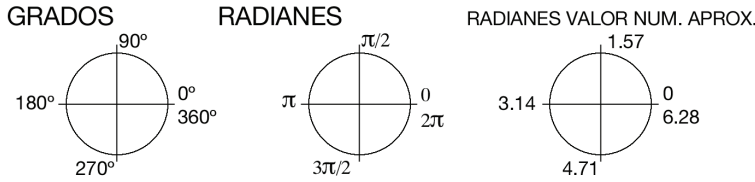
4.3. Curvas por trigonometría y funciones matemáticas

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:

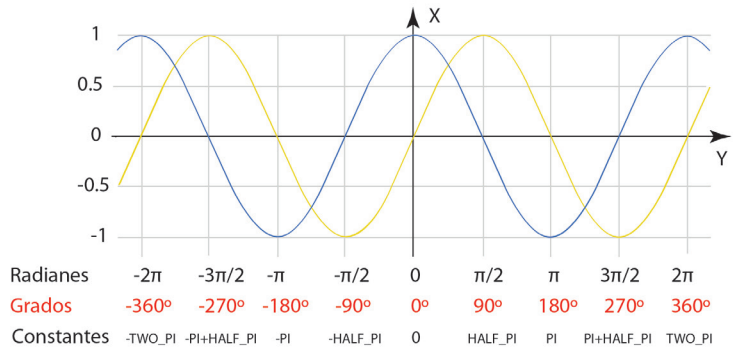
```
PI
TWO_PI
TAU     // Similar a TWO_PI
FOUR_PI
HALF_PI
```

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 ángulos[43].

43 Más información sobre funciones matemáticas de 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 integrales (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`

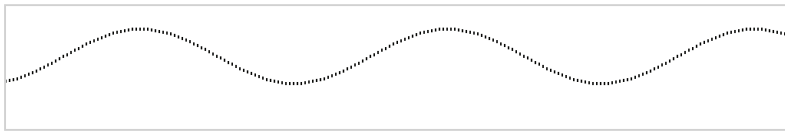
`math.modf(-5.3)` devuelve `-5` `-0.3`

`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 Reas[44], 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`.



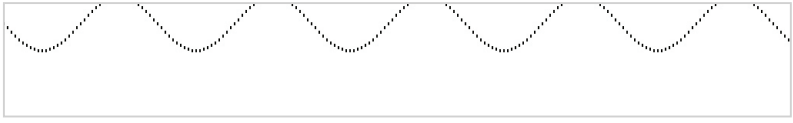
```
/*
 GAmuza 043        E-4-10
 ------------------------

 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.

pos = 25



pos = 75

amplitud=5.0

amplitud=45.0

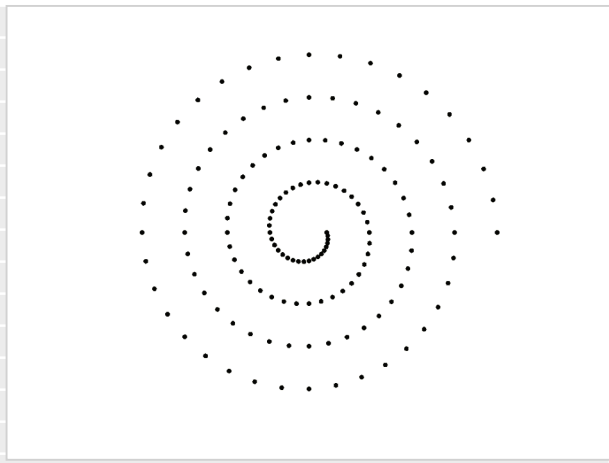
inc=PI/12.0

inc=PI/90.0

angle=HALF_PI

angle = PI

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-4-11
--------------------------------
 Trigonometría espiral


*/

radio = 30     //variable global

function setup()
   ofEnableSmoothing()
end

function draw()
   gaBackground(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
```

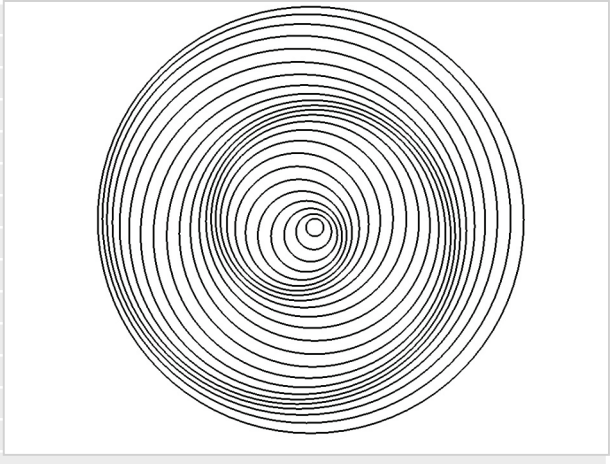
La espiral se genera por pequeños círculos cuyo centro se va alejando progresivamente del centro de la pantalla a la vez que giran alrededor de él.

