255,255,255,24)
    end
end

function Particle:draw()
    ofNoFill()
    ofSetColor(c)
    ofLine(self.ox+self.xx,self.oy+self.yy,self.ox+self.x,self.oy+self.y)
    ofLine(self.ox-self.xx,self.oy+self.yy,self.ox-self.x,self.oy+self.y)
end

/////////// archivo principal

function setup()
    for i=0,numKaons-1 do
        local emitx = math.ceil(3*math.sin(TWO_PI*i/numKaons)+OUTPUT_W/2)
        local emity = math.ceil(3*math.cos(TWO_PI*i/numKaons)+OUTPUT_H/2)
        local r = PI*i/numKaons
        kaons[i] = Particle(emitx,emity,r)
    end
end

function update()
    for i=0,numKaons-1 do
        kaons[i]:update()
    end

```

```

local r = ofRandom(0)
if r > 0.9991 then
    blackout = not blackout
end

function draw()
    gaBackground(0.0,0.001)
    for i=0,numKaons-1 do
        kaons[i]:draw()
    end
end

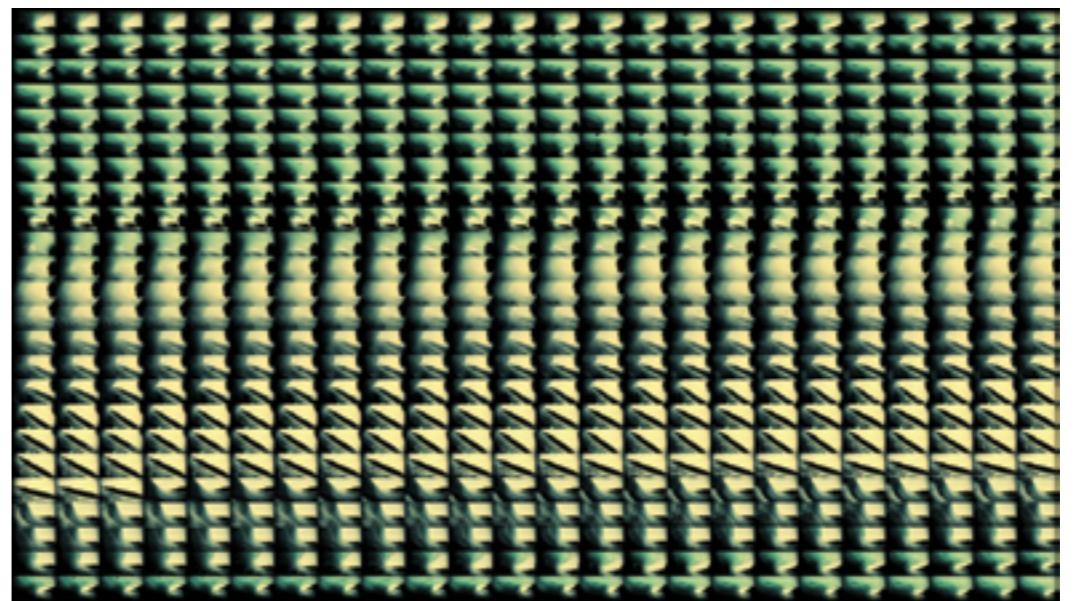
function mouseReleased()
    blackout = not blackout
end

```

15.2. Adaptación de Run Lola Run Lola Run Lola Run, Daniel Shiffman

Trabajando en una demo para el sistema MPE (Most Pixels Ever) que consta de una cuadrícula de celdas, se planteó la posibilidad de reproducir en cada una de esas celdas la película *Run Lola Run* en 60×32 píxeles. Cada célula reproduce un frame posterior (o anterior) al de su contigua a la izquierda (o derecha). La idea es, en última instancia, tener suficiente espacio de píxeles para mostrar la película entera de una sola vez, como en *Cinema Redux* de Brendan Dawes, sólo que en movimiento. Lástima que esta es una violación flagrante de los derechos de autor⁹³.

"Most Pixels Ever" (que no debe confundirse con "Best Pixels Ever") es un framework de código abierto de Java para expandir applets/aplicaciones a través de múltiples pantallas gráficas en tiempo real⁹⁴.



Interpretación de *Run Lola Run Lola Run Lola Run*, Daniel Shiffman, 2007

La adaptación en GAmuza no utiliza ese sistema de comunicación entre pantallas, sino lo que correspondería a la partición en células de cada una de las pantallas utilizadas por Shiffman. Aunque utilizáramos 2 tarjetas Matrox TripleHead no alcanzaríamos el tamaño de pantallas comunicadas entre sí del proyecto inicial.

La traducción del código en GAmuza es:

⁹³ Daniel Shiffman, "Run Lola Run Lola Run Lola Run". <<http://shiffman.net/2007/03/27/runlolarun/>> [20.02.2016]

⁹⁴ Daniel Shiffman, "Most Pixels Ever". <<http://shiffman.net/2007/03/02/most-pixels-ever/>> [20.02.2016]

```

/*
GAmuza 0434           E-15-2
-----
Matriz video 2d desde archivo video
Origen referencia: Run Lola Run Lola Run Lola Run, Daniel Shiffman, 2007
*/

video = ofVideoPlayer()
currentFrame = ofTexture()
currentPixels = ofPixels()
frameCache = {}
frameCount = 0

camW = 1280/20 // tamaño frame original/num de celdas
camH = 720/20

cols = math.floor(OUTPUT_W/camW)
rows = math.floor(OUTPUT_H/camH)
maxFrames = cols*rows

function setup()
    video:loadMovie(gaImportFile("video.mov"))
    video:setLoopState(OF_LOOP_NORMAL)
    video:play()

    currentFrame:allocate(camW,camH,GL_RGB)
    currentPixels:allocate(camW,camH,OF_PIXELS_RGB)
end

function update()
    video:update()

    if video:isFrameNew() then
        currentFrame = video:getTextureReference()
        currentFrame:readToPixels(currentPixels)
        frameCache[frameCount] = ofImage()
        frameCache[frameCount]:allocate(camW,camH,OF_IMAGE_COLOR)
        frameCache[frameCount]:setFromPixels(currentPixels)

        if frameCount > maxFrames then
            frameCount = maxFrames
        else
            frameCount += 1
        end
    end

```

```

if frameCount > 0 then
    for i=1,frameCount-1 do
        frameCache[i-1] = frameCache[i]
    end
end

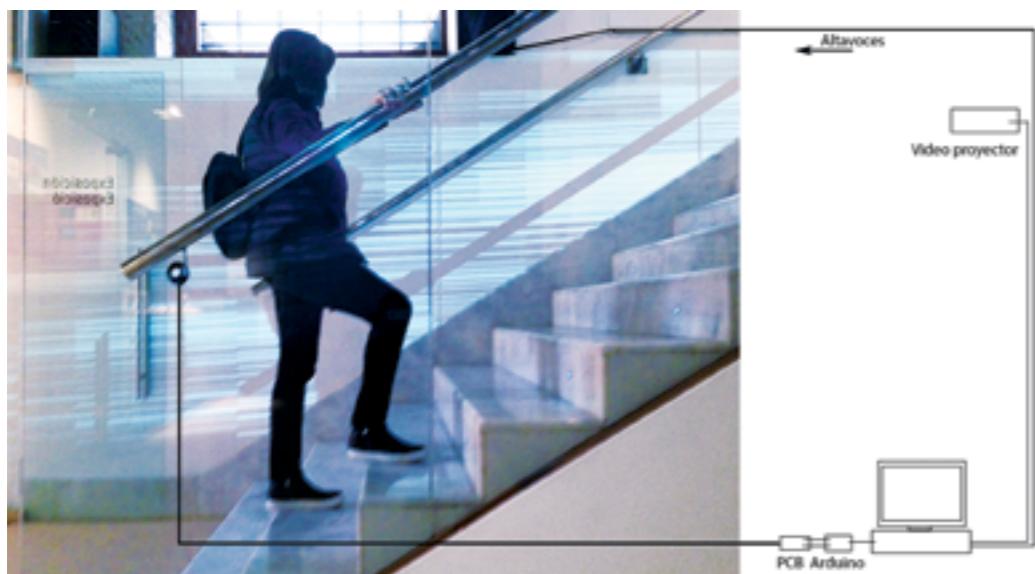
function draw()
    gaBackground(0.0,1.0)
    ofSetColor(255)
    c = 0
    r = 0
    for i=0, table.getn(frameCache)-1 do
        ofSetColor(255)
        frameCache[i]:draw(c*camW,r*camH,camW,camH)
        c += 1
        if c == cols then
            c = 0
            r += 1
        end
    end
end

```

15.3. On Sense, Laboratorio de Luz

El planteamiento del proyecto se basa en la relación luz-sonido como energía, con la intención de vincular esa energía con los espectadores que de forma consciente o no la generan. Para reflejarlo se utiliza la barandilla metálica de la escalera de acceso a un espacio como sensor capacitivo, que es controlado por Arduino. Cuando los visitantes apoyan la mano en la barandilla para bajar o subir la escalera envían un rango de datos que varía en función de la distancia de la mano al punto de conexión y también de su propia carga energética, porque el ser humano también es un dispositivo eléctrico que dispone de capacitancia. Si el campo de capacitancia normal del sensor (su estado de referencia) es alterado por otro campo de capacitancia, el del espectador o espectadores, el circuito electrónico controlado por Arduino registra esos datos de 'distorsión' como inputs.

Estos valores regulan el volumen y velocidad de reproducción de un archivo de audio, y, con los datos que se obtienen del análisis del espectro sonoro de ese audio, se genera a la vez una representación gráfica que es proyectada como líneas oscilantes de luz sobre la pared lateral de la escalera.



On Sense. Laboratorio de Luz, 2016

El proyecto se inició testando el sensor capacitivo con la placa PCB y Arduino⁹⁵, analizando el rango de datos que devolvía al puerto serial cuando alguien tocaba la barandilla, el análisis de estos datos sirvió para establecer un rango de valores desde un mínimo y un máximo, entre los cuales debía responder el sonido y la proyección de formas gráficas.

⁹⁵ Para la librería del sensor capacitivo en Arduino y el código ver: <<http://playground.arduino.cc/Main/CapacitiveSensor?from=Main.CapSense>> [22.02.2016]

El código de Arduino es:

```
#include <CapacitiveSensor.h> // Librería del sensor capacitivo

CapacitiveSensor cs_4_2 = CapacitiveSensor(4,2); //los pin de arduino
int sensorValue = 0; // el valor del sensor capacitivo

void setup() {
  cs_4_2.set_CS_AutoCal_Millis(0xFFFFFFFF); // desactivar autocalibrado en canal 1
  Serial.begin(9600);
}

void loop() {
  sensorValue = cs_4_2.capacitiveSensor(30);
    // mapea el rango de valores de entrada [0-1024] al rango del byte [0-256]
  mappedSensorValue = ceil(map(sensorValue,0, 1024, 0, 256));

    // envia el valor por el puerto serie como dato binario para Pure Data
  Serial.write((byte)mappedSensorValue); // 1 BYTE

  delay(10); // delay para limitar los datos en el puerto serial y no saturarlo
}
```

Y el código de GAMUZA:

```
/*
GAMUZA 0435 E-15-3
-----
On Sense - Laboratorio de Luz
Sensor capacitivo || comunicación con Arduino por puerto serie para
regular velocidad reproducción audio, volumen y visualización gráfica
*/

/////////inputBuffer
nBandsToGet = 16
val = memarray('float',nBandsToGet)
fftSmoothed = memarray('float', nBandsToGet)

///////// archivo sonido
mySound = ofSoundPlayer()
posxVol= 0.0 // para ajustar según volumen
posySpeed = 0.0 // ajuste velocidad reproducción del audio
```

```

////// Serial
baud = 9600
serial = ofSerial()
bytesReturned = ""
bytes = {}
count = 0

maxValue = 320 //PONER VALORES SERIAL de ARDUINO
minValue = 40
normalizedValue = 0.0

function setup()
    // ofSerial
    serial:setup("/dev/cu.usbmodem1411",baud)

    // iniciar memarrays
    for i=0,nBandsToGet-1 do
        val[i] = 0
        fftSmoothed[i] = 0
    end

    // archivo audio
    mySound:loadSound(gaImportFile("hypno00.wav"),true)
    mySound:setLoop(true)
    mySound:play()
end

function update()
    if serial:available() then // captura valores Serial
        bytesReturned = gaSerialReadBytes(serial,1)

        count = 0
        for word in string.gmatch(bytesReturned, "[^:]+") do
            bytes[count] = word
            sensorVal = string.byte(bytes[count])
            normalizedValue = ofMap(sensorVal,minValue,maxValue,1.0,0.0,true)
            count = count + 1
        end
    end
    // activar para monitorizar los valores que regulan el sonido
    // gaLog(string.format("Values: %s", normalizedValue))

    // FFT valores
    for i=0,nBandsToGet-1 do
        val[i] = gaGetSoundSpectrum(i, nBandsToGet)
    end

```

```

if fftSmoothed[i] < val[i] then
    fftSmoothed[i] = val[i]
end
fftSmoothed[i] = fftSmoothed[i]*0.96 //suaviza los valores muy altos
end

mySound:setVolume(normalizedValue) // asigna volumen
mySound:setSpeed(normalizedValue) // asigna velocidad
end

function draw()
    gaBackground(0.0,0.008)
    ofSetLineWidth(5)
    ofSetColor(255,80)
    w = (OUTPUT_W/nBandsToGet)
    j = 1
    for i=0,nBandsToGet-1 do
        ofLine((i*w),(fftSmoothed[i] * 800)* j,(i*(w*2)),(fftSmoothed[i] * 800)* j)
        // se mapea el incremento de j para visibilizar
        // las últimas frecuencias del audio con valor bajo
        j += ofMap(i, 0, nBandsToGet-1, 20, 400)
    end
end

```

15.4. Roomba, Laboratorio de Luz

Proyecto pensado como performance que reflexiona sobre las relaciones entre la emisión de imagen-movimiento por la red, la impresión y difusión de imágenes impresas como etiquetas adhesivas y el espacio del arte.

Una aspiradora robótica Roomba limpia un espacio de arte, trasladando sobre ella una cámara de video IP.

El software GAmuza recoge cada 4 minutos una imagen de la cámara, cuando tiene tres imágenes las compone en una tira y la envía a una impresora de etiquetas adhesivas. Las etiquetas van cayendo sobre una bandeja invitando a los espectadores a cogerlas, llevárselas como recuerdo, o pegarlas en algún lugar del espacio expositivo.



El código en GAmuza para gestionar las imágenes de la cámara, componer la tira y enviarla a la impresora es:

```
/*
GAmuza 0434          E-15-4
-----
Roomba - Laboratorio de Luz
capturar desde cámara IP, guardar y cargar las imágenes
componer 3 imágenes, recortar fondo e imprimirlas
*/
images = {}           // tabla para las 3 imágenes
currentImage = 0
numImages = 3

cameraStream = "http://192.168.1.38/videostream.cgi" // Camara IP
ipGrabber = ofxIpVideoGrabber()

pixels_pantalla = ofPixels() // para componer la tira de imágenes con crop
textura_pantalla = ofTexture()

gaTexture = ofTexture()
```

```
captureW = 320    //tamaño captura imagen
captureH = 240
imagTime = 240000 // 4 min    //Variables Tiempo
contar = 0
contar2 = 0

function setup()
    ipGrabber setUsername("admin") // gestionar la cámara IP
    ipGrabber setPassword("12345")
    ipGrabber setURI(cameraStream)
    ipGrabber connect()

    gaTexture allocate(OUTPUT_W, OUTPUT_H, GL_RGB)
    pixels_pantalla allocate(OUTPUT_W, OUTPUT_H, OF_PIXELS_RGB) // para imagen salida

    textura_pantalla allocate(960, 240, GL_RGB) // recoger píxeles, imagen imprimir
                                                // funciones propias para inicializar tabla imágenes y cargarlas
    setupImages(numImages)
    loadImageLibrary(numImages)
end

function update()
    ipGrabber update() // recoge píxeles imagen de la ventana salida
    gaTexture = gaGetOutputTexture()
    gaTexture readToPixels(pixels_pantalla)
end

function draw()
    gaBackground(0.0, 1.0)

    ofSetColor(255) // muestra la imagen capturada por IPcamera
    ipGrabber draw(0, 0, captureW, captureH) // dibuja los 3 frames guardados
    images[0].draw(300, 300, 320, 240)
    images[1].draw(620, 300, 320, 240)
    images[2].draw(940, 300, 320, 240)

    // tiempo espera
    if contar < imagTime then
        contar = contar+1
    else
        contar = 0
        ofSetColor(255) // guarda imágenes cámara
```

```

_path = gaDataPath(string.format("images/export%i.jpg", currentImage))
gaSaveFrame(_path, ipGrabber:getTextureReference())

pixels_pantalla:crop(300, 300, 960, 240) // recorta pantalla salida
//asigna los píxeles a la textura de la imagen a imprimir
textura_pantalla:loadData(pixels_pantalla)
_path2 = gaDataPath("pantalla.tif")
gaSaveFrame(_path2, textura_pantalla) // guarda la tira de imágenes como tiff
if currentImage < 2 then // para renumerar las imágenes que guarda
    currentImage += 1
else
    currentImage = 0
end
// carga de nuevo las imágenes guardadas en la carpeta
loadImageLibrary(numImages)
end

//////// PARA IMPRIMIR
if contar2 < imagTime*3 then
    contar2 +=1
else
    _path2 = gaDataPath("pantalla.tif")
    gaSystem(string.format("lp-o:media=Custom.64x231mm-o:Halftone=B:ErrorDiffusion
-o Quality=B:Quality300x300dpi %s", _path2))
    contar2 = 0 // comunica con la impresora e imprime la imagen
end
end

//////// FUNCIONES (pueden estar en otro tab)

function setupImages(num)
for i=0,num-1 do
    images[i] = ofImage()
end
end

function loadImageLibrary(num)
for i=0,num-1 do
    file = gaDataPath(string.format("images/export%i.jpg",i))
    images[i]:loadImage(file)
end
end

```

15.5. Folding, Emanuele Mazza y Cristina Ghetti

Folding es una instalación audiovisual que genera patrones geométricos mediante un algoritmo principal de control del sistema, centrado en la temporización y cadencia del flujo de datos que va alimentando las estructuras de patrones visuales, al tiempo que va escribiendo una partitura dinámica en escala pentatónica para la construcción sonora. El proyecto está planteado como un estudio en proceso sobre la percepción, centrado en el uso de herramientas digitales para la realización de piezas que se pueden presentar en distintos formatos y enfocadas principalmente hacia una evolución del arte abstracto y cinético, más concretamente al op-art.

A nivel alto de código, el sistema generativo está constituido por un sistema de ecuaciones de oscilación bidimensional, aplicadas a superficies compuestas por QUADS OpenGL, y correctamente ordenadas para crear el patrón visual de líneas de distinto grosor y alternancia de colores aleatoria. Las ecuaciones de oscilación están embebidas en un sistema de generación pseudo-aleatoria controlada por varios parámetros restrictivos necesarios para lograr la estética deseada. El mismo sistema oscilatorio que genera y transforma los patrones visuales, es el que está modulando la generación sonora (notas-tiempo-silencio) y controlando su secuencia temporal⁹⁶.



⁹⁶ Descripción extraída de Cristina Ghetti y Emanuel Mazza. "Abstracción en movimiento. Proyecto Folding" en Actas 2º Congreso Internacional Arte Ciencia Ciudad 2015 <<http://ocs.editorial.upv.es/index.php/ASC/ASC15/paper/view/2049/1218>> [07.03.2016]

El código del proyecto está organizado con tres clases: `AbstractWave`, `ConstantWave` y `WaveState`. Estas clases, se muestran en el texto al final del archivo principal:

