Algoritmos Genéticos aplicados a la generación y producción de formas escultóricas

Emiliano Causa, emiliano.causa@gmail.com

Este trabajo se hizo en el marco de los proyectos de investigación "Nuevos territorios de la Generatividad en las Artes Electrónicas: su convergencia con la Robótica, la Realidad Aumentada y el Net.Art" (Dirigido por Emiliano Causa – Fac. Bellas Artes UNLP) y "El Arte Generativo en las Artes Multimediales." (Dirigido por Carmelo Saitta y Codirigido por Emiliano Causa – Área de Artes Multimediales IUNA).

Resumen

El presente trabajo aborda la creación de formas físicas tridimensionales a partir de la aplicación de Algoritmos Genéticos. La idea consiste en crear volúmenes mediante el montaje de planos seriados, dichos planos serían cortados mediante una fresa de CNC, una máquina de corte por control numérico. El artículo muestra el proceso completo de formalización matemática y su traducción formal en un volumen que puede ser montado con planos seriados. Esta experiencia forma parte de la colaboración entre dos grupos de investigación que permitió construir una fresa de CNC para poner a prueba técnicas de fabricación digital.

Palabras claves:

Algoritmos genéticos, Planos Seriados, Escultura, Tridimensionalidad, CNC, Fabricación Digital

Introducción

El presente texto es continuación de "*Los Algoritmos Genéticos y su Aplicación al Arte Generativo*", escrito por este autor. Dicho texto trata conceptos y antecedentes de este tipo de técnicas y por ende se recomienda su lectura antes de abordar el presente, ya que no se tratarán dichos conceptos y se darán por conocidos por el lector.
La aplicación fue desarrollada en el lenguaje de programación Processing (desarrollado por la UCLA y la NYU). Debido a que la extensión necesaria para explicar todas la decisiones y algoritmos implicados en el proceso excederían ampliamente el alcance de este texto, nos centraremos en las cuestiones más importantes y dejaremos para un anexo aquellas cuestiones periféricas.

Una escultura generativa

Al momento de iniciar este trabajo me propuse buscar alguna estrategia que permitiera realizar una escultura mediante procedimientos generativos, esto implicaba dos cuestiones:

1) La primera, que debía encontrar alguna manera de representar matemáticamente la "forma" de ese volumen.

2) La segunda, que debía encontrar una forma de traducir esa forma físicamente, es decir un método constructivo.

3) La tercera que debía encontrar un método algorítmico de diseñar dicha forma.

La fresa por CNC y los planos seriados

La primera y segunda cuestión, es decir: la representación matemática y el proceso constructivo, están íntimamente ligadao, ya que el método constructivo implica el uso de alguna o varias máquinas/herramientas que permita traducir una forma matemáticamente representada en un volumen concreto/físico de alguna materialidad (madera, piedra, plástico).
Como parte de nuestras investigaciones hicimos construir una fresa por CNC, una máquina de corte por control numérico, preparada para cortar piezas planas controlada por una computadora. Esta máquina, tiene 2,5 dimensiones, lo que significa que puede cortar 2 dimensiones (x e y) y que tiene un moderado control de una tercera dimensión. La CNC trabaja con una Arduino que la controla y que recibe órdenes (vía USB) desde una computadora. El software de la computadora transmite esta órdenes en GCode, un protocolo de comunicación para el control de este tipo de dispositivos.

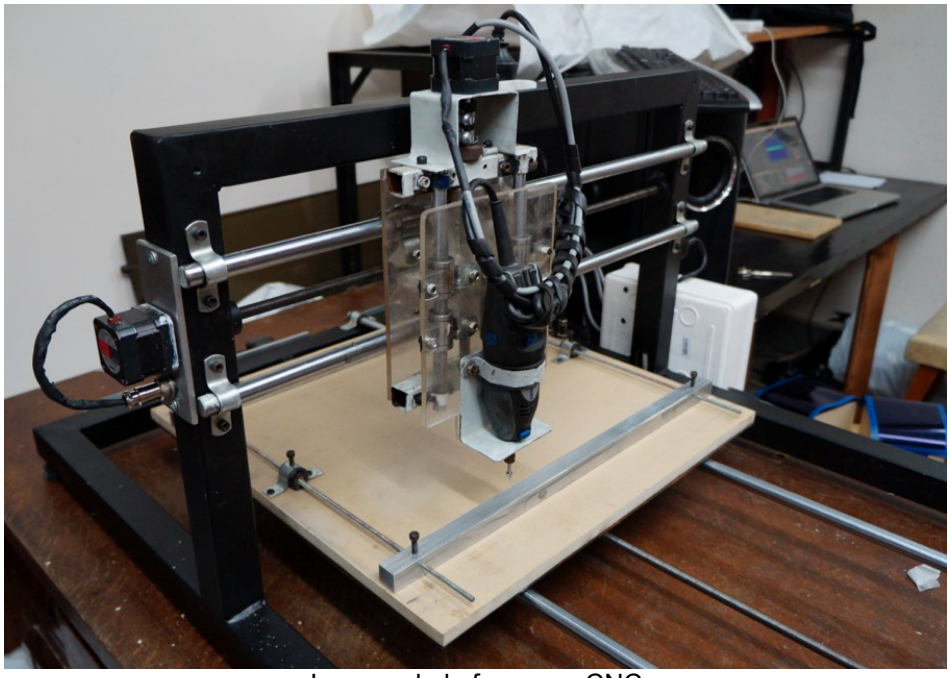


Imagen de la fresa por CNC

Dado que nuestro desarrollo se realizaría con el lenguaje de programación Processing, decidimos encontrar algún formato intermedio entre el GCode y este lenguaje. El formato más adecuado que encontramos fue el formato de gráficos vectoriales SVG, el cuál es un formato libre y abierto y que puede ser construido desde Processing, a la vez que existen diferentes softwares capaces de traducir SVG en GCode.
Ya que la CNC es capaz, principalmente, de construir planos, debía abordar una estrategia para construir un volumen con planos. Una de las formas más interesantes de hacerlo es mediante la técnica de “planos seriados” que consiste en ubicar una serie de planos que representan los diferentes cortes de un volumen y que al ubicarlos en forma continua lo reconstruye.

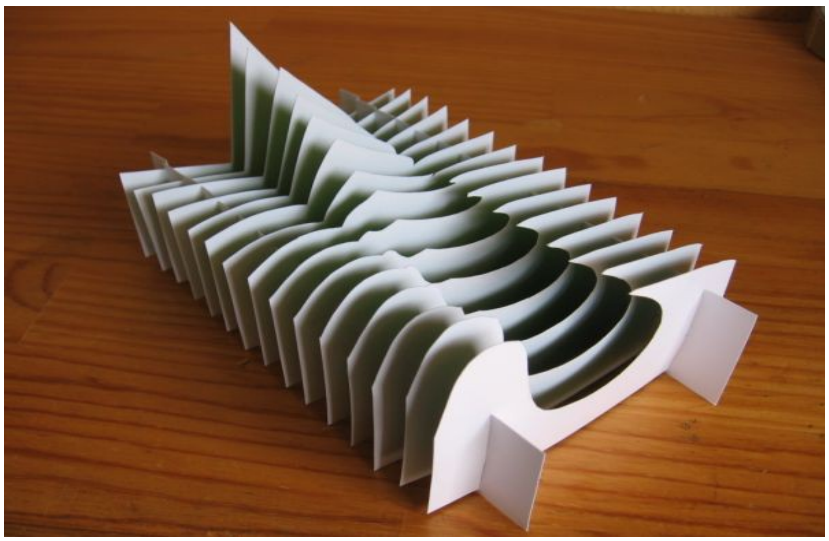


Imagen de Planos Seriados publicada por Cristina Laurent en:
http://1.bp.blogspot.com/_XrALJqvZKuY/TDusLHx3AEI/AAAAAAAAAK8/xZ6oZkKCjf0/s1600/IMG_1148.jpg

De esta forma, sería posible diseñar un volumen con la computadora mediante su descomposición en planos seriados, luego habría que cortar plano por plano en la CNC para, finalmente, montarlos uno por uno hasta reconstruir el volumen. A continuación mostramos un volumen creado con nuestra aplicación a partir de planos seriados. Para facilitar su lectura se resaltó el sexto plano.

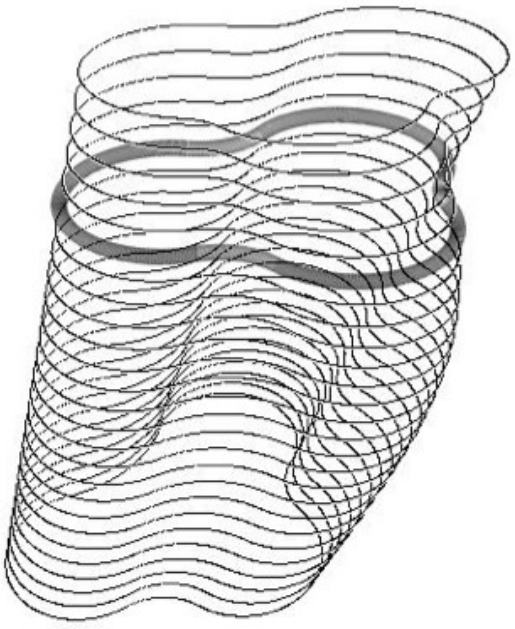


Imagen de planos seriados creado por nuestra aplicación

El modelado del volumen

En este punto nos enfrentamos a la tercera cuestión que enunciamos al inicio: encontrar un método algorítmico de diseñar dicha forma. En esta etapa opté por elegir los algoritmos genéticos. Tal como expliqué en mi texto “Los Algoritmos Genéticos y su Aplicación al Arte Generativo” (2011), los algoritmos genéticos son las técnicas que nos permiten simular el proceso evolutivo y aplicarlo a sistemas artificiales.

Para poder utilizar esta técnica es necesario traducir el fenómeno (el objeto a ser procesado) en términos matemáticos, es decir hay que traducir el “modelado de la forma escultura” en un conjunto de parámetros matemáticos que lo represente.

Me interesaba encontrar un método para diseñar formas plásticas, orgánicas, que privilegiara las curvas por sobre las formas angulosas y las aristas. El camino más sencillo que encontré fue crear cada plano a partir de círculos que se van empalmando. Debajo puede verse una imagen de una forma construida a partir del empalme de círculos, más abajo se puede ver la forma resultante.

