Metagrafic: hacia un lenguaje para la graficación por computadora

ALEJANDRO AGUILAR SIERRA

Se presenta un sistema para definición y almacenamiento de gráficas en 2D, que utiliza transformaciones lineales, estructuración jerarquizada, interacciones y recursividad; al tiempo que se introducen algunos conceptos de graficación por computadora.

Introducción

Uno de los principales problemas de la graficación por computadora es la representación y almacenamiento de información gráfica. Para empezar conviene distinguir entre el simple archivo de datos (suele llamársele *displayfile*), y un lenguaje de graficación propiamente dicho, que tendría que cumplir los requerimientos que cumple cualquier lenguaje computacional.

Un *displayfile* tiene las siguientes ventajas:

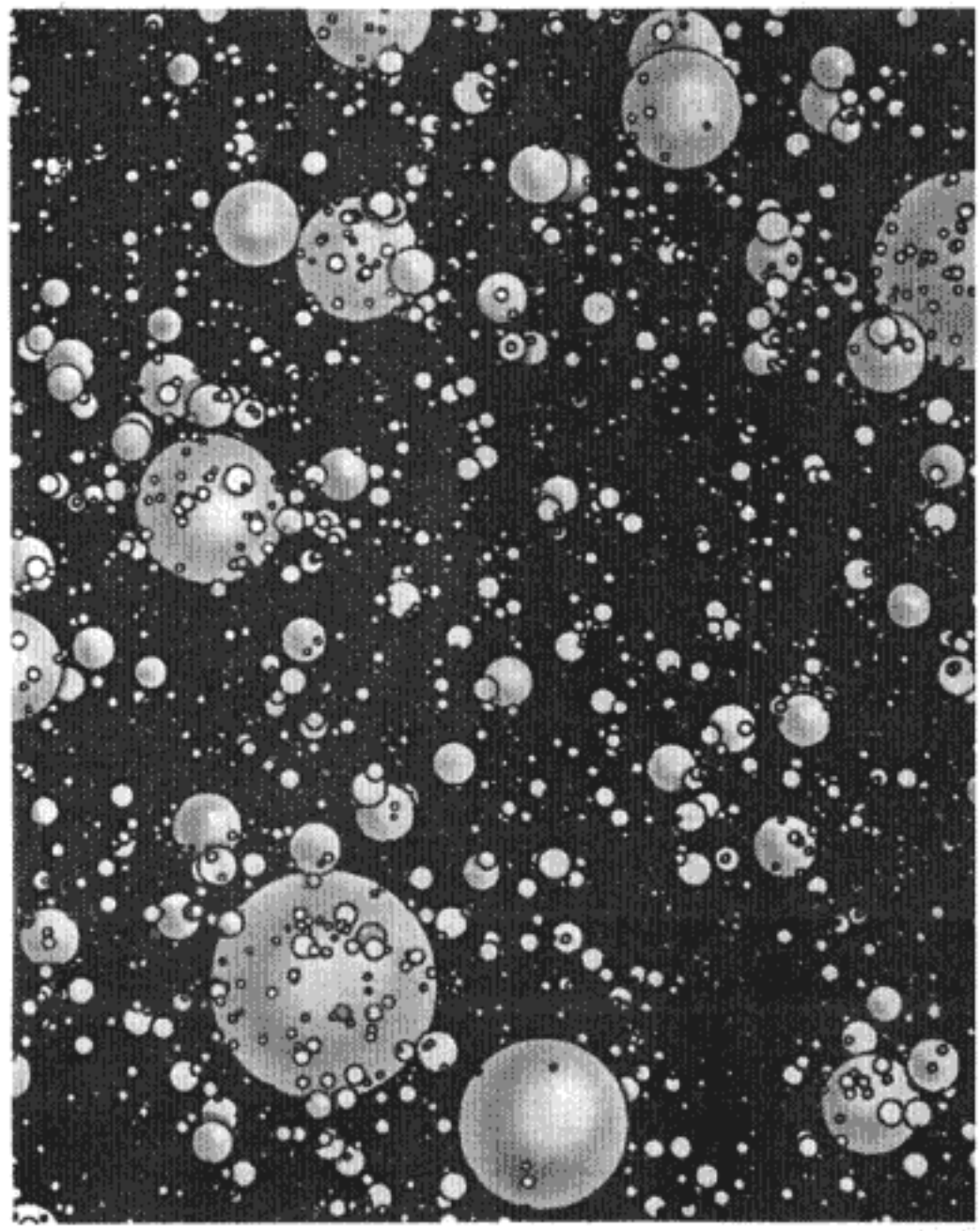
- Es un formato de datos para archivar "imágenes" y no tener que reconstruirlas cada vez que se necesiten.
- Puede ser un formato único para ser interpretado en distintos dispositivos graficadores.
- Puede "preverse" una imagen en la pantalla, antes de desplegarla en un medio lento o caro.

