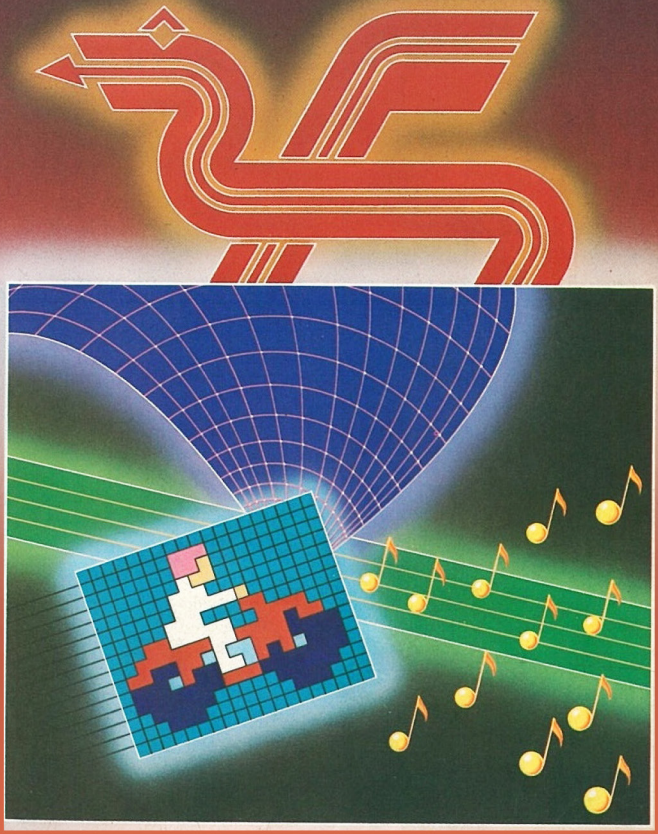
GRAFICOS Y SONIDOS PARA EL DRAGON

INCLUYE SUBRUTINAS EN CODIGO MAQUINA

K. y S. BRAIN



GRAFICOS Y SONIDOS PARA EL DRAGON

INCLUYE SUBRUTINAS EN CODIGO MAQUINA

K. y S. BRAIN

Editorial Gustavo Gili, S. A.

08029 Barcelona Rosellón, 87-89. Tel. 322 81 61
28006 Madrid Alcántara, 21. Tel. 401 17 02
1064 Buenos Aires Cochabamba, 154-158. Tel. 361 99 98
03100 México, D.F. Amores, 2027. Tels. 660 00 21 y 660 28 96
Bogotá Diagonal 45 N.° 16 B-11. Tel. 245 67 60
Santiago de Chile Vicuña Mackenna, 462. Tel. 222 45 67

GG

Título original
Advanced sound & graphics
for the Dragon computer

Primero publicado en inglés en 1983 por
Sunshine Books (an imprint of Scot Press Ltd.)
12/13 Little Newport Street
London WC2R 3LD.

Versión castellana de Jordi Abadal Berini, Ingeniero Industrial y Gabriel Abadal Berini

Printed in Spain
ISBN: 84-252-1232-4
Depósito Legal: B. 24.867-1985
Fotocomposición: TECFA, S. A. - Barcelona
Impresión: HUROPE, S. A. - Recaredo, 2 - Barcelona

Indice

Detalle del contenido

Capítulo 1
Sonido

Cómo ejecutar una melodía: cómo construir una secuencia de notas, cambiar el tempo, la octava y la longitud de la nota, cómo visualizar gráficamente la melodía y añadir palabras a la música. Cómo mejorar sus juegos con efectos sonoros.

Capítulo 2
Texto y baja resolución

Visualización de caracteres; CLS, caracteres gráficos, utilización de las órdenes PRINT TAB, PRINT USING, RESET, POINT y SCREEN.

Capítulo 3
Alta resolución

Selección de la pantalla en alta resolución, selección de la resolución y los colores, selección de PMODE, elección del color principal y el de fondo, tratamiento con puntos individuales en alta resolución.

Capítulo 4
Círculos

El CIRCLE es una orden muy versátil que puede generar muchos tipos distintos de curvas incluyendo elipses, arcos y espirales.

Capítulo 5
La orden DRAW

Cómo experimentar con las 15 órdenes distintas de DRAW para introducir características tales como escala, color, ángulo, movimiento y movimiento en blanco.

Capítulo 6
Combinación de las órdenes gráficas

El programa PIC-MAN demuestra cómo pueden combinarse la mayoría de las órdenes de dibujo en alta resolución en un único programa.

Capítulo 7
Movimientos en la pantalla

Los movimientos en la pantalla toman muchas formas. Este capítulo demuestra las distintas formas de mover dibujos en baja resolución y los combina para mover un dibujo más complicado, el de una nave espacial, sobre la pantalla.

Capítulo 8
Copia de la pantalla

Copia de páginas gráficas completas, almacenamiento y mejoras de las visualizaciones, superposición de dibujos, borrado selectivo, almacenaje de pantallas y reproducción en copias duras.

Capítulo 9
Presentación gráfica de los datos

Diagramas de barras, diagramas de línea, mapas de contorno y diagramas de sectores.

Capítulo 10
Tres dimensiones

La presentación de un objeto desde el punto de vista tridimensional es una forma muy efectiva de convertirlo en algo mucho más sólido. Cómo dibujar en tres dimensiones. Cómo realizar un rectángulo en tres dimensiones, un tubo y una gráfica.

Capítulo 11
Rotación de figuras

Órdenes de dibujo angular: dibujo con un ángulo determinado, formación de triángulos simétricos, rotación de triángulos y rectángulos.

Capítulo 12
Acceso instantáneo a las órdenes de alta resolución mediante el teclado

Cómo dibujar directamente en la pantalla mediante sencillas rutinas de tecla, incluyendo líneas, borrados, círculos, elipses, las órdenes GET y PUT, cambio de colores, rellenado, dibujo con las palancas y rotulado de los diagramas.

Capítulo 13
Las órdenes GET y PUT con caracteres de alta resolución

Cómo utilizar y guardar rutinas de la orden DRAW: transferencia de caracteres entre programas, cómo guardar caracteres en código máquina, dimensionado de tablas, las órdenes GET y PUT con caracteres.

Capítulo 14
Cómo trabajar con una retícula

El trabajar con una retícula es una guía muy útil cuando se quiere con seguridad que una figura encaje con un formato determinado. Formación de la retícula, movimiento, cómo realizar una copia real y guardarla.

Capítulo 15
Animación

Corredor, Sprinter, Volando muy alto y Oasis.

Capítulo 16
Síntesis de sonido

Algunas características del sintetizador: repetición de sonido mediante el teclado, cambio del tempo, control de volumen, cambio de las octavas, envolventes de sonido.

Capítulo 17
Editor gráfico de sonido

El editor le permite entrar una pieza de música, visualizarla según la notación musical estándar en la pantalla y después ejecutarla.

Capítulo 18
Más allá del BASIC

Exploración más profunda dentro del Dragon: cómo utilizar la orden POKE dentro de su programa, cambio de los nombres de las tablas, modos gráficos escondidos, semigráficos, utilización de subrutinas en código máquina, PCLS parcial y desplazamiento de la imagen de la pantalla.

Introducción

El objetivo principal de este libro es enseñarle a obtener el máximo provecho de los gráficos y sonidos en sus programas para el Dragon. El Dragon tiene unas posibilidades gráficas y de sonido de mucha utilidad y el BASIC Microsoft con color que emplea, contiene algunas órdenes muy potentes. Sin embargo, la gran variedad y versatilidad de esta implementación puede constituir una barrera para el principiante, ya que todas pueden parecerle demasiado complicadas hasta el punto de que no sepa por donde empezar. No obstante, la base del libro está en los primeros principios, ya que para desarrollar programas interesantes, utilizando estas características, usted deberá comprender de forma muy clara dos cosas: la manipulación básica de estas órdenes y las mejores maneras de utilizar cada una de ellas. Después de dar las explicaciones directas de cómo deben manejarse las órdenes gráficas y las de sonido, entramos en consideraciones más detalladas de problemas más complicados y en el desarrollo de una serie de herramientas y programas de utilidad. Su valor quizá puede estimarse por el hecho de que las figuras y las copias de pantalla de alta resolución de este libro fueron creadas mediante estos programas.

El formato básico es dar la explicación de una orden o una idea y, a continuación, las rutinas se construyen paso a paso, explorando y comparando las posibilidades alternativas siempre que sea posible. Además, se incluyen copias de las visualizaciones en pantalla de alta resolución, de forma que puede verse lo que hay que esperar de ello. En lugar de decirle sencillamente lo que hay que hacer y lo que no hay que hacer, le animamos a experimentar con las distintas posibilidades para que usted vea los resultados por sí mismo. Siempre que sea posible, se evita la reescritura de líneas, pero es frecuente la modificación de líneas. Todos los listados de este libro están formateados en 32 columnas de forma que aparezcan igual que los verá en la pantalla, excepto que los caracteres invertidos aparecerán como caracteres en minúscula. En la mayoría de los casos, los espacios y las comillas se han utilizado de forma liberal, para hacer que los listados sean más fáciles de leer, pero tenga en cuenta que algunos espacios y comillas son esenciales, por lo tanto no intente eliminarlos todos.