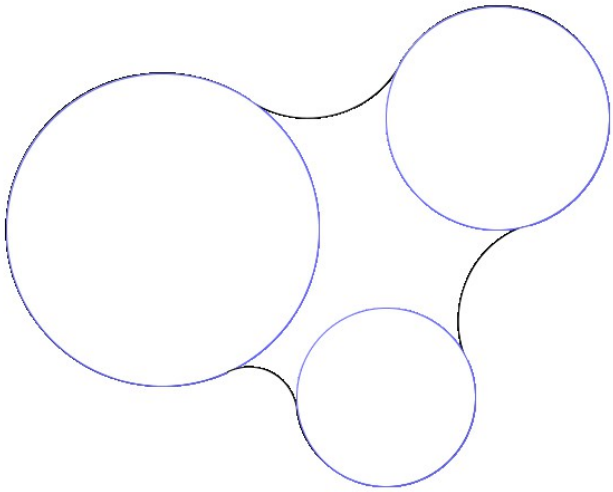


Imagen de círculos y sus empalmes

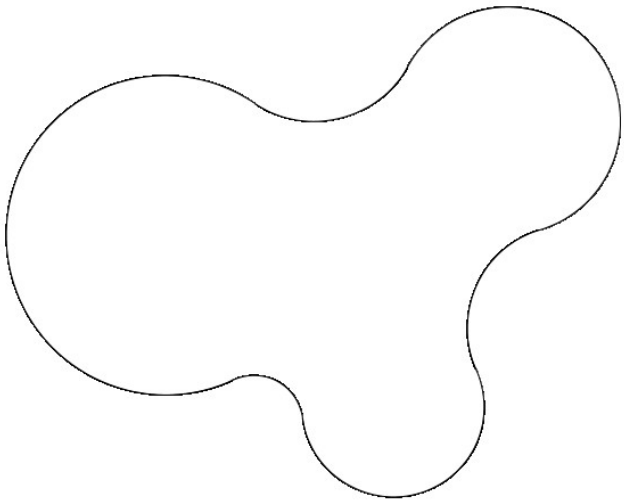


Imagen de la forma resultante

La sencillez de este método permite descomponer formas complejas en una de las figuras más sencillas, el círculo. Así el modelado del volumen, puede traducirse en el control de los parámetros de un conjunto de círculos, es decir: sus posiciones y diámetros. En la imagen de abajo puede observarse como a partir de tres círculos y la variación de sus parámetros se puede construir formas complejas.

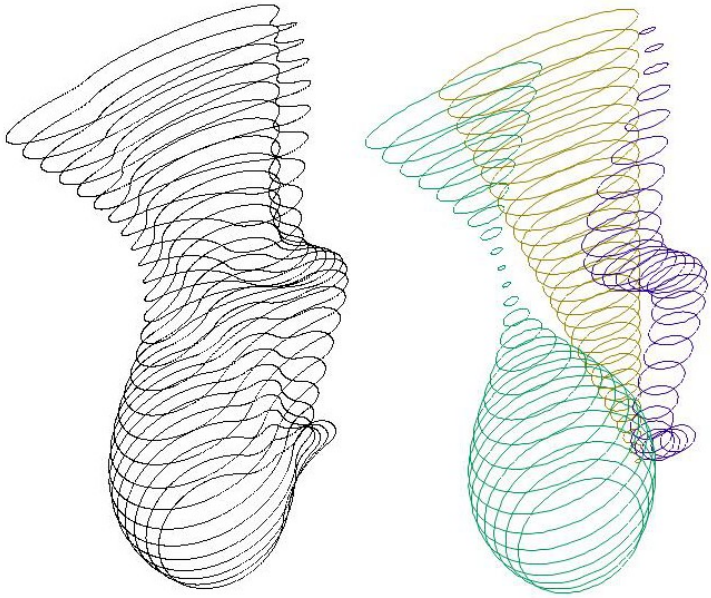


Imagen de construcción de formas complejas a partir de empalmes de círculos

De esta forma, un volumen se puede construir a partir de la configuración de un conjunto pequeño de parámetros:

1) Cantidad de círculos que participan de la construcción de la forma
2) Posición inicial de esos círculos
3) Cantidad de planos por los que se proyectan los círculos.
4) Evolución de dichas posiciones, traducible en la evolución de dos variables, X e Y.
5) Evolución de los diámetros de dichos círculos

Por ejemplo en los gráficos de abajo pueden verse diferentes configuraciones de cantidad de círculos intervinientes para generar la forma:

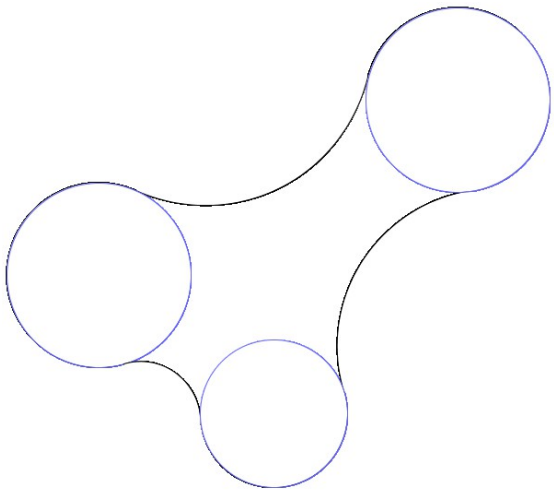


Imagen: configuración con 3 círculos

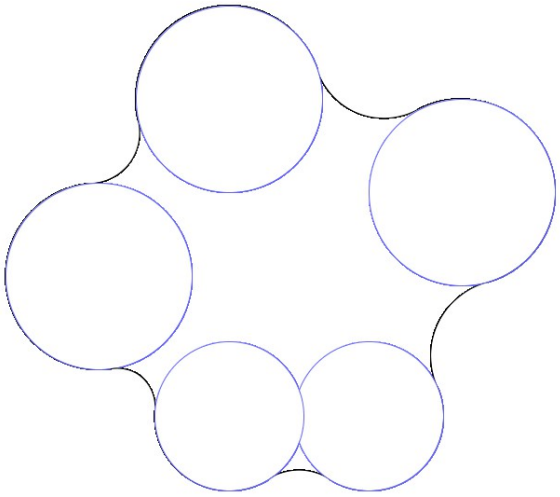
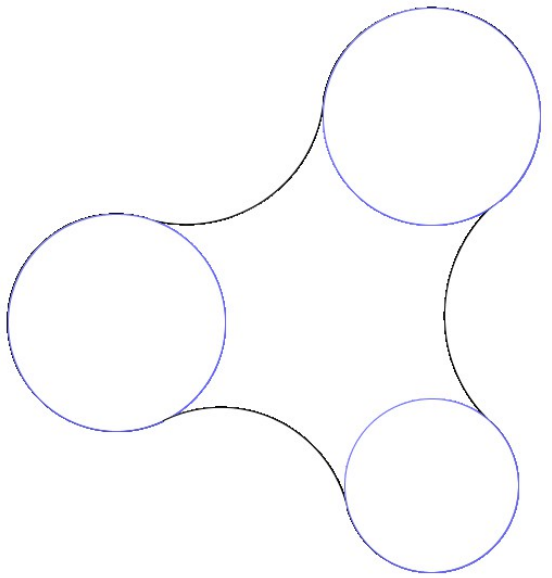
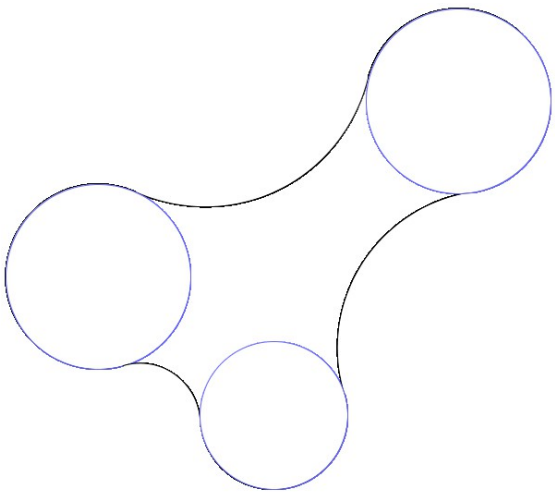


Imagen: configuración con 5 círculos

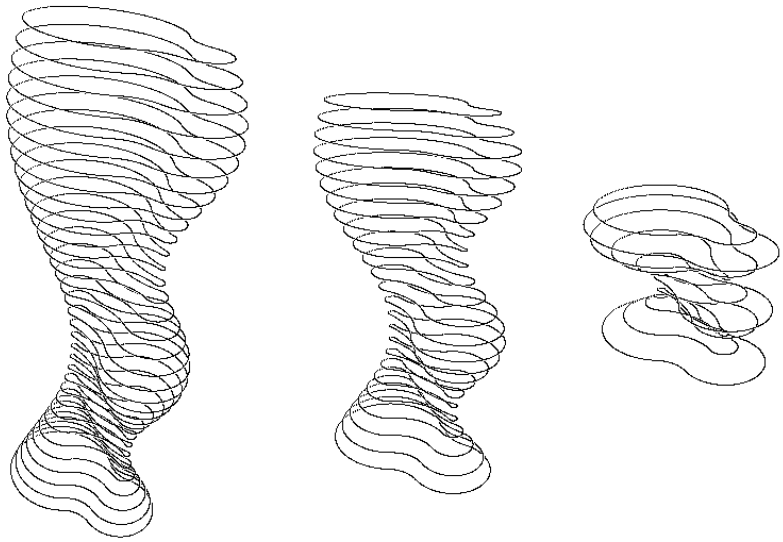
Respecto del segundo parámetro podemos ver a continuación ejemplos de diferentes posiciones iniciales:





El siguiente parámetro es la cantidad de planos o estratos por los que desarrollan o proyectan los círculos.

En la imagen de abajo puede observarse como las mismas formas de desarrollan en 35, 25 y 10 estratos, produciendo diferencias importantes en la forma resultante.



Respecto de la evolución de las parámetros de la posición de los círculos (posición en X e Y) podemos ver los siguientes ejemplos:

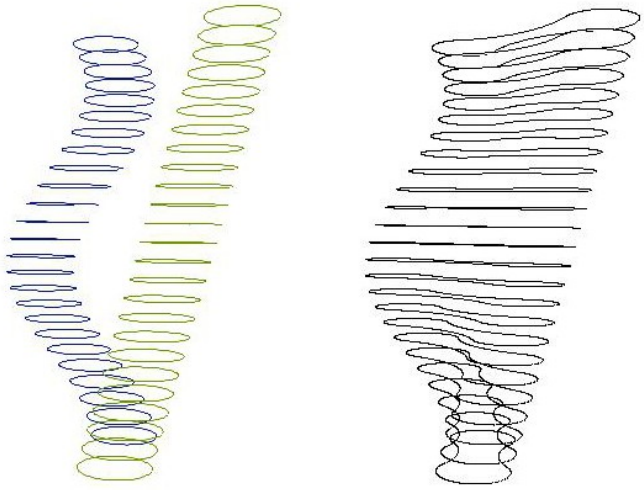


Imagen: primer ejemplo de la evolución de la posición

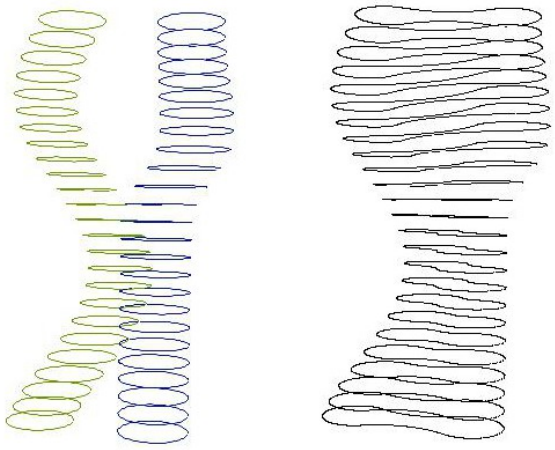


Imagen: segundo ejemplo de la evolución de la posición

En el primer ejemplo se pueden observar dos formas de evolución, una que sigue una curva de desarrollo lineal y otras que es la combinación de dos sinusoides (una en X y otra en Y). En cambio en el segundo ejemplo, además de una curva sinusoidal se puede observar una envolvente irregular. Independientemente de la particularidades de cada tipo de curva, se ve como la evolución de las curvas genera diferentes volúmenes.