```
/*
GAmuza 0435           E-15-5
-----
Folding - Cristina Ghetti + Emanuele Mazza
*/
// Syphon stuff
gaTex = ofTexture()
stream = ofxSyphonServer()

sampleRate = 192000
mainFreq = 120
fmFreq = mainFreq*PI
amFreq = mainFreq*1.029000
freq = mainFreq
ASCALE = 10
STEP = 80

rotatingBuffer = 0
vr = 0.0
incR= 0.01
zAngle = 20.000000
yAngle = 130.000000
meshZoom = 1.000000
planeZoom = 3.6
planeDim = OUTPUT_W*planeZoom
D2 = planeDim/2
AMP = 80.000000
ampFactor = 10.900000
yIncrement = 0;

waveX_a = AbstractWave()
waveY_a = AbstractWave()
waveX = AbstractWave()
waveY = AbstractWave()
fmod_a = AbstractWave()
amod_a = AbstractWave()
fmod = AbstractWave()
amod = AbstractWave()
    // colors
colCounter = 0
aPLength = 0
```

```
uno = ofColor(0,0,0,255)
dos = ofColor(255,255,255,255)

paleta = {}
function setup()
    ofEnableSmoothing()

    // Syphon init
    gaTex:allocate(OUTPUT_W,OUTPUT_H,GL_RGB)
    gaTex:clear()
    stream: setName("GAmuza")

    paleta[0] = uno
    paleta[1] = dos
    paleta[2] = uno
    paleta[3] = uno
    paleta[4] = uno
    paleta[5] = uno
    paleta[6] = dos
    paleta[7] = uno
    aPLength = table.getn(paleta)

    fmod_a:FMSquareWave(0,fmFreq, freq,0)
    amod_a:FMTriangleWave(0,amFreq,20.5,0)

    fmod:FMSquareWave(0,fmFreq, freq*ASCALE,0)
    amod:FMTriangleWave(0,amFreq,120.5*ASCALE,0)

    waveX_a:AMFMSineWave(0, freq,0,fmod_a,amod_a)
    waveY_a:SineWave(0,freq,10.5,0)

    waveX:AMFMSineWave(0, freq,0,fmod,amod)
    waveY:SineWave(0,0.9,0.5,0)

    gaWave(GA_PULSE,0)      // Pulse oscillator
    gaWave(GA_PHASOR,0)      // Phasor oscillator
    gaWaveVolume(0,0.2)
    gaWaveVolume(1,0.2)

    // Active AU plugins first time
    //auAddPlugin("Apple: AUDistortion")
    //auAddPlugin("Apple: AUGraphicEQ") // Load plugins activated
    auAddPluginFromPreset(gaImportFile("distorsion.aupreset"))
    auAddPluginFromPreset(gaImportFile("eq.aupreset"))

end
```

```

function update()
    gaTex = gaGetOutputTexture()

    if incR > 0.00001 then
        incR -= 0.00001
    else
        incR = 0.01
    end
    vr += incR
    rotatingBuffer = ofMap(ofNoise(vr),0.15,0.85,0,16200, true)

    waveX:setFrequency(rotatingBuffer*2)
    gaWaveFrequency(0, rotatingBuffer/1000)
    gaWaveFrequency(1, rotatingBuffer/1000)
end

function draw()
    gaBackground(1.0,1.0)

    ofPushMatrix()

    ofTranslate(OUTPUT_W*0.5,OUTPUT_H*0.5,0)
    ofRotateY(yAngle+yIncrement)
    ofRotateZ(zAngle)
    ofScale(meshZoom,meshZoom,meshZoom)

    prevY = waveY:update()
    waveY:push()
    colCounter = 0 // avoid mesh vertical flickering
    for y = 0, planeDim-1, STEP do
        valueY = waveY:update()
        waveX:push()
        glBegin(GL_QUAD_STRIP)

        // paleta length control
        if colCounter >= aPLength then
            colCounter = 0
        end

        ofSetColor(paleta[colCounter])
        for x = 0, planeDim-1, STEP do
            valueX = waveX:update()
            glVertex3d(x - D2, y - STEP - D2, (valueX + prevY) * AMP)
            glVertex3d(x - D2, y - D2, (valueX + valueY) * AMP)
        end
    end
end

```

```

colCounter += 1
glEnd()
waveX:pop()
prevY = valueY
end

waveY:pop()
waveX:update()
waveY:update()
fmod_a:update()
amod_a:update()
fmod:update()
amod:update()

ofPopMatrix()

yIncrement += 0.1
// publish video streaming
stream:publishTexture(gaTex)

//ofSetColor(255,0,0)
//ofDrawBitmapString(tostring(incR),200,200)
end

function mouseReleased()
    //auSavePluginPreset("Apple: AUDistortion",gaDataPath(""),"distorsion")
    //auSavePluginPreset("Apple: AUGraphicEQ",gaDataPath(""),"eq")
end

// -----
// ----- class AbstractWave

class 'AbstractWave'

function AbstractWave:_init()
    self.waveType = ""
    self.phase = 0
    self.frequency = 0
    self.amp = 0
    self.offset = 0
    self.value = 0
    self.origPhase = 0
    self.phaseInc = 0.01
    self.stateStack = {} // LIFO stack - Last IN First OUT
end

```

```

self.fmod = nil
self.amod = nil
end

function AbstractWave:setWave(p)
  self.phase = p
  self.frequency = 0
  self.amp = 1
  self.offset = 0
end

function AbstractWave:setWave(p,f)
  self.phase = p
  self.frequency = f
  self.amp = 1
  self.offset = 0
end

function AbstractWave:setWave(p,f,a,o)
  self:setPhase(p)
  self.frequency = f
  self.amp = a
  self.offset = o
end

function AbstractWave:AMFMSineWave(p,f,fm,am)
  self.waveType = "AMFMSineWave"
  self:setWave(p,f)
  self.fmod = fm
  self.amod = am
end

function AbstractWave:AMFMSineWave(p,f,o,fm,am)
  self.waveType = "AMFMSineWave"
  self:setWave(p,f,1,o)
  self.fmod = fm
  self.amod = am
end

function AbstractWave:FHarmonicSquareWave(p,f,a,o)
  self.waveType = "FHarmonicSquareWave"
  self:setWave(p,f,a,o)
  self.fmod = ConstantWave(0)
end

```

```

function AbstractWave:FMSawtoothWave(p,f,a,o)
  self.waveType = "FMSawtoothWave"
  self:setWave(p,f,a,o)
  self.fmod = ConstantWave(0)
end

function AbstractWave:FMSineWave(p,f,a,o)
  self.waveType = "FMSineWave"
  self:setWave(p,f,a,o)
  self.fmod = ConstantWave(0)
end

function AbstractWave:FMSquareWave(p,f,a,o)
  self.waveType = "FMSquareWave"
  self:setWave(p,f,a,o)
  self.fmod = ConstantWave(0)
end

function AbstractWave:FTriangleWave(p,f,a,o)
  self.waveType = "FTriangleWave"
  self:setWave(p,f,a,o)
  self.fmod = ConstantWave(0)
end

function AbstractWave:SineWave(p,f)
  self.waveType = "SineWave"
  self:setWave(p,f,1,0)
end

function AbstractWave:SineWave(p,f,a,o)
  self.waveType = "SineWave"
  self:setWave(p,f,a,o)
end

function AbstractWave:cyclePhase(f)
  self.phase -= self.phaseInc + (f/sampleRate * TWO_PI)
  if self.phase < 0.0 then
    self.phase += TWO_PI
  end
  return self.phase
end

function AbstractWave:setPhase(p)
  self.phase = p
  self:cyclePhase(0)
  self.origPhase = p
end

```

```

function AbstractWave:setFrequency(f)
    self.phase = self.origPhase
    self.frequency = f
end

function AbstractWave:push()
    table.insert(self.stateStack, WaveState(self.phase, self.frequency, self.amp, self.offset))
    if self.fmod ~= nil then
        self.fmod:push()
    end
    if self.amod ~= nil then
        self.amod:push()
    end
end

function AbstractWave:pop()
    s = table.remove(self.stateStack)

    self.phase = s.phase
    self.frequency = s.frequency
    self.amp = s.amp
    self.offset = s.offset

    if self.fmod ~= nil then
        self.fmod:pop()
    end
    if self.amod ~= nil then
        self.amod:pop()
    end
end

function AbstractWave:reset()
    self.phase = self.origPhase
end

function AbstractWave:update()
    if self.waveType == "AMFSineWave" then
        if self.amod ~= nil then
            self.amod:update()
        end
        self.value = self.amp * math.sin(self.phase) + self.offset
        if self.fmod ~= nil then
            self:cyclePhase(self.frequency + self.fmod:update())
        end
    elseif self.waveType == "FMHarmonicSquareWave" then
        self.value = 0
    end
end

```

```

for i=1,3,2 do
    self.value += 1.0 / i*math.sin(i*self.phase)
end
self.value *= self.amp
self.value += self.offset
self:cyclePhase(self.frequency + self.fmod:update())
elseif self.waveType == "FMSawtoothWave" then
    self.value = ((self.phase / TWO_PI) * 2 - 1) * self.amp + self.offset
    self:cyclePhase(self.frequency + self.fmod:update())
elseif self.waveType == "FMSineWave" then
    self.value = (math.sin(self.phase) * self.amp) + self.offset
    self:cyclePhase(self.frequency + self.fmod:update())
elseif self.waveType == "FMSquareWave" then
    if self.phase/TWO_PI < 0.5 then
        self.value = self.amp + self.offset
    else
        self.value = (self.amp * -1) + self.offset
    end
    self:cyclePhase(self.frequency + self.fmod:update())
elseif self.waveType == "FMTriangleWave" then
    self.value = 2 * self.amp * (math.abs(PI-self.phase) * 1/PI - 0.5) + self.offset
    self:cyclePhase(self.frequency + self.fmod:update())
elseif self.waveType == "SineWave" then
    self.value = (math.sin(self.phase) * self.amp) + self.offset
    self:cyclePhase(self.frequency)
end

return self.value
end

// -----
// ----- class ConstantWave

class 'ConstantWave'

function ConstantWave:__init(v)
    self.value = v
end

function ConstantWave:update()
    return self.value
end

```

```

function ConstantWave:push()
    return nil
end

function ConstantWave:pop()
    return nil
end

// -----
// ----- class WaveState

class 'WaveState'

function WaveState:_init(_phase,_freq,_amp,_offset)
    self.phase = _phase
    self.frequency = _freq
    self.amp = _amp
    self.offset = _offset
end

```

15.6. Bloomsday, Andrea Canepa

*Bloomsday*⁹⁷ es un video generado en tiempo real por ordenador y programado con GAmuza. El video muestra el texto de la novela *Ulises* de James Joyce, cuya trama representa el curso de un día cualquiera en la vida de Leopold Bloom, el personaje principal. Cada palabra del libro se proyecta por separado generando una secuencia que tarda 24 horas en reproducirse por completo. El libro en su idioma original, inglés, consta de 265,492 palabras; cada una de ellas aparece en la pantalla durante 0.325433 segundos.



El texto del libro está en un archivo .txt guardado en la carpeta **data** del script de programación. La tipografía utilizada es Courier New, pero como Joyce construye algunas palabras uniendo varias, como por ejemplo 'handsomemarriedwomanrubbedagainstwidebehindinClonskeatram' en el código se diferencia el tamaño de letra, las que tienen una longitud estándar su tamaño es 64, y para las que tienen mayor longitud se les aplica la función **calculateFontSize()**, y para comprobar que la palabra más larga, antes mencionada, se ajusta bien al ancho de la ventana de salida se hace un test que aparece comentado en el código: // TESTING. También con la función, **calculateWaitTime()**, se calcula el tiempo de exposición de cada palabra. El código completo para el proyecto es:

⁹⁷ Más información en <<http://andreacanepa.com/index.php?/artwork/bloomsday/>> [07.03.2016]

```
/*
GAmuza 043           E-15-6
-----
Bloomsday - Andrea Canepa
code by n3m3da | www.d3cod3.org
*/
// TIME VARIABLES
GAmuzaTime = 0
scriptTime = 0
myTime = 0
wait = 1000
totalTime = 86400000 // 24h in ms
waitToStart = 5000 // 5 seconds to start
lastReset = ofGetElapsedTimeMillis()

// SEMAPHORE VARIABLES
semaphore = true

// TEXTFILE VARIABLES
textFile = ofFile()
buffer = ofBuffer()
textLines = {}
textWords = {}
_line = 0
_word = 0
totalWords = 0

maxWordWidth = 0
longestWordIndex = 0
actualIndex = 0

// FONT VARIABLES
font = ofTrueTypeFont()
fontStandard = ofTrueTypeFont()
fontFile = gaImportFile("fonts/CourierNew.ttf")
fontSize = 42
standardFontSize = 64

function setup()
    // capture data from file, line by line, word by word
    textFile:open(gaImportFile("Ulysses.txt"))
    buffer = textFile:readToBuffer()
```

```
while not buffer:isLastLine() do
    textLines[_line] = buffer:getNextLine()
    for w in string.gmatch(textLines[_line], "[%a-zA-Z%-%.%,:;%!%?]+") do
        table.insert(textWords,_word,tostring(w))
        if string.len(textWords[_word]) > maxWordWidth then
            maxWordWidth = string.len(textWords[_word])
            longestWordIndex = _word
        end
        _word += 1
    end
    _line += 1
end

totalWords = _word

calculateFontSize()
calculateWaitTime()

// load true type font
font:loadFont(fontFile,fontSize,true,false)
fontStandard:loadFont(fontFile,standardFontSize,true,false)

scriptTime = ofGetElapsedTimeMillis()
end

function update()
    // get the time elapsed from the opening of GAmuza
    GAmuzaTime = ofGetElapsedTimeMillis() - scriptTime

    if GAmuzaTime > waitToStart then
        myTime = GAmuzaTime - lastReset

        if semaphore == true then
            // loop over all Ulysses words
            if actualIndex < totalWords then
                actualIndex += 1
            else
                actualIndex = 0
            end
            // parallel timeline reset
            lastReset = GAmuzaTime
            semaphore = false
        end
    end
end
```

```

if GAMuzaTime-lastReset > wait then
    semaphore = true
end

end
end
function draw()
    gaBackground(1.0,1.0)

    // find next word
    actualWord = textWords[actualIndex]

    // calculate margins to center word
    wordW = fontStandard:stringWidth(actualWord)

    // draw word
    ofSetColor(0)
    if wordW > OUTPUT_W-20 then
        wordWS = font:stringWidth(actualWord)
        posX = OUTPUT_W/2 - (wordWS/2)
        posY = OUTPUT_H/2 + (fontSize/3)
        font:drawString(actualWord,posX,posY)
    else
        posX = OUTPUT_W/2 - (wordW/2)
        posY = OUTPUT_H/2 + (standardFontSize/3)
        fontStandard:drawString(actualWord,posX,posY)
    end

    // TESTING
    //ofLine(0,OUTPUT_H/2,OUTPUT_W,OUTPUT_H/2)
    //www = font:stringWidth(textWords[longestWordIndex])
    //posX = OUTPUT_W/2 - (www/2)
    //font:drawString(textWords[longestWordIndex],posX,200)

end

function calculateFontSize()
    fontSize = math.floor(OUTPUT_W/maxWordWidth * 1.26)
end

function calculateWaitTime()
    wait = math.ceil(totalTime/totalWords)
end

```

16. Estructura del código fuente de GAMuza

16.1. Integración de openFrameworks

Se listan a continuación las funciones y clases (con sus métodos) de openFrameworks que están integradas en GAMuza. Se han organizado por tipologías para facilitar su uso, si bien caben combinaciones entre clases para tipos de aplicaciones que cruzan transversalmente el orden que hemos utilizado. Se han excluido las clases:

ofArduino -- gestionado desde el módulo GUI Arduino + el framework de GAMuza
 ofSoundStream -- gestionado desde el módulo GUI AudioAnalysis + el framework de GAMuza
 ofLog, ofLogFatalError, ofLogError, ofLogVerbose, ofLogWarning, ofLogNotice -- cubierto por el panel consola de GAMuza
 ofPoint -- utiliza ofVec3f en su lugar
 ofRendererCollection
 ofEvents
 ofPtr

MATH

ofSeedRandom()
 ofSeedRandom(int)
 ofRandom(float)
 ofRandom(float, float)
 ofRandom()
 ofRandomf()
 ofNextPow2(int)
 ofNormalize()
 ofMap(float, float, float, float, float, bool)
 ofClamp(float, float, float)
 ofLerp(float, float, float)
 ofDist(float, float, float, float)
 ofDistSquared(float, float, float, float)
 ofSign(float)
 ofInRange(float, float, float)
 ofRadToDeg(float)
 ofDegToRad(float)
 ofLerpDegrees(float, float, float)
 ofLerpRadians(float, float, float)
 ofAngleDifferenceDegrees(float, float)
 ofAngleDifferenceRadians(float, float)
 ofWrapRadians(float, float, float)

```
ofWrapDegrees(float, float, float)
ofNoise(float)
ofNoise(float, float)
ofNoise(float, float, float)
ofNoise(float, float, float, float)
ofSignedNoise(float)
ofSignedNoise(float, float)
ofSignedNoise(float, float, float)
ofSignedNoise(float, float, float, float)
ofInsidePoly(float, float, vector<ofPoint>)
ofInsidePoly(ofPoint, vector<ofPoint>)
ofLineSegmentIntersection(ofPoint, ofPoint, ofPoint, ofPoint)
ofBezierPoint(ofPoint, ofPoint, ofPoint, ofPoint, float)
ofCurvePoint(ofPoint, ofPoint, ofPoint, ofPoint, float)
ofBezierTangent(ofPoint, ofPoint, ofPoint, ofPoint, float)
ofCurveTangent(ofPoint, ofPoint, ofPoint, ofPoint, float)
```

UTILS

```
ofBufferFromFile(string, bool)
ofBufferToFile(string, ofBuffer, bool)
ofGetElapsedTimeMillis()
ofGetElapsedTimeMicros()
ofGetElapsedTimef()
ofGetSeconds()
ofGetMinutes()
ofGetHours()
ofLaunchBrowser()
ofGetVersionInfo()
ofToInt(string)
ofToFloat(string)
ofSaveScreen()
ofSaveFrame()
ofSaveViewport()
ofSplitString(string, string, bool)
ofGetYear()
ofGetMonth()
ofGetDay()
ofGetWeekday()
ofResetElapsedTimeCounter()
ofGetUnixTime()
ofGetSystemTime()
ofGetSystemTimeMicros()
ofGetTimestampString()
```

```
ofGetTimestampString(string)
ofToChar(string)
ofToBool(string)
ofToBinary(string)
ofBinaryToInt(string)
ofBinaryToChar(string)
ofBinaryToFloat(string)
ofBinaryToString(string)
ofJoinString(vector<string>, string)
ofHexToInt(string)
ofHexToChar(string)
ofHexToFloat(string)
ofHexToString(string)
ofStringReplace(string, string, string)
ofIsStringInString(string, string)
ofSleepMillis(int)
```

GRAPHICS

```
ofBackground(int)
ofBackground(int, int)
ofBackground(int, int, int)
ofBackground(int, int, int, int)
ofBackgroundGradient(ofColor, ofColor, ofGradientMode)
ofSetCircleResolution(int)
ofSetRectMode(int)
ofGetRectMode()
ofRect(float, float, float, float)
ofTriangle(float, float, float, float, float, float)
ofCircle(float, float, float)
ofEllipse(float, float, float, float)
ofLine(float, float, float, float)
ofCurve(float, float, float, float, float, float, float, float)
ofBezier(float, float, float, float, float, float, float, float)
ofSetPolyMode(int)
ofBeginShape()
ofVertex(float, float)
ofVertex(float, float, float)
ofCurveVertex(float, float)
ofCurveVertex(float, float, float)
ofBezierVertex(float, float, float, float, float, float)
ofBezierVertex(float, float, float, float, float, float, float, float)
ofNextContour(bool)
ofEndShape(bool)
```