Y, abusando un poco de los términos, un "lenguaje" de graficación agregaría las siguientes:

- Posiblidad de modificar fácilmente la "imagen"; algo así como "editar" o "depurar".
- Concisión. Más imágenes con menos palabras.
- Posiblidad de integrar imágenes construidas separadamente.
- Flexiblidad, jerarquización entre partes de la imagen.

Muchos paquetes personales de graficación (que pueden usarse en una PC) almacenan la información en forma digital, lo cual reúne todas las desventajas: las líneas no horizontales ni verticales se ven horribles, consumo excesivo de memoria, redundancia, dificultad para cambiar la escala de la imagen, o rotarla, imposiblidad de desplegar en dispositivos vectoriales, (donde la información se da en coordenadas cartesianas) etc. Otros manejan lenguajes especializados, que sólo ellos entienden, lo que limita la flexiblidad y aumenta la dependencia. Los sistemas basados en modelos estandarizados (tipo GKS, PHIGS) tienen la lata adicional de que hay que hacer un programa para cada aplicación específica, y son tan vastos que uno no puede aprovecharlos al límite.

Para aplicaciones en que se necesita calidad de trazo de línea, independencia del dispositivo de graficación a emplear, no consumir más memoria de la necesaria, y facilidad de usarse en combinación con un lenguaje de alto nivel, interactivamente, o bien escribiendo un programa fuente, creamos Metagrafic, una primera propuesta en 2D.



Alejandro Aguilar Sierra: Centro de Ciencias de la Atmósfera, UNAM.

Descripción del sistema Metagrafic

MG es algo así como el eslabón perdido entre un *displayfile* y un lenguaje de graficación. La definición de los objetos a graficar se hace mediante primitivas simples, transformaciones y atributos. La estructura de datos básica es la lista de puntos (parejas de números reales). Cada primitiva tiene la siguiente sintaxis:

nombre primitiva lista de puntos *NL*

un identificador, seguido de una lista de puntos, y al final el indicador de fin de lista *NL*. Pero hay excepciones:

PL x_1 y_1 ... x_m y_m *ng* x_{m+1} y_{m+1} ... x_n y_n *NL*

PL traza una poligonal tomando como vértices los puntos de la lista. La subinstrucción *ng* interrumpe el trazo de la poligonal, de modo que los puntos m y m+1 no son unidos con una línea.

CR r x_1 y_1 ... x_n y_n *NL*

En un policírculo se generan n círculos centrados en cada uno de los puntos x_i y_i, y radio r, por lo que sólo es necesario definir r una vez (al principio).

Las coordenadas se definen en el espacio del objeto. Se llama ventana a la región del espacio del objeto que deseamos visualizar. En *MG* la ventana por omisión es [0,1] x [0,1]. Las coordenadas normalizadas tienen la ventaja de que mapean cualquier puerto de visión con un mínimo de cálculos. Es posible ajustar la ventana a los datos del usuario. La instrucción es *WW* Xmin Xmax Ymin Ymax.

En el caso de las transformaciones, ya se sabe cuántos datos se necesitan, y no es necesario finalizar con *NL*. Aquí no se utilizan listas de puntos, sino matrices. Las instrucciones son básicamente:

RT	theta	Rotar un ángulo theta.
SC	sx sy	Cambiar de escala en ambas dimensiones.

