Experimentos para la producción de esculturas robóticas
Ariel Uzal, Emiliano Causa
Arte e Investigación (N.° 10), pp. 89-100, noviembre 2014
ISSN 2469-1488

Experimentos para la producción de esculturas robóticas

Ariel Uzal
arieluzal@gmail.com
Emiliano Causa
emiliano.causa@gmail.com

Emmelab
Facultad de Bellas Artes
Universidad Nacional de La Plata
Argentina

Resumen

En este trabajo se presentan las exploraciones realizadas en el marco del proyecto artístico «Extremófilos», cuyo objetivo es producir una serie de esculturas robóticas. Dichas exploraciones fueron divididas en dos etapas: la primera, relacionada con el diseño digital de la forma tridimensional de la escultura; y la segunda, vinculada a la fabricación y a la motorización de la escultura.

Los temas abordados en este trabajo están relacionadas con el diseño paramétrico de formas tridimensionales, con el desarrollo de herramientas de software para tal fin, con la implementación de técnicas de prototipado rápido y con las estrategias de robotización aplicadas al arte, junto con posibles intersecciones entre las técnicas de arte generativo y los desarrollos realizados a lo largo de este proyecto.

Palabras clave

Arte robótico, multimedia, software libre, prototipado rápido, diseño paramétrico

Abstract

In this work the explorations carried out in the context of the art project «Extremófilos» [«Extremophiles»], which aim is to produce a series of robotic sculptures, are presented. These explorations were divided into two stages: the first one is related to the digital design of the tridimensional shape of the sculpture, and the second one is linked to the production and the motorization of the sculpture.

The topics presented in this work are related to the parametric design of tridimensional shapes, with the use of software tools, the implementation of rapid prototyping and the robotization strategies applied to art, along with potential junctions between generative art techniques and the development occurred during this project.

Key words

Robotic art, multimedia, free software, rapid prototyping, parametric design

El proyecto artístico «Extremófilos»,[1] tiene como objetivo producir una serie de esculturas robóticas. Las exploraciones y los desarrollos realizados en el marco del Proyecto fueron divididos en dos etapas. La primera se relaciona con el diseño digital de la forma tridimensional de la escultura; la segunda, con la fabricación y con la motorización de la escultura –es decir, con la posibilidad de dotarla de movimiento–. En el presente texto, entonces, se presentarán las exploraciones y los desarrollos realizados según las etapas del Proyecto.

La primera parte de este escrito se divide en los siguientes puntos: el desarrollo de la discusión acerca de la propuesta morfológica –junto con posibles aproximaciones al diseño de formas tridimensionales– para explicar por qué la técnica de los planos seriados será la utilizada en este proyecto; la descripción de los algoritmos computacionales necesarios para describir dichas formas tridimensionales; la caracterización del desarrollo de un *software* de interfaz gráfica de usuario, que posibilita la implementación de los algoritmos mencionados anteriormente y que, por ende, permite realizar diseños de forma; y por último, la presentación de un ejemplo de diseño de forma para el que se utilizaron las herramientas desarrolladas y descriptas.

La segunda parte se divide en la descripción y en la argumentación de la implementación de servomotores como forma de motorización para la escultura; la caracterización del proceso de diseño de piezas articulares que permiten el acople de dichos servomotres a la estructura de la escultura; la descripción del diseño y de la fabricación de un sistema de control electrónico que posibilita la animación del movimiento de la escultura; y la discusión acerca del armado de un prototipo que reúne a los motores con las piezas de la escultura. Por último, se ofrece una conclusión en la que se discute acerca del futuro de los desarrollos realizados en este trabajo, en particular, sobre la base de su carácter de código/*hardware* abierto.[2]

[1] Proyecto dirigido por Emiliano Causa en el marco del proyecto de investigación «El algoritmo y la generatividad aplicados al arte».

El diseño de la forma

El trabajo en torno al diseño de formas tridimensionales fue desarrollado según varios pasos. En principio se relevaron distintas estrategias de diseño posibles y se eligió, finalmente, la generación de volúmenes por planos seriados. Una vez decidida la estrategia de diseño, se desarrollaron los algoritmos necesarios. Luego, se creó una herramienta de *software* con interfaz gráfica para facilitar dicho diseño y, finalmente, se realizó el diseño de la forma utilizando las herramientas y las técnicas desarrolladas previamente.

La propuesta morfológica: planos seriados

En pos de diseñar y de construir la forma de esta escultura robótica, se buscó una técnica sencilla que permitiera alcanzar formas tridimensionales mediante técnicas de diseño generativo y de prototipado. Las principales técnicas de prototipado, y las más accesibles, son aquellas que posibilitan producir formas bidimensionales, como el corte láser o el corte por control numérico. A su vez, estas técnicas se pueden combinar con el uso de un *software* de gráfica vectorial. Una de las ventajas de la utilización de formas bidimensionales es que se puede llegar hasta el uso de tijeras, de papeles o de cartones para su producción. Por todo esto, se pensó en producir formas tridimensionales a partir de figuras bidimensionales. Existen varias técnicas posibles para este tipo de diseño, pero la producción de volúmenes por planos seriados es una de las más sencillas (además, la habíamos usado en un proyecto anterior).[3]