```

ofNoFill()
ofFill()
ofSetColor(int)
ofSetColor(int,int)
ofSetColor(int,int,int)
ofSetColor(int,int,int,int)
ofSetHexColor(int)
ofEnableAlphaBlending()
ofDisableAlphaBlending()
ofEnableSmoothing()
ofDisableSmoothing()
ofDrawBitmapString(string,ofPoint)
ofDrawBitmapString(string,float,float)
ofDrawBitmapString(string,float,float,float)
ofDrawBitmapStringHighlight(string,ofPoint,ofColor,ofColor)
ofDrawBitmapStringHighlight(string,int,int,ofColor,ofColor)
ofRotate(float,float,float)
ofRotateX(float)
ofRotateY(float)
ofRotateZ(float)
ofPushStyle()
ofPopStyle()
ofSetStyle(ofStyle)
ofGetStyle(ofStyle)
ofSetLineWidth(float)
ofPushMatrix()
ofPopMatrix()
ofTranslate(float,float,float)
ofScale(float,float,float)
ofBeginSaveScreenAsPDF(string,bool,bool,ofRectangle)
ofEndSaveScreenAsPDF()
ofPushView()
ofPopView()
ofViewport(ofRectangle)
ofViewport(float,float,float,float,bool)
ofGetCurrentViewport()
ofGetWidth()
ofGetHeight()
ofTranslate(ofPoint)
ofRotate(float)
ofSetupGraphicDefaults()
ofSetupScreen()
ofSetCurveResolution(int)
ofSetSphereResolution(int)
ofGetFill()

```

```

ofSetColor(ofColor)
ofSetColor(ofColor,int)
ofEnableBlendMode(ofBlendMode)
ofDisableBlendMode()
ofEnablePointSprites()
ofDisablePointSprites()
ofClear(float,float,float,float)
ofClear(float,float)
ofClearAlpha()
ofTriangle(float,float,float,float,float,float,float,float)
ofTriangle(ofPoint,ofPoint,ofPoint)
ofCircle(float,float,float)
ofCircle(ofPoint,float)
ofEllipse(float,float,float,float)
ofEllipse(ofPoint,float)
ofLine(float,float,float,float,float)
ofLine(ofPoint,ofPoint)
ofRect(ofRectangle)
ofRect(ofPoint,float)
ofRect(float,float,float,float)
ofVertex(ofPoint)
ofCurveVertex(ofPoint)
ofBezierVertex(ofPoint,ofPoint,ofPoint)
ofSphere(float,float,float)
ofSphere(float)
ofSphere(ofPoint,float)
ofSphere(float)
ofBox(float,float,float)
ofBox(float,float)
ofBox(ofPoint,float)
ofBox(float)
ofCone(float,float,float,float)
ofCone(float,float,float)
ofCone(ofPoint,float)
ofCone(float)
ofSetDrawBitmapMode(ofDrawBitmapMode)

```

URL

```

ofLoadURL(string)
ofLoadURLAsync(string,string)
ofSaveURLTo(string,string)
ofSaveURLAsync(string,string)
ofRemoveURLRequest(int)
ofRemoveAllURLRequest()

```

CLASES

Métodos de la clase `ofNode()` para 3D

```
ofNode()
ofNode:setParent(ofNode)
ofNode:clearParent()
ofNode:getParent()
ofNode:getPosition()
ofNode:getX()
ofNode:getY()
ofNode:getZ()
ofNode:getXAxis()
ofNode:getYAxis()
ofNode:getZAxis()
ofNode:getSideDir()
ofNode:getLookAtDir()
ofNode:getUpDir()
ofNode:getPitch()
ofNode:getHeading()
ofNode:getRoll()
ofNode:getOrientationQuat()
ofNode:getOrientationEuler()
ofNode:getScale()
ofNode:getLocalTransformMatrix()
ofNode:getGlobalTransformMatrix()
ofNode:getGlobalPosition()
ofNode:getGlobalOrientation()
ofNode:setTransformMatrix(ofMatrix4x4)
ofNode:setPosition(float, float, float)
ofNode:setPosition(ofVec3f)
ofNode:setGlobalPosition(float, float, float)
ofNode:setGlobalPosition(ofVec3f)
ofNode:setOrientation(ofQuaternion)
ofNode:setOrientation(ofVec3f)
ofNode:setGlobalOrientation(ofQuaternion)
ofNode:setScale(float, float, float)
ofNode:setScale(ofVec3f)
ofNode:move(float, float, float)
ofNode:move(ofVec3f)
ofNode:truck(float)
ofNode:boom(float)
ofNode:dolly(float)
ofNode:tilt(float)
ofNode:pan(float)
```

```
ofNode:roll(float)
ofNode:rotate(ofQuaternion)
ofNode:rotate(float, ofVec3f)
ofNode:rotate(float, float, float, float)
ofNode:rotateAround(ofQuaternion, ofVec3f)
ofNode:rotateAround(float, ofVec3f, ofVec3f)
ofNode:lookAt(ofVec3f, ofVec3f)
ofNode:orbit(float, float, float, ofVec3f)
ofNode:orbit(float, float, float, ofNode)
ofNode:transformGL()
ofNode:restoreTransformGL()
ofNode:resetTransform()
ofNode:draw()
```

Métodos de la clase `ofMesh()` para 3D con vectores

```
ofMesh()
ofMesh:setMode(ofPrimitiveMode)
ofMesh:getMode()
ofMesh:clear()
ofMesh:setupIndicesAuto()
ofMesh:getVertex(int)
ofMesh:removeVertex(ofIndexType)
ofMesh:clearVertices()
ofMesh:getNormal(int)
ofMesh:removeNormal(ofIndexType)
ofMesh:clearNormals()
ofMesh:getColor(int)
ofMesh:removeColor(ofIndexType)
ofMesh:clearColors()
ofMesh:getTexCoord(int)
ofMesh:removeTexCoord(ofIndexType)
ofMesh:clearTexCoords()
ofMesh:getIndex(int)
ofMesh:addIndex(ofIndexType)
ofMesh:removeIndex(ofIndexType)
ofMesh:setIndex(int, ofIndexType)
ofMesh:clearIndices()
ofMesh:addTriangle(ofIndexType, ofIndexType, ofIndexType)
ofMesh:getNumVertices()
ofMesh:getNumColors()
ofMesh:getNumNormals()
ofMesh:getNumTexCoords()
ofMesh:getNumIndices()
```

```

ofMesh:getVerticesPointer()
ofMesh:getColorsPointer()
ofMesh:getNormalsPointer()
ofMesh:getTexCoordsPointer()
ofMeshgetIndexPointer()
ofMesh:getName(string)
ofMesh:haveVertsChanged()
ofMesh:haveColorsChanged()
ofMesh:haveNormalsChanged()
ofMesh:haveTexCoordsChanged()
ofMesh:haveIndicesChanged()
ofMesh:hasVertices()
ofMesh:hasColors()
ofMesh:hasNormals()
ofMesh:hasTexCoords()
ofMesh:hasIndices()
ofMesh:drawVertices()
ofMesh:drawWireframe()
ofMesh:drawFaces()
ofMesh:draw()
ofMesh:addVertex(ofVec3f)
ofMesh:setVertex(int,ofVec3f)
ofMesh:addNormal(ofVec3f)
ofMesh:setNormal(int,ofVec3f)
ofMesh:addGroup(ofFloatColor)
ofMesh:setColor(int,ofFloatColor)
ofMesh:addTexCoord(ofVec2f)
ofMesh:setTexCoord(int,ofVec2f)
ofMesh:addIndices(ofIndexType*,int)
ofMesh:getCentroid()

```

Métodos de la clase `ofCamera()` que proporciona una cámara para una escena 3D

```

ofCamera()
ofCamera:setFov(float)
ofCamera:setNearClip(float)
ofCamera:setFarClip(float)
ofCamera:setLensOffset(ofVec2f)
ofCamera:setAspectRatio(float)
ofCamera:setForceAspectRatio(bool)
ofCamera:getFov()
ofCamera:getNearClip()
ofCamera:getFarClip()
ofCamera:getLensOffset()

```

```

ofCamera:getForceAspectRatio()
ofCamera:getAspectRatio()
ofCamera:setupPerspective(bool,float,float,float,ofVec2f)
ofCamera:setupOffAxisViewPortal(ofVec3f,ofVec3f,ofVec3f)
ofCamera:enableOrtho()
ofCamera:disableOrtho()
ofCamera:getOrtho()
ofCamera:getModelViewMatrix()
ofCamera:setPosition(float,float,float)
ofCamera:setPosition(ofVec3f)
ofCamera:setScale(float)
ofCamera:setScale(float,float,float)
ofCamera:setScale(ofVec3f)
ofCamera:move(float,float,float)
ofCamera:move(ofVec3f)
ofCamera:truck(float)
ofCamera:boom(float)
ofCamera:dolly(float)
ofCamera:tilt(float)
ofCamera:pan(float)
ofCamera:roll(float)
ofCamera:rotate(float,ofVec3f)
ofCamera:rotate(float,float,float,float)
ofCamera:lookAt(ofVec3f,ofVec3f)
ofCamera:orbit(float,float,float,ofVec3f)
ofCamera:transformGL()
ofCamera:restoreTransformGL()
ofCamera:resetTransform()
ofCamera:draw()
ofCamera:getImagePlaneDistance(ofRectangle)
ofCamera:getProjectionMatrix(ofRectangle)
ofCamera:getModelViewProjectionMatrix(ofRectangle)
ofCamera:worldToScreen(ofVec3f,ofRectangle)
ofCamera:screenToWorld(ofVec3f,ofRectangle)
ofCamera:worldToCamera(ofVec3f,ofRectangle)
ofCamera:cameraToWorld(ofVec3f,ofRectangle)
ofCamera:beginCamera(ofRectangle)
ofCamera:endCamera()

```

Métodos de la clase `ofEasyCam()` para ver escenas 3D

```

ofEasyCam()
ofEasyCam:beginCamera(ofRectangle)
ofEasyCam:endCamera()

```

```
ofEasyCam:reset()
ofEasyCam:setTarget(ofVec3f)
ofEasyCam:setTarget(ofNode)
ofEasyCam:getTarget()
ofEasyCam:setDistance(float)
ofEasyCam:getDistance()
ofEasyCam:setDrag(float)
ofEasyCam:getDrag()
ofEasyCam:enableMouseInput()
ofEasyCam:disableMouseInput()
ofEasyCam:getMouseInputEnabled()
```

Métodos de la clase `ofMatrix3x3()` para mover vértices 2D

```
ofMatrix3x3()
ofMatrix3x3(float, float, float, float, float, float, float, float)
ofMatrix3x3:set(float, float, float, float, float, float, float, float)
ofMatrix3x3:transpose()
ofMatrix3x3:determinant(ofMatrix3x3)
ofMatrix3x3:invert()
ofMatrix3x3:inverse(ofMatrix3x3)
```

Métodos de la clase `ofMatrix4x4()` para calcular datos necesarios en 3D

```
ofMatrix4x4()
ofMatrix4x4(ofMatrix4x4)
ofMatrix4x4(ofQuaternion)
ofMatrix4x4:getRowAsVec3f(int)
ofMatrix4x4:getRowAsVec4f(int)
ofMatrix4x4:isValid()
ofMatrix4x4:isNaN()
ofMatrix4x4:set(ofMatrix4x4)
ofMatrix4x4:isIdentity()
ofMatrix4x4:makeIdentityMatrix()
ofMatrix4x4:makeScaleMatrix(ofVec3f)
ofMatrix4x4:makeScaleMatrix(float, float, float)
ofMatrix4x4:makeTranslationMatrix(ofVec3f)
ofMatrix4x4:makeTranslationMatrix(float, float, float)
ofMatrix4x4:makeRotationMatrix(float, ofVec3f)
ofMatrix4x4:makeRotationMatrix(float, float, float, float)
ofMatrix4x4:makeRotationMatrix(ofQuaternion)
ofMatrix4x4:makeRotationMatrix(float, ofVec3f, float, ofVec3f, float, ofVec3f)
ofMatrix4x4:makeInvertOf(ofMatrix4x4)
ofMatrix4x4:makeOrthoNormalOf(ofMatrix4x4)
```

```
ofMatrix4x4:makeFromMultiplicationOf(ofMatrix4x4, ofMatrix4x4)
ofMatrix4x4:getInverse()
ofMatrix4x4:makeOrthoMatrix(double, double, double, double, double, double)
ofMatrix4x4:makeOrtho2DMatrix(double, double, double, double)
ofMatrix4x4:makeFrustumMatrix(double, double, double, double, double, double)
ofMatrix4x4:makePerspectiveMatrix(double, double, double, double)
ofMatrix4x4:makeLookAtMatrix(ofVec3f, ofVec3f, ofVec3f)
ofMatrix4x4:makeLookAtViewMatrix(ofVec3f, ofVec3f, ofVec3f)
ofMatrix4x4:getOrtho(double, double, double, double, double, double)
ofMatrix4x4:getFrustum(double, double, double, double, double, double)
ofMatrix4x4:getPerspective(double, double, double, double)
ofMatrix4x4:getLookAt(ofVec3f, ofVec3f, ofVec3f, float)
ofMatrix4x4:decompose(ofVec3f, ofQuaternion, ofVec3f, ofQuaternion)
ofMatrix4x4:setRotate(ofQuaternion)
ofMatrix4x4:setTranslation(ofVec3f)
ofMatrix4x4:rotate(ofQuaternion)
ofMatrix4x4:translate(ofVec3f)
ofMatrix4x4:scale(ofVec3f)
ofMatrix4x4:glRotate(ofQuaternion)
ofMatrix4x4:glTranslate(ofVec3f)
ofMatrix4x4:glScale(ofVec3f)
ofMatrix4x4:getRotate()
ofMatrix4x4:getTranslation()
ofMatrix4x4:getScale()
```

Métodos de la clase `ofQuaternion()` para rotaciones en 3D

```
ofQuaternion()
ofQuaternion(float, float, float, float)
ofQuaternion(ofVec4f)
ofQuaternion(ofVec3f)
ofQuaternion(float, ofVec3f, float, ofVec3f, float, ofVec3f)
ofQuaternion:asVec4()
ofQuaternion:asVec3()
ofQuaternion:set(float, float, float, float)
ofQuaternion:set(ofVec4f)
ofQuaternion:set(ofMatrix4x4)
ofQuaternion:get(ofMatrix4x4)
ofQuaternion:zeroRotation()
ofQuaternion:length()
ofQuaternion:length2()
ofQuaternion:conj()
ofQuaternion:inverse()
ofQuaternion:makeRotate(float, float, float, float)
```

```
ofQuaternion:makeRotate(float,ofVec3f)
ofQuaternion:makeRotate(float,ofVec3f,float,ofVec3f,ofVec3f,ofVec3f)
ofQuaternion:makeRotate(ofVec3f,ofVec3f)
ofQuaternion:makeRotate_original(ofVec3f,ofVec3f)
ofQuaternion:getEuler()
ofQuaternion:slerp(float,ofQuaternion,ofQuaternion)
```

Métodos de la clase `ofVec2f()` para vectores en 2D

```
ofVec2f()
ofVec2f(float,float)
ofVec2f(ofVec3f)
ofVec2f(ofVec4f)
ofVec2f:set(float,float)
ofVec2f:set(ofVec2f)
ofVec2f:match(ofVec2f,float)
ofVec2f:align(ofVec2f,float)
ofVec2f:alignRad(ofVec2f,float)
ofVec2f:getScaled(float)
ofVec2f:scale(float)
ofVec2f:rotate(float)
ofVec2f:rotateRad(float)
ofVec2f:rotate(float,ofVec2f)
ofVec2f:rotateRad(float,ofVec2f)
ofVec2f:getMapped(ofVec2f,ofVec2f,ofVec2f)
ofVec2f:map(ofVec2f,ofVec2f,ofVec2f)
ofVec2f:distance(ofVec2f)
ofVec2f:squareDistance(ofVec2f)
ofVec2f:getInterpolated(ofVec2f,float)
ofVec2f:interpolate(ofVec2f,float)
ofVec2f:getMiddle(ofVec2f)
ofVec2f:middle(ofVec2f)
ofVec2f:average(ofVec2f,int)
ofVec2f:getNormalized()
ofVec2f:normalize()
ofVec2f:getLimited(float)
ofVec2f:limit(float)
ofVec2f:getPerpendicular()
ofVec2f:perpendicular()
ofVec2f:length()
ofVec2f:squareLength()
ofVec2f:angle(ofVec2f)
ofVec2f:angleRad(ofVec2f)
ofVec2f:dot(ofVec2f)
```

```
ofVec2f:rescaled(float)
ofVec2f:rescale(float)
ofVec2f:normalized()
ofVec2f:limited(float)
ofVec2f:perpendicularized()
ofVec2f:lengthSquared()
ofVec2f:interpolated(ofVec2f,float)
ofVec2f:middled(ofVec2f)
ofVec2f:mapped(ofVec2f,ofVec2f,ofVec2f)
ofVec2f:distanceSquared(ofVec2f)
```

Métodos de la clase `ofVec3f()` para vectores en 3D

```
ofVec3f()
ofVec3f(float,float,float)
ofVec3f(ofVec2f)
ofVec3f(ofVec4f)
ofVec3f:set(float,float,float)
ofVec3f:set(ofVec3f)
ofVec3f:match(ofVec3f,float)
ofVec3f:align(ofVec3f,float)
ofVec3f:alignRad(ofVec3f,float)
ofVec3f:getScaled(float)
ofVec3f:scale(float)
ofVec3f:rotate(float,float,float)
ofVec3f:rotateRad(float,float,float)
ofVec3f:rotate(float,ofVec3f)
ofVec3f:rotateRad(float,ofVec3f)
ofVec3f:getMapped(ofVec3f,ofVec3f,ofVec3f)
ofVec3f:map(ofVec3f,ofVec3f,ofVec3f)
ofVec3f:distance(ofVec3f)
ofVec3f:squareDistance(ofVec3f)
ofVec3f:getInterpolated(ofVec3f,float)
ofVec3f:interpolate(ofVec3f,float)
ofVec3f:getMiddle(ofVec3f)
ofVec3f:middle(ofVec3f)
ofVec3f:average(ofVec3f,int)
ofVec3f:getNormalized()
ofVec3f:normalize()
ofVec3f:getLimited(float)
ofVec3f:limit(float)
ofVec3f:getCrossed(ofVec3f)
ofVec3f:cross(ofVec3f)
ofVec3f:getPerpendicular()
```

```
ofVec3f:perpendicular()
ofVec3f:length()
ofVec3f:squareLength()
ofVec3f:angle(ofVec3f)
ofVec3f:angleRad(ofVec3f)
ofVec3f:dot(ofVec3f)
ofVec3f:rescaled(float)
ofVec3f:rescale(float)
ofVec3f:normalized()
ofVec3f:limited(float)
ofVec3f:perpendiculared()
ofVec3f:lengthSquared()
ofVec3f:interpolated(ofVec3f, float)
ofVec3f:middled(ofVec3f)
ofVec3f:mapped(ofVec3f, ofVec3f, ofVec3f)
ofVec3f:distanceSquared(ofVec3f)
```

Métodos de la clase `ofVec4f()` para vectores 4D

```
ofVec4f()
ofVec4f(float, float, float, float)
ofVec4f(ofVec2f)
ofVec4f(ofVec3f)
ofVec4f:set(float, float, float, float)
ofVec4f:set(ofVec4f)
ofVec4f:match(ofVec4f, float)
ofVec4f:getScaled(float)
ofVec4f:scale(float)
ofVec4f:distance(ofVec4f)
ofVec4f:squareDistance(ofVec4f)
ofVec4f:getInterpolated(ofVec4f, float)
ofVec4f:interpolate(ofVec4f, float)
ofVec4f:getMiddle(ofVec4f)
ofVec4f:middle(ofVec4f)
ofVec4f:average(ofVec4f, int)
ofVec4f:getNormalized()
ofVec4f:normalize()
ofVec4f:getLimited(float)
ofVec4f:limit(float)
ofVec4f:length()
ofVec4f:squareLength()
ofVec4f:dot(ofVec4f)
ofVec4f:rescaled(float)
ofVec4f:rescale(float)
```