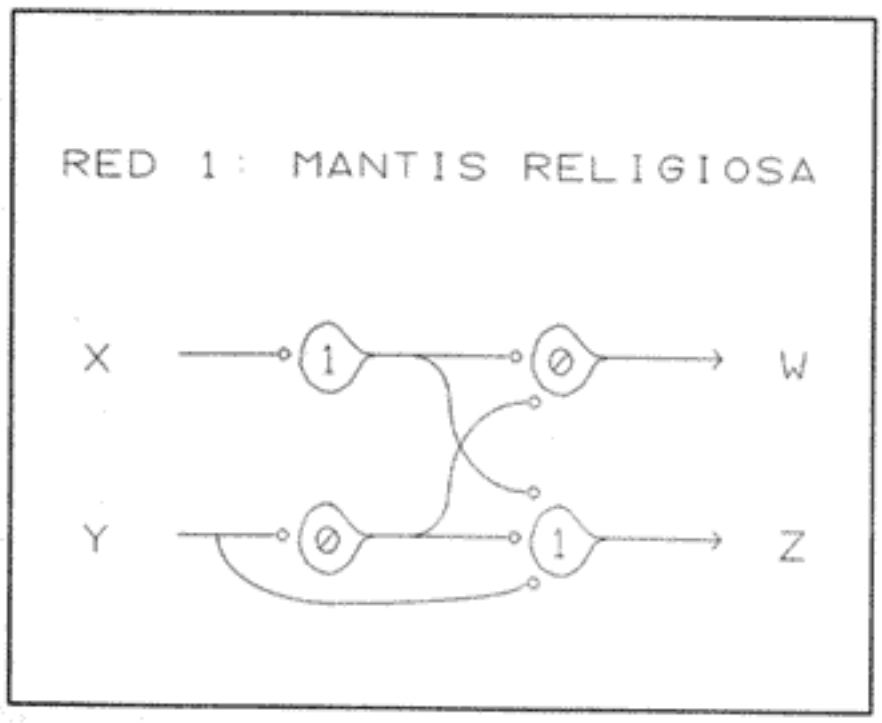


Figura 1.

TL	tx ty	Trasladarse al punto tx, ty.
IM		Asignar la Idéntica a la matriz.
MT	m_{11}...m_{13}	Matriz de transformación definida
	m_{31}...m_{33}	por el usuario.

Las transformaciones se realizan con matrices de 3x3 en coordenadas homogéneas (ver recuadro). Es posible que la instrucción no necesite ningún dato, como es el caso de *IM* que asigna la matriz idéntica a la matriz de transformación.

Lo que hace realmente poderoso a *MG* es el uso de estructuras. Las estructuras son subrutinas gráficas que una vez definidas pueden utilizarse como si fuesen primitivas.

La sintaxis es la siguiente:

OPST Identificador estructura
Primitivos, transformaciones, atributos y otras estructuras.
CLST

El identificador de la estructura referirá lo que se encuentre entre *OPST* y *CLST*. Funciona, pues, como una referencia (un apuntador, no una copia, como sería el caso de una macroinstrucción). Esto representa un buen ahorro de tiempo y memoria al evitar redundancias. Otras ventajas de utilizar estructuras se ennumeran a continuación:

- Jerarquía. Como una estructura puede contener otras estructuras, las transformaciones que se apliquen a la estructura "madre" se compondrán a las de la "hija".
- Recursividad. Una estructura puede contenerse a sí misma, permitiendo sorprendentes efectos.

Toda estructura lleva asociados un punto de referencia (un "origen" relativo) y una matriz de transformación. Lo más usual es desplegar la estructura en cada punto de una lista:

Nombre estructura lista de puntos *NL*

Pero también es posible ajustar una estructura a un área específica del espacio, con las siguientes instrucciones:

MKST Identificador estructura Marca la estructura.