A partir de la experiencia obtenida en el proyecto mencionado,

[2] Para conocer las definiciones de código abierto y hardware abierto, consultar las siguientes direcciones: http://opensource.org/docs/osd y http://www.oshwa.org/definition/spanish/

[3] Dicha experiencia se desarrolla en el texto «Algoritmos Genéticos aplicados a la generación y producción de formas escultóricas» (Causa, 2013).

se realizaron formas más complejas: en lugar de trabajar con círculos, se utilizaron elipses a las que se le agregan o se le quitan apéndices. Con una secuencia de elipses, que varían en tamaño, en posiciones y en apéndices, se construyó un cuerpo. Por último, los varios cuerpos pueden operarse entre sí para producir cuerpos más complejos. Ya que el objetivo del proyecto es producir estructuras zoomorfas, se buscó una figura que favoreciera la simetría bilateral. En este caso, la forma deseada es la de los peces, así que se partió de la idea de construir un tubo, como una secuencia de planos elípticos que varíen durante su desarrollo.

El desarrollo de los algoritmos de la forma

Sobre la base de las necesidades detalladas en los párrafos anteriores, se creó un conjunto de clases y de métodos en *Processing*[4] (un lenguaje de programación libre que utilizamos con frecuencia), que permitiera construir una forma de manera paramétrica. La estrategia que se eligió es la que ya fue expuesta: una serie de elipses construyen un *cuerpo* –a estas elipses se les puede llamar «costillas»–; a estas elipses se les puede agregar o quitar *apéndices*. Tanto las *elipses* como sus *apéndices* pueden variar durante la progresión siguiendo ciertas curvas. Varios cuerpos pueden ser operados mediantes adiciones y substracciones para crear cuerpos más complejos. Para esto, se programó la clase *Cuerpo* –es la clase principal y la encargada de construir el cuerpo– que posee un conjunto de métodos combinables con algunas funciones.

`Cuerpo( int cantidad ): el método constructor de la clase en el que el parámetro «cantidad» determina la cantidad de costillas (plano) que tendrá en cuerpo.`

`setColumna( float[ ] columnaX_, float[ ] columnaY_, float[ ] columnaZ_ ): este método determina la posición de la` ***`columna vertebral`***`, es decir, cómo se ubican los centros de los planos (las costillas) en la progresión.`

`setElipses( float anchos[ ], float altos[ ] ): este método determina los tamaños de los diámetros verticales y horizontales de las elipses.`

`apendiceSuave ( int resolucion, int desdeCostilla, int hastaCostilla, float[ ] angDesde, float[ ] angHasta, float[ ] agregado ): genera un apéndice que se ubica (en forma angular por coordenadas polares) entre el ángulo` ***`angDesde`*** `hasta el` ***`angHasta`***`. El parámetro agregado determina el nivel de prominencia del apéndice.`

`apendiceAgudo( int resolucion, int desdeCostilla, int hastaCostilla, float[ ] angDesde, float[ ] angHasta, float[ ] agregado ): este método es como el anterior, pero la forma del apéndice que genera es más aguda.`

`operar(Cuerpo otro, float dx, float dy, int indiceDesde, int indiceHasta, int indiceOffSet, float angulo, boolean horarioUno, boolean horarioDos): este método permite operar dos cuerpos entre sí para obtener uno resultante de la adición o sustracción entre los dos. Se determina la posición relativa entre los cuerpos (con «dx» y «dy») así como el índice que determinan en cuáles planos se ejecuta la operación.`

`cambiarCostillasCon( Cuerpo otro, float dx, float dy, int indiceDesde, int indiceHasta, int indiceOffSet ): este método permite que un cuerpo reemplace sus costillas con las de otro.`

Como puede observarse, en gran parte de los métodos citados los parámetros son arreglos (aquellos que terminan con corchetes). Esto se debe a que el parámetro no se aplica a una sola elipse, sino a una serie de elipses (el cuerpo) y, por tanto, debe ser una serie de valores. Para complementar estos métodos, se usó un conjunto de funciones que construyen arreglos que respetan ciertas progresiones:

`float[ ] vecConst( int cantidad, float valorC ): esta función devuelve un arreglo de un tamaño determinado por`

[4] Página oficial del proyecto Processing: https://processing.org/

cantidad, en donde todos los valores poseen el valor constante «valorC».

float[ ] vecLinea(int cantidad, float desde, float hasta): en este caso, la función devuelve un arreglo con una progresión lineal que va entre los dos valores límites («desde», «hasta»).

float[ ] vecSeno(int cantidad, float desde, float hasta, float ang1, float ang2): la progresión de esta función es sinusoidal, moviendo la función entre dos valores límites («desde», «hasta») y moviendo la fase del seno entre dos ángulos.

float[ ] vecEnvolvente(int cantidad, float[ ] t, float[ ] v): en función genera una envolvente, una curva que transita entre una secuencia de puntos, los puntos están definidos por dos arreglos «t» y «v» que describen los tiempo y valores de los puntos por los que pasa la curva.