Por último, podemos ver en las siguientes imágenes ejemplos de la evolución del diámetro de los círculos.

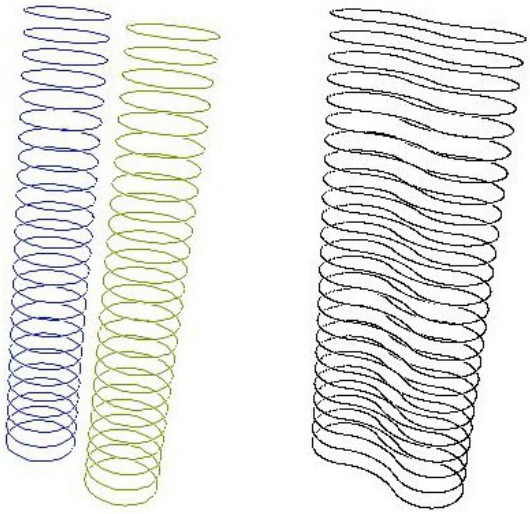


Imagen: primer ejemplo de evolución de diámetro

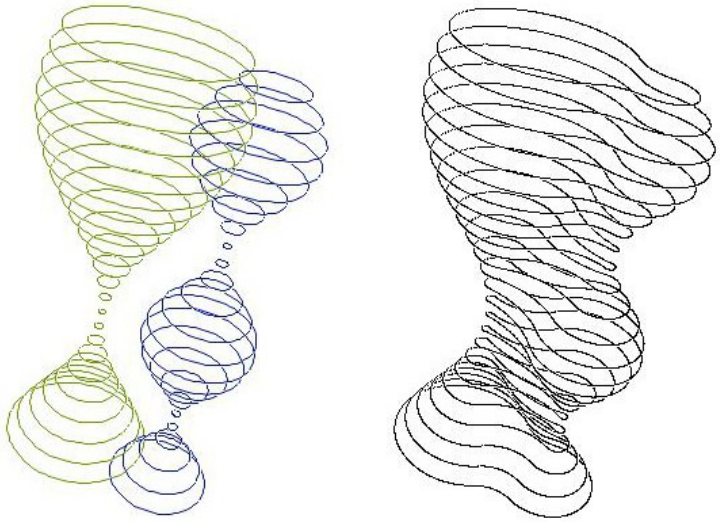


Imagen: segundo ejemplo de evolución de diámetro

En el primer ejemplo podemos ver dos líneas de círculos que evolucionan en forma constante, mientras que en el segundo ejemplo, las líneas de círculos evolucionan con curvas sinusoidales con diferentes fases.

De esta forma, con los cuatro parámetros recién presentados, tenemos una relación entre éstos y la forma que producen. A partir de ahora es necesario establecer de qué manera serán tratados estos datos para poder procesarlos con los Algoritmos Genéticos.

El genotipo y el fenotipo

Tal como expliqué en mi texto “Los Algoritmos Genéticos y su Aplicación al Arte Generativo” (2011), a la hora de usar esta técnica prefiero separar la información de su manifestación, usando la metáfora del genotipo versus el fenotipo: en el material genético (el genotipo) están escritas las instrucciones para construir al individuo (el fenotipo). En este caso decidí organizar los parámetros que definen la forma (el genotipo) en un XML. Este XML está organizado en una secuencia de “genes”, lo que permite facilitar la operación de crossover.

```
<?xml version="1.0"?>
<ADN>

      <PADRES>
            <P>...</P>
            <M>...</M>
            <CADENA>...</CADENA>
      </PADRES>

      <GEN  id = "0">
            <CANT_PLANOS>...</CANT_PLANOS>
      </GEN>

      <GEN  id = "1">
            <PUNTOS_INICIALES>
                  <PUNTO>
                        <X>...</X>
                        <Y>...</Y>
                  </PUNTO>
                  ...
            </PUNTOS_INICIALES>
```

```
        </GEN>

        <GEN  id = "2">
              <EVOLUCIONES_DIAMETRO>
                    <EVOLUCION>
                          ...
                    </EVOLUCION>
                    ...
              </EVOLUCIONES_DIAMETRO>
        </GEN>

        <GEN  id = "3">
              <EVOLUCIONES_XY>
                    <PAR>
                          <EVOLUCION  id = "x">
                                ...
                          </EVOLUCION>
                          <EVOLUCION  id = "y">
                                ...
                          </EVOLUCION>
                    </PAR>
                    ...
              </EVOLUCIONES_XY>
        </GEN>

</ADN>
```

Código: Estructura del XML

En el esquema de arriba se puede ver la estructura del XML. En los lugares que está tres puntos ("...") son en donde van los datos. En el esquema no se profundiza el detalle de la etiqueta <EVOLUCION> que será explicada más adelante.
Como se puede ver, el XML se organiza bajo la etiqueta <ADN> de la cual se desprenden una etiqueta <PADRES> y cuatro etiquetas <GEN>.
La etiqueta <PADRES> es la que guarda los nombres de los archivos de los padres que se combinaron para crear este XML, la etiqueta <P> corresponde al padre y la <M> a la madre (la distinción padre, madre es netamente poética y sirve para distinguir a los dos progenitores, pero no tienen diferencia alguna) y la etiqueta <CADENA> muestra cuáles genes fueron tomados de cuál de los progenitores, por ejemplo, la cadena "MpMM" significa que el primer, tercer y cuarto gen fueron tomados de la madre, mientras que el segundo fue tomado del padre.
La estructura tiene cuatro etiquetas <GEN> que obviamente representan cada uno de los genes en los que se organiza la información. Los genes se vincula con algunos de los parámetros que describimos anteriormente:

Gen 0: Cantidad de estratos del plano seriado. Contiene la etiqueta <CANT_PLANOS>.

Gen 1: Cantidad y posición inicial de los círculos. Contiene la etiqueta <PUNTOS_INICIALES>, que se divide en etiquetas <PUNTO>, que a su vez se dividen en <X> e <Y>.

Gen 2: Evolución de los diámetros de los círculos. Contiene la etiqueta <EVOLUCIONES_DIAMETRO> que se divide en etiquetas <EVOLUCION> (más adelante profundizaremos esta etiqueta).

Gen 3: Evolución de las posiciones, traducible en la evolución de dos variables, X e Y. Contiene la etiqueta <EVOLUCIONES_XY>, que se divide en etiquetas <PAR>, que a su vez poseen una etiqueta <EVOLUCION>.

En el gen 2 y 3 hay etiquetas <EVOLUCION>, las cuales se encargan de crear evoluciones dinámicas de parámetros, tales como las que describimos en el apartado anterior. Esta etiqueta fue pensada para poder elegir entre diferentes tipos de curvas.

Por ejemplo, el siguiente fragmento de XML produce un secuencia con valores constantes, como la que muestra el gráfico inmediatamente debajo del mismo:

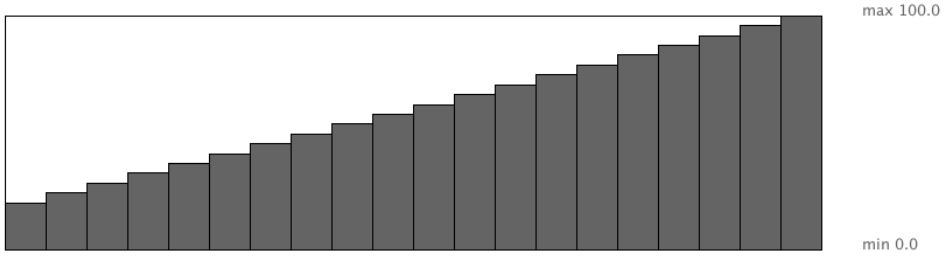
```
<EVOLUCION>
      <TIPO> constante </TIPO>
      <VALOR> 100 </VALOR>
</EVOLUCION>
```

Código XML de la Evolución tipo Constante

El gráfico se muestra en forma segmentada, ya que cada columna representa un valor discreto de la curva. Para hacer una evolución lineal se puede usar el XML que figura a continuación en el que se especifica el valor inicial y el final:

```
<EVOLUCION>
      <TIPO> linea </TIPO>
      <DESDE> 20 </DESDE>
      <HASTA> 100 </HASTA>
</EVOLUCION>
```

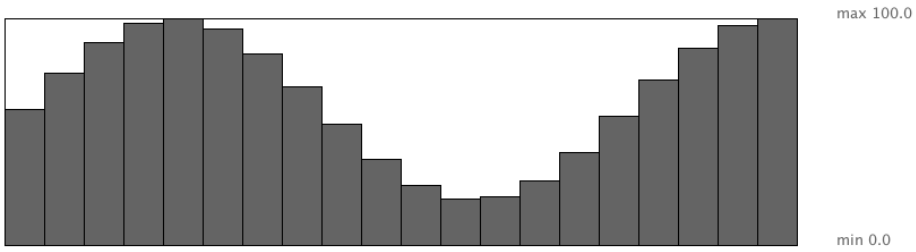
Código XML de la Evolución tipo Línea



Otra posibilidad es crear una curva de forma sinusoidal. Para hacer una evolución sinusoidal hay que especificar un mínimo y un máximo y los ángulos (en grados) que determinan los límites de la porción de la curva sinusoidal que se quiere utilizar en la evolución:

```
<EVOLUCION>
      <TIPO> seno </TIPO>
      <DESDE> 20 </DESDE>
      <HASTA> 100 </HASTA>
      <ANG1> 0 </ANG1>
      <ANG2> 450 </ANG2>
</EVOLUCION>
```

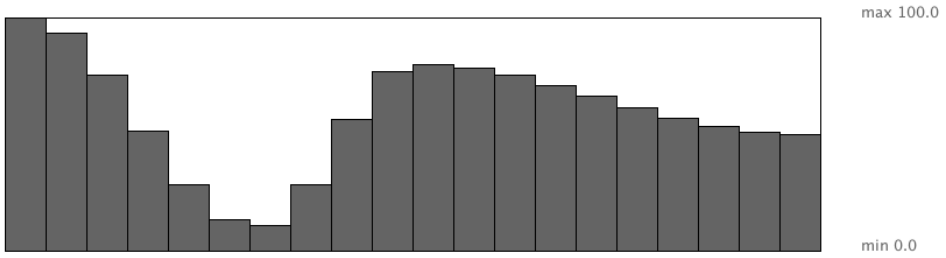
Código XML de la Evolución tipo Seno



Por último, existe una forma de evolución a la que denominé “envolvente” y que consiste en una secuencia de evoluciones lineales. En el XML de abajo se muestra una envolvente que está determinada por cuatro puntos, cada punto es una etiqueta <P>, que contiene una etiqueta <T> que indica el lugar en el tiempo (eje x) en forma normalizada, es decir que va desde cero hasta uno, por ejemplo la mitad del período es 0,5. Cada punto tiene también una etiqueta <V> que muestra el valor en ese punto.

```
<EVOLUCION>
      <TIPO> envolvente </TIPO>
      <PUNTOS>
            <P>
                  <T> 0.0 </T>
                  <V> 100 </V>
            </P>
            <P>
                  <T> 0.3 </T>
                  <V> 10 </V>
            </P>
            <P>
                  <T> 0.5 </T>
                  <V> 80 </V>
            </P>
            <P>
                  <T> 1.0 </T>
                  <V> 50 </V>
            </P>
      </PUNTOS>
</EVOLUCION>
```

Código XML de la Evolución tipo Envolvente



A continuación mostraré un XML completo que corresponde forma del gráfico de abajo. En el XML se podrá ver que el GEN 0 define 25 estratos para los planos seriados. En GEN 1 indica que hay dos puntos iniciales, es decir que hay dos líneas evolutivas de círculos. En el GEN 2 los diámetros evolucionan con curvas sinusoidales con diferentes fases. El GEN 3 mantiene a los círculos en su lugar con una curva constante.