```
ofVec4f:normalized()
ofVec4f:limited(float)
ofVec4f:lengthSquared()
ofVec4f:interpolated(ofVec4f, float)
ofVec4f:middled(ofVec4f)
ofVec4f:distanceSquared(ofVec4f)
```

Métodos de la clase `ofMaterial()` para mapear objetos 3D

```
ofMaterial()
ofMaterial:setColors(ofFloatColor, ofFloatColor, ofFloatColor, ofFloatColor)
ofMaterial:setDiffuseColor(ofFloatColor)
ofMaterial:setAmbientColor(ofFloatColor)
ofMaterial:setSpecularColor(ofFloatColor)
ofMaterial:setEmissiveColor(ofFloatColor)
ofMaterial:setShininess(float)
ofMaterial:getDiffuseColor()
ofMaterial:getAmbientColor()
ofMaterial:getSpecularColor()
ofMaterial:getEmissiveColor()
ofMaterial:getShininess()
ofMaterial:beginMaterial()
ofMaterial:endMaterial()
```

Métodos de la clase `ofFbo()` [OpenGL Framebuffer object] para trabajar texturas

```
ofFbo(ofFbo)
ofFbo:allocate(int, int, int, int)
ofFbo:draw(float, float)
ofFbo:draw(float, float, float, float)
ofFbo:setAnchorPercent(float, float)
ofFbo:setAnchorPoint(float, float)
ofFbo:resetAnchor()
ofFbo:setDefaultTextureIndex(int)
ofFbo:getDefaultTextureIndex()
ofFbo:getTextureReference()
ofFbo:getTextureReference(int)
ofFbo:getDepthTexture()
ofFbo:setUseTexture(bool)
ofFbo:beginFbo()
ofFbo:endFbo()
ofFbo:readToPixels(ofPixels, int)
ofFbo:readToPixels(ofShortPixels, int)
```

```
ofFbo:readToPixels(ofFloatPixels,int)
ofFbo:getWidth()
ofFbo:getHeight()
ofFbo:bind()
ofFbo:unbind()
ofFbo:getNumTextures()
ofFbo:getFbo()
ofFbo:getDepthBuffer()
ofFbo:getStencilBuffer()
ofFbo:checkGLSupport()
ofFbo:maxColorAttachments()
ofFbo:maxDrawBuffers()
ofFbo:maxSamples()
```

Métodos de la clase `ofVbo()` [OpenGL Vertex buffer object] para vectores 3D

```
ofVbo()
ofVbo(ofVbo)
ofVbo:getVertId()
ofVbo:getColorId()
ofVbo:getNormalId()
ofVbo:getTexCoordId()
ofVbo:getIndexId()
ofVbo:getIdAllocated()
ofVbo:getUsingVerts()
ofVbo:getUsingColors()
ofVbo:getUsingNormals()
ofVbo:getUsingTexCoords()
ofVbo:getUsingIndices()
ofVbo:draw(int,int,int)
ofVbo:drawElements(int,int)
ofVbo:bind()
ofVbo:unbind()
ofVbo:clear()
ofVbo:setMesh(ofMesh,int)
ofVbo:setVertex3Data(ofVec3f*,int,int)
ofVbo:setVertex2Data(ofVec2f*,int,int)
ofVbo:setColorData(ofFloatColor*,int,int)
ofVbo:setNormalData(ofVec3f*,int,int)
ofVbo:setTexCoordData(ofVec2f*,int,int)
ofVbo:setIndexData(ofIndexType*,int,int)
ofVbo:setColorData(float*,int,int,int)
ofVbo:setNormalData(float*,int,int,int)
ofVbo:setTexCoordData(float*,int,int,int)
```

```
ofVbo:updateMesh(ofMesh)
ofVbo:updateVertex3Data(ofVec3f*,int)
ofVbo:updateVertex2Data(ofVec2f*,int)
ofVbo:updateColorData(ofFloatColor*,int)
ofVbo:updateNormalData(ofVec3f*,int)
ofVbo:updateTexCoordData(ofVec2f*,int)
ofVbo:updateIndexData(ofIndexType*,int)
ofVbo:updateVertexData(float*,int)
ofVbo:updateColorData(float*,int)
ofVbo:updateNormalData(float*,int)
ofVbo:updateTexCoordData(float*,int)
```

Métodos de la clase `ofVboMesh()` extensión de `ofMesh()`

```
ofVboMesh(ofMesh)
ofVboMesh:setUsage(int)
ofVboMesh:setMode(ofPrimitiveMode)
ofVboMesh:getMode()
ofVboMesh:clear()
ofVboMesh:setupIndicesAuto()
ofVboMesh:getVertex(int)
ofVboMesh:clearVertices()
ofVboMesh:getNormal(int)
ofVboMesh:clearNormals()
ofVboMesh:getColor(int)
ofVboMesh:clearColors()
ofVboMesh:getTexCoord(int)
ofVboMesh:clearTexCoords()
ofVboMesh:getIndex(int)
ofVboMesh:addIndex(ofIndexType)
ofVboMesh:setIndex(int,ofIndexType)
ofVboMesh:clearIndices()
ofVboMesh:addTriangle(ofIndexType,ofIndexType,ofIndexType)
ofVboMesh:getNumVertices()
ofVboMesh:getNumColors()
ofVboMesh:getNumNormals()
ofVboMesh:getNumTexCoords()
ofVboMesh:getNumIndices()
ofVboMesh:getVerticesPointer()
ofVboMesh:getColorsPointer()
ofVboMesh:getNormalsPointer()
ofVboMesh:getTexCoordsPointer()
ofVboMesh:getIndexPointer()
ofVboMesh:setName(string)
```

```

ofVboMesh::haveVertsChanged()
ofVboMesh::haveColorsChanged()
ofVboMesh::haveNormalsChanged()
ofVboMesh::haveTexCoordsChanged()
ofVboMesh::haveIndicesChanged()
ofVboMesh::hasVertices()
ofVboMesh::hasColors()
ofVboMesh::hasNormals()
ofVboMesh::hasTexCoords()
ofVboMesh::hasIndices()
ofVboMesh::drawVertices()
ofVboMesh::drawWireframe()
ofVboMesh::drawFaces()
ofVboMesh::draw()
ofVboMesh::addVertex(ofVec3f)
ofVboMesh::setVertex(int,ofVec3f)
ofVboMesh::addNormal(ofVec3f)
ofVboMesh::setNormal(int,ofVec3f)
ofVboMesh::addColor(ofFloatColor)
ofVboMesh::setColor(int,ofFloatColor)
ofVboMesh::addTexCoord(ofVec2f)
ofVboMesh::setTexCoord(int,ofVec2f)

```

Métodos de la clase `ofShader()` para Shaders GL Shading Language

```

ofShader()
ofShader::load(string)
ofShader::load(string,string,string)
ofShader::beginShader()
ofShader::endShader()
ofShader::setUniformTexture(char*,ofBaseHasTexture,int)
ofShader::setUniformTexture(char*,ofTexture,int)
ofShader::setUniform1i(char*,int)
ofShader::setUniform2i(char*,int,int)
ofShader::setUniform3i(char*,int,int,v)
ofShader::setUniform4i(char*,int,int,int,int)
ofShader::setUniform1f(char*,float)
ofShader::setUniform2f(char*,float,float)
ofShader::setUniform3f(char*,float,float,float)
ofShader::setUniform4f(char*,float,float,float,float)
ofShader::unload()

```

Métodos de la clase `ofGLRenderer()` para renderizar objetos de OpenGL

```

ofGLRenderer(bool)
ofGLRenderer::getType()
ofGLRenderer::setCurrentFB0(ofFbo*)
ofGLRenderer::update()
ofGLRenderer::draw(ofMesh,bool,bool,bool)
ofGLRenderer::draw(ofMesh,ofPolyRenderMode,bool,bool,bool)
ofGLRenderer::draw(ofPolyline)
ofGLRenderer::draw(ofPath)
ofGLRenderer::draw(ofImage,float,float,float,float,float,float,float,float)
ofGLRenderer::draw(ofFloatImage,float,float,float,float,float,float,float,float)
ofGLRenderer::draw(ofShortImage,float,float,float,float,float,float,float,float)
ofGLRenderer::rendersPathPrimitives()
ofGLRenderer::pushView()
ofGLRenderer::popView()
ofGLRenderer::viewport(ofRectangle)
ofGLRenderer::viewport(float,float,float,float,bool)
ofGLRenderer::setupScreenPerspective(float,float,ofOrientation,bool,float,float,float)
ofGLRenderer::setupScreenOrtho(float,float,ofOrientation,bool,float,float)
ofGLRenderer::getCurrentViewport()
ofGLRenderer::getViewportWidth()
ofGLRenderer::getViewportHeight()
ofGLRenderer::pushMatrix()
ofGLRenderer::popMatrix()
ofGLRenderer::translate(ofPoint)
ofGLRenderer::translate(float,float,float)
ofGLRenderer::scale(float,float,float)
ofGLRenderer::rotate(float,float,float,float)
ofGLRenderer::rotate(float)
ofGLRenderer::rotateX(float)
ofGLRenderer::rotateY(float)
ofGLRenderer::rotateZ(float)
ofGLRenderer::setupGraphicDefaults()
ofGLRenderer::setupScreen()
ofGLRenderer::setFillMode(ofFillFlag)
ofGLRenderer::getFillMode()
ofGLRenderer::setCircleResolution(int)
ofGLRenderer::setRectMode(ofRectMode)
ofGLRenderer::getRectMode()
ofGLRenderer::setLineWidth(float)
ofGLRenderer::setLineSmoothing(bool)
ofGLRenderer::setBlendMode(ofBlendMode)
ofGLRenderer::enablePointSprites()

```

```

ofGLRenderer:disablePointSprites()
ofGLRenderer:setColor(int,int,int)
ofGLRenderer:setColor(int,int,int,int)
ofGLRenderer:setColor(ofColor)
ofGLRenderer:setColor(ofColor,int)
ofGLRenderer:setColor(int)
ofGLRenderer:setHexColor(int)
ofGLRenderer:getBgColor()
ofGLRenderer:bClearBg()
ofGLRenderer:background(ofColor)
ofGLRenderer:background(float)
ofGLRenderer:background(int,float)
ofGLRenderer:background(int,int,int,int)
ofGLRenderer:setBackgroundAuto(bool)
ofGLRenderer:clear(float,float,float,float)
ofGLRenderer:clear(float,float)
ofGLRenderer:clearAlpha()
ofGLRenderer:drawLine(float,float,float,float,float)
ofGLRenderer:drawRectangle(float,float,float,float,float)
ofGLRenderer:drawTriangle(float,float,float,float,float,float,float,float)
ofGLRenderer:drawCircle(float,float,float,float)
ofGLRenderer:drawSphere(float,float,float,float)
ofGLRenderer:drawEllipse(float,float,float,float,float)
ofGLRenderer:drawString(string,float,float,float,ofDrawBitmapMode)

```

Métodos de la clase `ofLight()` para control de iluminación en 3D

```

ofLight()
ofLight(ofLight)
ofLight:destroy()
ofLight:enable()
ofLight:disable()
ofLight:isEnabled()
ofLight:directional()
ofLight:isDirectional()
ofLight:spotlight(float,float)
ofLight:isSpotlight()
ofLight:spotlightCutOff(float)
ofLight:spotConcentration(float)
ofLight:pointLight()
ofLight:isPointLight()
ofLight:attenuation(float,float,float)
ofLight:type()
ofLight:setAmbientColor(ofFloatColor)

```

```

ofLight:setDiffuseColor(ofFloatColor)
ofLight:setSpecularColor(ofFloatColor)
ofLight:getAmbientColor()
ofLight:getDiffuseColor()
ofLight:getSpecularColor()
ofLight:getLightID()
ofLight:customDraw()
ofLight:getPosition()
ofLight:getX()
ofLight:getY()
ofLight:getZ()
ofLight:getXAxis()
ofLight:getYAxis()
ofLight:getZAxis()
ofLight:getSideDir()
ofLight:getLookAtDir()
ofLight:getUpDir()
ofLight:getPitch()
ofLight:getHeading()
ofLight:getRoll()
ofLight:orientationQuat()
ofLight:orientationEuler()
ofLight:getScale()
ofLight:getLocalTransformMatrix()
ofLight:getGlobalTransformMatrix()
ofLight:getGlobalPosition()
ofLight:getGlobalOrientation()
ofLight:transformMatrix(ofMatrix4x4)
ofLight:position(float,float,float)
ofLight:position(ofVec3f)
ofLight:globalPosition(float,float,float)
ofLight:globalPosition(ofVec3f)
ofLight:orientation(ofQuaternion)
ofLight:orientation(ofVec3f)
ofLight:globalOrientation(ofQuaternion)
ofLight:scale(float,float,float)
ofLight:scale(ofVec3f)
ofLight:move(float,float,float)
ofLight:move(ofVec3f)
ofLight:truck(float)
ofLight:boom(float)
ofLight:dolly(float)
ofLight:tilt(float)
ofLight:pan(float)
ofLight:roll(float)

```

```
ofLight:rotate(ofQuaternion)
ofLight:rotate(float,ofVec3f)
ofLight:rotate(float,float,float)
ofLight:rotateAround(ofQuaternion,ofVec3f)
ofLight:rotateAround(float,ofVec3f,ofVec3f)
ofLight:lookAt(ofVec3f,ofVec3f)
ofLight:orbit(float,float,float,ofVec3f)
ofLight:orbit(float,float,float,ofNode)
```

Métodos de la clase `ofTexture()` para trabajar con texturas

```
ofTexture()
ofTexture:allocate(int,int,int)
ofTexture:allocate(int,int,int,bool)
ofTexture:clear()
ofTexture:loadData(ofPixels)
ofTexture:loadScreenData(int,int,int,int)
ofTexture:setAnchorPercent(float,float)
ofTexture:setAnchorPoint(float,float)
ofTexture:resetAnchor()
ofTexture:readToPixels(ofPixels)
ofTexture:draw(ofRectangle)
ofTexture:draw(ofPoint)
ofTexture:draw(ofPoint, float, float)
ofTexture:draw(float,float)
ofTexture:draw(float,float,float)
ofTexture:draw(float,float,float,float,float)
ofTexture:bAllocated()
ofTexture:getCoordFromPoint(float,float)
ofTexture:getCoordFromPercent(float,float)
ofTexture:setTextureWrap(GLint,GLint)
ofTexture:setTextureMinMagFilter(GLint,GLint)
ofTexture:setCompression(ofTexCompression)
ofTexture:isAllocated()
ofTexture:getGLInternalFormat(ofPixels)
ofTexture:getGLFormatAndType(int,int,int)
ofTexture:getImageTypeFromGLType(int)
ofTexture:getGLPolyMode(ofPolyRenderMode)
ofTexture:getOFPolyMode(GLuint)
ofTexture:getGLPrimitiveMode(ofPolyRenderMode)
```

Métodos de la clase `ofPath()` para crear alineamientos de puntos

```
ofPath()
ofPath:clear()
ofPath:newSubPath()
ofPath:close()
ofPath:lineTo(ofPoint)
ofPath:lineTo(float,float,float)
ofPath:moveTo(ofPoint)
ofPath:moveTo(float,float,float)
ofPath:curveTo(ofPoint)
ofPath:curveTo(float,float)
ofPath:curveTo(float,float,float)
ofPath:bezierTo(ofPoint,ofPoint,ofPoint)
ofPath:bezierTo(float,float,float,float,float)
ofPath:bezierTo(float,float,float,float,float,float,float)
ofPath:quadBezierTo(ofPoint,ofPoint,ofPoint)
ofPath:quadBezierTo(float,float,float,float,float)
ofPath:quadBezierTo(float,float,float,float,float,float,float,float)
ofPath:arc(ofPoint,float,float,float)
ofPath:arc(float,float,float,float)
ofPath:arc(float,float,float,float,float)
ofPath:setPolyWindingMode(ofPolyWindingMode)
ofPath:setFilled(bool)
ofPath:setStrokeWidth(float)
ofPath:setColor(ofColor)
ofPath:setHexColor(int)
ofPath:setFillColor(ofColor)
ofPath:setFillHexColor(int)
ofPath:setStrokeColor(ofColor)
ofPath:setStrokeHexColor(int)
ofPath:getWindingMode()
ofPath:isFilled()
ofPath:getFillColor()
ofPath:getStrokeColor()
ofPath:getStrokeWidth()
ofPath:hasOutline()
ofPath:draw(float,float)
ofPath:draw()
ofPath:getOutline()
ofPath:getTessellation()
ofPath:simplify(float)
ofPath:flagShapeChanged()
ofPath:setCurveResolution(int)
ofPath:getCurveResolution()
```

```
ofPath:setArcResolution(int)
ofPath:getArcResolution()
ofPath:setUseShapeColor(bool)
ofPath:getUseShapeColor()
ofPath:tessellate()
ofPath:translate(ofPoint)
ofPath:rotate(float,ofVec3f)
ofPath:scale(float,float)
```

Métodos de la clase `ofColor()` para trabajar Color

```
ofColor()
ofColor(float,float,float,float)
ofColor(ofColor)
ofColor(ofColor,float)
ofColor(float,float)
ofColor:getHex()
ofColor:clamp()
ofColor:invert()
ofColor:normalize()
ofColor:getClamped()
ofColor:getInverted()
ofColor:getNormalized()
ofColor:getHue()
ofColor:getSaturation()
ofColor:getBrightness()
ofColor:getLightness()
ofColor:setHue(float)
ofColor:setSaturation(float)
ofColor:setBrightness(float)
ofColor:setHsb(float,float,float,float)
ofColor:setHsb(float,float,float)
ofColor:set(float,float,float,float)
ofColor:set(float,float)
ofColor:set(ofColor const)
ofColor:setHex(int,float)
ofColor:lerp(ofColor, float)
ofColor:getLerped(ofColor, float)
ofColor:getHsb(float,float,float)
```

Métodos de la clase `ofFloatColor()` para trabajar Color

```
ofFloatColor()
ofFloatColor(float,float,float,float)
```

```
ofFloatColor(ofFloatColor)
ofFloatColor(ofFloatColor,float)
ofFloatColor(float,float)
ofFloatColor:getHex()
ofFloatColor:clamp()
ofFloatColor:invert()
ofFloatColor:normalize()
ofFloatColor:getClamped()
ofFloatColor:getInverted()
ofFloatColor:getNormalized()
ofFloatColor:getHue()
ofFloatColor:getSaturation()
ofFloatColor:getBrightness()
ofFloatColor:getLightness()
ofFloatColor:setHue(float)
ofFloatColor:setSaturation(float)
ofFloatColor:setBrightness(float)
ofFloatColor:setHsb(float,float,float,float)
ofFloatColor:setHsb(float,float,float)
ofFloatColor:set(float,float,float,float)
ofFloatColor:set(float,float)
ofFloatColor:set(ofFloatColor const)
ofFloatColor:setHex(int,float)
ofFloatColor:lerp(ofFloatColor, float)
ofFloatColor:getLerped(ofFloatColor, float)
ofFloatColor:getHsb(float,float,float)
```

Métodos de la clase `ofShortColor()` para trabajar Color

```
ofShortColor()
ofShortColor(float,float,float,float)
ofShortColor(ofShortColor)
ofShortColor(ofShortColor,float)
ofShortColor(float,float)
ofShortColor:getHex()
ofShortColor:clamp()
ofShortColor:invert()
ofShortColor:normalize()
ofShortColor:getClamped()
ofShortColor:getInverted()
ofShortColor:getNormalized()
ofShortColor:getHue()
ofShortColor:getSaturation()
ofShortColor:getBrightness()
```