Ya hemos revisado la clase Cuerpo y las principales funciones para generar sus parámetros, pasaremos, entonces, a ver ejemplos concretos de su uso. En el código que se expone debajo se puede observar (en la segunda línea) que se declara un objeto «k» de tipo Cuerpo al que se le pasa como parámetro una variable *cantidad* con el valor 100. Esto significa que el cuerpo es una secuencia de cien planos seriados; estos planos son elipses. Debajo del código, en la Figura 1, se muestra un gráfico con la secuencia de elipses que conforman el volumen.

Debajo de la declaración del Cuerpo se hay tres líneas que declaran tres arreglos (series de números), d[], p[] y z[], que son cargados en el caso de d[] y p[] con valores constantes (60 para d[] y 0 para p[]) y en el caso de z[] con una progresión lineal de 0 a 600. La línea «k.setElipses(d, d);» determina que los diámetros horizontales y verticales de las elipses toman el valor 60 extraído del arreglo d[] (el cual posee 60 en toda su serie). Por último, la línea «k.setColumna(p, d, z);» organiza las posiciones de los planos, asignándoles los parámetros (X,Y,Z); en este caso, las posiciones en X son tomadas de la serie p[] (que posee 0 en todas sus posiciones), los valores de Y se toman de d[] (que posee 60 en todas sus posiciones) y a Z se le asigna el arreglo z[] que avanza progresivamente de 0 a 600 en 100 pasos (es decir: 0, 6, 12, 18...). Este último determina la separación en profundidad de los planos.

```
int cantidad = 100;

k = new Cuerpo( cantidad );

float d[] = vecConst( cantidad, 60 );
float p[] = vecConst( cantidad, 0 );
float z[] = vecLinea( cantidad, 0, 600 );

k.setElipses( d, d );
k.setColumna( p, d, z );
```

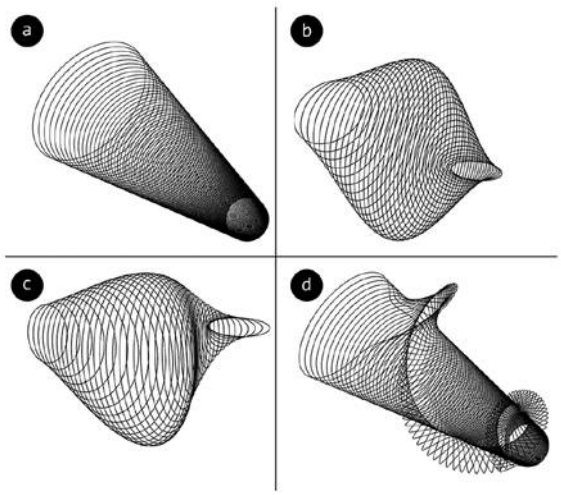

Figura 1. Cuatro variaciones posibles, resultantes de distintas configuraciones de vectores y de la aplicación de apéndices

La Figura 1-a muestra el volumen resultante del código anterior. Para explicar mejor la estrategia, en el ejemplo siguiente modificamos los diámetros de las elipses y agregamos un nuevo arreglo

llamado d2[], al que se le carga una serie que sigue una curva sinusoidal, declarado en la cuarta línea del código. En la anteúltima línea se puede ver cómo en «k.setElipses(d, d2);» se usa d2[] para determinar los diámetros verticales. Esto hace que en la figura 1-b el cilindro se engrose en altura hacia la mitad de su recorrido, también se puede ver que dicha progresión sigue la curva sinusoidal antes descripta:

```
int cantidad = 40;

k = new Cuerpo( cantidad );

float d[] = vecConst( cantidad, 60 );
float d2[] = vecSeno( cantidad, 20 , 140 , radians(-45)
, radians(270) );
float p[] = vecConst( cantidad, 0 );
float z[] = vecLinea( cantidad, 0, 400 );

k.setElipses( d, d2 );
k.setColumna( p, d, z );
```

En el siguiente ejemplo se reemplazó a d[] por d1[], un arreglo al que se le asignó una serie que también sigue una sinusoidal (en la tercer línea del código). Si bien d1[] y d2[] son sinusoides que organizan valores en un rango que va de 20 a 140 (como puede verse en sus parámetros), la principal diferencia es que están corridos en fase, ya que d1[] copia la forma de la sinusoide entre los 0 y los 360 grados, mientras que d2[] lo hace entre los -45 y los 270 grados. El nuevo arreglo, d1[], determina los diámetros horizontales de las elipses. Los resultados de estos cambios pueden verse en la Figura 1-b. Al mover ambos diámetros, la forma tubular de la serie se pierde y da lugar a una forma más plástica y sutil:

```
int cantidad = 40;

k = new Cuerpo( cantidad );

float d1[] = vecSeno( cantidad, 20, 140, radians(0),
radians(360) );
float d2[] = vecSeno( cantidad, 20, 140, radians(-45),
radians(270) );
float p[] = vecConst( cantidad, 0 );
float y[] = vecSeno( cantidad, 20, 70, radians(-45),
radians(270) );
float z[] = vecLinea( cantidad, 0, 400 );

k.setElipses( d1, d2 );
k.setColumna( p, y, z );
```

La Figura 1-c muestra el volumen resultante del código anterior.