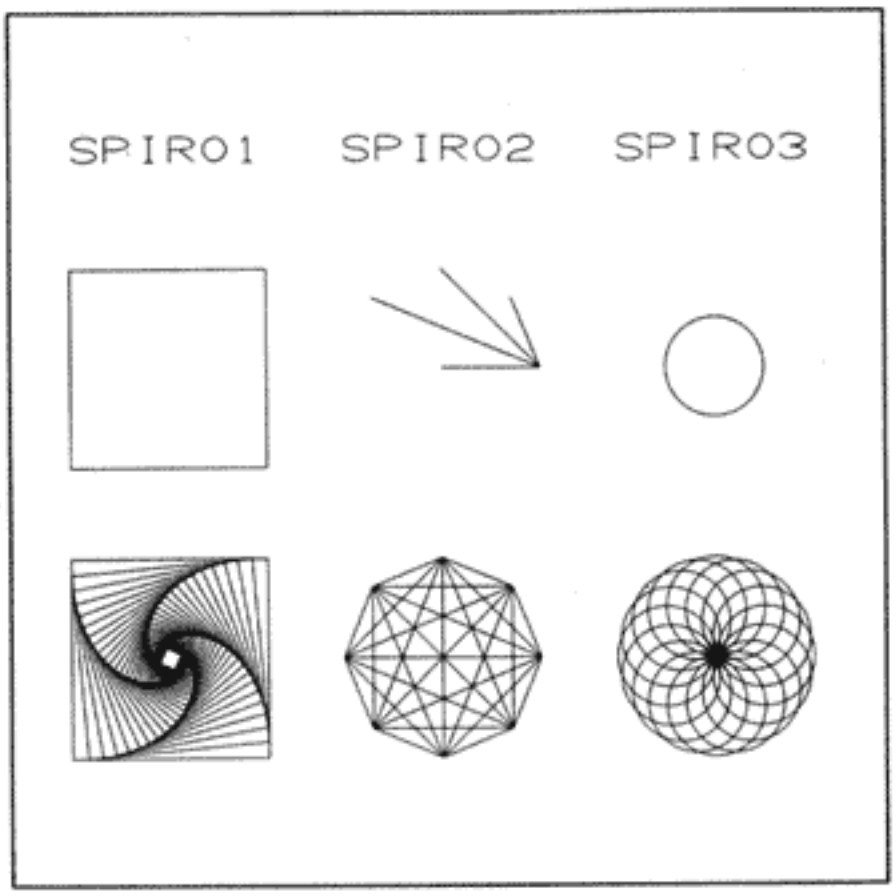


Figura 2.

PSWT x_1 y_1 x_2 y_2 Despliega la estructura marcada dentro de los límites del rectángulo encerrado entre los puntos 1 y 2. Cambiando el orden de estos puntos se obtienen las reflexiones simples.

Otras matrices de transformación actúan con las estructuras. Una es la matriz de iteración, que multiplica repetidas veces la matriz asociada a la estructura marcada. En la instrucción *RPST* n se despliega n veces dicha estructura cada vez con una transformación diferente. Otra matriz altera iterativamente el punto de referencia de la estructura. Cada vez que se indica la instrucción *DPST* nombre estructura cambia la posición, y no es necesario marcar la estructura, pero sólo la despliega una vez. La aplicación de estas transformaciones secundarias puede controlarse con banderas especiales.

Mediante la instrucción INPUT nombre archivo [.grf] podemos cargar estructuras definidas en otro archivo. Se vale anidar, es decir, abrir archivos que a su vez abren otros archivos, pero hasta cierto límite.

Una aplicación inmediata, dadas las facilidades descritas, es la creación de texto. Cada letra del alfabeto puede definirse en forma muy concisa, y la letra misma puede usarse como identificador de su estructura. El usuario podría, muy fácilmente, crear su propio alfabeto y manejar texto junto con gráficas. *MG* provee algunas abreviaturas:

INTXT	Carga un alfabeto definido en el archivo ALFA.GRF.
XYTXT x y	Posición del texto a desplegar.
SCTXTsx sy	Dimensiones de la letra.
TDTXT Ix Iy	Incremento de la posición.
RTTXT theta	Rotación de la letra.
DPTXT texto	Despliega el texto según lo definido arriba.

Conviene poder desplegar texto junto con las gráficas. Algunos dispositivos de graficación no incluyen esta facilidad.

Es útil intercalar comentarios entre las instrucciones de un programa, para su posterior revisión. Esto se hace, en *MG*, iniciando cada comentario con el caracter %.

Por último, cabe recordar que todas las coordenadas están definidas en el espacio del objeto. El espacio imagen depende del dispositivo donde se va a interpretar el archivo.

Ejemplos

En esta sección no se pretende demostrar todas las posibilidades de *MG*, ya que sólo se usan segmentos de línea y círculos. La idea es obtener figuras aparentemente complejas a partir de diseños simples, aprovechando algún tipo de simetría.