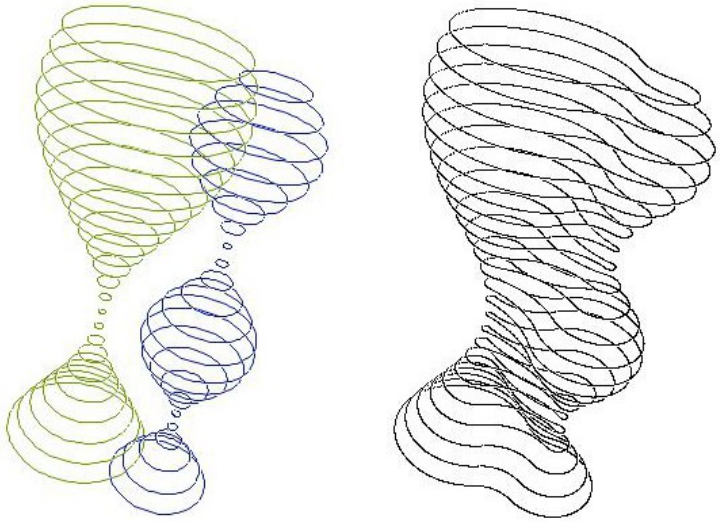


Imagen de la forma definida por el XML detallado a continuación

```
<?xml version="1.0"?>
<ADN>

	<PADRES>
		<P>ADN_06_13h14_25_57r67.xml</P>
		<M>ADN_06_11h01_36_42r86.xml</M>
		<CADENA>pppM</CADENA>
	</PADRES>

	<GEN  id = "0">
		<CANT_PLANOS>25</CANT_PLANOS>
	</GEN>

	<GEN  id = "1">
		<PUNTOS_INICIALES>
			<PUNTO>
				<X>-100.0</X>
				<Y>-100.0</Y>
			</PUNTO>
			<PUNTO>
				<X>100.0</X>
				<Y>100.0</Y>
			</PUNTO>
		</PUNTOS_INICIALES>
	</GEN>

	<GEN  id = "2">
		<EVOLUCIONES_DIAMETRO>
			<EVOLUCION>
				<TIPO>seno</TIPO>
				<DESDE>10.0</DESDE>
				<HASTA>250.0</HASTA>
				<ANG1>45</ANG1>
				<ANG2>400</ANG2>
			</EVOLUCION>
			<EVOLUCION>
				<TIPO>seno</TIPO>
				<DESDE>10.0</DESDE>
```

```
                        <HASTA>150.0</HASTA>
                        <ANG1>0</ANG1>
                        <ANG2>780</ANG2>
                  </EVOLUCION>
            </EVOLUCIONES_DIAMETRO>
      </GEN>

      <GEN  id = "3">
            <EVOLUCIONES_XY>
                  <PAR>
                        <EVOLUCION  id = "x">
                              <TIPO>constante</TIPO>
                              <VALOR> 0 </VALOR>
                        </EVOLUCION>
                        <EVOLUCION  id = "y">
                              <TIPO>constante</TIPO>
                              <VALOR> 0 </VALOR>
                        </EVOLUCION>
                  </PAR>
            </EVOLUCIONES_XY>
      </GEN>

</ADN>
```

Código XML del ADN

Implementación de la clase Escultura en Processing

Tal como dijimos en la sección anterior repliqué la metáfora del genotipo y el fenotipo. El genotipo está representado por el ADN configurado en este caso por el XML que acabamos de mostrar. El fenotipo en nuestra aplicación está representado por la clase Escultura que es capaz de crear la forma escultórica en función de un ADN (XML) recibido. El código que mostraremos a continuación está implementado en el lenguaje Processing. Como puede verse abajo, la clase Escultura esta constituida por una serie de miembros datos (propiedades) que se encargan de almacenar diferentes elementos.

```
class Escultura {

  EmpalmesMultiples empalmes[];
  int cantPlanos;

  Punto puntosIniciales[];
  int cantCirculos;

  VectorArbolXML evDiametro[];
  int cantEvDiametro;

  VectorArbolXML evX[];
  VectorArbolXML evY[];
  int cantEvXY;

  VectoresCirculo vc[];
...
```

Código: inicio clase Escultura

Recorrámoslos en el orden que aparecen definidos en la clase. El arreglo “empalmes” pertenece a la clase EmpalmesMultiples: esta clase genera y almacena los arcos que se empalman a partir de los círculos generadores, es decir, cada estrato de los planos seriados está representado por un objeto de tipo EmpalmesMultiples. El objeto “empalmes” al ser un arreglo de este tipo, se encarga de guardar toda la escultura. La variable “cantPlanos” simplemente almacena la cantidad de planos (estratos) de la escultura y se relaciona con el GEN 0.

El arreglo “puntosIniciales”, almacena las posiciones iniciales de los círculos generadores y la variables “cantCirculos” define la cantidad de círculos que expandirán sus líneas. Estos datos corresponden al GEN 1.
El arreglo “evDiametro” es del tipo VectorArbolXML y es el encargado de gestionar la evolución de los diámetros de los círculos durante su proyección (desarrollo) en los planos seriados. Mientras que la variable “cantEvDiametro” determina la cantidad de curvas de evolución que hay.
Un aspecto importante a tener en cuenta es que la cantidad de curvas de evolución no coinciden necesariamente con la cantidad de círculos iniciales, ya que este parámetro está pensado para combinarlo independientemente del otro. Así, si la cantidad de curvas es mayor a la cantidad de círculos, sólo se asignan las necesarias y en orden. Mientras que si la cantidad de evoluciones es menor, se repiten en forma cíclica, hasta cubrir todos los círculos. Por ejemplo, si la cantidad de evoluciones (en este caso de los diámetros) son 3 (las nombrares A,B y C para identificarlas) y la cantidad de círculos es sólo dos, entonces se asignarán sólo A y B. Debajo se muestra la forma que se asignarían tres evoluciones a diferentes cantidades de círculos:

Cantidad de Círculos	Asignaciones
2	A – B
3	A – B – C
4	A – B – C – A
5	A – B – C – A – B
6	A – B – C – A – B – C
7	A – B – C – A – B – C – A

Tabla: ejemplo de asignaciones de 3 parámetros según la cantidad de círculos

El tipo de dato VectorArbolXML será explicado más adelante en profundidad, pero en principio diremos que se encargar de levantar el tipo de evolución del XML y de aplicarla a un arreglo de números.
Los arreglos “evX” y “evY[]” también son del tipo VectorArbolXML y se encargan de almacenar y gestionar las evoluciones de las posiciones de los círculos en su desarrollo. La variable “cantEvXY” define la cantidad de evoluciones vinculadas a la posición.

Tal como dijimos antes, la clase Escultura se encarga de recibir un ADN y seguir sus órdenes (configuraciones) para construir la forma, por esto es de esperarse que su constructor tenga dos funciones principales.

```
Escultura( ArbolXML adn ) {

      cargarADN( adn );
      armarEstructura();
}
```

Código: Constructor clase Escultura

Traduciendo el ADN (XML) a vectores

La primera función, “cargarADN()”, se encarga de recibir el XML del ADN y decodificarlo, es decir de configurar las estructuras de datos internas al objeto y prepararlas para la construcción de la forma. La segunda función, “armarEstructura()” es la que construye la forma a partir de los parámetros decodificados por la primera función.

```
void cargarADN( ArbolXML adn ) {

      iniciarGenCERO(  devolverGen( adn, "0" )  );
```

```
        iniciarGenUNO(  devolverGen( adn, "1" )  );

        iniciarGenDOS(  devolverGen( adn, "2" )  );

        iniciarGenTRES(  devolverGen( adn, "3" )  );
}
```

Código: método cargarADN() de la clase Escultura

A su vez, el método “cargarADN()” se encarga de cortar la cadena total, y separarla en cada uno de los genes. Los métodos “iniciarGenCERO()”, “iniciarGenUNO()”, “iniciarGenDOS()”, e “iniciarGenTRES()”, toman las respectivas porciones de ADN y configuran sus variables internas. Los cortes del XML lo hacen gracias al método “devolverGen()”, el cuál recibe el XML (llamado “adn” aquí) y la etiqueta que identifica al gen en cuestión.

```
ArbolXML devolverGen( ArbolXML adn, String identidad ) {

        ArbolXML resultado = buscarHijoConNombreId( adn,
              "GEN", identidad );

        return resultado;
}
```

Código: método devolverGen() de la clase Escultura

El método “devolverGen()” utiliza un objeto de tipo ArbolXML. En el momento de la implementación de esta clase, la versión de Processing que utilicé fue la 1.5.1 que no permitía editar los XML o construirlos, sólo leerlos, por eso cree esta clase que es capaz de hacer todas estas cosas. Por ejemplo, la función “buscarHijoConNombreId()” permite a partir de un XML, buscar una etiqueta determinada que tenga una propiedad “id” con una valor específico.
Como se mostró en el ejemplo de XML, se puede ver que las etiquetas <GEN> poseen un atributo “id” que determina el número de gen:

```
<GEN  id = "2">
```

En la línea siguiente la función “buscarHijoConNombreId()” recibe un XML llamado “adn” y busca una etiqueta <GEN> que tenga un atributo “id” igual al valor que figura en la variable “identidad”:

```
ArbolXML resultado = buscarHijoConNombreId( adn,
        "GEN", identidad );
```

Por otra parte los métodos “iniciarGen...()”, como el método “iniciarGenCERO()”, recibe la porción de ADNy busca etiqueta específicas para cargar sus correspondientes variable, en el caso del GEN 0, la variable “cantPlanos”:

```
void iniciarGenCERO( ArbolXML adn ) {
        if ( adn != null ) {
                cantPlanos = valorIntHijo( adn, "CANT_PLANOS" );
        }
        else {
                println("ERROR: falta xml GEN 0");
        }
}
```

Código: método “iniciarGenCERO()” de la clase “Escultura”

Así “iniciarGenUNO()” carga los valores de las variables “puntosIniciales[]” y “ cantCirculos”. El método “iniciarGenDOS()” carga los valores del arreglo “evDiametro[]” y la variable “cantEvDiametro”. Y por último “iniciarGenTRES()” carga los arreglos “evX[]” y “evY[]” y la variable “cantEvXY”. Una vez ejecutados estos métodos, la carga del ADN termina y todas las estructuras de datos están cargadas con sus correspondientes valores.