Todas las rutinas se han comprobado rigurosamente y los listados comprobados en su totalidad, por lo que esperamos que no encuentre ningún error. Es una triste realidad de la vida que la mayoría de los errores aparecen como resultado de errores de escritura cometidos por el usuario. Las comas y los puntos y comas pueden parecer muy insignificantes, pero su ausencia puede producir efectos profundos. Si se encuentra con problemas en los listados de este libro, o con sus propios programas, no se olvide de que el Dragon tiene la función TRON que le permite seguir la ejecución de sus programas, y que el pulsar el SHIFT y la @ detendrá la ejecución del programa hasta que pulse cualquier otra tecla.

Si esto no resuelve el problema, recuerde que las variables no se cambian hasta que se ejecute el programa o utilice el EDIT, por lo que puede visualizarlas mediante la orden PRINT y ver si sus valores son correctos. Los errores de sintaxis (?SN ERROR) generalmente son resultado de la falta (o la excesiva utilización) de las comillas, las comas o los signos de texto ($), o el resultado de errores en la escritura de palabras, pero también pueden ser el resultado de quitar espacios esenciales. Los errores de función (?FC ERROR) generalmente aparecen porque se está intentando utilizar un valor ilegal en una función. Los errores de escritura (?TM ERROR) aparecen si se mezclan variables sencillas y de texto. La utilización de gran cantidad de texto puede dar como resultado el quedarse sin espacio para textos (?OS ERROR) que generalmente se solventa reservando más espacio para textos mediante la orden CLEAR. La longitud máxima de un texto es 255 caracteres, por lo tanto si intenta juntar dos textos muy largos obtendrá un ?LS ERROR. Si se olvida de DIMensionar una tabla, ésta es demasiado pequeña o busca un valor negativo obtendrá un error de subíndice incorrecto (?BS ERROR). Si el programa alcanza un NEXT o un RETURN sin haber ejecutado el correspondiente FOR o GOSUB obtendrá un ?NF ERROR o un ?RG ERROR. Finalmente, si intenta utilizar un GOTO o un GOSUB hacia una línea inexistente obtendrá un ?UL ERROR. Esto quizá sea debido a que ha olvidado colocar una línea o quizá la haya borrado accidentalmente.

Los dragones míticos eran bestias ruidosas de gran colorido y esperamos que este libro le ayude a convertir su Dragon contemporáneo para que muestre características similares, sin pillarse los dedos o gastar demasiada energía por las noches.

Keith y Steven Brain
Groeswen, julio de 1983

1. Sonido

Primeros sonidos

Todo el mundo empieza en la vida con sonidos sencillos, pero a lo largo de los años su vocabulario va aumentando. Así, pues, ¿por qué no aprender acerca del sonido de ordenadores de la misma forma? Empezaremos con la orden SOUND que es la manera más sencilla de crear sonidos en el Dragon. Necesita dos parámetros, tono y duración.

```
SOUND n,n
```

El primer número es el tono y puede ser cualquier número entre 1 y 255 ambos inclusive. El tono 89 corresponde al DO medio en un piano.

El segundo número es la duración, que también puede ser entre 1 y 255 ambos inclusive. Un valor de 16 para la duración corresponde aproximadamente a un segundo.

Por ejemplo:

```
SOUND 1,16
```

generará una nota baja durante un segundo y si cambiamos el tono a un valor más alto:

```
SOUND 255,16
```

generará una nota alta durante un segundo.

De la misma forma, cambiando el segundo parámetro:

```
SOUND 1,160
```

generará una nota baja durante diez segundos y

```
SOUND 255,160
```

producirá una nota alta durante diez segundos.

Conseguir sonidos sencillos no es muy interesante, por lo tanto coloque la orden SOUND dentro de un bucle FOR... NEXT con lo que

obtendrá todos los tonos que pueden conseguirse con las órdenes de sonido del Dragon, una a una y por orden ascendente.

```
10 FOR N=1 TO 255
20 SOUND N,1
30 NEXT N
```

Si invierte el bucle se producirá una serie descendente.

```
10 FOR N=255 TO 1 STEP-1
20 SOUND N,1
30 NEXT N
```

La duración también puede variarse dentro del bucle. Si ahora ustituimos la duración de 1 por N, el sonido será cada vez más largo.

```
20 SOUND N,N
```

Cuando llegue a este punto, probablemente intentará detener el terrible ruido que se está produciendo, ya que parece que vaya a continuar para siempre (255 da una duración de cerca de 16 segundos). Sin embargo, quizá le sorprenda descubrir que la tecla BREAK parece no funcionar. De hecho, cuando se está ejecutando cualquier orden SOUND la CPU está tan ocupada que no puede recorrer el teclado, por lo tanto la tecla BREAK funciona únicamente en el pequeño espacio de tiempo entre dos sonidos.

Si se quiere relacionar la duración con la variable N dentro del bucle, entonces necesitará dividirla de alguna manera para obtener una longitud adecuada. Cuando haga esto deberá tener cuidado de no generar valores ilegales (menores que 1). La solución más sencilla es añadir siempre uno al valor calculado.

```
20 SOUND N,N/20+1
```

Naturalmente puede utilizarse cualquier valor para el STEP del bucle FOR-NEXT.

```
10 FOR N=255 TO 1 STEP-5
20 SOUND N,1
30 NEXT N
```

Un STEP impar producirá un sonido más interesante.

```
10 FOR N=1 TO 255 STEP RND(5)
```

Pero aunque se utilicen los bucles FOR...NEXT se queda restringido a una secuencia, pero se podría colocar los valores del tono y la

duración en sentencias de DATA que podrían leerse (READ) cuando fuese necesario.

```
10 DATA 10,5,39,27,234,221,56,44
,37,11,33,24,75,64,31,53,20,24,4
8,65
20 FOR N=1 TO 10
30 READ P,D
40 SOUND P,D
50 NEXT N
```

El programa anterior generará una extraña secuencia de notas y aunque se puede utilizar este método para ejecutar sencillas melodías, un método mejor es utilizar la orden PLAY. Por lo tanto, la orden SOUND es mejor dejarla para generar notas largas o para ejecutar secuencias de notas fijas.

Ejecución de una melodía (PLAY)

El Dragon tiene un método más potente para crear sonido a través de la orden PLAY, que tiene la sintaxis:

```
PLAY A$
```

donde A$ es un texto que puede tener hasta 255 caracteres de largo. Permite definir una serie completa de notas y también permite controlar mejor la forma en que éstas son ejecutadas.

La forma más fácil para decidir qué notas se van a ejecutar es utilizar las letras A-G para indicar las notas A-G. Estas están ordenadas en una secuencia fija (o escala para los que entiendan de música). La escala de C es:

C D E F G A B

Compare con el pentagrama musical mostrado en la figura 1.1.

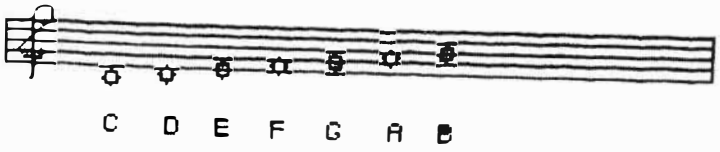


Fig. 1.1 Notas sobre el pentagrama.

Para ejecutar la escala de C se puede escribir esto como una orden directa:

```
PLAY"C;D;E;F;G;A;B"
```

Obsérvese que los puntos y comas son opcionales y que serán ignorados cuando se ejecute la serie de notas. Si se omiten, no se apreciará ninguna diferencia.

```
PLAY"CDEFGAB"
```

En general, existe la tendencia de omitir los puntos y comas ya que ocupan espacio, pero a veces tiene cierta utilidad el colocar algunos ya que puede hacer que las secuencias sean más fáciles de leer.

(Aunque es posible definir las notas mediante los números del 1 al 12 esto puede producir cierta confusión, por lo tanto olvídese de esta idea por el momento.)

Si se tiene la partitura de una canción, pueden copiarse las notas en un texto, pero hay que tener en cuenta una serie de otros factores. Para ilustrar los distintos parámetros que deben tenerse en consideración, veamos cómo entrar la melodía de un conocido villancico. Las notas son las siguientes:

```
10 PLAY"FFCFGCAGABAGFFEDEFGAEDCC
CBABAGAFGEDCFFEFGFCAGABAGABAGFEF
BAGFF"
```

Si se ejecuta esta orden y se está muy atento, quizá reconozca la melodía que está intentando ejecutar el ordenador, pero está algo desafinada. Para corregir esto necesitaremos indicarle qué notas deben ser sostenidas. Los bemoles y los sostenidos se indican añadiendo otro carácter después de la nota. Para indicar un bemol hay que añadir el signo «−» después de la letra correspondiente a la nota, y para indicar una nota sostenida hay que añadir el signo «#» o el signo «+» después de la letra. La misma melodía con los bemoles y sostenidos adecuados sería:

```
10 PLAY"FFCFGCAGAB-AGFFEDEFGAEDC
CCBABAGAFGEDCFFEFGFCAGAB-AGAB-AG
FEFB-AGFF"
```

El ordenador puede detectar si ha añadido bemoles y sostenidos a las notas de forma aleatoria, ya que es más inteligente que usted y rechaza un B# o un C− ya que no forman parte de los doce tonos de la escala musical.