En el primer ejemplo (figura 1) se expone una muestra típica de las aplicaciones que motivaron la creación de MG. Se trata de una red neuronal muy sencilla que simula un comportamiento de la mantis religiosa. X y Y representan los estímulos, W y Z las respuestas, y los números dentro del cuerpo de las neuronas, los umbrales. El texto y las neuronas están definidas con splines.

En la figura 2 se muestra, en la parte de arriba, la estructura básica, y más abajo, el resultado después de haber aplicado transformaciones reiterativas; rotando la estructura básica (como en SPIRO1 y en SPIRO2) o su origen (como en SPIRO3), o bien cambiando la escala (como en SPIRO1).

Con el uso de la recursividad podemos conseguir objetos tan curiosos como los árboles fractales, llamados así por su autosimilitud (simetría escalar: a cualquier escala se observa el mismo patrón). No es difícil concluir que una curva que pasara por todos los puntos de una de las ramas del árbol, tendría longitud infinita. Son aparentemente muy complicados (ver figura 4) pero en realidad la estructura base es una simple línea (el tronco), en uno de cuyos extremos se repite la estructura dos veces, a una escala menor y con ángulos distintos de rotación (dos ramificaciones). Queda como ejercicio al lector la programación, en *MG*, de las curvas de Hilbert o de Sierpinski.

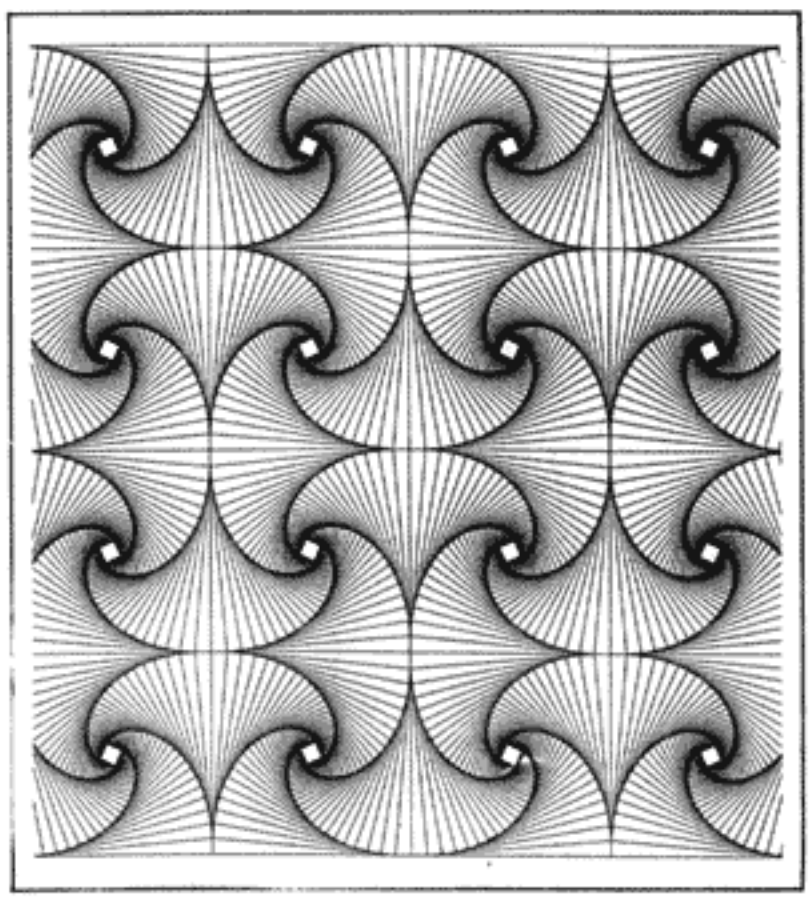


Figura 3.