Vamos a revisar un caso concreto paso por paso, primero presentaremos el código XML completo y luego lo desarmaremos para ver la función de cada parte.

```
<?xml version="1.0"?>
<ADN>
	<PADRES>
		<P>ADN_06_13h14_25_57r67.xml</P>
		<M>ADN_06_11h01_36_42r86.xml</M>
		<CADENA>pppM</CADENA>
	</PADRES>

	<GEN  id = "0">
		<CANT_PLANOS>15</CANT_PLANOS>
	</GEN>

	<GEN  id = "1">
		<PUNTOS_INICIALES>
			<PUNTO>
				<X>-100.0</X>
				<Y>-100.0</Y>
			</PUNTO>
			<PUNTO>
				<X>100.0</X>
				<Y>100.0</Y>
			</PUNTO>
			<PUNTO>
				<X>-200.0</X>
				<Y>100.0</Y>
			</PUNTO>
		</PUNTOS_INICIALES>
	</GEN>

	<GEN  id = "2">
		<EVOLUCIONES_DIAMETRO>
			<EVOLUCION>
				<TIPO>constante</TIPO>
				<VALOR> 100 </VALOR>
			</EVOLUCION>
			<EVOLUCION>
				<TIPO>seno</TIPO>
				<DESDE>20.0</DESDE>
				<HASTA>150.0</HASTA>
				<ANG1>45</ANG1>
				<ANG2>400</ANG2>
			</EVOLUCION>
			<EVOLUCION>
				<TIPO>linea</TIPO>
				<DESDE>10</DESDE>
				<HASTA>300</HASTA>
			</EVOLUCION>
		</EVOLUCIONES_DIAMETRO>
	</GEN>

	<GEN  id = "3">
		<EVOLUCIONES_XY>
			<PAR>
				<EVOLUCION  id = "x">
					<TIPO>seno</TIPO>
					<DESDE>-150.0</DESDE
							<HASTA>150.0</HASTA>
					<ANG1>45</ANG1>
```

```
                        <ANG2>400</ANG2>
                    </EVOLUCION>
                    <EVOLUCION  id = "y">
                        <TIPO>linea</TIPO>
                        <DESDE>0</DESDE>
                        <HASTA>-300</HASTA>
                    </EVOLUCION>
                </PAR>
                <PAR>
                    <EVOLUCION  id = "x">
                        <TIPO>linea</TIPO>
                        <DESDE>0</DESDE>
                        <HASTA>100</HASTA>
                    </EVOLUCION>
                    <EVOLUCION  id = "y">
                        <TIPO>seno</TIPO>
                        <DESDE>-150.0</DESDE>
                        <HASTA>100.0</HASTA>
                        <ANG1>300.24396</ANG1>
                        <ANG2>705.0759</ANG2>
                    </EVOLUCION>
                </PAR>
            </EVOLUCIONES_XY>
        </GEN>
    </ADN>
```

Código: segundo ejemplo de XML de un ADN

Empecemos por revisar el primer <GEN>, que es el que define la cantidad de planos de la estructura:

```
...
<GEN  id = "0">
    <CANT_PLANOS>15</CANT_PLANOS>
</GEN>
...
```

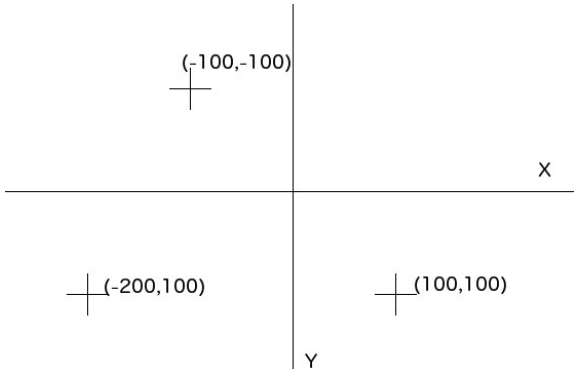
Este <GEN> inicializa la variable “cantPlanos” que adopta el valor 15. Esto se hace con el método “iniciarGenCERO()“ que mostramos anteriormente.
Si vemos el segundo <GEN> que mostramos debajo, veremos que este se encarga de establecer las posiciones iniciales de los círculos. Estos valores se cargan en el arreglo “puntosIniciales[ ]”:

```
...
<GEN  id = "1">
    <PUNTOS_INICIALES>
        <PUNTO>
            <X>-100.0</X>
            <Y>-100.0</Y>
        </PUNTO>
        <PUNTO>
            <X>100.0</X>
            <Y>100.0</Y>
        </PUNTO>
        <PUNTO>
            <X>-200.0</X>
            <Y>100.0</Y>
        </PUNTO>
    </PUNTOS_INICIALES>
</GEN>
...
```

En este caso la operación la realiza el método “iniciarGenUNO()”, que establece que la variable

“cantCirculos” sería igual a 3 y el arreglo “puntosIniciales[ ]” se cargaría de la siguiente forma:

puntosIniciales
[0]-> Punto[X = -100 Y = -100]
[1]-> Punto[X = 100 Y = 100]
[2]-> Punto[X = -200 Y = 100]

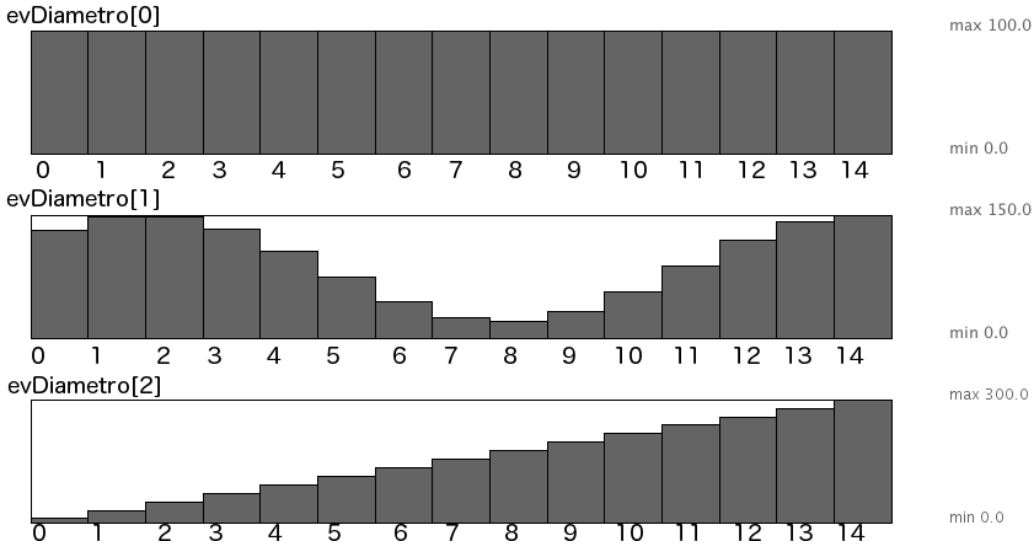


Esquema: posiciones iniciales de los círculos

En el caso del GEN 2, que es gestionado por “iniciarGenDOS()”, se carga el arreglo “evDiametro[ ]” con las curvas indicadas por cada etiqueta <EVOLUCION>:

```
...
<GEN  id = "2">
      <EVOLUCIONES_DIAMETRO>
            <EVOLUCION>
                  <TIPO>constante</TIPO>
                  <VALOR> 100 </VALOR>
            </EVOLUCION>
            <EVOLUCION>
                  <TIPO>seno</TIPO>
                  <DESDE>20.0</DESDE>
                  <HASTA>150.0</HASTA>
                  <ANG1>45</ANG1>
                  <ANG2>400</ANG2>
            </EVOLUCION>
            <EVOLUCION>
                  <TIPO>linea</TIPO>
                  <DESDE>10</DESDE>
                  <HASTA>300</HASTA>
            </EVOLUCION>
      </EVOLUCIONES_DIAMETRO>
</GEN>
...
```

En el ejemplo de arriba pueden verse tres etiquetas <EVOLUCION> que corresponden a una evolución de tipo constante, sinusoidal y lineal, respectivamente. Una vez ejecutado el método, “evDiametro[ ]” queda cargado con las siguientes curvas:



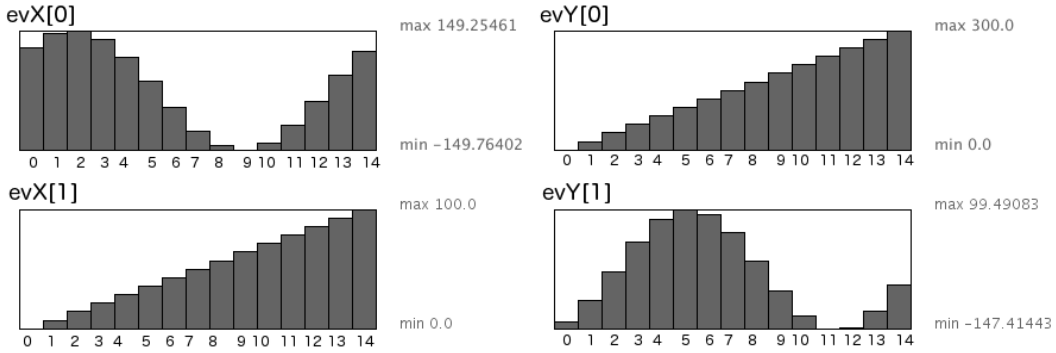
Esquema: carga del arreglo “evDiametro[ ]” luego de ejecutar “iniciarGenDOS()”

Cabe aclarar que el arreglo “evDiametro[ ]” tiene en cada una de sus celdas un objeto de tipo “VectorArbolXML”. En rigor, estos objetos no guardan el vector de valores, sino el XML con la configuración necesaria para construir el vector, pero este sólo se construye cuando se invoca al método “crearVector( )”. De todos modos, a fines de hacer más sencillo el recorrido, diremos que el vector queda cargado con dichos valores.
El <GEN> 3 contiene la evolución de las posiciones de los círculos para cada estrato de los planos seriados. Esta información es gestionada por el método “iniciarGenTRES()” y se organiza por pares, es decir que va una evolución en la coordenada X acompañada de otra en Y. En el caso de la porción de código citada debajo, el <GEN> 3 contiene dos pares, el primero en que X evoluciona con una curva sinusoidal e Y con una lineal, mientras que en el segundo par X evoluciona con otra linea e Y con un seno. Estos datos son cargados en los arreglos “evX[ ]” y “evY[ ]”, ambos pertenecientes a la clase “VectorArbolXML”.

```
...
<GEN  id = "3">
      <EVOLUCIONES_XY>
            <PAR>
                  <EVOLUCION  id = "x">
                        <TIPO>seno</TIPO>
                        <DESDE>-150.0</DESDE
                        <HASTA>150.0</HASTA>
                        <ANG1>45</ANG1>
                        <ANG2>400</ANG2>
                  </EVOLUCION>
                  <EVOLUCION  id = "y">
                        <TIPO>linea</TIPO>
                        <DESDE>0</DESDE>
                        <HASTA>-300</HASTA>
                  </EVOLUCION>
            </PAR>
            <PAR>
                  <EVOLUCION  id = "x">
                        <TIPO>linea</TIPO>
                        <DESDE>0</DESDE>
                        <HASTA>100</HASTA>
                  </EVOLUCION>
                  <EVOLUCION  id = "y">
                        <TIPO>seno</TIPO>
                        <DESDE>-150.0</DESDE>
                        <HASTA>100.0</HASTA>
```

```
                                    <ANG1>300.24396</ANG1>
                                    <ANG2>705.0759</ANG2>
                                </EVOLUCION>
                            </PAR>
                        </EVOLUCIONES_XY>
                    </GEN>
                    ...
```

En el esquema de abajo se muestra como quedan cargados los arreglos “evX[ ]” y “evY[ ]” luego de la ejecución del método “iniciarGenTRES( )”:



Esquema: carga de los arreglos “evX[ ]” y “evY[ ]” luego de ejecutar “iniciarGenTRES()”