```
ofShortColor:getLightness()
ofShortColor:setHue(float)
ofShortColor:setSaturation(float)
ofShortColor:setBrightness(float)
ofShortColor:setHsb(float,float,float,float)
ofShortColor:setHsb(float,float,float)
ofShortColor:set(float,float,float,float)
ofShortColor:set(float,float)
ofShortColor:set(ofShortColor const)
ofShortColor:setHex(int,float)
ofShortColor:lerp(ofShortColor, float)
ofShortColor:getLerped(ofShortColor, float)
ofShortColor:getHsb(float,float,float)
```

Métodos de la clase `ofPixels()` para trabajar con pixels de Imagen o Texturas

```
ofPixels()
ofPixels:allocate(int,int,int)
ofPixels:set(unsigned char)
ofPixels:setFromPixels(unsigned char*,int,int,int)
ofPixels:setFromPixels(unsigned char*,int,int,ofImageType)
ofPixels:setFromExternalPixels(unsigned char*,int,int,int)
ofPixels:setFromAlignedPixels(unsigned char*,int,int,int,int)
ofPixels:swapRgb()
ofPixels:clear()
ofPixels:getPixels()
ofPixels:getPixelIndex(int,int)
ofPixels:getColor(int,int)
ofPixels:setColor(int,int,ofColor)
ofPixels:isAllocated()
ofPixels:getWidth()
ofPixels:getHeight()
ofPixelsgetBytesPerPixel()
ofPixels:getBitsPerPixel()
ofPixels:getImageType()
ofPixels:crop(int,int,int,int)
ofPixels:rotate90(int)
ofPixels:mirror(bool,bool)
ofPixels:resize(int,int)
```

Métodos de la clase `ofImage()` para trabajar con archivos de imagen

```
ofImage(string)
ofImage:allocate(int,int,int)
ofImage:clear()
ofImage:clone(ofImage)
ofImage:setUseTexture(bool)
ofImage:getTextureReference()
ofImage:loadImage(string)
ofImage:saveImage(string, ofImageQualityType)
ofImage:saveImageBuffer(ofBuffer, ofImageQualityType)
ofImage:getPixels()
ofImage:getPixelsRef()
ofImage:setFromPixels(ofPixels)
ofImage:setImageType(int)
ofImage:resize(int,int)
ofImage:grabScreen(int,int,int,int)
ofImage:update()
ofImage:draw(float,float)
ofImage:draw(float,float,float,float)
ofImage:getWidth()
ofImage:getHeight()
ofImage:getColor(int,int)
ofImage:setColor(int,int,ofColor)
ofImage:setImageType(ofImageType)
ofImage:crop(int,int,int,int)
ofImage:cropFrom(ofImage,int,int,int,int)
ofImage:rotate90(int)
ofImage:mirror(bool,bool)
ofImage:setAnchorPoint(float,float)
ofImage:draw(ofRectangle)
ofImage:draw(ofPoint,float,float)
ofImage:draw(float,float,float)
ofImage:draw(ofPoint)
ofImage:draw(float,float,float,float,float)
ofImage:bAllocated(bool)
ofImage:reloadTexture()
```

Métodos de la clase `ofFloatImage()` para trabajar con archivos de imagen

```
ofFloatImage()
ofFloatImage(string)
ofFloatImage:allocate(int,int,int)
ofFloatImage:clear()
```

```

ofFloatImage:clone(ofFloatImage)
ofFloatImage:setUseTexture(bool)
ofFloatImage:getTextureReference()
ofFloatImage:loadImage(string)
ofFloatImage:saveImage(string, ofImageQualityType)
ofFloatImage:saveImageBuffer(ofBuffer, ofImageQualityType)
ofFloatImage:getPixels()
ofFloatImage:setFromPixels(ofPixels)
ofFloatImage:setImageType(int)
ofFloatImage:resize(int,int)
ofFloatImage:grabScreen(int,int,int,int)
ofFloatImage:update()
ofFloatImage:draw(float,float)
ofFloatImage:draw(float,float,float,float)
ofFloatImage:getWidth()
ofFloatImage:getHeight()
ofFloatImage:getColor(int,int)
ofFloatImage:setColor(int,int,ofColor)
ofFloatImage:setImageType(ofImageType)
ofFloatImage:crop(int,int,int,int)
ofFloatImage:cropFrom(ofFloatImage,int,int,int,int)
ofFloatImage:rotate90(int)
ofFloatImage:mirror(bool,bool)
ofFloatImage:setAnchorPoint(float,float)
ofFloatImage:draw(ofRectangle)
ofFloatImage:draw(ofPoint,float,float)
ofFloatImage:draw(float,float,float)
ofFloatImage:draw(ofPoint)
ofFloatImage:draw(float,float,float,float,float)
ofFloatImage:bAllocated(bool)
ofFloatImage:reloadTexture()

```

Métodos de la clase `ofFile()` para trabajar con archivos

```

ofFile()
ofFile(ofFile)
ofFile:openFile()
ofFile:close()
ofFile:create()
ofFile:exists()
ofFile:path()
ofFile:getExtension()
ofFile:getFileName()
ofFile:getBaseName()

```

```

ofFile:getEnclosingDirectory()
ofFile:getAbsolutePath()
ofFile:canRead()
ofFile:canWrite()
ofFile:canExecute()
ofFile:isFile()
ofFile:isLink()
ofFile:isDirectory()
ofFile:isDevice()
ofFile:isHidden()
ofFile:setWriteable(bool)
ofFile:setReadOnly(bool)
ofFile:setExecutable(bool)
ofFile:copyTo(string,bool,bool)
ofFile:moveTo(string,bool,bool)
ofFile:renameTo(string,bool,bool)
ofFile:remove(bool)
ofFile:getSize()
ofFile:readToBuffer()
ofFile:writeFromBuffer(ofBuffer)
ofFile:copyFromTo(string,string,bool,bool)
ofFile:moveFromTo(string,string,bool,bool)
ofFile:doesFileExist(string,bool)
ofFile:removeFile(string,bool)

```

Métodos de la clase `ofDirectory()` para trabajar con directorios de archivos

```

ofDirectory()
ofDirectory(string)
ofDirectory:open(string)
ofDirectory:close()
ofDirectory:create(bool)
ofDirectory:exists()
ofDirectory:path()
ofDirectory:isDirectory()
ofDirectory:isHidden()
ofDirectory:setWriteable(bool)
ofDirectory:setReadOnly(bool)
ofDirectory:setExecutable(bool)
ofDirectory:setShowHidden(bool)
ofDirectory:copyTo(string,bool,bool)
ofDirectory:moveTo(string,bool,bool)
ofDirectory:renameTo(string,bool,bool)
ofDirectory:remove(bool)

```

```

ofDirectory:allowExt(string)
ofDirectory:listDir(string)
ofDirectory:listDir()
ofDirectory:getName(unsigned int)
ofDirectory:getPath(unsigned int)
ofDirectory:getFile(unsigned int,OfFile)
ofDirectory:getFiles()
ofDirectory:getShowHidden()
ofDirectory:reset()
ofDirectory:sort()
ofDirectory:size()
ofDirectory:numFiles()
ofDirectory:createDirectory(string,bool,bool)
ofDirectory:isDirectoryEmpty(string,bool)
ofDirectory:doesDirectoryExist(string,bool)
ofDirectory:removeDirectory(string,bool,bool)
ofDirectory:getFile(unsigned int,OfFile)

```

Métodos de la clase `ofFileDialogResult()` para trabajar con archivos

```

ofFileDialogResult:getName()
ofFileDialogResult:getPath()

ofSystemAlertDialog(string)

```

Métodos de la clase `ofBuffer()` para trabajar con buffer de datos

```

ofBuffer(string)
ofBuffer(ofBuffer)
ofBuffer:set(string)
ofBuffer:set(istream)
ofBuffer:writeTo(ostream)
ofBuffer:clear()
ofBuffer:allocate(long)
ofBuffer:setBinaryBuffer(int,char)
ofBuffer:getBinaryBuffer(int)
ofBuffer:getBinaryBuffer()
ofBuffer:getText()
ofBuffer:size()
ofBuffer:getNextLine()
ofBuffer:getFirstLine()
ofBuffer:isLastLine()
ofBuffer:resetLineReader()

```

Métodos de la clase `ofThread()` para cargar imágenes de una cámara

```

ofThread()
ofThread:isThreadRunning()
ofThread:startThread(bool,bool)
ofThread:lock()
ofThread:unlock()
ofThread:stopThread(bool)
ofThread:waitForThread(bool)

```

Métodos de la clase `ofVideoPlayer()` para reproducir archivos de video

```

ofVideoPlayer()
ofVideoPlayer:loadMovie(string)
ofVideoPlayer:closeMovie()
ofVideoPlayer:close()
ofVideoPlayer:update()
ofVideoPlayer:idleMovie()
ofVideoPlayer:play()
ofVideoPlayer:stop()
ofVideoPlayer:isFrameNew()
ofVideoPlayer:getPixels()
ofVideoPlayer:getPosition()
ofVideoPlayer:getSpeed()
ofVideoPlayer:getDuration()
ofVideoPlayer:getIsMovieDone()
ofVideoPlayer:setPosition(float)
ofVideoPlayer:setVolume(int)
ofVideoPlayer:setLoopState(ofLoopType)
ofVideoPlayer:getLoopState()
ofVideoPlayer:setSpeed(float)
ofVideoPlayer:setFrame(int)
ofVideoPlayer:setUseTexture(bool)
ofVideoPlayer:getTextureReference()
ofVideoPlayer:draw(float,float)
ofVideoPlayer:draw(float,float,float,float)
ofVideoPlayer:draw(ofPoint)
ofVideoPlayer:draw(ofRectangle)
ofVideoPlayer:setAnchorPercent(float,float)
ofVideoPlayer:setAnchorPoint(float,float)
ofVideoPlayer:resetAnchor()
ofVideoPlayer:setPaused(bool)
ofVideoPlayer:getCurrentFrame()
ofVideoPlayer:getTotalNumFrames()

```

```
ofVideoPlayer:firstFrame()
ofVideoPlayer:nextFrame()
ofVideoPlayer:previousFrame()
ofVideoPlayer:getHeight()
ofVideoPlayer:getWidth()
ofVideoPlayer:isPaused()
ofVideoPlayer:isLoaded()
ofVideoPlayer:isPlaying()
```

Métodos de la clase `ofVideoGrabber()` para capturar desde cámara de video

```
ofVideoGrabber()
ofVideoGrabber:isFrameNew()
ofVideoGrabber:update()
ofVideoGrabber:close()
ofVideoGrabber:initGrabber(int,int)
ofVideoGrabber:setPixelFormat(ofPixelFormat)
ofVideoGrabber:getPixelFormat()
ofVideoGrabber:videoSettings()
ofVideoGrabber:getPixels()
ofVideoGrabber:getPixelsRef()
ofVideoGrabber:getTextureReference()
ofVideoGrabber:setVerbose(bool)
ofVideoGrabber:setDeviceID(int)
ofVideoGrabber:setDesiredFrameRate(int)
ofVideoGrabber:setUseTexture(bool)
ofVideoGrabber:draw(float,float,float,float)
ofVideoGrabber:draw(float,float)
ofVideoGrabber:setAnchorPercent(float,float)
ofVideoGrabber:setAnchorPoint(float,float)
ofVideoGrabber:resetAnchor()
ofVideoGrabber:getWidth()
ofVideoGrabber:getHeight()
ofVideoGrabber:isInitialized()
```

Métodos de la clase `ofSoundPlayer()` para reproducir archivos de sonido

```
ofSoundPlayer()
ofSoundUpdate()
ofSoundGetSpectrum(int)
ofSoundPlayer:loadSound(string,bool)
ofSoundPlayer:unloadSound()
ofSoundPlayer:play()
```

```
ofSoundPlayer:stop()
ofSoundPlayer:setVolume(float)
ofSoundPlayer:setPan(float)
ofSoundPlayer:setSpeed(float)
ofSoundPlayer:setPaused(bool)
ofSoundPlayer:setLoop(bool)
ofSoundPlayer:setMultiPlay(bool)
ofSoundPlayer:setPosition(float)
ofSoundPlayer:setPositionMS(int)
ofSoundPlayer:getPositionMS()
ofSoundPlayer:getPosition()
ofSoundPlayer:getIdPlaying()
ofSoundPlayer:getSpeed()
ofSoundPlayer:getPan()
ofSoundPlayer:getVolume()
ofSoundPlayer:isLoaded()
```

Métodos de la clase `ofTrueTypeFont()` para archivos de fuentes de texto

```
ofTrueTypeFont()
ofTrueTypeFont:loadFont(string,int,bool,bool)
ofTrueTypeFont:getLineHeight()
ofTrueTypeFont:setLineHeight(float)
ofTrueTypeFont:stringWidth(string)
ofTrueTypeFont:stringHeight(string)
ofTrueTypeFont:getStringBoundingBox(string,float,float)
ofTrueTypeFont:drawString(string,float,float)
ofTrueTypeFont:drawStringAsShapes(string,float,float)
ofTrueTypeFont:isLoaded()
ofTrueTypeFont:isAntiAliased()
ofTrueTypeFont:hasFullCharacterSet()
ofTrueTypeFont:getSize()
ofTrueTypeFont:getLetterSpacing()
ofTrueTypeFont:setLetterSpacing(float)
ofTrueTypeFont:getSpaceSize()
ofTrueTypeFont:setSpaceSize(float)
ofTrueTypeFont:getNumCharacters()
ofTrueTypeFont:getCharacterAsPoints(int)
ofTrueTypeFont:getStringAsPoints(string)
ofTrueTypeFont:bind()
ofTrueTypeFont:unbind()
```

Métodos de la clase `ofPolyline()` para combinar puntos en vectores

```
ofPolyline()
ofPolyline:clear()
ofPolyline:addVertex(float, float, float)
ofPolyline:addVertices(vector<ofPoint>)
ofPolyline:lineTo(ofPoint)
ofPolyline:lineTo(float, float, float)
ofPolyline:arc(ofPoint, float, float, float, int)
ofPolyline:arc(float, float, float, float, float, int)
ofPolyline:arc(float, float, float, float, float, float, int)
ofPolyline:curveTo(ofPoint, int)
ofPolyline:curveTo(float, float, float, int)
ofPolyline:bezierTo(ofPoint, ofPoint, ofPoint, int)
ofPolyline:bezierTo(float, float, float, float, float, float, int)
ofPolyline:bezierTo(float, float, float, float, float, float, float, float, int)
ofPolyline:quadBezierTo(ofPoint, ofPoint, ofPoint, int)
ofPolyline:quadBezierTo(float, float, float, float, float, float, int)
ofPolyline:quadBezierTo(float, float, float, float, float, float, float, float, int)
ofPolyline:getSmoothed(int, float)
ofPolyline:getResampledBySpacing(float)
ofPolyline:getResampledByCount(int)
ofPolyline:getBoundingBox()
ofPolyline:getClosestPoint
ofPolyline:simplify(float)
ofPolyline:size()
ofPolyline:resize(size_t)
ofPolyline:setClosed(bool)
ofPolyline:isClosed()
ofPolyline:close()
ofPolyline:hasChanged()
ofPolyline:getVertices()
ofPolyline:getPerimeter()
ofPolyline:draw()
```

Métodos de la clase `ofTessellator()` para convertir ofPolylines en ofMeshes

```
ofTessellator(ofTessellator)
```

Métodos de la clase `ofRectangle()` para trabajar con formas rectangulares

```
ofRectangle(float, float, float, float)
ofRectangle(ofRectangle)
ofRectangle:set(float, float, float, float)
ofRectangle:set(ofPoint, float, float)
ofRectangle:set(ofRectangle)
ofRectangle:setX(float)
ofRectangle:setY(float)
ofRectangle:setWidth(float)
ofRectangle:setHeight(float)
ofRectangle:setPosition(float, float)
ofRectangle:setPosition(ofPoint)
ofRectangle:setFromCenter(float, float, float, float)
ofRectangle:setFromCenter(ofPoint, float, float)
ofRectangle:translate(float, float)
ofRectangle:translate(ofPoint)
ofRectangle:translateX(float)
ofRectangle:translateY(float)
ofRectangle:scale(float)
ofRectangle:scale(float, float)
ofRectangle:scale(ofPoint)
ofRectangle:scaleWidth(float)
ofRectangle:scaleHeight(float)
ofRectangle:scaleFromCenter(float)
ofRectangle:scaleFromCenter(float, float)
ofRectangle:scaleFromCenter(ofPoint)
ofRectangle:scaleTo(ofRectangle, ofScaleMode)
ofRectangle:scaleTo(ofRectangle, ofAspectRatioMode, ofAlignHorz, ofAlignVert)
ofRectangle:scaleTo(ofRectangle, ofAspectRatioMode, ofAlignHorz, ofAlignVert, ofAlignHorz, ofAlignVert)
ofRectangle:alignToHorz(float, ofAlignHorz)
ofRectangle:alignToHorz(ofRectangle, ofAlignHorz)
ofRectangle:alignToHorz(ofRectangle, ofAlignHorz, ofAlignHorz)
ofRectangle:alignToVert(float, ofAlignVert)
ofRectangle:alignToVert(ofRectangle, ofAlignVert)
ofRectangle:alignToVert(ofRectangle, ofAlignVert, ofAlignVert)
ofRectangle:alignTo(ofPoint, ofAlignHorz, ofAlignVert)
ofRectangle:alignTo(ofRectangle, ofAlignHorz, ofAlignVert)
ofRectangle:alignTo(ofRectangle, ofAlignHorz, ofAlignVert, ofAlignHorz, ofAlignVert)
ofRectangle:growToInclude(float, float)
ofRectangle:growToInclude(ofPoint)
ofRectangle:growToInclude(ofRectangle)
ofRectangle:growToInclude(ofPoint, ofPoint)
ofRectangle:getIntersection(ofRectangle)
ofRectangle:getUnion(ofRectangle)
```

```
ofRectangle:standardize()
ofRectangle:getStandardized()
ofRectangle:isStandardized()
ofRectangle:getArea()
ofRectangle:getPerimeter()
ofRectangle:getAspectRatio()
ofRectangle:isEmpty()
ofRectangle:getMin()
ofRectangle:getMax()
ofRectangle:getMinX()
ofRectangle:getMaxX()
ofRectangle:getMinY()
ofRectangle:getMaxY()
ofRectangle:getLeft()
ofRectangle:getRight()
ofRectangle:getTop()
ofRectangle:getBottom()
ofRectangle:getTopLeft()
ofRectangle:getTopRight()
ofRectangle:getBottomLeft()
ofRectangle:getBottomRight()
ofRectangle:getHorzAnchor(ofAlignHorz)
ofRectangle:getVertAnchor(ofAlignVert)
ofRectangle:getPosition()
ofRectangle:getCenter()
ofRectangle:getX()
ofRectangle:getY()
ofRectangle:getWidth()
ofRectangle:getHeight()
```

Métodos de la clase `ofHttpRequest()` para trabajar con la Web

```
ofHttpRequest()
ofHttpRequest(string,string,bool)
ofHttpRequest::getID()
```

Métodos de la clase `ofURLFileLoader()` para cargar archivos URL

```
ofURLFileLoader()
ofURLFileLoader::get(string)
ofURLFileLoader::getAsync(string,string)
ofURLFileLoader::saveTo(string,string)
ofURLFileLoader::saveAsync(string,string)
```

```
ofURLFileLoader::remove(int)
ofURLFileLoader::clear()
```

Métodos de la clase `ofSerial()` para trabajar con puerto serial y dispositivos

```
ofSerial()
ofSerial::listDevices()
ofSerial::close()
ofSerial::setup(string,int)
ofSerial::setup(int,int)
ofSerial::readBytes(unsigned char*,int)
ofSerial::writeBytes(unsigned char*,int)
ofSerial::writeByte(unsigned char)
ofSerial::readByte()
ofSerial::flush(bool,bool)
ofSerial::available()
ofSerial::drain()
```

16.2. Integración de OpenGL⁹⁸

Se listan a continuación las funciones de OpenGL que están integradas en GAMUZA, asociando en sus parámetros las variables de sistema de OpenGL que les corresponde.