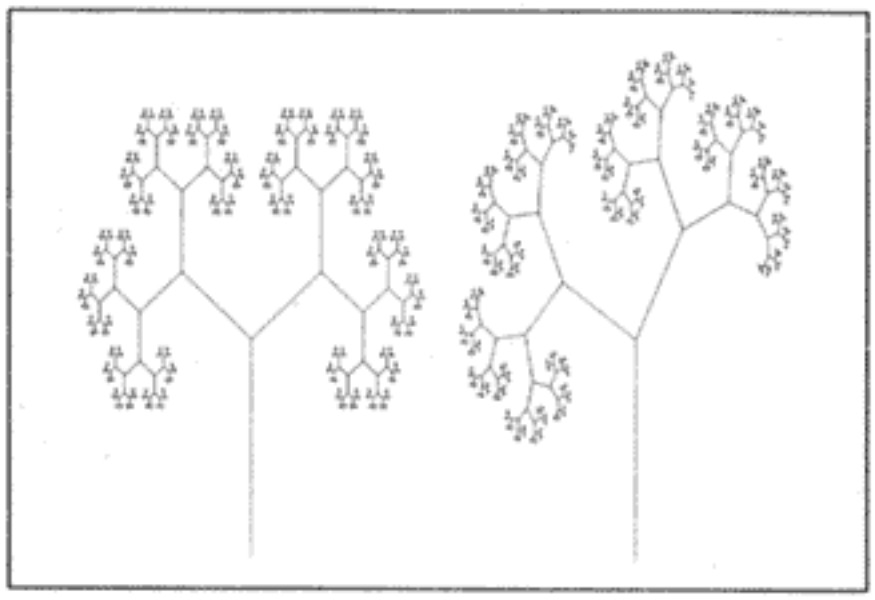


Figura 4.

Hasta aquí sólo se han visto curvas. Dejo para otra ocasión los objetos con áreas cerradas. Veremos cómo construir diseños en 2D tales como los de M. C. Escher; o como los mosaicos no periódicos de Roger Penrose.

Conclusión

MG se perfila como algo más que un *displayfile*, ya que toda la información gráfica está contenida en el archivo; y es un lenguaje de graficación, con instrucciones sencillas y fáciles de memorizar, pero con posiblidades casi infinitas. Esta parcialmente inspirado en sistemas de graficación estandarizados (tales como *PHIGS*) y en el "graphics language" de los plotters, pero con énfasis en la sencillez y en un mínimo de instrucciones. Hay que tomar en cuenta que *MG* no es un producto acabado. Es más, se aceptan sugerencias. ❖

Reconocimientos

El autor da las gracias a Ana Luisa Solís, Guillermo Correa y Homero Ríos por sus valiosos comentarios.