En el siguiente ejemplo se refleja cómo funcionan los apéndices. Volvimos al cuerpo del primer ejemplo, en el que los diámetros se comportan de forma constante, dado que se encuentran vinculados a un un arreglo d[] que posee el valor 60 en todas sus posiciones. Al final del código se pueden apreciar cuatro líneas en las que primero se declaran tres arreglos (angDesde[], angHasta[] y agregado[]), que luego son usados en el método que tienen los cuerpos para agregar apéndices: «k.apendiceAgudo(resolucion, 0, cantidad-1, angDesde, angHasta, agregado);». La idea es la siguiente: los apéndices son salientes que se generan en las elipses, la posición de estas se determinan por una variable angular. Por ejemplo, el ángulo «0» se ubica en la parte superior, el de 90 grados en el costado derecho, el de 180 grados abajo y así siguiendo el giro. Por eso, en la Figura 1-d se observa que los apéndices de las elipses se corren según una progresión, lo que hace que efectúen un giro alrededor del tubo, como en una helicoide.

```
int cantidad = 100;

k = new Cuerpo( cantidad );

float d[] = vecConst( cantidad, 60 );
float p[] = vecConst( cantidad, 0 );
float z[] = vecLinea( cantidad, 0, 600 );

k.setElipses( d, d );
```

```
   k.setColumna( p, d, z );

    float angDesde[] = vecLinea( cantidad, radians(0),
radians(600) );
    float angHasta[] = vecLinea( cantidad, radians(80),
radians(600+80) );
    float agregado[] = vecConst( cantidad, 50 );

   k.apendiceAgudo( resolucion, 0, cantidad-1, angDesde,
angHasta, agregado );
```

En este último ejemplo, mostraremos la última acción posible: la de operar cuerpos entre sí. El código está dividido en tres partes por gráficos que reflejan a cada una de las partes. La primera parte del código muestra cómo se construye un cuerpo según los lineamientos explicados en los anteriores ejemplos. Este cuerpo se llama «k» y, debido a la variación de sus dos diámetros, produce una forma similar a una curva recostada con cierto corrimiento de su eje central.

```
   k = new Cuerpo( cantidad );
    float d[] = vecSeno( cantidad, 10, 80, radians(0),
radians( 400) );
    float p[] = vecConst( cantidad, 0 );
    float z[] = vecLinea( cantidad, 0, 600 );
    float y[] = vecSeno( cantidad, 10, 50, radians(0),
radians( 400) );
    k.setElipses( d, d );
    k.setColumna( p, y, z );
```

En la segunda parte, se construye otro cuerpo llamado «l», que es de forma muy similar al anterior aunque con un eje más centrado.

```
   l = new Cuerpo( cantidad );
    float d2[] = vecSeno( cantidad, 10, 90, radians(0),
radians( 400) );
    float p2[] = vecConst( cantidad, 80 );
    float y2[] = vecSeno( cantidad, 10, 50, radians(0),
radians( 400) );
    l.setElipses( d2, d2 );
    l.setColumna( p, p2, z );
```

En la tercera y última parte, el cuerpo «k» es operado con el cuerpo «l», en este caso, mediante una substracción. Es decir que al cuerpo «k» se le sustrae el cuerpo «l».

```
   k.operar( l, 0, 10, 0, 70, 20, radians(180), true,
false );
```

La Figura 2 muestra, a la izquierda, al cuerpo «k»; a la derecha, al cuerpo «l», y abajo, el resultado de la operación de ambos.

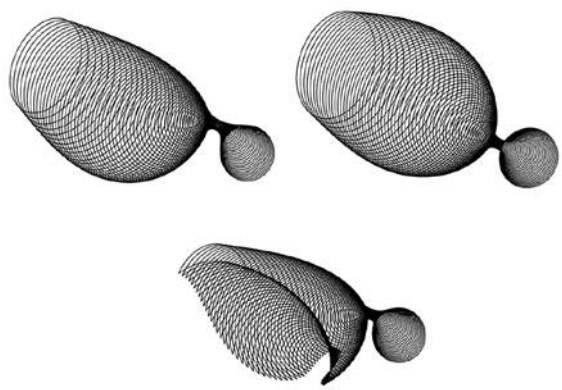

Figura 2. Dos configuraciones de cuerpo posibles y la intersección de ambos

Estas son, en líneas generales, las variantes morfológicas posibles a partir de los algoritmos desarrollados.

El desarrollo de la interfaz gráfica de usuario (IGU)

Una vez desarrollados los algoritmos de *software* descritos arriba, se creó un sistema de interfaz gráfica que permita utilizarlos de manera práctica. Como ya se explicó, los arreglos que definen

a la forma de los cuerpos pueden ser definidos manualmente o a través de funciones de código específicas para la generación de los mismos. Ambas opciones, al ser implementadas en el código de la aplicación, no permiten la realización de cambios en tiempo real. El código debe modificarse, debe ejecutarse la aplicación y, si se desean realizar cambios en la configuración formal del volumen, debe modificarse nuevamente el código y volver a ejecutar la aplicación para visualizar dichos cambios.