Incluso cuando ejecute esta versión modificada notará que todavía sigue habiendo algo incorrecto. Para empezar, la melodía debe cambiar de octava, en ciertos puntos, y pasar a un grupo más alto de notas A-G. La figura 1.2 muestra dos octavas en el pentagrama. Para cambiar de octava se utiliza:

```
On
```

donde n es un número del 1 al 5. El valor por defecto es 2. Por ejemplo:

```
PLAY"CDEFGAB"
```

es distinto que

```
PLAY"O3CDEFGAB"
```

ya que la segunda escala está ocho notas por encima (una octava).

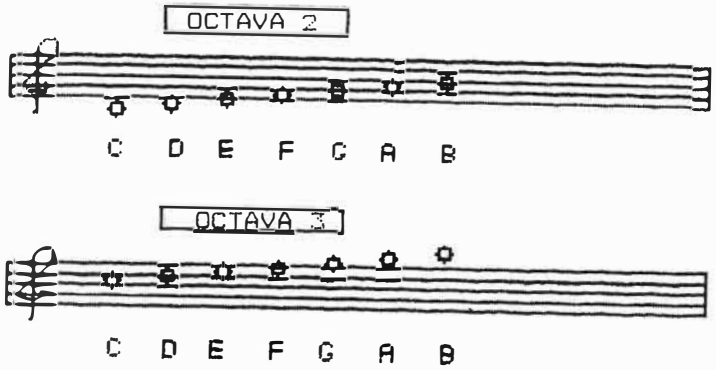


Fig. 1.2 Dos octavas.

Cuando se cambia el parámetro correspondiente a la octava, esta nueva octava será utilizada hasta que se haga otro cambio; por lo tanto hay que recordar el restablecerla si fuese necesario. La melodía corregida sería:

```
10 PLAY"O2FFCFGCAGAB-AGFFEDEFGAE
DCCO3CO2BABAGAFGEDCFFEFGFCAGAB-A
GAB-AGFEFB-AGFF"
```

El 02 del principio no es estrictamente necesario en este caso, ya que se utiliza la octava 2 por defecto. Pero es una buena idea incluirlo

ya que de otra forma podrían aparecer complicaciones cuando se terminase el texto en una octava distinta y después se repitiese la secuencia. Entre estas líneas e intente ver las diferencias entre la primera vez que se ejecuta la melodía y las repeticiones.

```
100 PLAY"CDEFGABO3CDEFGAB"
110 GOTO 110
```

La primera vez se utiliza Ø2 para la primera escala y Ø3 para la segunda, pero después Ø3 se utiliza para ambas escalas en las repeticiones. Para repetir la primera cada vez, hay que añadir el Ø2 al principio o al final del texto.

```
100 PLAY"O2CDEFGABO3CDEFGAB"
```

o

```
100 PLAY"CDEFGABO3CDEFGABO2"
```

Sin embargo, nuestro villancico todavía no está del todo correcto ya que algunas notas son más cortas de lo que debieran ser, y algunas son demasiado largas. El parámetro de longitud de la nota (L) resolverá este problema.

Ln

donde n es cualquier número del 1 al 255 y el valor por defecto es 4. A medida de que el número aumenta, la longitud de la nota disminuye. L1 es una nota completa, L2 media nota, L4 un cuarto de nota, etc. Si se quieren obtener valores intermedios se puede «puntillear» la nota. Añadiendo un punto después del número se conseguirá que el valor sea de nuevo la mitad de largo.

Por ejemplo L2.

sería $\frac{1}{2}$ nota + $\frac{1}{4}$ nota = $\frac{3}{4}$ nota.

A continuación se muestra la melodía con las longitudes de las notas corregidas.

```
10 PLAY"O2FL2FL4CFL2GCL4AGAB-L2A
L4GFL2FL4EDEFGAL2EL4DL8CL2CO3CO2
L4BAL2BAL4GAFGL4EL8DL4CFFEFGL2FL
4CAGAB-L2AL4GAB-AGFL2EL4FB-L2AL4
GL8FL2F"
```


Finalmente, podemos acelerar el proceso un poco, cambiando el parámetro Tempo (T).

Tn

donde n es un número del 1 al 255.

La longitud de la nota establece el tiempo que dura cada nota con relación a las otras notas, pero el Tempo altera la relación con que todas las notas son ejecutadas, en la misma proporción.

El valor por defecto de T es 2, por lo tanto cámbielo por un 5 y observe que aunque la melodía se ejecuta ahora más rápidamente, la longitud de las notas mantiene la misma proporción.

```
10 PLAY"O2T5FL2FL4CFL2GCL4AGAB-L
2AL4GFL2FL4EDEFGAL2EL4DL8CL2CO3C
O2L4BAL2BAL4GAFGL4EL8DL4CFFEFGL2
FL4CAGAB-L2AL4GAB-AGFL2EL4FB-L2A
L4GL8FL2F"
```

Ya hemos visto que un texto podía incluirse dentro de un bucle para repetir la melodía, pero ¿cómo generar una pausa para descansar un poco al final de cada verso? La sintaxis de la pausa es como la de N y T, pero en este caso tan sólo espera el tiempo especificado (1-255) sin generar ningún sonido. Si añadimos una orden PLAY con una pausa antes de repetir la melodía crearemos un descanso entre los versos.

```
20 PLAY"P50"
30 GOTO 10
```

Si tenemos distintos grupos en el coro cantando los distintos versos, quizá deberíamos alterar el volumen en cada repetición. Naturalmente, puede cambiarse el volumen utilizando los mandos del televisor, pero también puede conseguirse que el Dragon controle el volumen, utilizando el parámetro volumen (V). La orden Volumen tan sólo se diferencia de los otros parámetros en que utiliza valores comprendidos entre 1 y 31, en lugar de entre 1 y 255, por lo que no puede intentarse competir con el equipo estéreo del vecino utilizando un valor muy alto. El valor por defecto es «V15» (medio volumen).

Si añadimos

```
30 PLAY"V31"
40 GOTO 10
```

la primera vez el volumen será la mitad (V=15) y en las repeticiones el volumen será el máximo (V=31).

Si se quiere repetir una determinada secuencia musical o un conjunto de órdenes, puede ser de utilidad el definir éstas como subtextos que pueden ejecutarse dentro de la orden PLAY mediante la «X». La sintaxis es:

```
10 PLAY"XA$"
```

donde A$ es un texto musical válido y el punto y coma es ESENCIAL. Se pueden ejecutar y reejecutar distintas frases para que constituyan una melodía completa. Por ejemplo, la melodía de un verso del villancico anterior podría definirse como un texto y entonces ejecutarse con tres valores distintos de Volumen y Tempo.

```
10 A$="O2T5FL2FL4CFL2GCL4AGAB-L2
AL4GFL2FL4EDEFGAL2EL4DL8CL2CO3CO
2L4BAL2BAL4GAFGL4EL8DL4CFFEFGL2F
L4CAGAB-L2AL4GAB-AGFL2EL4FB-L2AL
4GL8FL2F"
20 PLAY"T5V15XA$;T4V8XA$;T6V31XA
$;"
```

Cómo ver lo que se está haciendo

Es posible visualizar las notas que se están ejecutando si se definen éstas como un texto y no únicamente ejecutarlas mediante la orden PLAY, sino también visualizarlas con la orden PRINT. Si se utilizan notas únicamente entonces todo es muy sencillo.

```
10 A$="CDEFGAB"
20 PLAY A$
30 PRINT A$
```

Quedaría mejor si se fuera visualizando cada nota a medida que se va ejecutando, por lo tanto descompongamos el texto y visualicemos cada nota exactamente antes de que se ejecute.

```
20 FOR P=1 TO LEN(A$)
30 B$=MID$(A$,P,1)
40 PRINT B$;:PLAY B$
50 NEXT P
```

Ahora es más fácil el detectar, caso que haya alguna, qué nota es incorrecta.

Naturalmente, los textos «reales» que se van a ejecutar tienden a ser más complicados que éste, y no todas las órdenes son de la misma longitud, por lo que el descomponer el texto puede ser más difícil. Si añadimos sostenidos y bemoles y los valores de Volumen, Longitud de la nota, Tempo y Octava que son menos de 1Ø, todavía podemos utilizar únicamente dos posiciones en el texto para definir cada orden y después descomponer el texto en secciones de dos caracteres cada uno. Debe tomarse alguna precaución con respecto a la forma en que se entra cada carácter, ya que no todas las ordenaciones son aceptables. La regla, a la hora de descomponerlo, es que el último carácter no debè ser un espacio, por lo tanto si una nota no es ni bemol ni sostenida debe ir precedida por un espacio.

```
10 A$="T5O1F# G AB- E F C"
20 FOR P=1 TO LEN(A$) STEP 2
30 B$=MID$(A$,P,2)
```

Incluso pueden acomodarse órdenes más largas (sin espacios al final) aunque eso tendrá tendencia a ocupar mucho más espacio, ya que deberán insertarse muchos espacios para acomodar cada orden a la longitud de la orden más larga

```
10 A$="V31 T5 O1 F#  G  A B-  E
 F  C"
20 FOR P=1 TO LEN(A$) STEP 3
30 B$=MID$(A$,P,3)
```

y

```
10 A$=" V31T100  O1  F#   G   A
 B-   E   F   C"
20 FOR P=1 TO LEN(A$) STEP 4
30 B$=MID$(A$,P,4)
```

Para hacer realmente más fácil el ver lo que sucede cuando se está copiando música o componiendo sus propias melodías, hemos diseñado un programa gráfico de música que visualiza la música exactamente de la misma forma como aparece sobre papel. Esto se describe con detalle más adelante, pero resista la tentación de saltar directamente hasta allí, ya que primero deberá estudiar las órdenes gráficas para que pueda comprender cómo funcionan.