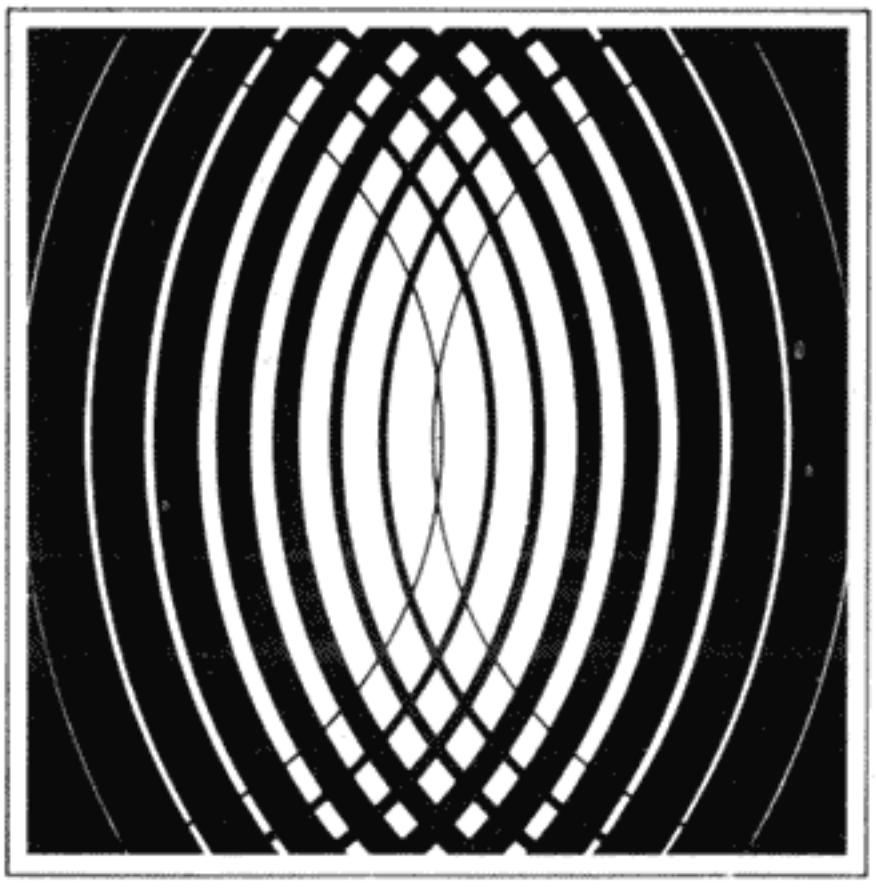
Bibliografía

1. W. Newman, R. Sproull. 1979. *Principles of Interactive Computer Graphics*. McGraw-Hill.
2. J. Foley, A. Van Dam. 1982. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley.
3. D. Rogers, J. Adams. 1976. *Mathematical Elements for Computer Graphics*. McGraw-Hill.
4. S. Harrington. 1987. *Computer Graphics: A Programming Aproach*. McGraw-Hill. (2a ed.).
5. R. Sproull, W. Sutherland, M. Ullner. 1985. *Device Independent Graphics*. McGraw Hill.

Apéndice 1

Algunos listados

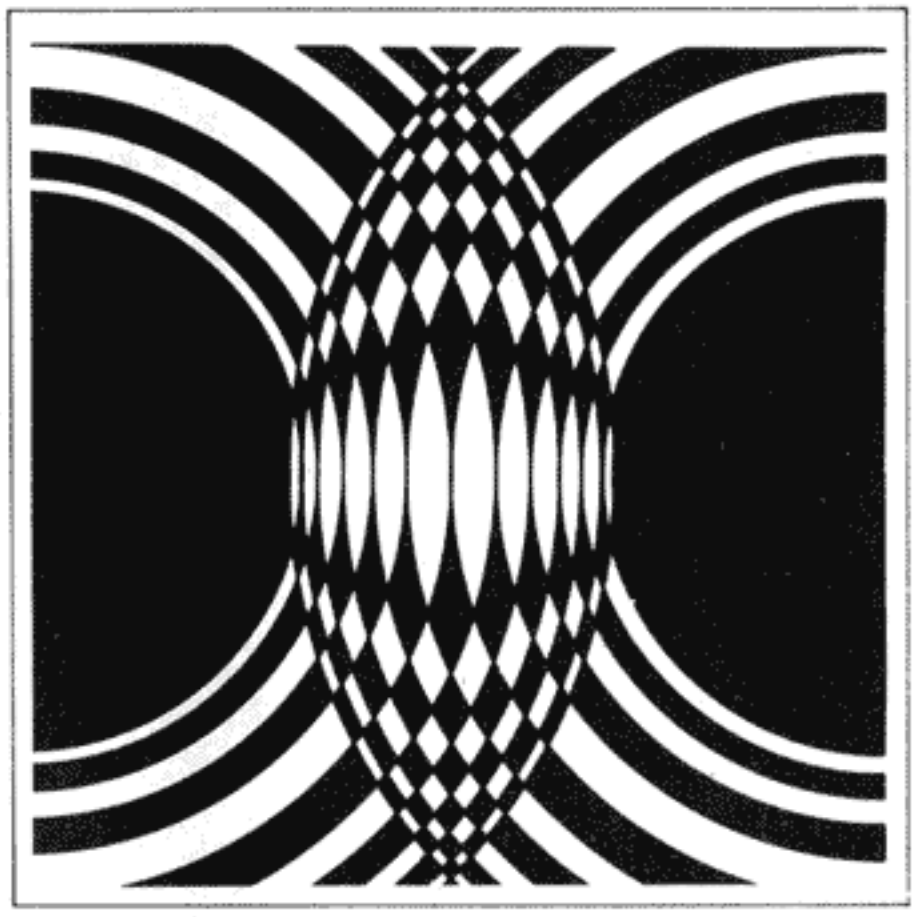
```
& Figura basada en la rotación de un cuadri
OPST SPIRO1
OPST C4
Pl -1 -1 -1 1 1 1 1  -1 -1 -1  NL
CLST
MKST C4
XYST 00 0
SIST 0.93306235 0.92306235
RIST 5
SFMST
RPST 32
CLST

% Figura basada en la rotación de cuatro líneas
OPST SPIRO 2
OPST R3
PL 1 0. 0.9071 0.7071 NG 1 0 0 1 NG
   1 0 -0.7071 0.7071 NG 1 0 0 0 NL
CLST
MKST R3
YXST 0 0
RIST 45
SFMST
RPST 8
CLST

%  Figura basada en la translación de un círculo
OPST SPIRO3
OPST cr
CR 0.5 0 0 NL
CLST
MKST cr
XYST 0.5 0
RDST 22.5
RPST 22.5
CLST

% Árbol fractal (figura recursiva)
OPST AF
VAR THETA PHI
PL 0 0 0 0.5 NL
STST THETA
AF 0.6 0.6
```