```
void glBegin(mode) || mode = GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP, GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_QUADS, GL_QUAD_STRIP o GL_POLYGON
void glEnd()
void glVertex2d(x, y)
void glVertex3d(x, y, z)
void glRectd(x1, y1, x2, y2)
void glEdgeFlag(flag) || GL_TRUE o GL_FALSE
void glRotated(angle, x, y, z) // el ángulo en grados
void glTranslated(x, y, z)
void glScaled(x, y, z)
void glMultMatrixd(m) // m = puntos para 16 valores consecutivos que se utilizan como elementos de una matriz 4 x 4.
void glFrustum(left,right,bottom,top,nearVal,farVal)
void glOrtho(left,right,bottom,top,nearVal,farVal)
void glLoadMatrixd(m)
void glLoadIdentity()
void glMatrixMode(mode) || GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE, GL_COLOR
void glPushMatrix()
void glPopMatrix()
void glDepthRange(nearVal,farVal)
void glViewport(x, y, w, h)
void glColor3d(R, G, B)
void glColor4d(R, G, B, alpha)
void glIndexd(c) // c especifica el nuevo valor para el index de color actual
void glNormal3d(x, y, z)
void glLightf(light,pname,param) || light = GL_LIGHTi, i puede ir de 0 al valor de GL_MAX_LIGHTS - 1 || pname puede ser GL_SPOT_EXPONENT,GL_SPOT_CUTOFF, GL_CONSTANT_ATTENUATION,GL_LINEAR_ATTENUATION o GL_QUADRATIC_ATTENUATION
void glLightfv(light,pname,param)
void glMaterialf(face,pname,param) || face puede ser GL_FRONT, GL_BACK, o GL_FRONT_AND_BACK || pname = GL_SHININESS
void glMaterialfv(face,pname,param) || pname = GL_AMBIENT,GL_DIFFUSE,GL_SPECULAR,GL_EMISSION,GL_SHININESS,GL_AMBIENT_AND_DIFFUSE o GL_COLOR_INDEXES
void glLightModelf(pname,param) || pname = GL_LIGHT_MODEL_LOCAL_VIEWER, GL_LIGHT_MODEL_COLOR_CONTROL o GL_LIGHT_MODEL_TWO_SIDE
void glShadeModel(mode) || mode = GL_FLAT o GL_SMOOTH
void glFrontFace(mode) || mode = GL_CW o GL_CCW
```

```
void glColorMaterial(face,mode) || face = GL_FRONT, GL_BACK, o GL_FRONT_AND_BACK || mode = GL_EMISSION,GL_AMBIENT,GL_DIFFUSE,GL_SPECULAR o GL_AMBIENT_AND_DIFFUSE
void glGetLightfv(light,pname,param)
void glGetMaterialfv(face,pname,param)
void glClipPlane(plane, planeEquation) || GL_CLIP_PLANEi, i = valor entre 0 y GL_MAX_CLIP_PLANES-1 || planeEquation= dirección de un array
void glGetClipPlane(plane, planeEquation)
void glRasterPos2d(x, y)
void glRasterPos3d(x, y, z)
void glRasterPos4d(x, y, z, w)
void glBitmap(width, height, xorig, yorig, xmove, ymove, ruta/bitmap)
void glPointSize(size)
void glLineWidth(width)
void glLineStipple(factor, pattern)
void glPolygonStipple(pattern)
void glGetPolygonStipple(pattern)
void glCullFace(mode) || mode= GL_FRONT, GL_BACK, o GL_FRONT_AND_BACK
void glPolygonMode(face, mode) || face = GL_FRONT, GL_BACK, o GL_FRONT_AND_BACK || mode= GL_POINT,GL_LINE o GL_FILL
void glReadBuffer(mode) || mode= GL_FRONT_LEFT,GL_FRONT_RIGHT,GL_BACK_LEFT,GL_BACK_RIGHT,GL_FRONT,GL_BACK,GL_LEFT, GL_RIGHT o GL_AUXi, i= un valor entre 0 y GL_AUX_BUFFERS-1
void glReadPixels(x, y, width, height, format, type, data) || format= GL_COLOR_INDEX,GL_STENCIL_INDEX,GL_DEPTH_COMPONENT,GL_RED,GL_GREEN,GL_BLUE,GL_ALPHA,GL_RGB,GL_BGR,GL_RGBA,GL_BGRA,GL_LUMINANCE o GL_LUMINANCE_ALPHA || type=GL_UNSIGNED_BYTE,GL_BYTE,GL_BITMAP,GL_UNSIGNED_SHORT,GL_SHORT,GL_UNSIGNED_INT, GL_INT,GL_FLOAT, GL_UNSIGNED_BYTE_3_3_2, GL_UNSIGNED_BYTE_2_3_3_REV, GL_UNSIGNED_SHORT_5_6_5, GL_UNSIGNED_SHORT_5_6_5_REV, GL_UNSIGNED_SHORT_4_4_4, GL_UNSIGNED_SHORT_4_4_4_REV, GL_UNSIGNED_SHORT_5_5_1, GL_UNSIGNED_SHORT_1_5_5_5_REV, GL_UNSIGNED_INT_8_8_8_8, GL_UNSIGNED_INT_8_8_8_REV, GL_UNSIGNED_INT_10_10_10_2, o GL_UNSIGNED_INT_2_10_10_10_REV
void glDrawPixels(width, height, format, type, data)
void glCopyPixels(x, y, width, height, format, type) || type= GL_COLOR,GL_DEPTH o GL_STENCIL
void glPixelStorei(pname,param) || pname = GL_PACK_SWAP_BYTES, GL_PACK_LSB_FIRST, GL_PACK_ROW_LENGTH, GL_PACK_IMAGE_HEIGHT, GL_PACK_SKIP_PIXELS, GL_PACK_SKIP_ROWS, GL_PACK_SKIP_IMAGES o GL_PACK_ALIGNMENT. U otros 6 para unpacking pixel data de la memoria: GL_UNPACK_SWAP_BYT, GL_UNPACK_LSB_FIRST, GL_UNPACK_ROW_LENGTH, GL_UNPACK_IMAGE_HEIGHT, GL_UNPACK_SKIP_PIXELS, GL_UNPACK_SKIP_ROWS, GL_UNPACK_SKIP_IMAGES, o GL_UNPACK_ALIGNMENT.
void glPixelStoref(pname,param)
void glPixelTransferf(pname,param) || pname = GL_MAP_COLOR, GL_MAP_STENCIL, GL_INDEX_SHIFT, GL_INDEX_OFFSET, GL_RED_SCALE, GL_RED_BIAS, GL_GREEN_SCALE, GL
```

98 Ver documentación completa en <http://www.opengl.org/sdk/docs/man2/> [20.08.2013]

```

GREEN_BIAS, GL_BLUE_SCALE, GL_BLUE_BIAS, GL_ALPHA_SCALE, GL_ALPHA_BIAS, GL_
DEPTH_SCALE o GL_DEPTH_BIAS
void glPixelMapfv(map,mapsize,value) || map = GL_PIXEL_MAP_I_TO_I, GL_PIXEL_MAP_S_
TO_S, GL_PIXEL_MAP_I_TO_R, GL_PIXEL_MAP_I_TO_G, GL_PIXEL_MAP_I_TO_B, GL_
PIXEL_MAP_I_TO_A, GL_PIXEL_MAP_R_TO_R, GL_PIXEL_MAP_G_TO_G, GL_PIXEL_MAP_B_
TO_B o GL_PIXEL_MAP_A_TO_A
void glGetPixelMapfv(map,data)
void glPixelZoom(xfactor,yfactor)
void glTexParameterf(target, pname, param) || target = GL_TEXTURE_1D, GL_TEXTURE_2D,
GL_TEXTURE_3D o GL_TEXTURE_CUBE_MAP || pname = GL_TEXTURE_MIN_FILTER, GL_
TEXTURE_MAG_FILTER, GL_TEXTURE_MIN_LOD, GL_TEXTURE_MAX_LOD, GL_TEXTURE_
BASE_LEVEL, GL_TEXTURE_MAX_LEVEL, GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T,
GL_TEXTURE_WRAP_R, GL_TEXTURE_PRIORITY, GL_TEXTURE_COMPARE_MODE, GL_TEXTURE_
COMPARE_FUNC, GL_DEPTH_TEXTURE_MODE o GL_GENERATE_MIPMAP.
void glTexEnvf(target, pname, param) || target = GL_TEXTURE_ENV, GL_TEXTURE_FILTER_
CONTROL o GL_POINT_SPRITE || pname = GL_TEXTURE_ENV_MODE, GL_TEXTURE_LOD_BIAS,
GL_COMBINE_RGB, GL_COMBINE_ALPHA, GL_SRC0_RGB, GL_SRC1_RGB, GL_SRC2_RGB, GL_
SRC0_ALPHA, GL_SRC1_ALPHA, GL_SRC2_ALPHA, GL_OPERAND0_RGB, GL_OPERAND1_RGB,
GL_OPERAND2_RGB, GL_OPERAND0_ALPHA, GL_OPERAND1_ALPHA, GL_OPERAND2_ALPHA,
GL_RGB_SCALE, GL_ALPHA_SCALE o GL_COORD_REPLACE
void glTexCoord1d(s)
void glTexCoord2d(s,t)
void glTexCoord3d(s,t,r)
void glTexCoord4d(s,t,r,q) // s, t, r, q = coordenadas de la textura
void glTexGend(coord, pname, param) || coord = GL_S, GL_T, GL_R o GL_Q
|| pname = GL_TEXTURE_GEN_MODE || param = GL_OBJECT_LINEAR, GL_EYE_LINEAR,
GL_SPHERE_MAP, GL_NORMAL_MAP, or GL_REFLECTION_MAP
void glTexImage1D(target, level, internalFormat, width, border, format, type, data)
|| target = GL_TEXTURE_1D o GL_PROXY_TEXTURE_1D || internalFormat = nº de
componentes de color en la textura: 1, 2, 3, 4, o las constantes de color de
openGL || format y type = mismas opciones que glReadPixels
void glTexImage2D(target, level, internalFormat, width, border, format, type, data) || 
target = GL_TEXTURE_2D, GL_PROXY_TEXTURE_2D, GL_TEXTURE_CUBE_MAP_POSITIVE_X,
GL_TEXTURE_CUBE_MAP_NEGATIVE_X, GL_TEXTURE_CUBE_MAP_POSITIVE_Y, GL_TEXTURE_
CUBE_MAP_NEGATIVE_Y, GL_TEXTURE_CUBE_MAP_POSITIVE_Z, GL_TEXTURE_CUBE_MAP_
NEGATIVE_Z o GL_PROXY_TEXTURE_CUBE_MAP
void glGetTexEnvfv(target, pname, params) || target = GL_TEXTURE_ENV, GL_TEXTURE_
FILTER_CONTROL o GL_POINT_SPRITE || pname = GL_TEXTURE_ENV_MODE, GL_TEXTURE_
ENV_COLOR o GL_TEXTURE_LOD_BIAS, GL_COMBINE_RGB, GL_COMBINE_ALPHA, GL_SRC0_
RGB, GL_SRC1_RGB, GL_SRC2_RGB, GL_SRC0_ALPHA, GL_SRC1_ALPHA, GL_SRC2_ALPHA,
GL_OPERAND0_RGB, GL_OPERAND1_RGB, GL_OPERAND2_RGB, GL_OPERAND0_ALPHA, GL_
OPERAND1_ALPHA, GL_OPERAND2_ALPHA, GL_RGB_SCALE, GL_ALPHA_SCALE o GL_COORD_
REPLACE
void glGetTexGenfv (coord, pname, params) || coord = GL_S, GL_T, GL_R o GL_Q || 
pname= GL_TEXTURE_GEN_MODE, GL_OBJECT_PLANE o GL_EYE_PLANE

```

```

void glGetTexImage(target, level, format, type, img) || target = GL_TEXTURE_1D,
GL_TEXTURE_2D, GL_TEXTURE_3D, GL_TEXTURE_CUBE_MAP_POSITIVE_X, GL_TEXTURE_
CUBE_MAP_NEGATIVE_X, GL_TEXTURE_CUBE_MAP_POSITIVE_Y, GL_TEXTURE_CUBE_MAP_
NEGATIVE_Y, GL_TEXTURE_CUBE_MAP_POSITIVE_Z o GL_TEXTURE_CUBE_MAP_NEGATIVE_Z
|| format = GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA, GL_RGB, GL_BGR, GL_RGBA, GL_
BGRA, GL_LUMINANCE o GL_LUMINANCE_ALPHA
void glGetTexLevelParameterfv(target, level, pname, params) || taget = a la anterior
|| pname = GL_TEXTURE_WIDTH, GL_TEXTURE_HEIGHT, GL_TEXTURE_DEPTH, GL_TEXTURE_
INTERNAL_FORMAT, GL_TEXTURE_BORDER, GL_TEXTURE_RED_SIZE, GL_TEXTURE_GREEN_
SIZE, GL_TEXTURE_BLUE_SIZE, GL_TEXTURE_ALPHA_SIZE, GL_TEXTURE_LUMINANCE_SIZE,
GL_TEXTURE_INTENSITY_SIZE, GL_TEXTURE_DEPTH_SIZE, GL_TEXTURE_COMPRESSED o GL_
TEXTURE_COMPRESSED_IMAGE_SIZE
void glGetTexParameterfv(target, pname, params) || taget = GL_TEXTURE_1D, GL_
TEXTURE_2D, GL_TEXTURE_3D o GL_TEXTURE_CUBE_MAP || pname = GL_TEXTURE_MAG_
FILTER, GL_TEXTURE_MIN_FILTER, GL_TEXTURE_MIN_LOD, GL_TEXTURE_MAX_LOD, GL_
TEXTURE_BASE_LEVEL, GL_TEXTURE_MAX_LEVEL, GL_TEXTURE_WRAP_S, GL_TEXTURE_
WRAP_T, GL_TEXTURE_WRAP_R, GL_TEXTURE_BORDER_COLOR, GL_TEXTURE_PRIORITY, GL_
TEXTURE_RESIDENT, GL_TEXTURE_COMPARE_MODE, GL_TEXTURE_COMPARE_FUNC, GL_DEPTH_
TEXTURE_MODE o GL_GENERATE_MIPMAP
void glGenTextures(n, textures) // n = núm texturas // textures = array con los
nombres de las texturas
void glBindTexture(target, texture) || target = GL_TEXTURE_1D, GL_TEXTURE_2D, GL_
TEXTURE_3D o GL_TEXTURE_CUBE_MAP || texture = nombre de una textura
void glDeleteTextures(n, textures)
void glFogf(pname, param) || pname = GL_FOG_MODE, GL_FOG_DENSITY, GL_FOG_START,
GL_FOG_END, GL_FOG_INDEX o GL_FOG_COORD_SRC
void glScissor(x, y, width, height)
void glAlphaFunc(func, ref) || func = GL_NEVER, GL_LESS, GL_EQUAL, GL_LEQUAL, GL_
GREATER, GL_NOTEQUAL, GL_GEQUAL o GL_ALWAYS || ref = entre 0 y 1, 0 es el más
bajo valor de alpha y 1 el más alto
void glStencilFunc(func, ref, mask)
void glStencilOp(sfail, dpfail, dppass) || sfail, dpfail y dppass aceptan GL_KEEP, GL_
ZERO, GL_REPLACE, GL_INCR, GL_INCR_WRAP, GL_DECR, GL_DECR_WRAP o GL_INVERT
void glDepthFunc(func)
void glBlendFunc(sfactor, factorDestino) || ambos parámetros aceptan GL_ZERO,
GL_ONE, GL_SRC_COLOR, GL_ONE_MINUS_SRC_COLOR, GL_DST_COLOR, GL_ONE_MINUS_DST_
COLOR, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA, GL_DST_ALPHA, GL_ONE_MINUS_DST_
ALPHA, GL_CONSTANT_COLOR, GL_ONE_MINUS_CONSTANT_COLOR, GL_CONSTANT_ALPHA, GL_
ONE_MINUS_CONSTANT_ALPHA o GL_SRC_ALPHA_SATURATE
void glLogicOp(opcode) || opcode = GL_CLEAR, GL_SET, GL_COPY, GL_COPY_INVERTED,
GL_NOOP, GL_INVERT, GL_AND, GL_NAND, GL_OR, GL_NOR, GL_XOR, GL_EQUIV, GL_AND_
REVERSE, GL_AND_INVERTED, GL_OR_REVERSE o GL_OR_INVERTED
void glClear(mask) || mask = GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT, GL_ACCUM_
BUFFER_BIT o GL_STENCIL_BUFFER_BIT.
void glClearAccum(red, green, blue, alpha)

```

```

void glClearColor(red, green, blue, alpha)
void glClearDepth(depth) // depth es el valor de profundidad usado cuando se borra el
    búfer de profundidad. El valor inicial es 1.
void glClearIndex(s) // s es el index usado cuando se borra el búfer de color index.
    El valor inicial es 0.
void glClearStencil(s)
void glDrawBuffer(mode) || mode = GL_NONE, GL_FRONT_LEFT, GL_FRONT_RIGHT, GL_BACK_
    LEFT, GL_BACK_RIGHT, GL_FRONT, GL_BACK, GL_LEFT, GL_RIGHT, GL_FRONT_AND_BACK
    o GL_AUXi, donde i es un valor entre 0 y GL_AUX_BUFFERS-1
void glIndexMask(mask)
void glColorMask(red, green, blue, alpha)
void glDepthMask(flag)
void glStencilMask(mask)
void glAccum(op, value) || op = GL_ACCUM, GL_LOAD, GL_ADD, GL_MULT o GL_RETURN
void glRenderMode(mode) || mode = GL_RENDER, GL_SELECT o GL_FEEDBACK
void glSelectBuffer(size, buffer)
void glFeedbackBuffer(size, type, buffer) || type = GL_2D, GL_3D, GL_3D_COLOR,
    GL_3D_COLOR_TEXTURE o GL_4D_COLOR_TEXTURE
void glPassThrough(token) || token vinculado a GL_PASS_THROUGH_TOKEN
void glInitNames()
void glLoadName(name)
void glPushName(name)
void glPopName(name)
void glNewList(list, mode) || mode = GL_COMPILE o GL_COMPILE_AND_EXECUTE
void glEndList()
void glDeleteLists(list, range)
void glCallList(list)
void glCallLists(n, type, lists) || type = GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT,
    GL_UNSIGNED_SHORT, GL_INT, GL_UNSIGNED_INT, GL_FLOAT, GL_2_BYTES, GL_3_BYTES
    o GL_4_BYTES
void glGenLists(range)
void glIsList(list)
void glListBase(base)
void glEnable(cap) || cap puede ser alguna de las constantes de la capacidad de GL
void glDisable(cap)
void gl.IsEnabled(cap)
void glFinish()
void glFlush()
void glHint(target, mode) || target = GL_FOG_HINT, GL_GENERATE_MIPMAP_HINT, GL_
    LINE_SMOOTH_HINT, GL_PERSPECTIVE_CORRECTION_HINT, GL_POINT_SMOOTH_HINT,
    GL_POLYGON_SMOOTH_HINT, GL_TEXTURE_COMPRESSION_HINT o GL_FRAGMENT_SHADER_
    DERIVATIVE_HINT || mode = GL_FASTEST, GL_NICEST o GL_DONT_CARE
void glGetError()
void glGetString(name) || name = GL_VENDOR, GL_RENDERER, GL_VERSION, GL_SHADING_
    LANGUAGE_VERSION o GL_EXTENSIONS

```