Dado que en la creación del tipo de volumen que planteamos hay varios parámetros y vectores de control en juego,[5] cambiar el código cada vez que se quiere modificar una variable resulta engorroso. Con esto en cuenta, se desarrolló una aplicación de *software*, denominada *extremoForm,* que permite el control de la forma del cuerpo mediante la implementación de módulos de interfaz gráfica. Estos módulos posibilitan la configuración detallada de un vector numérico (de los cuatros tipos preestablecidos detallados anteriormente). Al mismo tiempo, la aplicación grafica en tiempo real los efectos que los cambios en la configuración provocan en el volumen diseñado.

La Figura 3 presenta distintas instancias un módulo de interfaz gráfica, cuyo nombre asignado es «anchos elipses». La instancia de la izquierda es el módulo en su estado inicial, que permite elegir un tipo de vector. La instancia del centro muestra las opciones de configuración para un vector del tipo sinusoidal (nótese que todos sus parámetros pueden ser modificados). La instancia de la derecha corresponde a las opciones de configuración para un vector del tipo lineal. Es importante notar que se puede saltar de un tipo de vector a otro en tiempo real.

La Figura 4 muestra un volumen diseñado con esta estrategia. Todos los parámetros que definen su forma (menos las posiciones de los elipses en el eje z) son controlados a través de objetos de interfaz: el ancho de los elipses responde a una progresión sinusoidal, al igual que su alto (aunque no a la misma función sinusoidal), mientras que la altura de los elipses responde a una progresión lineal. El módulo de interfaz «paleta 1» es auxiliar y permite controlar la paleta de colores con la que se visualiza el volumen. La gran ventaja de la implementación de los módulos de interfaz es que conviven con la visualización del volumen, por lo tanto, cualquier modificación de parámetros (que a su vez hace efecto de manera instantánea) puede realizarse con la visualización como referencia y facilita el proceso de diseño.

Figura 3. Módulo de interfaz, utilizado para configurar un vector, en tres instancias posibles del proceso de configuración

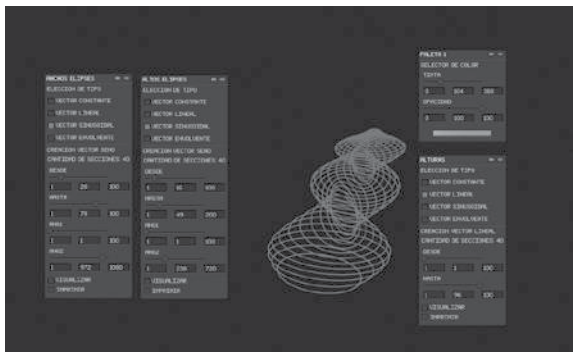

Figura 4. Volumen diseñado a partir de la configuración de vectores a través de módulos de interfaz gráfica

[5] Un volumen está definido por varios vectores, incluso, puede estar definido a partir de la operación de varios cuerpos.

Estos módulos de interfaz gráfica pueden ser reemplazados completamente por código. Una vez que un vector ha sido configurado, el módulo de interface que lo controla puede generar el código equivalente a dicha configuración. Este código puede implementarse en el código mismo de la aplicación y así liberar espacio en el entorno de la misma, para luego configurar otros parámetros a través de nuevos módulos de interfaz gráfica. De esta forma, podemos entender al *software extremoForm* como un híbrido entre configuración por código y configuración por interfaz gráfica.

En cuanto a la estructura del código, todo lo relacionado con el módulo de interfaz gráfica se encuentra encapsulado dentro de un objeto llamado «interfaz». Este objeto, a su vez, contiene el vector numérico que emerge de su configuración. Dentro del objeto *interfaz*, se encuentra una serie de objetos del tipo *menú* (hay uno específico para cada tipo de vector, más algunos auxiliares), que a su vez contienen los objetos de interfaz gráfica necesarios (*sliders*, *checkboxes*, campos de texto, botones, etcétera). Al mismo tiempo, el objeto *interfaz* posee métodos para exportar el código equivalente al vector configurado y para visualizar el mismo.

Resultaba importante que la implementación de estos objetos de interfaz fuera lo más sencilla posible (para poder reemplazarlos por su código equivalente de manera rápida), por lo que toda la funcionalidad del objeto interfaz está controlada por un sólo método. Este método devuelve el vector configurado a través de la interfaz. Su implementación es la siguiente:

```
float vector[] = interfaz.procesar("NOMBRE INTERFAZ",
numero_elementos);
```

De esta forma, el método *procesar* (cuyo argumentos es el nombre de la interfaz, dato únicamente de referencia para el usuario, y el número de elementos que tendrá el vector devuelto), se encarga de mostrar la interfaz en la pantalla, de procesar las cambios realizados en la misma y de reflejarlos en el vector que devuelve. Esta estrategia permite que al momento de reemplazar la interfaz por su código equivalente, tan solo haya que reemplazar una línea de código.