Construyendo la estructura

Luego de que se ha logrado cargar el ADN en las estructuras de datos de la clase “Escultura”, se ejecuta el método “armarEstructura()”, este método se encarga de construir, uno por uno, los planos seriados. Cada plano de la estructura es un objeto de la clase “EmpalmesMultiples”. Cada estrato se construye a partir de una conjunto de círculos empalmados entre sí. El método encargado de realizar la configuración de cada estrato es el llamado “armarEstructura()”, que se muestra debajo:

```
void armarEstructura() {

      cargarVectores();

      empalmes = new EmpalmesMultiples[ cantPlanos ];

      for ( int i=0 ; i<cantPlanos ; i++ ) {

            Circulo c[] = new Circulo[ cantCirculos ];
            for ( int j=0 ; j<cantCirculos ; j++ ) {
                  c[j] = new Circulo(
                        puntosIniciales[j].x + vc[j].x[i],
                        puntosIniciales[j].y + vc[j].y[i],
                        vc[j].d[i]
                  );
            }

            empalmes[i] = new EmpalmesMultiples( c, radians(60) );
      }
}
```

Código: método “armarEstructura()” de la clase “Escultura”

Entre las primeras líneas de este método se encuentra “cargarVectores()“, qué es un método clave de la

clase, ya que se encarga de traducir los parámetros obtenidos del ADN (cargado en las variables y arreglos descriptos en la sección anterior) en parámetros de los círculos, realizando las vinculaciones entre estas estructuras de datos.
Una de las principales dificultades que enfrenta este algoritmo, es el de conciliar estructura que tiene distintas cantidades de datos, por ejemplo, en el caso recién mostrado, la cantidad de círculos por planos es tres, al igual que la cantidad de evoluciones de diámetro pero, en cambio, la cantidad de pares de evolución de X e Y es de dos. Esta situación se produce debido a que en el entrecruzamiento de genes que se produce en la reproducción, cada gen viene de una padre diferente y por ende posee diferentes cantidades. Esto ya se mostró en la tabla “ejemplo de asignaciones de 3 parámetros según la cantidad de círculos”, expuesto algunas páginas atrás.
Debajo podemos ver el método “cargarVectores()”:

```
void cargarVectores() {

      vc = new VectoresCirculo[ cantCirculos ];

      if ( cantEvDiametro>0 ) {
            if ( cantEvXY>0 ) {

                  for ( int i=0 ; i<cantCirculos ; i++ ) {

                        vc[i] = new VectoresCirculo();
                        int cual = i % cantEvDiametro;
                        int cualXY = i % cantEvXY;

                        vc[i].d =
                        evDiametro[ cual ].crearVector( cantPlanos );
                        vc[i].x = evX[ cualXY ].crearVector( cantPlanos );
                        vc[i].y = evY[ cualXY ].crearVector( cantPlanos );
                  }
            }
            else {
                  println("ERROR: faltan EvXY");
            }
      }
      else {
            println("ERROR: faltan EvDiametro");
      }
}
```

Código: método “cargarVectores()” de la clase “Escultura”

El método se ocupa de armar las líneas de evolución para los círculos que intervienen en la forma de los planos. En el caso que venimos revisando, la cantidad de círculos es tres, por eso el “ciclo-for i” se encarga de construir tres líneas. Para poder pasar los parámetros a la clase “EmpalmeMultiples”, el método “cargarVectores( )” se encarga de cargar un arreglo llamado “vc”, que pertenece a la clase “VectoresCirculo” que se muestra debajo:

```
class VectoresCirculo {

      float x[];
      float y[];
      float d[];

      VectoresCirculo() {
      }
}
```

Código: clase “VectoresCirculo”

La clase “VectoresCirculo” posee un arreglo para guardar la evolución del diámetro y otro dos para las

evoluciones de la posición. En las siguientes líneas, extraídas del método, se usan los arreglos “evDiametro[ ]” , “evX[ ]” y “evY[ ]” (que son de la clase “VectorArbolXML”) para crear los valores de los vectores “d”, “x” e “y” de la clase “VectoresCirculo”:

```
...
vc[i].d = evDiametro[ cual ].crearVector( cantPlanos );
vc[i].x = evX[ cualXY ].crearVector( cantPlanos );
vc[i].y = evY[ cualXY ].crearVector( cantPlanos );
...
```

En dichas líneas, el contador “i” designa el número de círculo, mientras que las variables “cual” y “cualXY” designa de cuáles evoluciones se toman los valores de estos arreglos. En el caso que estamos revisando, la variables “i” adoptará valores entre 0 y 2 (inclusive), al igual que la variable “cual”, pero “cualXY” sólo adopta dos valores, 0 y 1. Como se puede ver, se ejecuta el método “crearVector( )” para cargar los valores del arreglo destino: “vc[ ]”. Debajo se muestra una tabla con los valores de los vectores “x[ ]”, “y[ ]” y “d[ ]” de cada uno de los elementos “VectoresCirculos” (llamados aquí Circulo 0, 1 y 2) del arreglo “vc[ ]”. Notaremos que los arreglos “x[ ]” e “y[ ]” de los círculos 0 y 2 coinciden ya que provienen de las mismas curvas de evolución:

planos	Circulo 0			Circulo 1			Circulo 2		
	d	x	y	d	x	y	d	x	y
[0]->	100,00	106,07	0,00	130,96	0,00	-132,99	10,00	106,07	0,00
[1]->	100,00	141,27	-21,43	146,22	7,14	-89,08	30,71	141,27	-21,43
[2]->	100,00	149,25	-42,86	149,68	14,29	-29,19	51,43	149,25	-42,86
[3]->	100,00	128,48	-64,29	140,67	21,43	31,74	72,14	128,48	-64,29
[4]->	100,00	82,95	-85,71	120,94	28,57	78,52	92,86	82,95	-85,71
[5]->	100,00	21,43	-107,14	94,29	35,71	99,49	113,57	21,43	-107,14
[6]->	100,00	-44,21	-128,57	65,84	42,86	89,42	134,29	-44,21	-128,57
[7]->	100,00	-101,34	-150,00	41,09	50,00	50,82	155,00	-101,34	-150,00
[8]->	100,00	-138,94	-171,43	24,79	57,14	-6,69	175,71	-138,94	-171,43
[9]->	100,00	-149,76	-192,86	20,10	64,29	-68,76	196,43	-149,76	-192,86
[10]->	100,00	-131,73	-214,29	27,92	71,43	-119,92	217,14	-131,73	-214,29
[11]->	100,00	-88,32	-235,71	46,73	78,57	-147,41	237,86	-88,32	-235,71
[12]->	100,00	-27,89	-257,14	72,92	85,71	-144,38	258,57	-27,89	-257,14
[13]->	100,00	37,92	-278,57	101,43	92,86	-111,58	279,29	37,92	-278,57
[14]->	100,00	96,42	-300,00	126,78	100,00	-57,19	300,00	96,42	-300,00

Tabla de valores del arreglo “vc[ ]” una vez cargado por el método “cargarVectores()”

Cada línea de la tabla de arriba representa un plano de la forma, en la que se describe la posición (en X e Y) y el diámetro de tres círculos. Las columnas representan la evolución de cada parámetro, por ejemplo, la tercer columna representa la evolución de la variable Y del primer círculo, que debido a una curva lineal evoluciona de 0 a -300. De esta forma, si tomamos la primer línea (que tiene índice 0) y construimos los círculos y sus empalmes obtenemos este plano:

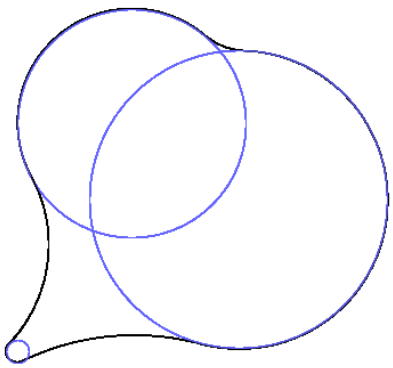


Imagen del primer estrato obtenido a partir de los parámetros

Mientras que si tomamos la sexta línea (que tiene índice 5) obtenemos la siguiente forma:

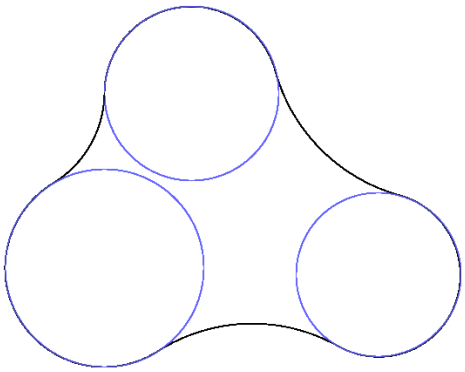
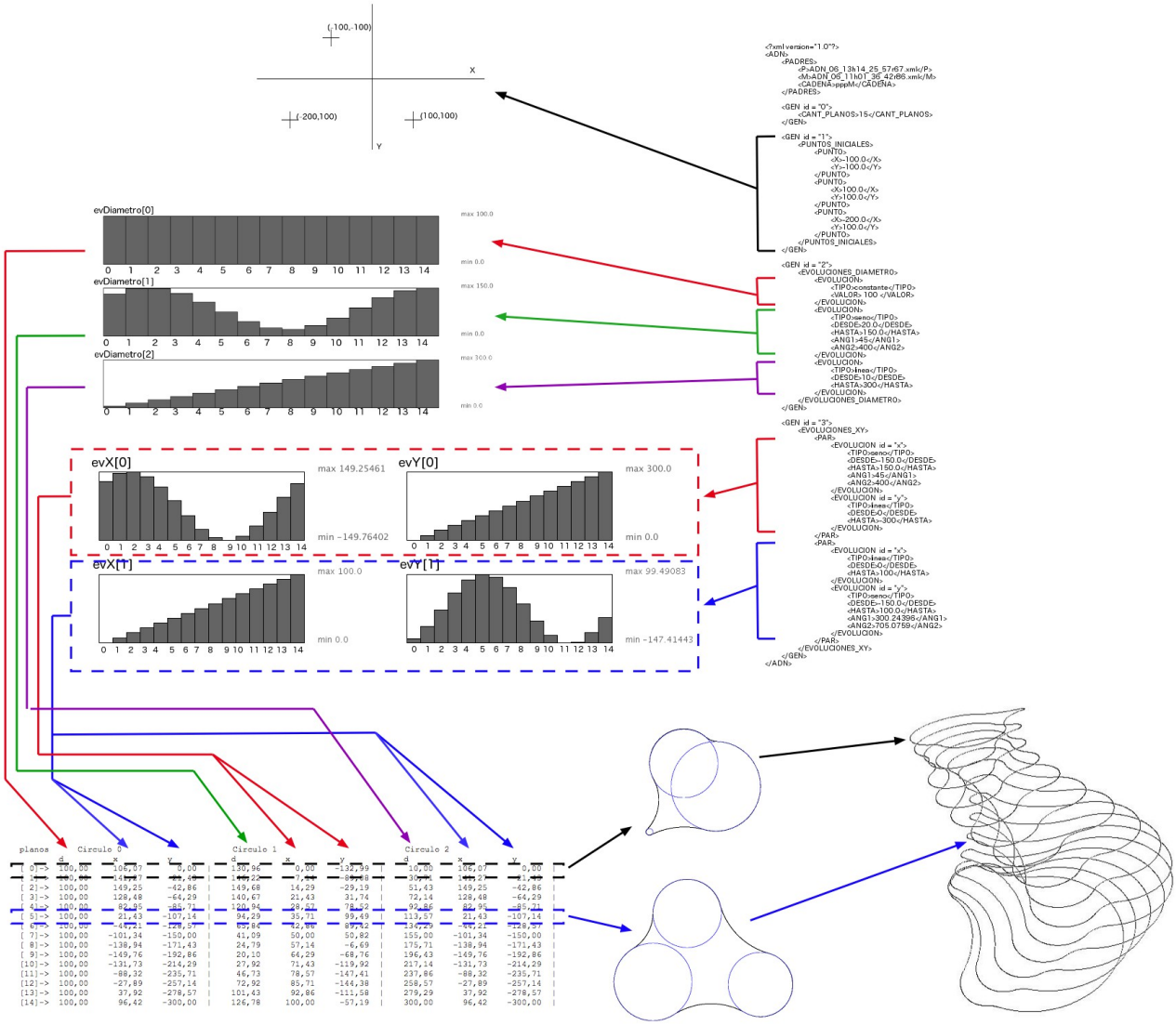