Para calcular la posición de esos centros se utiliza una estructura `for` que recorre cuatro veces los 360º 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.

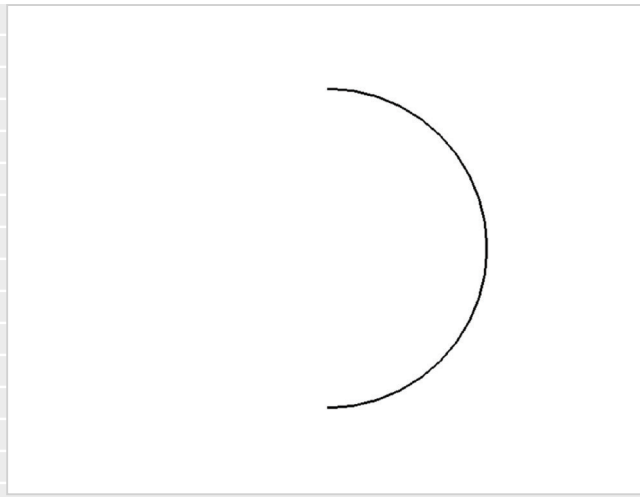


```
/*
 GAmuza 043            E-4-12
 ---------------------------
 Artists/Duchamp - Rotoreliefs
 creado por n3m3da && mj

*/

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 45º. La diferencia de sus radios, `spiralFactor`, se corresponde al valor de ese mismo circulo invisible que distribuye la tangencia de los otros, es esa tangencia lo que produce la percepción de una espiral.

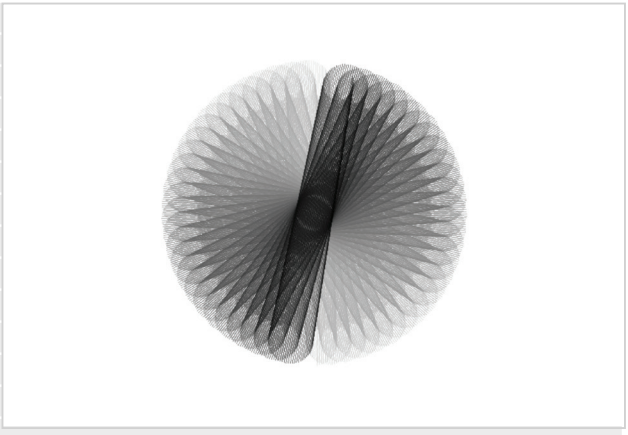
También es usual utilizar las funciones de generación de formas irregulares para dibujar curvas con trigonometría. El siguiente ejemplo muestra cómo dibujar un arco de circunferencia con `ofVertex()` más las funciones de seno y coseno.



```
/*
GAmuza 0.4.3      E-4-13
 ----------------------
 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-4-14
 ------------------------------
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
```

4.4. Transformar: traslación, rotación, escalar

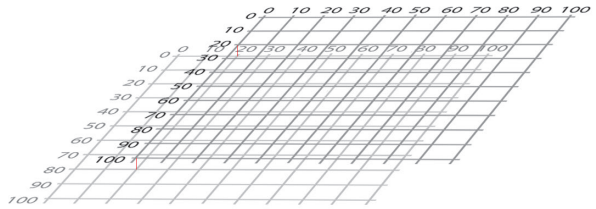
> Ahora bien, sabemos, por un conocido pasaje de la República, que el espín de la peonza le había planteado a Platón el difícil problema del reposo y el movimiento. Porque dando vueltas es estable, contradicción. Platón se las ingenia para afirmar que ésta está en reposo en relación con lo recto, y en 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.



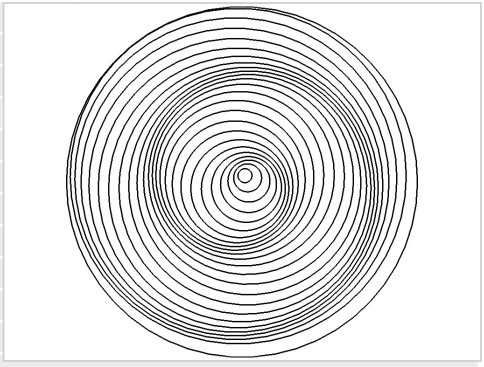
La función `ofTranslate(x, y, z)` mueve la textura de salida desde su coordenada `(0, 0, 0)` hasta el punto que se indiquen los parámetros, si se está trabajando en 2D la coordenada `z` es `0`. Solo las figuras situadas después de la función se verán afectadas por la translación.

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.

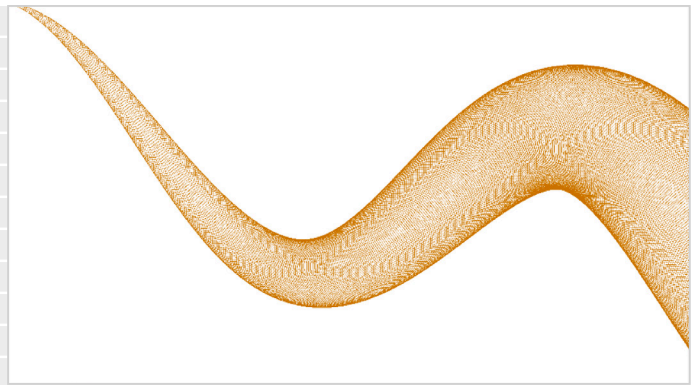
Retomamos el ejemplo *Rotating Disc* de Duchamp para ver estas funciones con la espiral en movimiento.