```

void glGetBooleanv(pname, params)
void glGetDoublev(pname, params)
void glGetFloatv(pname, params)
void glGetIntegerv(pname, params)
void glPushAttrib(mask)
void glPopAttrib(mask)

```

16.3. Funciones de Lua

MATH

`math.abs(x)` Devuelve el valor absoluto de x
`math.acos(x)` Devuelve el arco coseno de x (en radianes)
`math.asin(x)` Devuelve el arco seno de x (en radianes)
`math.atan(x)` Devuelve el arco tangente de x (en radianes)
`math.atan2(y,x)` Devuelve el arco tangente de y/x (en radianes), pero usa los signos de ambos argumentos para determinar el cuadrante del resultado
`math.ceil(x)` Devuelve el menor entero mayor o igual que x.
`math.cos(x)` Devuelve el coseno de x (en radianes)
`math.deg(x)` Devuelve en grados sexagesimales el valor de x (dado en radianes)
`math.exp(x)` Devuelve el valor de e^x
`math.floor(x)` Devuelve el mayor entero menor o igual que x
`math.fmod(x, y)` Devuelve el resto de la división de x por y
`math.frexp(x)` Devuelve m y e tales que $x = m \cdot 2^e$, e es un entero y el valor absoluto de m está en el intervalo [0.5, 1) (o cero cuando x es cero).
`math.ldexp(m,e)` Devuelve m $\cdot 2^e$ (e debe ser un entero)
`math.log(x)` Devuelve el logaritmo natural de x
`math.log10(x)` Devuelve el logaritmo decimal (base 10) de x
`math.max(x, y,...)` Devuelve el mayor valor de entre sus argumentos
`math.min(x, y,...)` Devuelve el menor valor de entre sus argumentos
`math.pi()` El valor de pi.
`math.pow(x)` Devuelve x^y . (Se puede también usar la expresión x^y para calcular este valor.)
`math.rad(x)` Devuelve en radianes el valor del ángulo x (dado en grados sexagesimales)
`math.random([m[,n]])` Esta función es un interface a `rand`, generador simple de números pseudo-aleatorios
`math.randomseed(x)` Establece x como "semilla" para el generador de números pseudoaleatorios: iguales semillas producen iguales secuencias de números
`math.sin(x)` Devuelve el seno de x (en radianes)
`math.sqrt(x)` Devuelve la raíz cuadrada de x. (Se puede usar también $x^{0.5}$ para calcular este valor.)
`math.tan(x)` Devuelve la tangente de x (en radianes)

STRINGS

`string.byte('char')` Devuelve el código numérico interno del carácter que se ponga como parámetro
`string.char(...)` Recibe cero o más enteros. Devuelve un string con igual longitud que el número de argumentos, en el que cada carácter tiene un código numérico interno igual a su correspondiente argumento.
`string.dump(function)` Devuelve un string con la representación binaria de la función dada
`string.find(string, patrón)` Busca la primera aparición de patrón en el string s
`string.format(formato,...)` Devuelve una versión formateada de sus argumentos. Por ejemplo:

```
text = string.format("Position X: %f", gaMouseX())
ofDrawBitmapString(text, x, y)
string.gmatch(string, patrón) Devuelve una función iteradora que, cada vez que se invoca, retorna las siguientes capturas del patrón en el string s
string.gsub(string, patrón, reemplazar) Devuelve una copia de s en la que todas las apariciones del patrón han sido reemplazadas por el dato especificado
string.len(string) Recibe un string y devuelve su longitud
string.lower(string) Recibe un string y devuelve una copia del mismo con todas las letras mayúsculas cambiadas a minúsculas
string.match(string, patrón) Busca la primera aparición del patrón en el string s
string.rep(string, n) Devuelve un string que es la concatenación de n copias del string s
string.reverse(string) Devuelve un string que es el original s invertido
string.sub(string,1,j) Devuelve un prefijo de s con longitud j. Si fuera string.sub(string, -i) devuelve un sufijo de s con longitud i
string.upper(string) Recibe un string y devuelve una copia del mismo con todas las letras minúsculas cambiadas a mayúsculas.
```

TABLAS

```
table.concat (tabla [, separador [, i [, j]]]) Dado una tabla donde todos sus elementos son strings o números devuelve: tabla[i]..separador..tabla[i+1] ... separador..tabla[j].
table.foreach(myTable, myFunction) Recorre cada índice de la tabla y le asigna la función.
table.getn(myTable) Para saber el número de elementos de una tabla
table.sort(myTable [,comparador]) Reordena los elementos de la tabla en un orden dado modificando la propia tabla
table.insert(myTable, [position], value) Añade un valor al final de la tabla. Si se señala una posición exacta, para colocar el valor los índices siguientes se desplazan en consecuencia. // ejemplo (myTable, 5, true)
table.remove(myTable,[pos]) Elimina un elemento de la tabla y mueve hacia arriba los índices restantes si es necesario. Si no se asigna posición borra el último elemento. // ejemplo (myTable, 3)
```

UTILS

```
os.clock() Devuelve una aproximación al total de segundos de CPU usados por el programa.
os.date([formato [, tiempo]]) Devuelve un string o una tabla conteniendo la fecha y hora, formateada de acuerdo con el string dado en formato. Si no lleva parámetros, date devuelve una representación razonable de la fecha y hora del ordenador y el sistema local (os.date() equivale a os.date("%c")).
os.difftime(t1, t2) Devuelve el número de segundos desde el instante t1 hasta el t2.
```

`os.execute([comando])` Pasa la orden comando para que sea ejecutada en el intérprete de comandos del sistema operativo.

`os.exit([código])` Invoca la función exit de C, con un código entero opcional, para terminar el programa.

`os.getenv(variable)` Devuelve el valor asignado a la variable del sistema, o `nil` si esa variable no está definida.

`os.remove(nombre_fichero)` Elimina el fichero o directorio dado. Los directorios deben estar vacíos para poder ser eliminados.

`os.rename(nombre_viejo, nombre_nuevo)` Renombra un fichero o directorio.

`os.setlocale(local [categoría])` Establece valores en el sistema local del programa. `local` es un string que especifica un valor local; `categoría` es un string opcional que describe qué categoría cambiar: `"all"`, `"collate"`, `"ctype"`, `"monetary"`, `"numeric"`, o `"time"`; la categoría por defecto es `"all"`. Esta función devuelve el nombre del nuevo local o `nil` si la petición no pudo ser aceptada.

`os.time([tabla])` Devuelve el tiempo actual cuando se llama sin parámetros, o un tiempo representando la fecha y hora especificadas en la tabla dada.

`os.tmpname()` Devuelve un `string` con un nombre de fichero que puede ser usado como fichero temporal.

16.4. Framework de GAMUZA

CAMERA TRACKING

```
class gaCameraTracking()
    Clase de GAMUZA para el panel Tracking video
void gaCameraTracking:setup(int camID,int captureWidth,int captureHeight)
    Inicializa el panel de tracking con los datos de la cámara y tamaño de captura.
void gaCameraTracking:update()
    Actualiza los datos del panel.
void gaCameraTracking:draw()
    Dibuja el panel en la ventana de salida.
void gaCameraTracking:setHaarFile(string gaImportFile("fileHaar.xml"))
    Carga un archivo con el tipo de Haar a utilizar.
void gaCameraTracking:setMovieFile(string gaImportFile("video.mov"))
    Carga un archivo de video pregrabado para trackear.
void gaCameraTracking:setGuiSettingsFile(string gaDataPath("camTrackingSettings.xml"))
    Guarda en la carpeta data la configuración del panel en un archivo XML.
void gaCameraTracking:mousePressed(int gaMouseX(),int gaMouseY())
    Permite ajustar los datos de la interfaz del panel con el ratón.
void gaCameraTracking:mouseDragged(int gaMouseX(),int gaMouseY())
    Permite ajustar los datos de la interfaz del panel con el ratón.
void gaCameraTracking:mouseReleased(int gaMouseX(),int gaMouseY())
    Permite ajustar los datos de la interfaz del panel con el ratón.
ofTexture gaCameraTracking:getCameraTexture()
    Coloca la imagen capturada por la cámara sobre una textura de ofTexture
ofTexture gaCameraTracking:getCameraTextureMod()
    Coloca la imagen capturada por la cámara sobre una textura de ofTexture con el modo previamente especificado.
ofPixelsRef gaCameraTracking:getCameraPixels()
    Coloca la imagen capturada por la cámara sobre una superficie generada con ofPixelsRef.
void gaCameraTracking:captureBackground()
    Captura la imagen del fondo para el algoritmo Backgroung Difference.
float gaCameraTracking:getMotionQ()
    Devuelve la cantidad de movimiento que registra la cámara expresado entre 0.0 y 1.0
float gaCameraTracking:getMotionX()
    Devuelve la coordenada X del punto medio de la masa de movimiento.
float gaCameraTracking:getMotionY()
    Devuelve la coordenada Y del punto medio de la masa de movimiento.
int gaCameraTracking:getNumBlobs()
    Devuelve el número de blobs detectados.
float gaCameraTracking:getBlobX(int blobID)
    Devuelve la coordenada X del blob especificado en su parámetro.
```

```

float gaCameraTracking:getBlobY(int blobID)
    Devuelve la coordenada Y del blob especificado en su parámetro.

float gaCameraTracking:getBlobW(int blobID)
    Devuelve la anchura del blob especificado en su parámetro.

float gaCameraTracking:getBlobH(int blobID)
    Devuelve la altura del blob especificado en su parámetro.

float gaCameraTracking:getBlobAngle(int blobID)
    Devuelve el ángulo del blob especificado en su parámetro.

int gaCameraTracking:getBlobContourSize(int blobID)
    Devuelve el tamaño del contorno del blob especificado en su parámetro.

float gaCameraTracking:getBlobCPointX(int blobID, int pointContourID)
    Devuelve la coordenada X del punto del contorno y del blob especificados en sus parámetros.

float gaCameraTracking:getBlobCPointY(int blobID, int pointContourID)
    Devuelve la coordenada Y del punto del contorno y del blob especificados en sus parámetros.

int gaCameraTracking:getBlobGeometrySize(int blobID)
    Devuelve el tamaño de la geometría del blob especificado en su parámetro.

float gaCameraTracking:getBlobGLineX1(int blobID, int lineID)
    Devuelve la coordenada X1 de la línea geométrica y el blob especificados en sus parámetros.

float gaCameraTracking:getBlobGLineY1(int blobID, int lineID)
    Devuelve la coordenada Y1 de la línea geométrica y el blob especificados en sus parámetros.

float gaCameraTracking:getBlobGLineX2(int blobID, int lineID)
    Devuelve la coordenada X2 de la línea geométrica y el blob especificados en sus parámetros.

float gaCameraTracking:getBlobGLineY2(int blobID, int lineID)
    Devuelve la coordenada Y2 de la línea geométrica y el blob especificados en sus parámetros.

float gaCameraTracking:getOpticalFlowX(int indexGridPosition)
    Devuelve la coordenada X de la posición del grid optical Flow especificado en su parámetro.

float gaCameraTracking:getOpticalFlowY(int indexGridPosition)
    Devuelve la coordenada Y de la posición del grid optical Flow especificado en su parámetro.

float gaCameraTracking:getOpticalFlowVX(int indexGridPosition)
    Devuelve la coordenada de velocidad VX de la posición del grid optical Flow especificado en su parámetro.

float gaCameraTracking:getOpticalFlowVY(int indexGridPosition)
    Devuelve la coordenada de velocidad VY de la posición del grid optical Flow especificado en su parámetro.

int gaCameraTracking:getNumHaars()
    Devuelve el número de Haars detectados.

float gaCameraTracking:getHaarX(int haarID)
    Devuelve la coordenada X del Haar especificado en su parámetro.

float gaCameraTracking:getHaarY(int haarID)
    Devuelve la coordenada Y del Haar especificado en su parámetro.

float gaCameraTracking:getHaarW(int haarID)
    Devuelve la anchura del Haar especificado en su parámetro.

float gaCameraTracking:getHaarH(int haarID)
    Devuelve la altura del Haar especificado en su parámetro.

```

bool gaCameraTracking:getTrigger(int triggerID)	Devuelve si está activo o no el Trigger especificado en su parámetro.
Constantes	
MAX_BLOBS	Constante que indica el máximo número de blobs. Se utiliza como límite en el recorrido en loop sobre todos los blobs detectado.
OPTICAL_FLOW_COLS_STEP	Constante que indica el número de columnas de la matriz utilizada en el algoritmo Optical Flow.
OPTICAL_FLOW_ROWS_STEP	Constante que indica el número de filas de la matriz utilizada en el algoritmo Optical Flow.
KINECT SENSOR TRACKING	
class gaKinectTracking()	Clase de GAmuza para el panel Tracking por sensor Kinect
void gaKinectTracking:setup(int IDsensor, bool IR, bool video, int LedMode)	Inicializa el panel del sensor Kinect con los datos del ID del sensor, si se activa o no el sistema infrarrojos, si se activa o no la imagen video y modo del led (0: LED_OFF, 1: LED_GREEN, 2: LED_RED, 3: LED_YELLOW, 4: LED_BLINK_GREEN y 6: LED_BLINK_YELLOW_RED)
void gaKinectTracking:update()	Actualiza los datos del panel.
void gaKinectTracking:draw()	Dibuja el panel en la ventana de salida.
void gaKinectTracking:close()	Desactiva el sensor Kinect.
void gaKinectTracking:setGuiSettingsFile(string gaDataPath("kinectSettings.xml"))	Guarda en la carpeta data la configuración del panel en un archivo XML.
void gaKinectTracking:mousePressed(int gaMouseX(), int gaMouseY())	Permite ajustar los datos de la interfaz del panel con el ratón.
void gaKinectTracking:mouseDragged(int gaMouseX(), int gaMouseY())	Permite ajustar los datos de la interfaz del panel con el ratón.
void gaKinectTracking:mouseReleased(int gaMouseX(), int gaMouseY())	Permite ajustar los datos de la interfaz del panel con el ratón.
ofTexture gaKinectTracking:getCameraTexture()	Coloca las formas capturadas por el sensor sobre una textura de ofTexture
ofTexture gaKinectTracking:getDepthTexture()	Devuelve la profundidad de la textura openGL.
ofPixelsRef gaKinectTracking:getCameraPixels()	Coloca las formas capturadas por el sensor sobre una superficie generada con ofPixelsRef.

```

int gaKinectTracking:getNumBlobs()
    Devuelve el número de blobs detectados.

float gaKinectTracking:getBlobX(int blobID)
    Devuelve la coordenada X del blob especificado en su parámetro.

float gaKinectTracking:getBlobY(int blobID)
    Devuelve la coordenada Y del blob especificado en su parámetro.

float gaKinectTracking:getBlobW(int blobID)
    Devuelve la anchura del blob especificado en su parámetro.

float gaKinectTracking:getBlobH(int blobID)
    Devuelve la altura del blob especificado en su parámetro.

float gaKinectTracking:getBlobAngle(int blobID)
    Devuelve el ángulo del blob especificado en su parámetro.

int gaKinectTracking:getBlobContourSize(int blobID)
    Devuelve el tamaño del contorno del blob especificado en su parámetro.

float gaKinectTracking:getBlobCPointX(int blobID, int pointContourID)
    Devuelve la coordenada X del punto del contorno y del blob especificados en sus parámetros.

float gaKinectTracking:getBlobCPointY(int blobID, int pointContourID)
    Devuelve la coordenada Y del punto del contorno y del blob especificados en sus parámetros.

int gaKinectTracking:getBlobGeometrySize(int blobID)
    Devuelve el tamaño de la geometría del blob especificado en su parámetro.

float gaKinectTracking:getBlobGLineX1(int blobID, int lineID)
    Devuelve la coordenada X1 de la línea geométrica y el blob especificados en sus parámetros.

float gaKinectTracking:getBlobGLineY1(int blobID, int lineID)
    Devuelve la coordenada Y1 de la línea geométrica y el blob especificados en sus parámetros.

float gaKinectTracking:getBlobGLineX2(int blobID, int lineID)
    Devuelve la coordenada X2 de la línea geométrica y el blob especificados en sus parámetros.

float gaKinectTracking:getBlobGLineY2(int blobID, int lineID)
    Devuelve la coordenada Y2 de la línea geométrica y el blob especificados en sus parámetros.

float gaKinectTracking:getOpticalFlowX(int indexGridPosition)
    Devuelve la coordenada X de la posición del grid optical Flow especificado en su parámetro.

float gaKinectTracking:getOpticalFlowY(int indexGridPosition)
    Devuelve la coordenada Y de la posición del grid optical Flow especificado en su parámetro.

float gaKinectTracking:getOpticalFlowVX(int indexGridPosition)
    Devuelve la coordenada VX de la posición del grid optical Flow especificado en su parámetro.

float gaKinectTracking:getOpticalFlowVY(int indexGridPosition)
    Devuelve la coordenada VY de la posición del grid optical Flow especificado en su parámetro.

bool gaKinectTracking:getTrigger(int triggerID)
    Devuelve si está activo o no el Trigger especificado en su parámetro.

float gaKinectTracking:getAccelX()
    Devuelve la coordenada de inclinación X del acelerómetro del sensor kinect.

float gaKinectTracking:getAccelY()
    Devuelve la coordenada de inclinación Y del acelerómetro del sensor kinect.

float gaKinectTracking:getAccelZ()
    Devuelve la coordenada de inclinación Z del acelerómetro del sensor kinect.

```

APPLICATION

```

void gaKey()
    Devuelve en Byte el valor ASCII de última la tecla presionada o liberada.

void gaMouseX()
    Devuelve la coordenada X de la posición del ratón.

void gaMouseY()
    Devuelve la coordenada Y de la posición del ratón.

void gaLog(string string.format("Data: %f",data))
    Muestra la información solicitada en la consola de GAMUZA. Ver en el apartado 4.1.1 los códigos para interpretar el tipo de datos, por ejemplo %f para float.

string gaImportFile(string "fileName.type")
    Da acceso directo a la carpeta data.

string gaDataPath(string "fileName.type")
    Da acceso directo a la carpeta data.

int gaGetFrameNum()
    Devuelve el número del frame.

void gaSystem(string "secuencias de comandos")
    Comunica con el sistema de c; puede abrir programas o ejecutar secuencias de comandos.