Palabras y música

Si piensa por un momento en cómo hemos visualizado las órdenes, probablemente se dará cuenta que en lugar de visualizar la orden que se estaba ejecutando podríamos visualizar algo en su lugar, sien-

do lo más obvio el que sean las palabras de la canción. Cuando se añadan las palabras será necesario colocar la sílaba adecuada junto a la nota correspondiente y también asegurarse de que no se visualiza nada en las órdenes que no son notas (por ejemplo en los cambios de octava o de la longitud de la nota). Las sílabas se colocan como datos en las líneas 1 y 2, añadiéndole los espacios correspondientes para obtener una visualización correcta. Necesitaremos leer estos datos y visualizarlos sólo en el caso de que la orden que se está ejecutando sea una nota y por ello las órdenes son clasificadas mediante la orden INSTR que las compara con X$, un texto que contiene todas las notas utilizadas en la canción. Cuando el fragmento siguiente no es una nota no se leerá nada.

```
1 DATAO ,COME ,ALL ,YE ,FAITH,FU
LL          ,JOY,FULL ,AND ,TRI,U
M,PHANT           ,O ,COME ,YE ,O
 ,CO,ME ,YE ,TO ,BE,TH,LE,HEM,CO
ME ,AND ,BE,HOLD ,HIM
  ,BORN ,THE ,KING ,OF ,A,N,GELS
         ,O ,COME ,LET ,US ,AD,O
RE ,HIM
2 DATACOME ,LET ,US ,AD,ORE ,HIM
           ,O ,COME ,LET ,US ,AD
,ORE ,H,IM          ,CHRI,ST ,THE
 ,LORD ,
10 X$="B- A B C D E F G"
20 CLS
30 A$="O2T5 FL2 FL4 C FL2 G CL4
A G AB-L2 AL4 G FL2 FL4 E D E F
G AL2 EL4 DL8 CL2 CF4L2O3 CO2L4
B AL2 B AL4 G A F GL4 EL8 DL4 C
F F E F GL2 FL4 C A G AB-L2 AL4
G AB- A G FL2 EL4 FB-L2 AL4 GL8
FL2 F"
40 FOR N=1 TO LEN(A$) STEP 2
50 B$=MID$(A$,N,2)
60 X=INSTR(1,X$,B$)
70 IF X<>0 THEN READ C$:PRINT C$
;
80 PLAY B$
90 NEXT N
```


Efectos sonoros

Además del interés que tienen para producir música, las órdenes SOUND y PLAY también son de mucha utilidad para generar efectos sonoros. Los posibles efectos sonoros son muchos y variados, pero suelen utilizar complejos cambios de parámetros para conseguir sus efectos.

Estos cambios pueden formar parte de una secuencia preestablecida o pueden relacionarse con variables del programa. Los problemas más comunes relacionados con la generación de efectos sonoros son la introducción de cambios ligados al programa y a la generación accidental de valores de parámetros ilegales.

La orden SOUND es la más fácil de utilizar ya que las variables pueden alterarse directamente como ya se ha descrito, aunque el ruido que generan no es muy interesante, a menos que se repita cambiando los parámetros. Un tono de Ø no está permitido, por lo tanto vaya con cuidado de que no caiga en este valor. Una manera sencilla de evitar esto es añadir siempre 1 a la variable utilizada en la orden SOUND. Esta rutina visualiza y genera cada uno de los tonos desde 1 a 255 aleatoriamente.

```
10 A=255
20 B=RND(255)
30 PRINT (A-B)
40 SOUND (A-B)+1,1
50 GOTO 20
```

Los distintos sonidos pueden relacionarse a cada una de las teclas si se utiliza la orden INKEY$ y se relaciona el tono con el valor ASCII de la tecla. Ya que la orden SOUND necesita variables sencillas, primero deberá convertirse A$ en A.

```
10 A$=INKEY$:IF A$="" THEN 10
20 A=ASC(A$)
30 SOUND A,3
40 GOTO 10
```

El bucle sobre la misma línea que aparece en la línea 1Ø, cuando no se pulsa ninguna tecla, es esencial para evitar que el programa se pierda, lo que sucedería si la línea 2Ø intentase obtener el valor ASCII de un texto vacío. Las diferencias en una unidad en el tono no son fáciles de detectar, por lo tanto es mejor multiplicar A por un factor, para que las diferencias entre las teclas sean mayores. El mayor factor que no producirá valores ilegales es 2 ya que 2 × 127 = 254.

La orden PLAY es más versátil, pero la introducción de variables es un poco más complicada ya que PLAY actúa sobre un texto. Cualquier variable sencilla que deba utilizarse, primero deberá convertirse al formato de un texto mediante la orden STR$ y después añadirse a la letra que indica el parámetro que debe variarse. El siguiente programa ejecutará una escala en una octava aleatoria.

```
10 A=RND(4)
20 PLAY"O"+STR$(A)
30 PLAY"CDEFGAB"
40 GOTO 10
```

Ya que la orden PLAY actúa sobre un texto podrá generar directamente una sencilla melodía sobre el teclado ejecutando el contenido de INKEY$.

```
20 A$=INKEY$
30 PLAY A$
40 GOTO 20
```

No es necesario el comprobar si A$ está vacío, ya que la orden PLAY sobre un texto vacío está permitida. Naturalmente, tan sólo pueden pulsarse las teclas correspondientes a las notas sin que el programa se pierda. Una forma sencilla de asegurarse de que tan sólo se aceptan los valores legales es el utilizar la orden INSTR para comparar la tecla pulsada con otro texto (N$) que contenga una lista de las teclas válidas.

```
10 N$="ABCDEFG"
30 IF INSTR(1,N$,A$)>0 THEN PLAY
 A$
```

Aunque el Volumen, Octava, Tempo y la Longitud de la nota pueden alterarse cambiando los valores actuales, tal como se ha descrito anteriormente, a veces es más conveniente utilizar un método alternativo que incrementa o decrementa automáticamente los valores. Para utilizar este método automático tan sólo hay que añadir uno de estos sufijos al parámetro:

\+ añade uno al valor actual
\− resta uno del valor actual
\> multiplica el valor actual por dos
\< divide el valor actual por dos

Obsérvese que las dos primeras cambian lentamente en incrementos de una unidad, pero que las dos últimas hacen cambios más drásticos, ya que doblan o dividen por dos el valor actual a cada paso.

Un aspecto que hay que vigilar en estas órdenes es que es muy fácil alcanzar valores ilegales y perder el programa. Por ejemplo:

```
100 PLAY"V>"
```

doblará el nivel del Volumen por defecto (15) hasta casi el valor máximo de salida (3Ø), pero:

```
100 PLAY"V>"
110 GOTO 100
```

no funcionará ya que se calculará un valor de 6Ø para el Volumen.

El aumentar o disminuir el Volumen sirve para indicar que algo se acerca o se aleja. Por ejemplo, esto produce una sirena de policía que cada vez es más alta a medida que se acerca:

```
20 PLAY"L2T4FGV+"
30 GOTO 20
```

Tal como está, empieza por un volumen medio hasta que el programa se pierde cuando se alcanza un valor de V>31. Para empezar desde un volumen mínimo hay que definir esto de forma separada y fuera del bucle, en la línea 1Ø, y en la línea 3Ø comprobar que no se han realizado más de 3Ø bucles.

```
10 PLAY"V1"
30 L=L+1:IF L<31 THEN 20
```

Si quiere que se acerque de una forma más dramática puede sustituir el + por >, pero si empieza desde V1 deberá poner V al valor de 31 para la cuarta repetición, ya que otro V más daría 32.

```
10 PLAY"V1L2T4FGV>FGV>FGV>FGV>FG
V31FG"
```

Por regla general, los programas de juego necesitan lasers, explosiones, etc., y éstos utilizan casi siempre Tempos y/o Longitudes muy cortas y cambios en las Octavas y el Volumen para generar los efectos deseados. He aquí algunos ejemplos, pero puede trabajar sobre ellos un buen rato y obtener resultados más impresionantes.

```
10 PLAY"T255L255CDEFGAB":GOTO 10

(escala de acenso muy rapido)
```

```
10 PLAY"T255L255CDEFGABAGFEDC":G
OTO 10
```

(escala ascendente y descendente)

```
10 PLAY"O1":FOR N=1 TO 4:PLAY"T2
55CDEFGABO+":NEXT N:GOTO 10
```

(incremento de octava)

```
10 PLAY"O5":FOR N=1 TO 4:PLAY"T2
55CDEFGABO-":NEXT N:GOTO 10
```

(decremento de octava)

```
0 FOR N=1 TO 30:PLAY"V"+STR$(N)+
"O1T255DCDCDCDC":NEXT N
```

(incremento de volumen)

```
0 FOR N=30 TO 1 STEP-1:PLAY"V"+S
TR$(N)+"O1T255DCDCDCDC":NEXT N
```

(decremento de volumen)