El diseño de la forma con extremoForm

A continuación se detalla un ejemplo del flujo de trabajo posible con el *software* desarrollado. La Figura 5 muestra el entorno gráfico de *extremoForm*, con un posible diseño de un cuerpo. En este caso, su configuración está generada a partir de tres módulos de interfaz gráfica (o IGU) que controlan el ancho, el alto y el desplazamiento en el eje Y de los elipses que lo componen; además de dos vectores definidos directamente en el código, que dictan la distancia entre elipses (o sus posiciones en el eje Z) y su desplazamiento en el eje X.

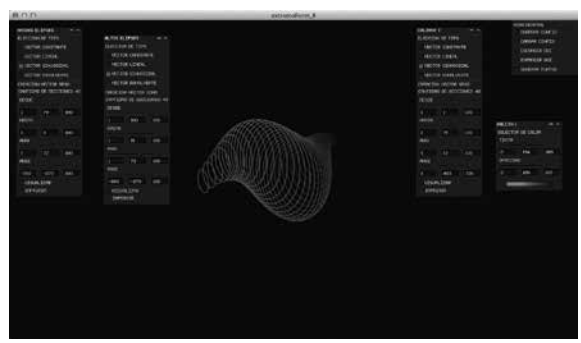

Figura 5. El entorno gráfico completo de *extremoForm*, con un posible diseño de cuerpo

El siguiente fragmento muestra el código necesario para lograr la configuración visible en la figura anterior. Nótese que los objetos *interfaz* fueron organizados en un arreglo (en este caso denominado «in») para facilitar su control.

```
Cuerpo k;
int cuantos = 40;
k = new Cuerpo( cuantos );
float w[] = in[0].procesar("ANCHOS ELIPSES", cuantos);
float h[] = in[0].procesar("ALTOS ELIPSES", cuantos);
float px[] = vecConst(cuantos, 0);
float py[] = in[0].procesar("COLUMNA Y", cuantos);
float z[] = vecLinea(cuantos, 0, -500);
```

```
  k.setElipses( w, h );
  k.setColumna( px, py, z );
  k.dibujar(); // este es un método propio de la clase
Cuerpo, se ocupa de dibujarlo en pantalla
```

Como puede observarse, cada módulo de IGU posee un botón denominado «IMPRIMIR», que copia al portapapeles del sistema operativo el código equivalente al vector creado a partir de la actual configuración del módulo. Una vez adquiridos los códigos equivalentes para todos los vectores, el fragmento de código necesario para la descripción completa de este volumen es el siguiente:

```
  Cuerpo k;
  int cuantos = 40;
  k = new Cuerpo( cuantos );
  float w[] = vecSeno(40, 79, 9, radians(72), ra-
dians(-273));
  float h[] = vecSeno(40, 100, 15, radians(72), ra-
dians(-273));
  float px[] = vecConst(cuantos, 0);
  float py[] = vecSeno(40, 1, 75, radians(13), ra-
dians(483));
  float z[] = vecLinea(cuantos, 0, -500);
  k.setElipses( w, h );
  k.setColumna( px, py, z );
  k.dibujar(); // este es un método propio de la clase
Cuerpo, se ocupa de dibujarlo en pantalla
```

En este fragmento de código, los arreglos w, h y py fueron generados a partir de módulos de IGU (y reemplazados por sus equivalentes en código), y los demás arreglos (px y z) fueron generados en el código mismo. Dado que los algoritmos desarrollados para el modelado del volumen permiten la adición de apéndices al cuerpo diseñado y también la intersección, suma y resta de cuerpos entre sí; y dado que todas estas operaciones respetan una lógica de funcionamiento basada en arreglos numéricos (y que, por lo tanto, pueden ser controladas mediante los módulos de IGU de *extremoForm*), resulta de gran importancia poder reemplazar dichos módulos por su código equivalente para liberar espacio en pantalla para poder implementar nuevos módulos de interfaz.

En la Figura 6 puede verse el mismo cuerpo, pero con la adición de dos apéndices, en forma de aletas. Todos los módulos de IGU implementados aquí están relacionados con parámetros de control de los apéndices.

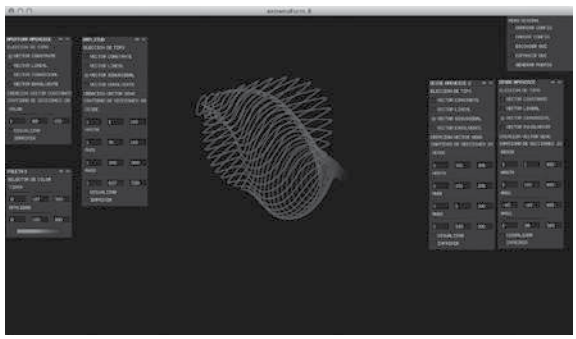

Figura 6. El mismo cuerpo de la figura anterior, pero con apéndices agregados y con los módulos de interfaz gráfica actualizados para controlarlos

Estos módulos fueron reemplazados, nuevamente, por sus equivalentes en código, a fin de liberar espacio en el entorno para nuevas operaciones de configuración.