RTST THETA
AF O 0.5 NL
RTST PHI
AS O 0.5 NL
CLST

Apendice 2

Algunas instrucciones
Primitivos básicos:
Lineales:
PL lista de puntos {g lista de puntos} NLPoligonal.
SP lista de puntos NL Spline (nodos)
CR r lista de puntos NL Policírculo.
El rx ry lista de puntos NL Polielipse.
Sólidos:
PG lista de puntos NLPolígono (la lista de puntos define la frontera del área a rellenar).
FL lista de puntos NLRellena el área cerrada que contiene a cada punto de la lista.
BR lista de puntos NLRellena el rectángulo refinido por cada grupo de cuatro puntos.
Atributos
LPATRN np Estilo de línea
SPATRN np Estilo de relleno de área sólida
LCOLOR nc Color de linea.
SCOLOR nc color de área sólida.
Transformaciones:
RTMT tetha Rota un ángulo theta.
SCMT sx sy Escala según sx sy
TLMT tx ty Traslada al punto tx ty.
MTMT matriz Matriz de 3x3 definida por el usuario.
IDMT Asigna la idéntica a la matriz de transformación.
Las transformaciones para estructuras son similares, cambiando las dos últimas letras por ST para texto igual, con TXT.



Operaciones con estructuras.
OPST nombre Abre la estructura nombre.
CLST Cierra la última estructura abierta.
MKST nombre Marca la estructura nombre.
PWST x1 y1 x2 y2 Despliega la estructura marcada en rectángulo definido por los datos.
RPST n Despliega la estructura marcada n veces.
DPST nombre Despliega la estructura nombre
Otras instrucciones como recorte, banderas de iteración, uso de variables dentro de una estructura etc. podrán consultarse en el manual del usuario cuando éste exista. ❖

TRANSFORMACIONES Y CONCEPTOS BÁSICOS

En la teoría de la graficación por computadora se fistinguen básicamente tres elementos: primitivaas, atributos y transformaciones. Las primitivas son tales como líneas, curvas, poligonos, etc. Los atributos van siempre asociados a las primitivas, y son el color, el estilo de línea, el patrón de relleno del polígono, etc. Las transformaciones más usadas son cambios de escala, rotaciones y traslaciones; y cada una de ellas puede ser representada con una matriz.
Las coordenadas homogéneas facilitan la composición de distintas transformaciones en una sola matriz, por medio del producto matricial. Si M es la matriz de transformación, en el caso de 2D tendríamos la ecuación vectorial $(x' \; y' \; h) = M \; (x \; y \; 1)^T$, donde la componente adicional h, en 2D no suele tomarse en cuenta. La matriz de transformación generalizada puede ser fividida en tres regiones

$$M = \left[\begin{array}{cc|c} a & b & Tx \\ c & d & Ty \\ \hline 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} I & II \\ \hline III & \end{array}\right]$$

donde la submatriz I de 2x2 produce cambios de escala, rotaciones, reflexiones y deformación. La submatriz II de 2x1 es específica para traslaciones y la III completa la matriz cuadrada, permitiendo el producto matricial.
Como el producto matricial no es conmutativo, es importante el orden en que se multiplican las matrices. Por ejemplo, en el caso

$$M = A \; b \; c \; d$$

$$(x' \; y' \; h) = M \; (x \; y \; 1)^T$$

la transformación en la matriz D será la primera en aplicarse sobre el punto, la siguiente será C y así sucesivamente. ¡En una sola matriz se compactan todas las transformaciones lineales que se deseen!