Audio

La última función relacionada con el sonido que no hemos considerado todavía es el AUDIO ON/OFF, que permite conectar y desconectar una señal externa proveniente de la entrada de cassette al altavoz del televisor. Mediante esto se puede ejecutar cualquier tipo de sonido que provenga de la cinta de cassette. Ya que el Dragon también permite conectar y desconectar el cassette (MOTOR ON/OFF) esto significa que se tiene un control total sobre la música. Una sencilla aplicación de esto es una versión controlada por el ordenador del juego «de las sillas musicales». Si coloca una cinta de música en el cassette, pulsa la tecla de PLAY y ejecuta el siguiente programa, entonces la música sonará durante un tiempo aleatorio.

```
10 AUDIO ON:MOTOR ON
20 FOR N=1 TO RND(100000)+10000:
NEXT N
30 AUDIO OFF:MOTOR OFF
40 GOTO 10
```


Otra utilización es generar instrucciones habladas para un programa mientras se está ejecutando la demostración. Deberá grabar el programa mediante la orden CSAVE y después grabar su propia voz a continuación, dejando un pequeño espacio. Esta secuencia puede incluirse como una línea de programa con los avisos correspondientes.

```
60000 CSAVE"nombre":MOTOR ON:FOR
 N=1 TO 5000:NEXT N:MOTOR OFF
60010 CLS:PRINT"CUANDO ESTES PRE
PARADO PARA GRABAR TU VOZ","PULS
A UNA TECLA"
60020 Q$=INKEY$:IF Q$="" THEN 60
020
60030 MOTOR ON:PRINT,,"GRABANDO"
,,"CUANDO TERMINES PULSA UNA TEC
LA"
60040 Q$=INKEY$:IF Q$="" THEN 60
040
60050 MOTOR OFF:PRINT,,"FIN DE L
A GRABACION"
```

Ahora tan sólo necesita añadir MOTOR ON: AUDIO ON cerca del principio de su programa y MOTOR OFF: AUDIO OFF cuando la grabación termine.

```
10 CLS:PRINT"HOLA",,"QUIERES INS
TRUCCIONES ORALES"
20 Q$=INKEY$:IF Q$="" THEN 20
30 IF Q$<>"S" THEN 100
40 AUDIO ON:MOTOR ON
50 CLS:PRINT"PARA PARAR EL PLAYB
ACK PULSA UNA TECLA"
60 Q$=INKEY$:IF Q$="" THEN 60
100 (resto del programa)
```

Una aplicación más seria es enlazar una cinta hablada con un programa de aprendizaje que compruebe la ortografía, vocabulario, etc. Entonces podría arreglarse de forma que se oyese una palabra cuya ortografía correcta o traducción debe entrarse. También podría colocarse un tiempo de espera para comprobarlo (utilizando la función TIMER) conectando y desconectando el motor si todos los fragmentos hablados fueran más o menos de la misma longitud. Un problema que puede aparecer con esto es el de la sincronización del programa y la voz, ya que la velocidad del motor puede ser bastante variable. Una

alternativa es hacer que el usuario pulse una tecla determinada para poner en marcha o detener la cinta.

```
100 AUDIO ON:MOTOR ON
110 PRINT"PULSA ENTER PARA PARAR
 LA CINTA",,,,
120 INPUT Q$:AUDIO OFF:MOTOR OFF
130 INPUT"RESPUESTA";A$
```

(rutina de comprobación)

```
200 GOTO 100
```


2. Texto y baja resolución

La pantalla de textos del Dragon contiene 512 posiciones en una matriz de 32 por 16 y las visualizaciones gráficas de baja resolución nos dan 2Ø48 puntos, que pueden controlarse individualmente en una matriz de 64 por 32. Ambas matrices utilizan la misma área de memoria (direcciones 1Ø24 a 1535) por lo que los textos y los gráficos de baja resolución pueden mezclarse fácilmente.

Caracteres

Los caracteres alfanuméricos y los de control de visualización están definidos por números del Ø al 127 y los números del 128 al 255 especifican los caracteres gráficos. Tan sólo algunos de estos caracteres pueden conseguirse directamente desde el teclado. Algunos otros pueden visualizarse utilizando la función CHR$ y el siguiente programa muestra todos los caracteres disponibles utilizando este método.

```
10 CLS
20 FOR N=0 TO 255
30 PRINT CHR$(N);
40 NEXT N
```

Los caracteres extra más importantes que están disponibles utilizando la función CHR$ son los caracteres gráficos (códigos del 128 al 255). Estos consisten en ocho conjuntos de 16 bloques en los cuales están sin iluminar distintos segmentos. Los caracteres de control de la visualización dan blancos y por lo tanto no pueden visualizarse.

Además, existen algunos otros caracteres que pueden visualizarse, pero únicamente si se cambia directamente el contenido de las posiciones de la memoria de pantalla, ya que el interpretador del BASIC no lo permite. Todos los caracteres disponibles en el Dragon pueden visualizarse añadiendo las siguientes líneas que visualizan mediante la orden POKE los números del Ø al 255 en la memoria de pantalla.

```
50 FOR N=0 TO 255
60 POKE 1280+N,N
70 NEXT N
```

Ya que la posición 128Ø es equivalente a la posición de visualización 256, estos caracteres aparecerán debajo del primer conjunto. Si analiza cuidadosamente las dos versiones verá que hay varias diferencias. Hay una línea entera de símbolos inversos y números en la versión que utiliza el POKE y además, cuando los caracteres aparecen en las dos secciones, no están siempre en el mismo lugar. Podrá utilizar estos caracteres invertidos extra en sus programas siempre que utilice la orden POKE, y las diferencias en el orden de los caracteres significa que debe irse con cuidado si se utiliza la orden PEEK para detectar lo que hay en un punto determinado de la pantalla. La tabla 2.1 compara los valores de CHR$ y POKE.

CLS

La pantalla de texto/baja resolución puede borrarse y dejarse en cualquiera de los nueve colores. La orden CLS tiene la sintaxis:

```
CLS n
```

donde n es un número del Ø al 8.

Los códigos de los nueve colores son:

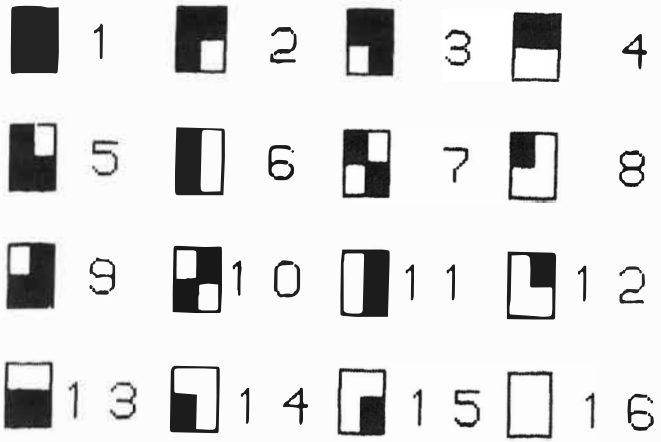
Ø – Negro
1 – Verde
2 – Amarillo
3 – Azul
4 – Rojo
5 – Marrón
6 – Cian
7 – Magenta
8 – Naranja

El CLS por defecto, si no se añade ningún número, es verde, pero hay que tener en cuenta que existen dos caracteres verdes que parecen iguales, pero tienen códigos distintos. CHR$ (143) es un carácter gráfico y CHR$ (96) es el carácter del espacio. Aunque CLS1 a CLS8 rellenan la memoria de pantalla con el carácter gráfico de color apropiado, el CLS utiliza el CHR$(96) en lugar del CHR$(143), lo que a veces puede producir alguna confusión. Para ver mejor este punto intente entrar CLS y CLS1 como órdenes directas y después ejecute PRINT PEEK(128Ø) para ver con qué se rellena la pantalla.

Caracteres gráficos

Como ya hemos visto los códigos 128 a 255 definen los bloques gráficos de color, y existen 16 caracteres distintos (fig. 2.1) disponibles en cada color. Estos bloques pueden combinarse fácilmente para realizar un dibujo (fig. 2.2).


```
10 CLS 0
30 PRINT CHR$(129);CHR$(135);CHR
$(131)
40 PRINT CHR$(128);CHR$(133);CHR
$(128)
50 PRINT CHR$(128);CHR$(142);CHR
$(138)
60 PRINT CHR$(128);CHR$(136);CHR
$(136)
```



+ 127 = Verde
+ 143 = Amarillo
+ 159 = Azul
+ 175 = Rojo
+ 191 = Marrón
+ 207 = Cian
+ 223 = Magenta
+ 239 = Naranja

Fig. 2.1 Caracteres gráficos.

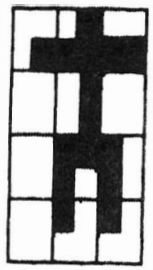


Fig. 2.2 Formación de un dibujo mediante caracteres gráficos.

Ya que la secuencia de bloques es la misma para cada color, el color del dibujo puede cambiarse fácilmente añadiendo múltiplos de 16 a los códigos de cada uno de los caracteres incluidos. La forma más sencilla es añadir una variable de incremento (I) a cada número CHR$ y después cambiarlo.

```
30 PRINT CHR$(129+I);CHR$(135+I)
;CHR$(131+I)
40 PRINT CHR$(128+I):CHR$(133+I)
;CHR$(128+I)
50 PRINT CHR$(128+I);CHR$(142+I)
;CHR$(138+I)
60 PRINT CHR$(128+I):CHR$(136+I)
;CHR$(136+I)
```

Si añade este bucle al programa anterior mostrará el mismo dibujo en el color superior cada vez que se pulse una tecla.

```
20 FOR N=0 TO 112 STEP 16
70 Q$=INKEY$:IF Q$="" THEN 70
80 NEXT N
```

PRINT @

La orden PRINT @ le permite definir una posición de visualización para el siguiente carácter, en cualquier lugar de la pantalla de texto. Tiene la sintaxis:

PRINT @ p, «mensaje»

donde p es un número entre Ø y 512.