    ÚSELO BAJO SU PROPIO RIESGO

```

STRING

```

int gaStringReplace(string textSource, string "SearchString", string "replaceString")
    Sustituye un string determinado de un texto

```

GRAPHICS

```

void gaBackground(float grayscale, float alpha)
void gaBackground(float red, float green, float blue, float alpha)
    Dibuja el fondo con los parámetros especificados, todos sus valores tienen un rango entre 0.0 y 1.0

void gaSaveFrame(string gaDataPath("fileName.type"))
    Guarda un archivo de imagen de la ventana de salida

```

```

void gaSaveFrame(string gaDataPath("fileName.type"), ofTexture TextureName)
    Guarda en un archivo la imagen de la textura que se especifique.

```

```

ofPixels gaGetOutputPixels()
    Devuelve la referencia de la ventana de salida en formato ofPixels

ofTexture gaGetOutputTexture()
    Devuelve la ventana de salida de GAMUZA como una ofTexture

```

Constantes

```

OUTPUT_W Ancho de la ventana de salida
OUTPUT_H Alto de la ventana de salida

```

3D

```
void gaDrawAxis(float length, ofColor color, bool true or false)
    Dibuja los ejes de coordenadas con la longitud, color y activando la zona positiva y negativa
    o solo la positiva según los datos especificados en los parámetros.

void gaDrawXAxis(float length, ofColor color, bool true or false)
void gaDrawYAxis(float length, ofColor color, bool true or false)
void gaDrawZAxis(float length, ofColor color, bool true or false)
    Dibuja el eje de coordenadas Z con la longitud, color y activando la zona positiva y negativa
    o solo la positiva según los datos especificados en los parámetros.
```

MATH

```
float gaGaussianFn(float position, float amplitude, float center, float width)
    Para calcular ecuaciones gaussianas
```

TIMELINE

```
void gaTimelineSetup(string gaDataPath("folder/"), string "timelineName")
    Inicializa el Módulo Timeline y guarda en la carpeta especificada, situada dentro de la
    carpeta data, el fichero de configuración en formato XML

void gaTimelineSetPageName(string "pageName")
    Para cambiar el nombre a una página de Timeline

void gaTimelineSetCurrentPage(string "pageName")
    Para ir a la página de Timeline especificada

void gaTimelinePlay()
void gaTimelineStop()
void gaTimelineDurationInFrames(int numFrames) Duración del Timeline en frames
void gaTimelineDurationInSeconds(int numSeconds) Duración en segundos del Timeline.
void gaTimelineEnableKeySnap(bool true or false)
    Activar o no el autoajuste del ratón a los eventos.

void gaTimelineSetLoopType(CONSTANT_OF_LOOP_TYPE)
    Para establecer el tipo de loop en la lectura del Timeline: OF_LOOP_NORMAL o OF_LOOP_
    PALINDROME.

void gaTimelineSetFrameRate(int numFrameRate)
    Para establecer el número de Frames por segundo

void gaTimelineSetBPM(int numBPM)
    Para establecer el número de Beats por minuto

void gaTimelineEnableBPMsnap(bool true or false)
    Para activar o no el autoajuste a las guías de BPM

void gaTimelineShowBPMGrid(bool true or false)
    Para mostrar o no las guías de BPM
```

```
void gaTimelineSetTimecontrolTrack(string "timecontrolTrackName")
    Para incorporar un track de control

void gaTimelineBringTrackToTop(string "trackToTopName")
    Para subir un track a la parte superior del Timeline.

void gaTimelineBringVideoTrackToTop(string "videoTrackToTopName")
    Para subir un track de video a la parte superior del Timeline.

void gaTimelineBringAudioTrackToTop(string "audioTrackToTopName")
    Para subir un track de audio a la parte superior del Timeline.

void gaTimelineAddPage(string "pageName")
    Para añadir una página al Timeline.

void gaTimelineAddCurves(string "curvesName", int minValue, int maxValue)
    Para añadir un track de curvas.

void gaTimelineAddBangs(string "bangTrackName")
    Para añadir un track de bangs.

void gaTimelineAddFlags(string "flagTrackName")
    Para añadir un track de flags.

void gaTimelineAddColors(string "colorTrackName")
    Para añadir un track de color.

void gaTimelineAddLFO(string "lfoTrackName")
    Para añadir un track de LFO.

void gaTimelineAddSwitches(string "switchTrackName")
    Para añadir un track de switches.

void gaTimelineAddNotes(string "noteTrackName")
    Para añadir un track de notas.

void gaTimelineAddAudioTrack(string "audioTrackName", string gaDataPath("file.type"))
    Para añadir un track de audio.

void gaTimelineAddVideoTrack(string "videoTrackName", string gaDataPath("file.type"))
    Para añadir un track de video.

float gaTimelineGetValue(string "curvesTrackName")
    Devuelve los valores registrados en un track de curvas.

bool gaTimelineGetSwitch(string "switchTrackName")
    Devuelve si se activa o no un switch en el track de switches.

bool gaTimelineGetNote(string "noteTrackName")
    Devuelve si se activa o no una nota situada en un track de notas.

int gaTimelineGetNotePitch(string "noteTrackName")
    Devuelve el valor del pitch la nota situada en un track de notas.

float gaTimelineGetNoteVelocity(string "noteTrackName")
    Devuelve la velocidad de la nota situada en un track de notas.

ofColor gaTimelineGetColor(string "colorTrackName")
    Devuelve los valores de color de un track de color.

string gaTimelineGetBang()
    Devuelve el nombre del bang situado en el track de bangs.

int gaTimelineGetINFrame()
    Devuelve la posición de un punto de entrada en el track control de tiempo.
```

```

void gaTimelineSetInFrame(float setInFramePosition)
    Establece un punto de entrada en el track control de tiempo.
int gaTimelineGetOutFrame()
    Devuelve la posición de un punto de salida en el track control de tiempo.
void gaTimelineSetOutFrame(float setOutFramePosition)
    Establece un punto de salida en el track control de tiempo.
void gaTimelineClearINOUT()
    Borra los puntos de entrada y salida en el track control de tiempo.
int gaTimelineGetAudioTrackDuration(string "audioTrackName")
    Devuelve la duración de un track de audio.
int gaTimelineGetAudioTrackFFTSize(string "audioTrackName")
    Devuelve el tamaño del buffer para FFT de un track de audio
float gaTimelineGetAudioTrackFFTBin(string "audioTrackName" int numBin)
    Devuelve el valor del Bin especificado de un track de audio
int gaTimelineGetVideoTrackPosition(string "videoTrackName")
    Devuelve la posición del cabezal lectura del video.
int gaTimelineGetVideoTrackDuration(string "videoTrackName")
    Devuelve la duración del video.
int gaTimelineGetVideoTrackNumFrames(string "videoTrackName")
    Devuelve el número de frames del video.
int gaTimelineGetVideoTrackCurrentFrame(string "videoTrackName")
    Devuelve el número del frame actual del video.
void gaTimelineVideoPlayerNextFrame(string "videoTrackName")
    Actualiza la posición de lectura del video al siguiente frame.
void gaTimelineVideoPlayerUpdate(string "videoTrackName")
    Actualiza los datos del player que lee el track de video
void gaTimelineVideoPlayerDraw(string "videoTrackName")
    Dibuja el video en la ventana de salida

```

AUDIO INPUT

```

float gaGetVolume(int channel)
    Devuelve el volumen del sonido que entra por el canal de audio especificado. El rango de
    valores va de 0.0 a 1.0.
float gaGetPitch(int channel)
    Devuelve el pitch del sonido que entra por el canal de audio especificado. El rango de
    valores va de 0.0 a 1.0.
float gaGetBin(int channel, int indexBin)
    Devuelve el valor de un Bin concreto (indexBin) analizado desde el canal de entrada de
    audio especificado. El rango de valores va de 0.0 a 1.0.
float gaGetFFT(int channel, int indexFFT)
    Devuelve el valor de un punto concreto del FFT (indexFFT) analizado desde el canal de
    entrada de audio especificado. El rango de valores va de 0.0 a 1.0.

```

```

float gaGetInputBuffer(int channel, int indexBuffer)
    Devuelve el valor de la señal del canal de entrada de audio en el index especificado. El rango
    de valores va de 0.0 a 1.0.

```

```

float gaGetOutputBuffer(int indexBuffer)
    Devuelve el valor de la señal del canal de salida de audio en el index de buffer especificado.
    El rango de valores va de 0.0 a 1.0.

```

```

float gaGetSoundSpectrum(int position, int totalPositions)
    Devuelve el valor del espectro sonoro del audio de un archivo en una posición concreta
    entre un número de posiciones máxima establecido. El rango de valores va de 0.0 a 1.0.

```

Constantes

AUDIO_INPUT_CHANNELS

Devuelve el número de canales de entrada de audio. Previamente se debe seleccionar el dispositivo en GAmuza/Preferences/Audio Streaming - Input Device.

AUDIO_OUTPUT_CHANNELS

Devuelve el número de canales de salida de audio. Como en la constante anterior se debe seleccionar el dispositivo en GAmuza/Preferences.

BUFFER_SIZE

Devuelve el tamaño del buffer de audio. Previamente se debe seleccionar el tamaño deseado en GAmuza/Preferences/Audio Streaming - Buffer Size.

FFT_BANDS

Devuelve el número de FFT Bins desde la escala de Bark que utiliza GAmuza.

AUDIO SYNTH

void gaWave(CONSTANT waveType, float frequency)

Genera un oscilador de ondas multicanal. Los tipos de ondas pueden ser: GA_SINE, GA_COSINE, GA_SAW, GA_TRI, GA_RECT, GA_NOISE, GA_PINK, GA_BROWN, GA_PHASOR, GA_PULSE

void gaMonoWave(CONSTANT waveType, float frequency, int channel)

Genera un oscilador de ondas monocanal, semejante al anterior pero su último parámetro indica el canal de salida del sonido.

void gaWaveVolume(int oscillatorID, float volumeLevel)

Ajusta el nivel de volumen de un oscilador de onda previamente creado.

void gaWaveFrequency(int oscillatorID, float frequency)

Reajusta la frecuencia de un oscilador previamente creado.

void gaWaveType(int oscillatorID, CONSTANT waveType)

Modifica el tipo de onda para un oscilador previamente creado.

void gaWaveTuning(int oscillatorID, float newValueTuning) ??????????????????

float gaNToF(CONSTANT musicalNote)

Convierte una nota musical en su frecuencia armónica. Las notas disponibles son DO_N, DOB_N, RE_N, REB_N, MI_N, FA_N, FAB_N, SOL_N, SOLB_N, LA_N, LAB_N, SI_N; siendo N un valor que puede ir de 0 a 8, por ejemplo LA_4 corresponde a 440 Hz.

AUDIO INPUT RECORDING

```
void gaStartRec(int inputChannel)
    Inicia la grabación de un sample en el canal de entrada especificado.
void gaStopRec()
    Detiene la grabación del sample previamente iniciado.
void gaRecSamplePlay(int indexSample)
    Inicia la reproducción de un sample identificado por su index.
void gaRecSampleStop(int indexSample)
    Detiene el sample de audio previamente grabado con el index especificado
void gaRecSampleVolume(int indexSample, float volumeLevel)
    Ajusta el volumen de la muestra de audio especificada previamente grabado. El rango de
    valores para el volumen va de 0,0 a 1,0.
void gaRecSampleLooping(int indexSample, bool true or false)
    Reproduce en bucle o una sola vez el sample de audio especificado y previamente
    grabado.
void gaRecSamplePaused(int indexSample, bool true or false)
    Pone en pausa o play el sample de audio con el index especificado y previamente grabado.
void gaRecSampleSpeed(int indexSample, float speed)
    Ajusta la velocidad (pitch) del sample de audio especificado y previamente grabado. El
    rango de valores para la velocidad va de -1,0 a 1,0.
void gaDrawRecHead(int indexSample, int x, int y, int w, int h)
    Dibuja la barra de reproducción del sample de audio especificado y previamente grabado
    dentro un área definida por los parámetros x, y, ancho y alto.
```

OSC

```
void gaGetOSCMessage(string "labelName", int _numVar)
    _numVar es el numero de variables que se asociarán con esta etiqueta, se suele trabajar
    con una variable x para la etiqueta, pero se puede enviar un array de variables asociadas
    con una sola etiqueta, por eso hay que decir cuantas variables va a tener asociada.
string gaGetOSCValue(string "labelName", int indexMessage)
    Lee en tiempo real el valor de un mensaje OSC asociado a la etiqueta al index de mensajes
    especificados como parámetro.
int gaGetOSCNumVars(string "labelName")
    Devuelve el número de valores de los mensajes OSC asociados con la etiqueta especificada.
void gaSetOSCMessage(string "labelName", CONSTANT DATA_TYPE)
    Inicializa un nuevo contenedor para enviar mensajes OSC, con la etiqueta y tipo de datos
    especificado en sus parámetros. Las opciones para el tipo son las constantes: OSC_INT,
    OSC_FLOAT y OSC_STRING
void gaSetOSCValue(string "labelName", string "value")
    Envía el valor del mensaje OSC asociado a la etiqueta especificada. El valor debe
    ser convertido a string, aunque sea un número. Para ello se puede utilizar la función
    toString() de Lua.
void gaSetOSCActive(string "labelName", bool true or false)
    Activa o desactiva un mensaje OSC específico. Esto es útil cuando se está trabajando con
    una gran cantidad de mensajes OSC y queremos aislar algunos.
string gaSendingOscTo()
    Devuelve la dirección IP a la que está enviando datos por OSC.
```

MIDI

```
int gaGetMidiChannel()
    Devuelve la referencia al dispositivo MIDI conectado y en comunicación. Se puede cambiar
    la configuración MIDI seleccionando el dispositivo en GAmuza / Preferences / pestaña MIDI.
int gaGetMidiByteOne()
    Devuelve la referencia al index del último elemento MIDI modificado, como un slider, botón,
    mando...
int gaGetMidiByteTwo()
    Devuelve el valor del último elemento MIDI modificado, como un slider, botón, mando...
int gaGetMidiPitch()
    Devuelve el pitch de la última tecla MIDI tocada, en el rango MIDI de 0-127.
int gaGetMidiVelocity()
    Devuelve la velocidad de la última tecla MIDI tocada, en el rango MIDI de 0-127.
float gaGetMidiValue(int deviceMIDI, int indexMIDI)
    El valor normalizado del index MIDI en el dispositivo MIDI seleccionados Por ejemplo:
    gaGetMidiValue(1,11) toma el valor de un slider con index 11 del dispositivo MIDI 1.
```

Constantes

OSC_SENDING_PORT	El puerto al que se están enviando datos OSC. Se debe configurar previamente en GAmuza/ Preferences/OSC - SENDING TO - PORT.
OSC RECEIVING_PORT	El puerto por el que se están recibiendo datos OSC. Se debe configurar previamente en GAmuza/Preferences/OSC - RECEIVING AT - PORT.
OSC_INT	Datos tipo int para funciones de OSC.
OSC_FLOAT	Datos tipo float para funciones de OSC.
OSC_STRING	Datos tipo string para funciones de OSC.

PURE DATA SYNTHESIS ENGINE

```
void pdAddToSearchPath(string "folder/")
    Indica la carpeta en la que están los archivos de pd. Esta carpeta debe estar dentro de data.
void pdOpenPatch(string gaDataPath("folderName/fileName.pd"))
    Abre el archivo de Pure data con la ruta/nombre especificados.
void pdClosePatch(string gaDataPath("folderName/fileName.pd"))
    Cierra el archivo de Pure data con la ruta/nombre especificados.
void pdStart()
    Inicia el archivo de Pure data.
void pdStop()
    Detiene el archivo de Pure data.
void pdComputeAudio(bool true or false)
    Activa o desactiva la opción Compute Audio de Pure data.
void pdSendBang(string "labelName")
    Envía un bang al objeto de pure data con la etiqueta especificada.
void pdSendFloat(string "labelName", float value)
    Envía un valor float al objeto de pure data con la etiqueta especificada.
void pdSendSymbol(string "labelName", string "symbolName")
    Envía un símbolo al objeto de pure data con la etiqueta especificada.
void pdStartMessage()
    Inicializa el sistema de envío de mensajes a Pure data.
void pdAddFloat(float value)
    Añade el valor float especificado en su parámetro.
void pdAddSymbol(string "symbolName")
    Añade el símbolo especificado en su parámetro.
void pdFinishList(string "labelName")
    Finaliza el envío de la lista de mensajes a la etiqueta especificada.
void pdFinishMessage(string "labelName", string "messageName")
    Finaliza el mensaje indicado de la etiqueta especificada en sus parámetros.
void pdSendNoteOn(int channel, int pitch, int velocity)
    Envía una nota con el pitch, velocidad y por el canal especificados.
void pdSendControlChange(int channel, int controller, int value)
    Envía un nuevo valor al controlador y por el canal especificados.
void pdSendProgramChange(int channel, int value)
    Función interna MIDI de Pure Data
void pdSendPitchBend(int channel, int value)
    Función interna MIDI de Pure Data
void pdSendAftertouch(int channel, int value)
    Función interna MIDI de Pure Data
void pdSendPolyAftertouch(int channel, int pitch, int value)
    Función interna MIDI de Pure Data
```

AUDIO UNIT PLUGINS

```
void auAddPlugin(string "filter")
    Añade el plugin especificado en su parámetro.
void auAddPluginFromPreset(string "preset")
    Carga un plugin AU desde un preset guardado previamente en .aupreset (Apple Logic Studio).
    En GAMUZA se puede guardar este tipo de presets con la función auSavePluginPreset().
void auSavePluginPreset(string "plugin", string "path", string "name")
    Guarda un archivo .aupreset del plugin AU, en la ruta y con el nombre especificados en sus
    parámetros.
void auPluginSetParam(string "plugin", string "parameter", float value)
    Pone el parámetro y valor especificados a un plugin previamente cargado.
void auListPlugins()
    Lista en la consola de GAMUZA los plugins de Audio Unit que reconoce el ordenador.
    Pueden verse también a través de la herramienta Logger en el menú Tools.
```

ARDUINO

```
string gaSerialDevice()
    Devuelve el nombre del puerto serial al que está conectado el dispositivo, que puede ser un
    Arduino u otro dispositivo como el controlador USB DMX. Recordad que previamente hay
    que configurar Arduino en GAMUZA/Preferences.
float gaAnalogRead(int analogPin)
    Devuelve el valor del pin analógico especificado en un rango de 0.0 a 1.0.
int gaDigitalRead(int)
    Devuelve el valor digital del pin especificado como 0.0 ó 1.0.
void gaDigitalWrite(int,int)
    Escribe el valor señalado en el pin digital de Arduino especificado. Si es un pin digital, utilice
    valores 0 ó 1, si es un pin PWM utilice el rango de 0 a 255.
    Recordad que hay que configurar previamente los pines en el Módulo Arduino de GAMUZA,
    disponible en el menú principal, módulos/ Arduino. Ejemplo: si está utilizando pin digital 10
    como PWM, id al Módulo y seleccionar en pin digital 10 la opción PWM.
void gaSetServoPin(int,bool)
    Activa / desactiva el pin de Arduino especificado como un pin de control para servomotor.
    Se debe elegir un pin de Arduino preparado para trabajar como pin servo; por ejemplo, en
    la placa Arduino UNO los pin digital 9 y 10 tienen la opción para ser pin servo.
void gaServoWrite(int,int)
    Escribe el valor señalado en el pin digital servo de Arduino especificado. El pin servo acepta
    un rango de valores entre 0 y 255.
    Se debe establecer previamente los pines de Arduino en el módulo Arduino de GAMUZA,
    disponible en el menú principal, Modules/Arduino.
```

SERIAL FUNCTIONS INTERFACE

string gaSerialReadBytes(ofSerial, int value)

Recibe por el puerto serial datos enviados por el dispositivo. No olvidar configurar previamente la clase **ofSerial()** con el puerto serial y el baudRate que utiliza el dispositivo conectado.

Ejemplo **serial:setup("/dev/cu.usbmodem1421", 9600)**

void gaSerialWriteBytes(ofSerial, string "data")

Escribe una secuencia de bytes en el puerto serie

void gaSerialWriteByte(ofSerial, string)

Escribe un único byte en el puerto serial

NET

void gaStartHTTPService()

Inicializa la conexión con internet

void gaStopHTTPService()

Detiene la conexión con internet

void gaAddHTTPForm(ofxHttpForm)

Función vinculada a la clase del addon de openFrameworks ofxHttpForm() para enviar formularios a una página web con los métodos GET o POST

void gaAddHTTPUrl(string url)

Conecta con la dirección web indicada en su parámetro

void gaSendHTTPReceivedCookies()**void gaSetHTTPBasicAuthentication(string user, string pass)**

Envía a una página web los datos básicos del formulario de identificación.

string gaGetHTTPReceivedResponse()

Devuelve la respuesta de un pedido http (si hay).

string gaGetHTTPReceivedStatus()

Devuelve el estado de la respuesta de un pedido http, 404 o 303 en el caso de errores, o un simple OK, en el caso de que ha habido respuesta correcta.

LIVE CREATIVE CODING

En la técnica subyace la naturaleza de los elementos que propician la interactividad, la generación de formas por el sonido, el movimiento de dispositivos por código..., por eso ella conoce bien cómo se construye la casa del arte digital, sus habitáculos y aperturas al exterior, su extenso jardín y el laberinto que contiene. Desvelar la estructura de la técnica es comprender ese complejo plano constructivo, y siguiendo sus indicaciones podremos empezar a esbozar diagramas, o planificar instrucciones que posibiliten procesos o comportamientos artísticos diferentes.