```
/*
 GAmuza 043    E-4-15
 -------------------------------------
  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-4-16
 ------------------------
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`.

4.5. 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 Ching [46].

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 45º 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-4-17
 --------------------
 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 45º.

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 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 el 16'6%.`

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

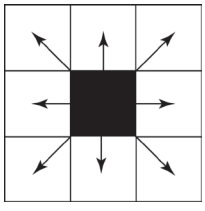
Daniel Shiffman propone otro ejemplo particular dentro del campo del azar, el *Camino aleatorio* (Random Walks), que de algún modo mantiene también relación con las acciones artísticas psicogeográficas de los Situacionistas.

> 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

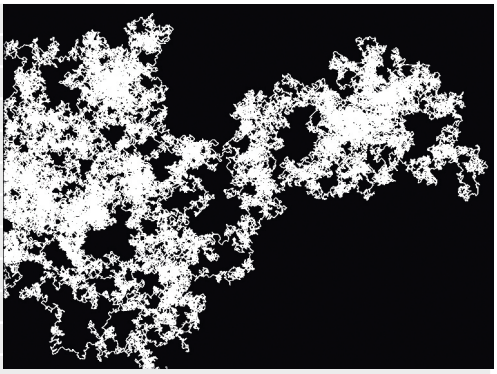
48 Daniel Shiffman (2012), *Nature of code*, autopublicado, pág. 7.

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 economía[49].

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 `ofRandomf()`, sin parámetros, porque son esos los que tiene establecidos por defecto.



```
/*
 GAmuza 043         E-4-18
 -------------------------
 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 * ofRandomf())
   ysig = y0 + (step * ofRandomf())

   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
```

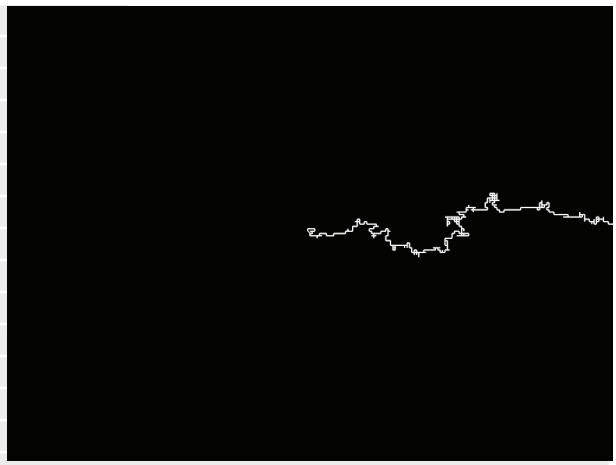


Este caminante recorre 4 px en cada paso, lo que transforma la opción de puntos en pequeñas líneas encadenadas. El segundo par de coordenadas `(xsig, ysig)` de cada línea será el primero de la siguiente, trazando así el recorrido de su deriva. Como el fondo en GAmuza se dibuja de nuevo en cada frame, 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 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 Shiffman[50] 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 euclidiana 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-4-19
 -------------------------
 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.

4.5.1. 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*.

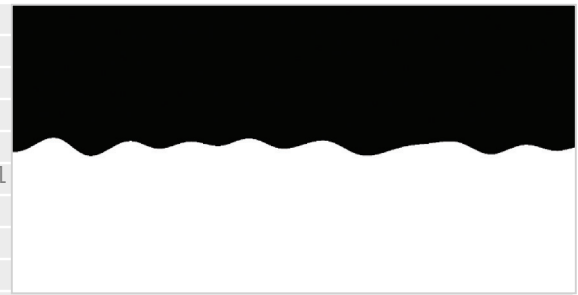
> El Ruido Perlin es una función matemática que utiliza interpolación entre un gran número de gradientes precalculados de vectores que construyen un valor que varía pseudo-aleatoriamente en el espacio o tiempo. Se parece al ruido blanco, y es frecuentemente utilizado en imágenes generadas por computadora para simular variabilidad en todo tipo de fenómenos[51].

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.

```
/*
 GAmuza 043      E-4-20
 ----------------------
 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 ente 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



```
/*
 GAmuza 043                          E-4-21
 ------------------------------------
 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
```