Puede utilizarse para posicionar tanto los textos como los caracteres gráficos.

PRINT TAB

Aunque no esté mencionado en el manual, la orden PRINT TAB está disponible y permite desplazar la siguiente posición de visualización un número especificado de posiciones, a partir de la posición actual. El formato es:

PRINT TAB(d); «mensaje»

donde d es el desplazamiento a partir de la posición actual de visualización.

PRINT USING

Esta orden permite controlar de forma precisa el formato de salida hacia la pantalla o hacia una impresora. Requiere que se defina el formato deseado y después se le dé una «lista de salida» de los elementos que deben visualizarse.

PRINT USING formato; lista de salida

Hay muchas formas distintas de formatear una salida, pero en los programas gráficos generalmente tan sólo estamos interesados con los textos. Por lo tanto, la única orden interesante de formato es el %. Si se define la longitud máxima de un texto a visualizar como el número de espacios comprendidos entre dos %, entonces cualquier texto más largo que esto aparecerá recortado para que quepa en el espacio asignado.

```
10 PRINT"COMO TE LLAMAS?"
20 INPUT A$
30 PRINT USING"%         %";A$
```

Esto puede tener su utilidad cuando se quiera evitar un valor inesperado, o la entrada de un mensaje, de forma que no estropee una visualización cuidadosamente planeada.

SET

La orden SET ilumina el «pixel» definido por las coordenadas X e Y con el color especificado.

SET (X, Y, Color)

La pantalla de baja resolución está controlada de tal forma que los pixels de distintos colores no pueden compartir el mismo espacio de carácter, de forma que en cualquier espacio de carácter tan sólo puede haber un color principal y un color de fondo y, de hecho, un pixel en la pantalla de baja resolución en realidad es tan sólo un cuarto de un carácter gráfico. Como demostración pruebe esto:

```
10 CLS
20 SET(10,10,2)
```

Primero se borra la pantalla y queda de color verde y a continuación verá que un espacio de carácter está ahora ocupado por un bloque negro con un cuadro amarillo en la parte superior izquierda. Si

mediante la orden SET se ilumina el punto situado a la derecha, a la izquierda, arriba o abajo con el mismo color, entonces la orden funciona tal como se espera, y el número de bloques amarillos aumenta.

```
30 SET(9,10,2)
40 SET(11,10,2)
50 SET(10,9,2)
60 SET(10,11,2)
```

Si se intenta iluminar el punto de la izquierda con un color distinto (3 = azul) esto funciona correctamente:

```
40 SET(10,9,3)
```

Pero si se intenta iluminar el punto de la derecha del primero con un color distinto, entonces no funciona.

```
50 SET(11,10,3)
```

En lugar de producir cuartos alternados de amarillo y azul, esto da como resultado que la mitad superior del carácter es totalmente azul. La razón de esto se explica más adelante, pero por el momento acepte el hecho de que no se pueden poner colores distintos en pixels adyacentes, a menos que la frontera entre ellos coincida con la frontera del carácter. En la práctica, la orden SET debe utilizarse siempre con un fondo negro, y debe evitar líneas adyacentes siempre que sea posible. Estas limitaciones significan que por lo general es mucho mejor utilizar la alta resolución para cualquier trabajo que implique un gráfico detallado, aunque la baja resolución siga teniendo como ventaja el disponer de nueve colores a la vez, y el acceso inmediato al texto.

RESET

RESET es la orden opuesta al SET y apaga un punto determinado. Se utiliza especificando únicamente las coordenadas X e Y, ya que el color de fondo en la baja resolución es siempre negro.

RESET (X,Y)

Obsérvese que no es posible realizar un RESET colocando el color Ø en la orden SET, ya que el sistema lo rechaza.

POINT

La orden POINT comprueba un pixel especificado y da como resultado el código del color.

POINT (X,Y)

Se utiliza principalmente en programas de comparación para comprobar el estado de un punto determinado antes de tomar una decisión:

```
100 IF POINT(X,Y)=6 THEN .......
```

Si el punto comprobado es negro, entonces devuelve como resultado el Ø y si no, si el punto está coloreado, entonces el resultado será uno de los números del 1 al 8. Sin embargo, si la orden POINT comprueba una posición que contiene un carácter alfanumérico, la respuesta será −1.

La orden SCREEN

Cuando se conecta el Dragon, automáticamente se selecciona la pantalla de texto con el texto en negro y el fondo en verde, y cuando termina un programa el sistema vuelve de nuevo a esta visualización. La definición de este estado es SCREEN Ø,Ø, donde el primer Ø indica baja resolución y el segundo Ø indica el primer conjunto de colores. Puede cambiarse el conjunto de colores en baja resolución, de forma que el texto aparezca en rojo o naranja mediante SCREEN Ø,1, pero este resultado continúa únicamente hasta que el contenido de la pantalla cambie o termine el programa.

Si quiere utilizarse esta característica hay que especificar SCREEN Ø,1, después del PRINT y retrasar la ejecución del programa.

```
10 PRINT"SCREEN 0,0"
20 Q$=INKEY$:IF Q$="" THEN 20
30 PRINT"SCREEN 0,1"
40 SCREEN 0,1
50 Q$=INKEY$:IF Q$="" THEN 50
60 GOTO 10
```

Ahora, cada vez que se pulse una tecla cambiará el conjunto de colores. Tan sólo se cambia el texto y los caracteres gráficos apare-

cen en la forma normal. La principal utilización de SCREEN Ø,1 es para resaltar algunos mensajes de la pantalla particularmente importantes.

Tabla 2.1

TABLA QUE MUESTRA LOS CODIGOS CHR$ Y POKE DE LOS PRIMEROS 128 CARACTERES

CARACTER	CODIGO CHR$	CODIGO POKE
@ INV	-	0
a	97	1
b	98	2
c	99	3
d	100	4
e	101	5
f	102	6
g	103	7
h	104	8
i	105	9
j	106	10
k	107	11
l	108	12
m	109	13
n	110	14
o	111	15
p	112	16
q	113	17
r	114	18
s	115	19
t	116	20
u	117	21
v	118	22
w	119	23
x	120	24
y	121	25
z	122	26
[INV	123	27
INV	124	28
] INV	125	29
INV	126	30
INV	127	31
ESPACIO INV	-	32
! INV	-	33
" INV	-	34

# INV	-	35
$ INV	-	36
% INV	-	37
& INV	-	38
' INV	-	39
(INV	-	40
) INV	-	41
* INV	-	42
+ INV	-	43
, INV	-	44
- INV	-	45
. INV	-	46
/ INV	-	47
0 INV	-	48
1 INV	-	49
2 INV	-	50
3 INV	-	51
4 INV	-	52
5 INV	-	53
6 INV	-	54
7 INV	-	55
8 INV	-	56
9 INV	-	57
: INV	-	58
; INV	-	59
< INV	-	60
= INV	-	61
> INV	-	62
? INV	-	63
@	64	64
A	65	65
B	66	66
C	67	67
D	68	68
E	69	69
F	70	70
G	71	71
H	72	72
I	73	73
J	74	74
K	75	75
L	76	76
M	77	77
N	78	78
O	79	79
P	80	80
Q	81	81

R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90
[	91
	92
]	93
	94
	95
(BARRA ESPACIO)	32
!	33
"	34
#	35
$	36
%	37
&	38
'	39
(	40
)	41
*	42
+	43
,	44
-	45
.	46
/	47
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57
:	58
;	59
<	60
=	61
>	62
?	63

	82
	83
	84
	85
	86
	87
	88
	89
	90
	91
	92
	93
	94
	95
	96
	97
	98
	99
	100
	101
	102
	103
	104
	105
	106
	107
	108
	109
	110
	111
	112
	113
	114
	115
	116
	117
	118
	119
	120
	121
	122
	123
	124
	125
	126
	127

3. Alta resolución

Selección de la pantalla en alta resolución

Antes de que podamos trabajar en alta resolución hay que poner el sistema a una determinada configuración de alta resolución y esto significa que hay que tomar una serie de decisiones.

Resolución

En primer lugar hay que seleccionar la resolución necesaria (la cantidad de detalles). Existen tres tamaños distintos del punto de la pantalla (fig. 3.1) y se corresponden con las siguientes distribuciones posibles.

128 horizontal por 96 vertical
128 horizontal por 192 vertical
256 horizontal por 192 vertical

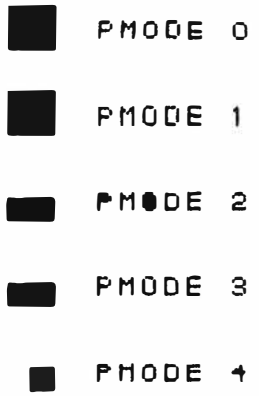


Fig. 3.1 Relación entre el tamaño de los puntos de pantalla.