En la Figura 7 se añadió un segundo cuerpo y todos los módulos de IGU implementados están vinculados a la definición de la forma del mismo. Los parámetros relacionados con la forma del cuerpo inicial fueron reemplazados por código. En este caso, el segundo cuerpo (que se encuentra ligeramente por debajo del primero), es sustraído del primer cuerpo. La Figura 8 muestra el resultado de la operación, junto con todos los módulos de IGU convertidos a sus equivalentes en código. De esta forma, el cuerpo en su estado actual se encuentra íntegramente definido a través de código, permitiendo la utilización de nuevos módulos de interfaz para nuevas operaciones (es posible agregar nuevos apéndices, nuevos cuerpos para operar, etcétera).

Por último, *extremoForm* tiene la capacidad de exportar la forma del cuerpo diseñado de forma tal que pueda ser importado por

otras aplicaciones de *software*, a fin de continuar su configuración y de prepararlo para su fabricación.

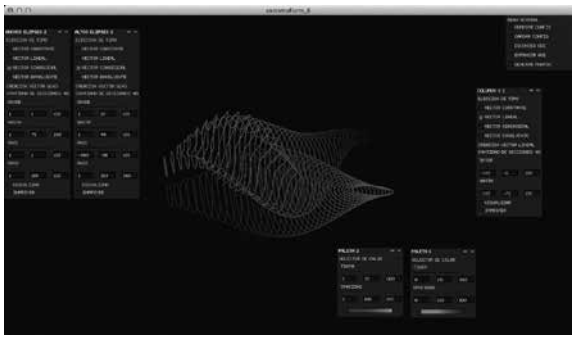

Figura 7. Dos cuerpos diseñados utilizando *extremoForm* con los módulos de interfaz vinculados a la forma del segundo

Figura 8. El cuerpo resultante de sustraer el segundo volumen al primero

La construcción y la motorización de la forma

En cuanto al trabajo relacionado con la construcción física de la forma diseñada y con su subsiguiente motorización/robotización, el trabajo fue realizado en los siguientes bloques. En primer lugar, se investigaron distintas estrategias y sistemas de motorización, y se decidió, finalmente, que el movimiento de nuestra escultura sería implementado a través de servomotores. Luego se diseñaron rótulas o articulaciones que permiten la vinculación de varios servomotores. A continuación se diseñó y se fabricó un sistema de control electrónico para los motores de la escultura y, finalmente, se realizó un prototipo de la escultura para realizar pruebas sobre el conjunto entero.

Los servomotores: animación y motorización

Producir piezas de arte robótico (como la que se busca lograr con este proyecto) requiere del control preciso de la posición y del movimiento de piezas y de mecanismos. Para este propósito, es conveniente la utilización de servomotores. Los servomotores son motores de alto torque que poseen un sensor de rotación acoplado a su eje. Generalmente están limitados a un rango de movimiento de 180 grados, por lo que (gracias al sensor de rotación) es posible conocer el ángulo de rotación del eje en todo momento. Esto permite un control muy preciso de la posición de elementos que estén acoplados al eje del servomotor (por ejemplo, una de las aplicaciones más comunes de este tipo de motores es el control de alerones en aeromodelismo). Al mismo tiempo, el acoplamiento de varios servomotores permite crear movimientos complejos con mucha exactitud. Otra ventaja de los servomotores es su relativo bajo costo (en comparación a tecnologías como actuadores neumáticos, por ejemplo). Además, existen en un gran rango de tamaños y de capacidades de torque.

Todas estas características hacen que los servomotores sean ideales para el desarrollo de piezas artísticas robotizadas.

El diseño de las rótulas

Sobre la base de lo anterior, y para poder dotar de movimiento a un cuerpo generado por nuestro *software*, fue necesario el desarrollo de una articulación robotizada, que permite la vinculación de dos piezas a un servomotor, permitiendo un giro de 180° de una pieza sobre otra. La articulación está compuesta de dos piezas de acrílico. El diseño fue realizado digitalmente. Se utilizó, para ello, una aplicación de dibujo vectorial teniendo en cuenta las medidas exactas de los servomotores que luego serían usados.

Las piezas fueron diseñadas para que pudieran ser cortadas mediante una máquina de corte láser– en una lámina de acrílico de 2 mm de espesor. Una vez cortadas, fueron plegadas mediante un proceso de termomoldeado. Para montar las piezas al servomotor se utilizaron bulones con tuercas y sellador de roscas anaeróbico (este último impide el desajuste de las piezas por consecuencia de las vibraciones de los motores).

La articulación está preparada para ser acoplada tanto a otras piezas mecánicas o estructurales como a otros servomotores (para lograr movimiento en dos ejes, por ejemplo), como también a otras articulaciones del mismo tipo. Para todo esto, las piezas cuentan con perforaciones de montaje que respetan las dimensiones estándar de ejes de servomotores. Al mismo tiempo, hay perforaciones en todas las caras de las piezas, de forma tal de proveer la mayor cantidad de posiciones de montaje posibles.

En el caso particular de este proyecto, se realizaron siete articulaciones y se montaron en línea (una con el extremo de otra) para lograr una configuración similar a una *columna vertebral* articulada para el cuerpo de nuestro Consumófago. La Figura 9 muestra un esquema del diseño de las articulaciones y de sus posibilidades de movimiento. En este caso particular, las costillas diseñadas con el *software* irían montadas entre las articulaciones.