Imagen del sexto estrato obtenido a partir de los parámetros

Por último, mostramos a continuación un esquema del proceso completo de transformación (traducción) desde la información del ADN (codificado en XML) hasta la forma final por planos seriados. En dicho esquema puede observarse como de cada <GEN> se obtienen las curvas de evolución que luego completan los parámetros que determinan la construcción de los planos:



Esquema completo de la construcción de la forma a partir del ADN

Entrecruzamiento genético

Una vez que se ha logrado organizar el ADN como se mostró en las secciones anteriores, la tarea de entrecruzamiento se facilita, ya que para obtener un espécimen mezcla de otros dos, sólo es necesario elegir un <GEN> al azar de cualquiera de los padres.

```
ArbolXML cruzarGenes( int padre, int madre ) {

  ArbolXML nuevoAdn = new ArbolXML( "ADN" );

  ArbolXML genesPadre = gestores[ padre ].adn;
  ArbolXML genesMadre = gestores[ madre ].adn;


  ArbolXML hijo = new ArbolXML( "PADRES" );

  ArbolXML nieto = new ArbolXML( "P" );
  nieto.agregarContenido( gestores[ padre ].archivo );
  hijo.agregarHijo( nieto );

  nieto = new ArbolXML( "M" );
  nieto.agregarContenido( gestores[ madre ].archivo );
  hijo.agregarHijo( nieto );

  String cadena = "";

  for ( int i=0 ; i<4 ; i++ ) {
    cadena += ( random( 100 )<50 ? 'p' : 'M' );
  }

  nieto = new ArbolXML( "CADENA" );
  nieto.agregarContenido( cadena );
  hijo.agregarHijo( nieto );

  nuevoAdn.agregarHijo( hijo );

  ArbolXML esteGen = null;

  for ( int i=0 ; i<4 ; i++ ) {

    String etiqueta = i+"";

    if ( cadena.charAt(i) == 'p' ) {
      esteGen = buscarHijoConNombreId( genesPadre, "GEN", etiqueta );
    }
    else {
      esteGen = buscarHijoConNombreId( genesMadre, "GEN", etiqueta );
    }
    nuevoAdn.agregarHijo( esteGen );
  }

  return nuevoAdn;
}
```

Código del método cruzarGenes

En el código de arriba podemos ver el método que se encarga de combinar genéticamente dos especímenes. Tal como se muestra en el recorte de abaja, una de las partes principales es la que elige al azar de cuál de los dos padres tomará cada gen. Esta información luego la guarda para poder seguir la forma en que se originó la herencia:

```
...
for ( int i=0 ; i<4 ; i++ ) {
  cadena += ( random( 100 )<50 ? 'p' : 'M' );
}
...
```

En el siguiente recorte del código se ve la forma en que se construye un nuevo XML (usando la clase ArbolXML) tomando uno por uno genes (al azar) del padre o la madre:

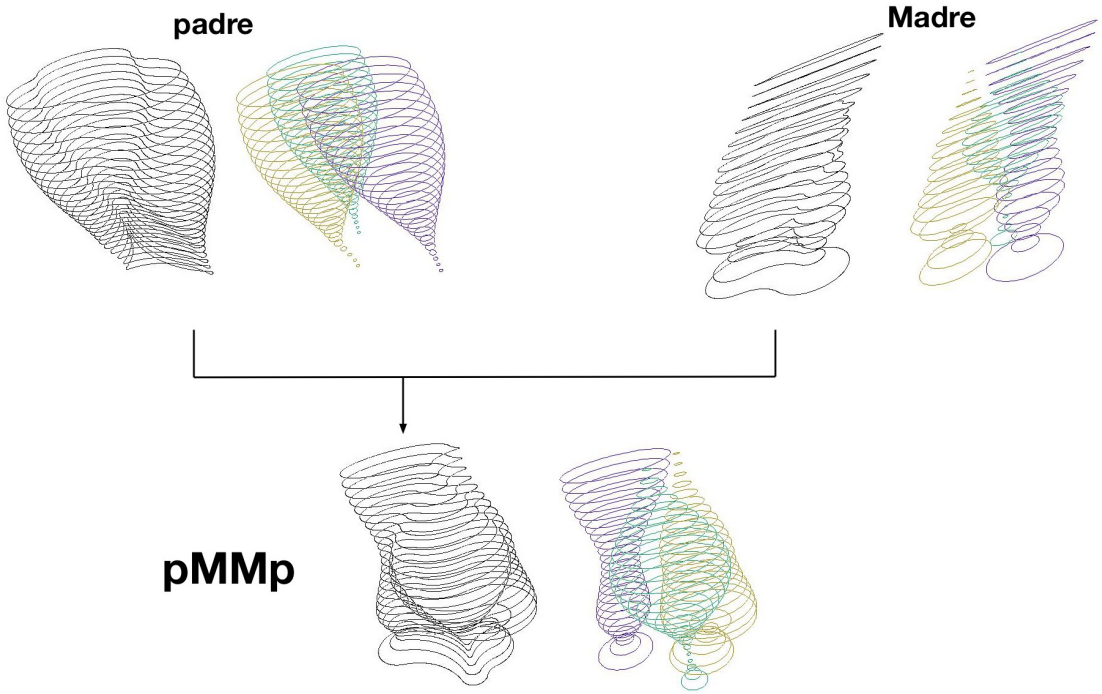
```
...
ArbolXML esteGen = null;

for ( int i=0 ; i<4 ; i++ ) {

  String etiqueta = i+"";

  if ( cadena.charAt(i) == 'p' ) {
    esteGen = buscarHijoConNombreId( genesPadre, "GEN", etiqueta );
  }
  else {
    esteGen = buscarHijoConNombreId( genesMadre, "GEN", etiqueta );
  }
  nuevoAdn.agregarHijo( esteGen );
}
...
```

En el esquema de abajo se puede ver el proceso de herencia a partir del entrecruzamiento (crossover) entre los genes de dos formas. Aclaro nuevamente, la distinción ente padre y madre sólo es útil a los fines de identificar el origen de los datos en la cruza. Por eso, al lado del “hijo”, notaremos que se encuentra la cadena “pMMp” que indica el orden en que fueron tomados los genes para producir este:
:

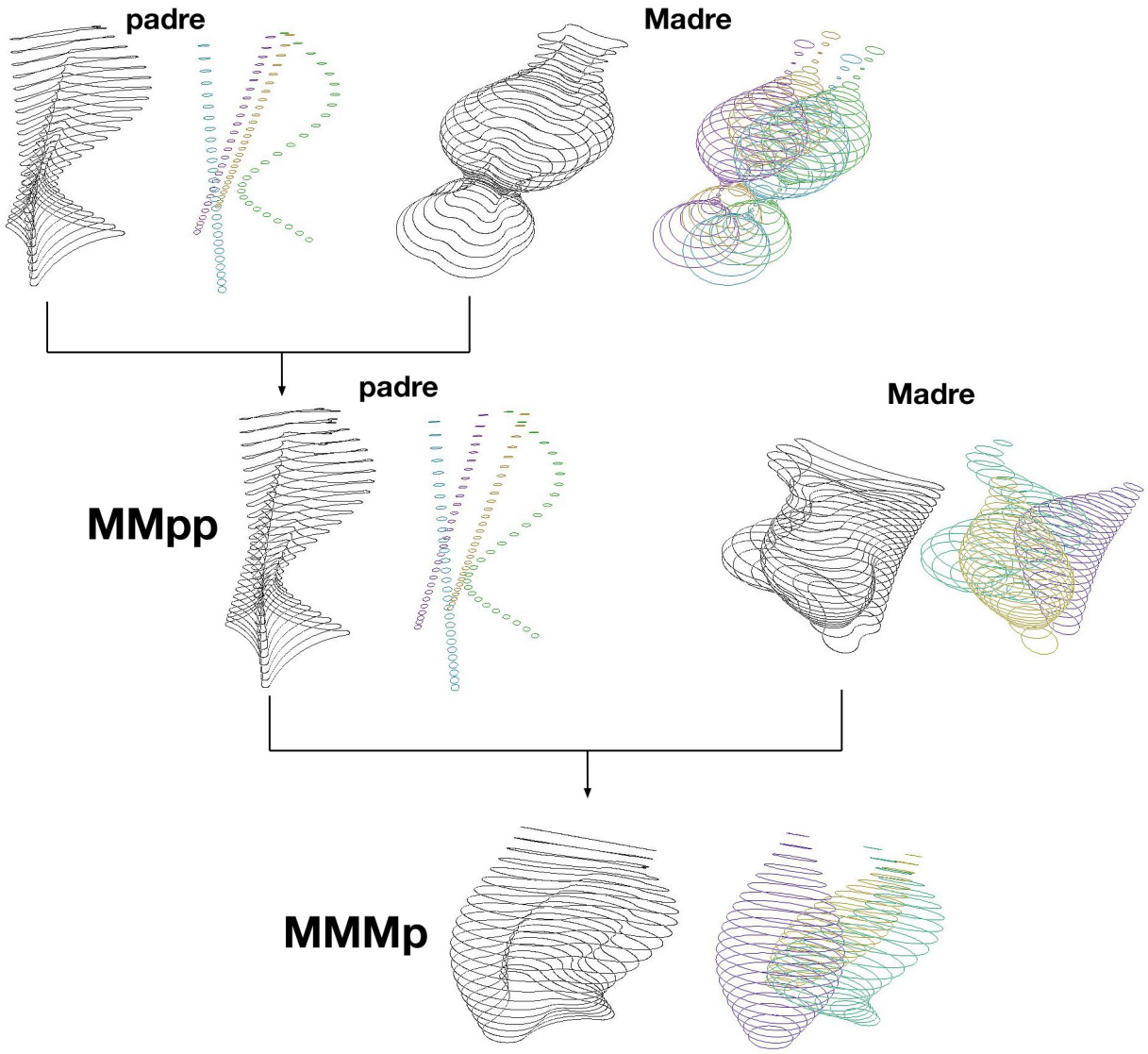


Esquema de entrecruzamiento

Esta cadena significa que el primer <GEN> fue tomado del padre, mientras que el segundo y tercer fueron tomado de la madre y el último del padre nuevamente. Es decir:

Del padre → Gen 0: Cantidad de estratos del plano seriado.
De la madre → Gen 1: Cantidad y posición inicial de los círculos.
De la madre → Gen 2: Evolución de los diámetros de los círculos.
Del padre -> Gen 3: Evolución de las posiciones, traducible en la evolución de dos variables, X e Y.