Un punto de pantalla en la resolución más alta corresponde a un cuarto del tamaño de un punto de pantalla en la resolución más baja, y la mitad del tamaño del punto de pantalla de la resolución media. Obsérvese que en la media resolución cada punto es más un rectángulo alargado horizontalmente que un cuadrado.

Colores

La siguiente consideración es el número de colores que quieran utilizarse. Aunque en el Dragon se dispone de un total de nueve colores (mejor dicho, ocho colores más el negro), sólo pueden utilizarse estos colores en alta resolución de forma restringida. Nada más pueden utilizarse dos o cuatro colores, y la mezcla de color está también fijada. La elección del número de colores se hace seleccionando una de las órdenes PMODE del Ø al 4 (tabla 3.1). Existen tres posibilidades con dos colores y dos con cuatro colores.

Tabla 3.1

RESOLUCION Y COLORES

NUMERO PMODE	PUNTOS HORIZONTALES	PUNTOS VERTICALES	NUMERO DE COLORES
0	128	96	DOS
1	128	96	CUATRO
2	128	192	DOS
3	128	192	CUATRO
4	256	192	DOS

Requerimientos de memoria y selección del PMODE

La cantidad de memoria necesaria para soportar la visualización en alta resolución depende del PMODE seleccionado y, en realidad, de la cantidad de detalle (resolución) y del color suministrado. Cuando se conecta inicialmente el Dragon, el área que va desde las direcciones 1536 a la 6143 queda reservada automáticamente para los gráficos de alta resolución, pero quizá las necesidades sean menos que

TABLA 3.2

REQUERIMIENTOS DE MEMORIA EN LOS DISTINTOS PMODES

PMODE	MEMORIA(bytes)	MEMORIA(páginas)
0	1536	UNO
1	3072	DOS
2	3072	DOS
3	6143	CUATRO
4	6143	CUATRO

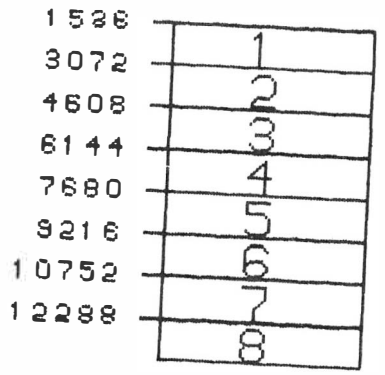


Fig. 3.2 Distribución de las páginas gráficas en memoria.

esto (tabla 3.2 y fig. 3.2). La memoria se utiliza en bloques de 1536 bytes, cada uno de los cuales se conoce como «página gráfica». Al conectar el Dragon quedan reservadas cuatro páginas, pero esto puede aumentarse hasta otras cuatro páginas. El número de páginas gráficas reservadas puede seleccionarse por el usuario mediante la orden PCLEAR. No hay ningún interés en reservar más memoria de la que se necesita (ya que entonces no podrá utilizarse para otros propósitos, tales como el programa o el almacenaje de variables), por lo tanto, si se está utilizando una relativa baja resolución, utilice

PCLEARn para seleccionar únicamente el número de páginas necesarias.

Por ejemplo, para PMODE Ø utilice PCLEAR 1

Por otra parte, para ciertas técnicas de programación serán necesarias más de cuatro páginas (véase más adelante), por lo que quizá sea necesario, mediante el PCLEAR, reservar un número de páginas superior a cuatro.

La orden PMODE tiene dos parámetros. El primero selecciona el PMODE y el segundo define en qué lugar de la memoria debe establecerse este PMODE. La posición más usual es la página 1, pero puede especificarse cualquiera de las páginas reservadas. Por ejemplo, PMODE Ø utiliza únicamente una página de memoria, pero esta página puede ser cualquiera de las ocho páginas disponibles. Naturalmente obtendrá un mensaje de error si intenta utilizar una página que no ha reservado.

PMODE Ø,1 = colocar PMODE Ø en la página 1
PMODE Ø,8 = colocar PMODE Ø en la página 8

Los PMODE más altos se extienden a más de una página, pero el segundo parámetro sigue definiendo la página inicial.

PMODE 2,1 = colocar PMODE 2 en las páginas 1 y 2
PMODE 2,2 = colocar PMODE 2 en las páginas 2 y 3

Pueden reservarse páginas distintas para propósitos distintos al mismo tiempo, y también pueden utilizarse con ellos PMODEs distintos. Por ejemplo, podríamos reservar cinco páginas y colocarlas de la siguiente forma: una página para PMODE Ø, dos páginas para PMODE 1 y dos páginas para PMODE 2 (fig. 3.3).

```
10 PCLEAR 5:PMODE 0,1:PCLS:PMODE
 1,2:PCLS:PMODE 2,4:PCLS
```

Obsérvese que se ha añadido la orden PCLS después de cada orden PMODE para asegurar que cada una de las páginas quede borrada. La orden PCLS funciona igual que la CLS en la pantalla de textos.

Conjunto de colores y SCREEN

Una vez que haya determinado cuántos colores necesita, deberá decidir qué colores va a utilizar. En cada PMODE hay dos conjuntos de colores alternativos, que se seleccionan mediante la orden SCREEN (tabla 3.3).

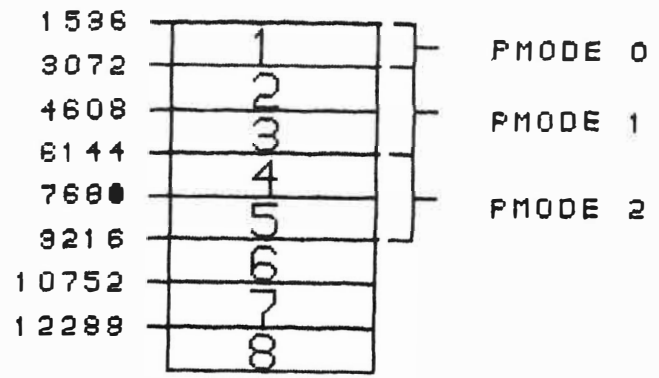


Fig. 3.3 Selección de distintos PMODE sobre distintas páginas.

Tabla 3.3

CONJUNTO DE COLORES

PMODE	CONJUNTO COLOR 0	CONJUNTO COLOR 1
0	negro/verde	negro/marrón
1	ver./amar./azul/rojo	marrón/cyan/mage./nar.
2	negro/verde	negro/marrón
3	ver./amar./azul/rojo	marrón/cyan/mage./nar.
4	negro/verde	negro/marrón

La orden SCREEN necesita dos parámetros, de los cuales el conjunto de colores es el segundo. El primer parámetro le dice al sistema si debe utilizar la memoria de texto o la de visualización en alta resolución. El número 1 selecciona la alta resolución. Por ejemplo, para seleccionar el conjunto de color 1 y la memoria de texto utilizaríamos:

```
30 SCREEN 0,1
```

Es importante recordar que, en realidad, la orden SCREEN es la que cambia la visualización de una parte de la memoria a otra. Si inserta la línea 15 GOTO 15 y la ejecuta, verá que parece que nada suceda. Pulse el BREAK, borre el 15 e inserte 1ØØ GOTO 1ØØ y ejecute de nuevo. La pantalla de texto normal quedará sustituida por la pantalla de alta resolución. Esta pantalla en alta resolución se estará visualizando hasta que se hayan ejecutado todas las órdenes de alta

resolución, y es cuando la orden SCREEN vuelve de forma maquinal a la memoria de visualización del texto (SCREEN Ø,Ø). Esto sucede automáticamente al final de un programa, de aquí la necesidad de la línea 1ØØ que forma un bucle sin fin.

La orden SCREEN siempre selecciona la visualización de acuerdo con el último PMODE especificado. Lo que en realidad estamos viendo ahora no es la página 1, sino las páginas 4 y 5 ya que el PMODE 2,4 era la última orden. Puede reconfirmar esto insertando otra orden PMODE en la línea 2Ø para colocar la última página de nuevo en dos, cuando la pantalla cambie a verde.

```
20 PMODE 1,2
```

Si cambia el conjunto de colores en la línea 3Ø a 1 y de nuevo ejecuta el programa, la pantalla aparecerá en marrón.

```
30 SCREEN 1,1
```

Color principal y de fondo

El primer color indicado en la tabla 3.3 es el color de fondo y el último es el color principal. Esto significa que la orden PCLS borrará la pantalla y la dejará del primer color, y cualquier orden gráfica en la que el color no esté especificado utilizará automáticamente el último color. Si se define un color determinado en una orden de borrado de pantalla mediante «PCLSn» la pantalla quedará de este color determinado. Pero es importante el darse cuenta que este efecto sólo es temporal y que no altera los colores principal y de fondo. Si se quieren cambiar los colores principales y de fondo debe utilizarse la orden COLOR que tiene el formato:

COLOR (principal,fondo)

Así, Color (1,2) producirá verde sobre amarillo y COLOR(4,3), rojo sobre azul. Esta selección seguirá siendo válida hasta que se altere deliberadamente.

Tratamiento de puntos individuales en alta resolución

PSET

La orden más sencilla en gráficos de alta resolución es PSET, que ilumina un único punto especificado en la pantalla. El formato es:

PSET(X,Y,C)

y para utilizarlo hay que establecer los tres parámetros X, Y y C.