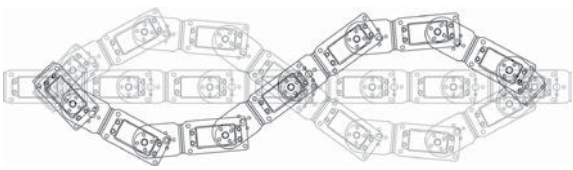

Figura 9. Vista esquemática de varias articulaciones funcionando en conjunto

El diseño de la electrónica de control

Para poder controlar todos los servomotres de la columna, se requiere de la utilización de un microcontrolador (al menos, esto resulta lo más práctico y versátil). La plataforma de microcontrolador elegida para este proyecto fue *Arduino*, una plataforma de código y de hardware abierto. Dado que la columna vertebral desarrollada cuenta con varios puntos de articulación, y que estos deben poder ser operados de manera simultánea, es necesario usar un elevado número de puertos de salida/entrada de la placa *Arduino*, que podrían ser útiles para controlar otro tipo de actuadores o de sensores. Por esto, se buscó una solución que permitiera el control de un gran número de servomotores sin comprometer todos los puertos de salida/entrada de la placa *Arduino*.

De este modo, se desarrolló un circuito electrónico basado en el *chip driver* de modulación por ancho de pulso (PWM, según sus siglas en inglés) de 16 canales TLC5940. El circuito fue basado en un diseño de hardware abierto y fue fabricado como una placa de circuito impreso. Se fabricaron tres placas (si bien este caso particular solo necesita de una), más una placa adaptadora que permite la fácil conexión del circuito a una placa *Arduino* modelo Mega.

El corte, el ensamblado y el montaje de las piezas

Una vez que diseñamos (utilizando nuestro *software extremoForm*) una forma de cuerpo con la que estuvimos conformes, se importó el diseño en un *software* de modelado 3d y se lo atravesó con un eje de referencia, para asegurar el correcto posicionamiento de las costillas posteriormente. Las mismas fueron exportadas del *software* 3d como curvas vectoriales e importadas en un *software* de dibujo vectorial. A partir de aquí, las costillas fueron organizadas para ocupar la menor cantidad de superficie posible (en función de reducir la cantidad de material necesario para su corte).

Una versión a escala de las costillas fue impresa en papel de alto gramaje y montada sobre un eje, de forma tal de lograr una maqueta a escala del volumen resultante del proceso descrito previamente, como puede observarse en la Figura 10.

De vuelta en el *software* de dibujo vectorial, se utilizaron los diseños vectoriales para las articulaciones robotizadas, para agregar las perforaciones de montaje de las mismas a las costillas (para poder montar las piezas sobre la columna creada con las articulaciones). Las mismas fueron impresas en papel a escala real, montadas sobre cartón y cortadas. Luego, fueron montadas en la columna de articulaciones para realizar pruebas de movimiento del conjunto en su totalidad (es decir, las costillas diseñadas

digitalmente, montadas sobre la columna motorizada, controlada mediante nuestro sistema electrónico). Una vez comprobado el correcto funcionamiento del sistema, los diseños vectoriales pueden ser utilizados para realizar cortes de las piezas en acrílico, para su ensamblaje final.

Figura 10. Maqueta a escala de un volumen diseñado mediante el uso de *extremoForm*

Conclusión

Las herramientas actualmente disponibles en el campo de la informática y del prototipado rápido permiten encontrar múltiples estrategias de producción de formas, es decir, se trata de un campo muy fértil para la investigación y para el desarrollo, particularmente, en su intersección con el campo artístico. En este sentido, es importante mencionar que la estrategia de diseño elegida para este trabajo será extendida para su implementación dentro de un entorno de algoritmos genéticos (extendiendo sus posibilidades de producción de forma al insertarse dentro del campo del arte generativo).

Finalmente, es importante notar la modularidad de las herramientas desarrolladas en este proyecto, junto con su carácter de código/hardware abierto. Muchas de las herramientas y de las técnicas desarrolladas (justamente gracias a su carácter modular), pueden utilizarse en otros proyectos. Al mismo tiempo, las estrategias de diseño elegidas permiten una producción de forma refinada que puede controlarse mediante procesos evolutivos y/o mediante interfaces intuitivas, pero que, a su vez, pueden materializarse de múltiples formas, algunas casi artesanales; esto aumenta la flexibilidad de las herramientas desarrolladas. Dado que las herramientas de *software* creadas están basadas en un lenguaje de programación de código abierto, y que los desarrollos electrónicos y de robótica realizados son también de naturaleza abierta, los mismos serán publicados y liberados a la comunidad artística, para que puedan ser utilizados en otros proyectos y modificados para servir a nuevas finalidades.

Referencias electrónicas

Causa, E. (2013). «Algoritmos genéticos aplicados a la generación y producción de formas escultóricas» [en línea]. Consultado el 17 de junio de 2015 en <http://www.invasiongenerativa.com.ar/descargas/INVASION%20GENERATIVA_1_1.pdf >.