Mostraremos, a continuación, un último ejemplo de herencia:



Esquema de entrecruzamiento

El ambiente de cruza

Hasta aquí hemos visto la forma en que se codificó la información constitutiva del volumen para poder cruzarlo con otros. Una vez superada esta etapa se desarrollo una interfaz que permite ver una cantidad

determinada de esculturas y a partir de esto, seleccionar las elegidas para poder cruzarlas entre sí. Por ejemplo, en la imagen de abajo podemos ver 9 esculturas, 4 de ellas (las que poseen un círculo verde en la esquina superior izquierda de su recuadro) son las seleccionadas, y las 5 restantes (las que poseen cuadrados cruzados azules) son los resultantes de la cruza entre los 4 anteriores. El cuadrado de color rojo, indica que aún no han sido guardados (agregados) en la colección.

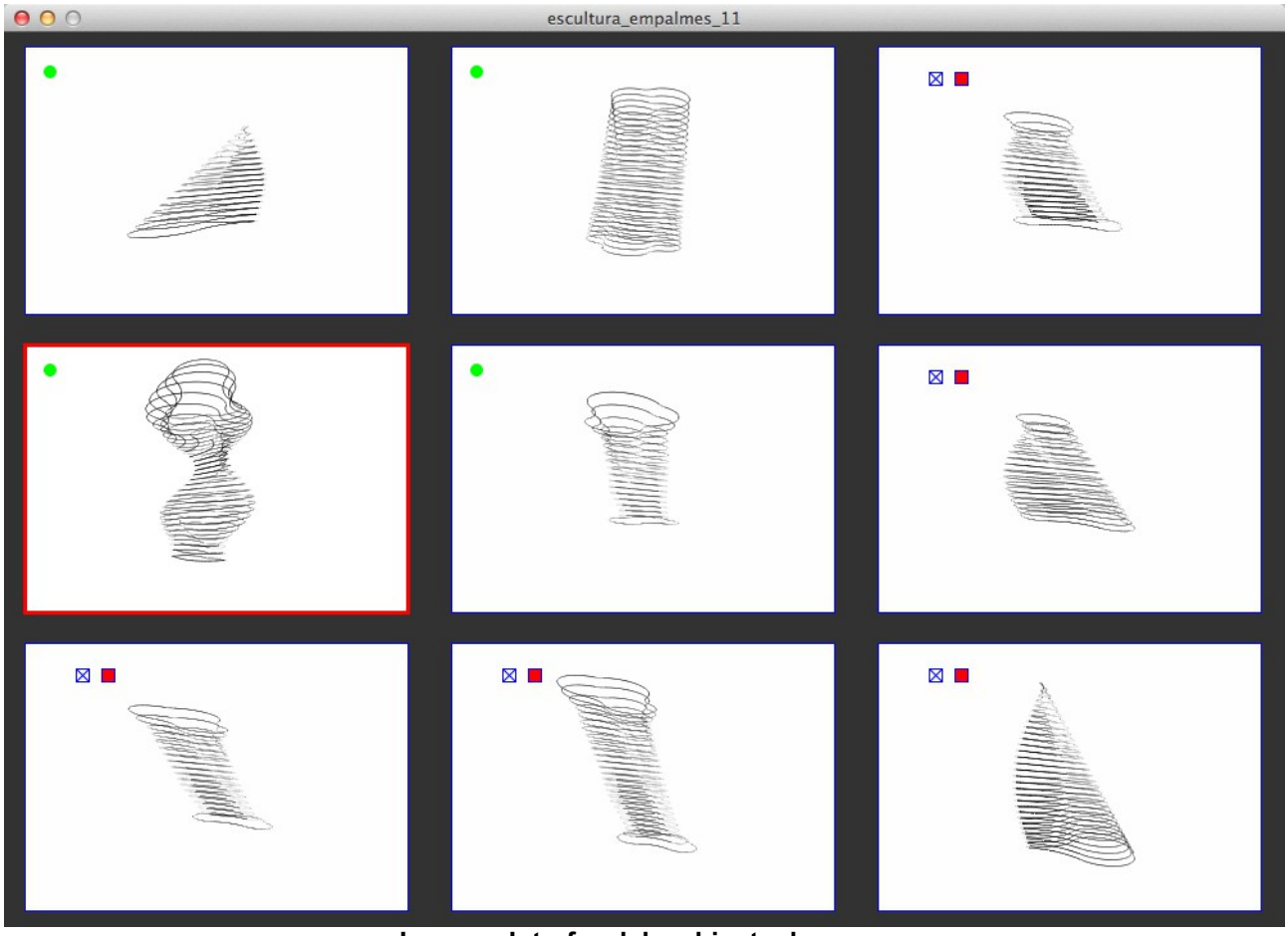


Imagen: Interfaz del ambiente de cruce

La dinámica de funcionamiento de esta interfaz es que en cada ciclo se selecciona aquellas esculturas que cumplen (o se acercan) al criterio deseado y se las cruza para suplantar a las no seleccionadas, en cada ciclo se puede redefinir la selección y volver a ejecutar la cruza para obtener nuevas. Paulatinamente se debería ir mejorando en las formas según el criterio deseado.

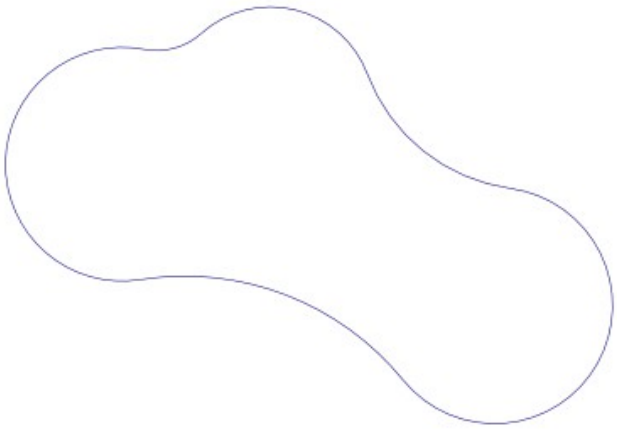
Traducción de la forma a SVG

Una vez finalizado el proceso de cruza y alcanzada la forma buscada, para poder crear la escultura a partir del diseño alcanzado, se traduce cada plano de la escultura a archivos SVG (*Scalable Vector Graphics* - Gráficos Vectoriales Redimensionables). Este formato es un tipo de archivo, de código libre y abierto, escrito en XML (lenguaje de marcas) y que constituye un standard. Uno de los principales software que los utiliza es el graficador vectorial InkScape, que es open-source, libre y multiplataforma.
La traducción de la escultura se hace escribiendo un archivo SVG por cada uno de los planos. Ya que los planos están conformados por secuencias de arcos empalmados, dicha traducción se hace escribiendo los parámetros de los arcos en el SVG. Debajo puede verse un ejemplo de un plano de una escultura escrito como SVG:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

```
<svg viewBox = "0 0 1000 700" version = "1.1">
    <g stroke = "navy" stroke-width = "1" fill = "none">
   <path d = "M 447.20108 139.45192 A 78.979546 78.979546 0 0 1 379.5034 158.27121"/>
   <path d = "M 375.15158 430.69196 A 138.31306 138.31306 0 1 1 379.50342 158.27118"/>
   <path d = "M 813.7698 322.84317 A 212.02072 212.02072 0 0 1 644.02936 187.47371"/>
   <path d = "M 447.20108 139.4519 A 122.68579 122.68579 0 0 1 644.0293 187.4737"/>
   <path d = "M 375.15158 430.69196 A 333.758 333.758 0 0 1 686.4615 549.5826"/>
   <path d = "M 813.7699 322.8432 A 139.77722 139.77722 0 1 1 686.4615 549.5825"/>
    </g>
</svg>
```

El SVG recién detallado produce la siguiente forma:



No es intención de este texto explicar el formato SVG, pero para acercarnos al tema, podemos decir que la forma de arriba está construida a partir de la concatenación de 6 arcos que producen los empalmes. Cada uno de esos arcos se representa con una etiqueta <path … > dentro del SVG. Dichas etiquetas sirven para hacer diferentes tipos de trazos a partir de sus parámetros, estas poseen los necesarios para hacer arcos a partir de establecer 6 valores numéricos: x1 e y1 (el punto inicial del arco), rx y ry (los radios vertical y horizontal del arco), x2 e y2 (el punto final). Debajo puede ver una línea del SVG y la posición de los parámetros más importantes:

```
             x1        y1          rx        ry              x2       y2
<path d = "M 447.20108 139.45192 A 78.979546 78.979546 0 0 1 379.5034 158.27121"/>
```

<path d = “M x1 y1 A rx ry rotation flag flag2 x2 y2” />

x1,y1 _ donde empieza el trazo
x2,y2 _ donde termina el trazo
rx,ry _ los radios de la elipse
rotation _ cuanto se rota en el eje x cuando los radios son diferentes
flag _ 1-es el lado largo 0-el lado corto
flag2 _ por que lado de la linea entre los puntos pasa la elipse

Tal como comentamos al principio de este texto, el formato SVG es un standard de formato vectorial, lo que permite escalar la imagen a diferentes tamaños sin perder resolución, por otra parte, existen diferentes aplicaciones que permiten traducir dicha forma al formato GCode para ser usado en dispositivos de corte como una fresa de CNC.

Conclusión

El trabajo expuesto en el presente texto está aún en desarrollo y es probable que los siguientes avances produzcan nuevas problemáticas y hallazgos, por lo que dicho texto intenta mostrar el actual estado de

avance del mismo, sin pretensiones de ser conclusivo. Quizás la pretensión del mismo ha sido la de mostrar como pasar del parámetro a la forma, de la forma a la codificación, de la codificación al proceso, del resultado del proceso a la factura física; es decir, mostrar la secuencia de procesos y transformaciones que permiten vincular los algoritmos genéticos con las formas escultóricas y a estas con el proceso de fabricación digital. Lejos de considerar esta secuencia de procedimientos, óptima, creo que aún queda mucho por modificar y mejorar. Una perspectiva más enfocada en el diseño, que en el procedimiento, revisaría seguramente los modos en que se produce la forma para proponer nuevas y más potentes estrategias, ese sería mi anhelo. Pero mi búsqueda aquí fue la de poner a prueba una metodología, que como el texto muestra, se presenta potente a la hora de diseñar con técnicas generativas. Debo para próximos trabajos abordar diseños más acabados, y espero que el actual texto sea de utilidad a otros que emprendan este camino.

2013
Emiliano Causa

Referencias

- Jakob Jenkov, “SVG Path elements”, en línea, consultado el 16/Dic/2013, http://tutorials.jenkov.com/svg/path-element.html

- “SVG Tutorial”, en línea, consultado el 16/Dic/2013, http://www.w3schools.com/svg/

- “SVG Reference”, en línea, consultado el 16/Dic/2013, http://www.w3schools.com/svg/svg_reference.asp

- “GCode Reference”, en línea, consultado el 16/Dic/2013, http://linuxcnc.org/docs/html/gcode.html

- “TxapuCNC”, en línea, consultado el 16/Dic/2013, http://txapuzas.blogspot.com.ar/2009/12/txapu-cnc-software.html

- “FabCentral”, en línea, consultado el 16/Dic/2013, http://fab.cba.mit.edu/