Los dos primeros son las coordenadas X (horizontal) e Y (vertical), que definen la posición del punto en la pantalla que se quiere cambiar. Aunque el número de puntos individuales de la pantalla (resolución) varía con el PMODE que se ha especificado, las coordenadas para esta orden utilizan siempre una matriz de 256 por 192, de forma que el centro de la pantalla es siempre (128,96). El tamaño correspondiente del punto generado sobre la pantalla dependerá naturalmente del PMODE seleccionado. Ya que se utilizan las mismas coordenadas para todos los modos, no debe sorprenderse al descubrir que utilizando coordenadas ligeramente distintas en las resoluciones más bajas quizá produzcan el mismo resultado. Por ejemplo en PMODE Ø no existe ninguna diferencia entre estas cuatro órdenes, ya que todas actúan sobre el punto situado en la parte superior izquierda de la pantalla.

PSET(Ø,Ø) PSET(Ø,1)
PSET(1,Ø) PSET(1,1)

El último parámetro es el número correspondiente al color requerido (que debe ser un color «permitido», es decir, un color incluido en el conjunto seleccionado). Si no se especifica ningún color, entonces se utiliza el color principal actual. En la práctica, si se quiere utilizar el color actual, de hecho la ejecución es más rápida si no se incluye el código de éste en la orden PSET.

Pueden comprobarse estos efectos de una forma clara si se utilizan las siguientes rutinas de bucle para iluminar mediante el PSET todos los puntos de la pantalla secuencialmente, bajo condiciones distintas, y se utiliza el reloj interno para comprobar el tiempo empleado.

En primer lugar en el modo de resolución más alto (PMODE 4):

```
10 PMODE 4,1:SCREEN 1,0:PCLS
20 TIMER=0
30 FOR Y=0 TO 191
40 FOR X=0 TO 255
50 PSET (X,Y)
60 NEXT X
70 NEXT Y
80 PRINT"TIEMPO EMPLEADO=";TIMER
/50;"SEGUNDOS"
```

Esto empleará alrededor de 237 segundos. Si ahora se cambia el PMODE en la línea 1Ø a Ø y de nuevo se ejecuta el programa, se obsevará que éste parece detenerse al final de cada línea. De hecho no hay una espera y lo que se está viendo es que el sistema está iluminando los mismos puntos que ya estaban iluminados. Ya que no

hay ningún interés en iluminar el mismo punto dos veces, quizá sea mejor colocar un STEP 2 al final de las líneas 3Ø y 4Ø, y ver cuanto tiempo ahorra esto.

```
30 FOR Y=0 TO 191 STEP 2
40 FOR X=0 TO 255 STEP 2
```

Pues bien, el resultado es de alrededor de 60 segundos, lo que ya debiera esperarse por el hecho de que tan sólo hay que realizar una cuarta parte del trabajo anterior.

El trabajo extra relacionado con la inclusión de un parámetro de color puede verse cambiando la línea 5Ø de forma que especifique el color 1.

```
50 PSET (X,Y,1)
```

El tiempo necesario aumenta hasta 85 segundos, y el aumento debido al cambio al definir el color en realidad es incluso más alto de lo que parece (50%) ya que se emplean 19 segundos para completar estas series de bucles FOR-NEXT cuando están completamente vacíos. La moraleja queda clara: ¡no defina parámetros a menos que deba hacerlo!

Los parámetros pueden deducirse en la orden, utilizando las funciones normales, tal como se demuestra en este programa que de manera aleatoria ilumina puntos en la pantalla de resolución más baja en alta resolución.

```
10 PMODE 0,1:SCREEN 1,0:PCLS
20 PSET (RND(255),RND(191))
1000 GOTO 20
```

Esta rutina es más interesante si se selecciona un modo con cuatro colores y el color también se elige aleatoriamente.

```
10 PMODE 1,1:SCREEN 1,0:PCLS
20 PSET (RND(255),RND(191),RND(4
))
1000 GOTO 20
```

PRESET

Lo contrario de PSET es PRESET que ilumina el punto especificado con el color de fondo (es decir, lo apaga).

Si utilizamos el PSET y el PRESET rápidamente generaremos un punto parpadeante en la pantalla. Pruebe el siguiente programa:

```
10 PMODE 1,1:SCREEN 1,0:PCLS
20 X=RND(255):Y=RND(191)
30 PSET (X,Y,RND(4))
40 PRESET (X,Y)
1000 GOTO 20
```

Obsérvese que no hay que especificar un color con la orden PRESET ya que se utiliza siempre el color de fondo. De hecho se puede utilizar el PSET para hacer exactamente lo mismo que hace el PRESET si se utiliza como tercer parámetro el del color de fondo. A veces esto puede ser de utilidad en programación, sobre todo cuando el color del punto que debe utilizarse se calcula en el programa. Por ejemplo, la siguiente rutina también produce un punto parpadeante, pero esta vez utilizando la resolución más alta, PMODE4.

```
10 PMODE 4,1:SCREEN 1,0:PCLS
20 X=RND(256)-1:Y=RND(192)-1
30 FOR C=1 TO 0 STEP-1
40 PSET (X,Y,C)
50 FOR N=1 TO 300:NEXT
60 NEXT C:GOTO 20
```

Obsérvese que debemos utilizar un STEP negativo para que el color Ø (fondo) se utilice en último lugar.

PPOINT

La última orden de alta resolución que actúa sobre un único punto de la pantalla es PPOINT, que da como resultado el color de la posición especificada. Para ver cómo funciona vamos a iluminar un punto aleatoriamente y después comprobaremos qué punto se ha iluminado utilizando la orden PPOINT y dos bucles FOR NEXT para recorrer toda la pantalla.

```
10 PMODE 0,1:SCREEN 1,0:PCLS
20 X=RND(256)-1:Y=RND(192)-1
30 PSET (X,Y,1)
40 TIMER=0
50 FOR X=0 TO 255
60 FOR Y=0 TO 191
70 IF PPOINT (X,Y)<>1 THEN NEXT
Y,X
80 PRINT "PUNTO";X;Y;"ILUMINADO"
90 PRINT TIMER/50;"SEGUNDOS"
```

(Obsérvese que hay que utilizar RND(256)−1 para obtener números entre Ø y 255 ya que RND(255) tan sólo da números entre 1 y 255.)

Observará que esta rutina es muy lenta. Por ejemplo, necesita 65 segundos para encontrar un punto iluminado en la posición 5Ø,11Ø y 198 segundos para encontrar un punto iluminado en 134,34. Naturalmente podemos acelerar un poco las cosas si sólo comprobamos un punto de cada cuatro, añadiendo STEP 2 a los bucles (ya que estamos en PMODE Ø). Ahora encontrará un punto en 68,156 en 28 segundos, pero sigue siendo un proceso muy lento el examinar la totalidad de la pantalla.

Una de las principales utilizaciones de la orden PPOINT es la detección de colisiones en los programas de juego. El motivo de que sea una proposición realista en tales casos es que allí tan sólo se seleccionan un número limitado de posiciones que deben comprobarse con PPOINT. Por ejemplo, si se tiene una diana amarilla y una bala roja y se busca el amarillo en las coordenadas del misil, podrá detectarse el impacto. Quizás algunos programadores con más experiencia se hayan dado cuenta de que de hecho puede evitarse utilizar el PPOINT en este ejemplo sí se guardan las coordenadas del misil y de la diana y se comparan directamente. Sin embargo, esta alternativa no es posible cuando estas coordenadas no se guardan como variable.

Las órdenes PSET y PRESET son relativamente lentas y muchas veces pueden utilizarse en su lugar las órdenes LINE y CIRCLE que son mucho más rápidas y potentes. Sin embargo, PSET/PRESET son importantes en ciertas aplicaciones, por ejemplo al dibujar datos no lineales y para generar un cursor que indique la posición en la pantalla (véase más adelante).

Líneas y cajetines

La orden PSET puede utilizarse para dibujar una serie de puntos adyacentes que formen una línea horizontal a través de la pantalla:

```
10 PMODE 1,1:SCREEN 1,0:PCLS
20 Y=96
30 C=2
40 FOR X=0 TO 255
50 PSET (X,Y,C)
70 NEXT X
1000 GOTO 20
```

Si añadimos la siguiente rutina de PRESET, nuestra línea será borrada desde el principio.


```
80 FOR T=1 TO 1000:NEXT T
90 FOR X=0 TO 255
100 PRESET (X,Y)
110 NEXT X
```

Por otra parte podríamos borrarla empezando por el otro extremo, mediante un bucle FOR...NEXT que se fuese decrementando.

```
90 FOR X=255 TO 0 STEP-1
```

O convertirla en una línea de puntos, borrando sólo algunos de ellos mediante la orden PRESET. Sin embargo, hay que recordar que en el PMODE1 los puntos se seleccionan por pares y que por lo tanto necesitamos utilizar STEP4.

```
90 FOR X=255 TO 0 STEP-4
```

Si mediante la orden PSET se quiere dibujar una línea que no sea ni horizontal ni vertical, entonces serán necesarios algunos cálculos. Para dibujar una línea desde Ø,Ø hasta 1ØØ,1ØØ por ejemplo, no hay ningún problema, ya que es evidente que las coordenadas X e Y deben incrementarse en una unidad para cada punto. Sin embargo, si se quiere dibujar la línea entre otros puntos, sigue siendo fácil el calcular el tamaño adecuado del STEP dividiendo la distancia que debe moverse sobre el eje Y (YE-YS) por la distancia que debe moverse a lo largo del eje X (XEXS).

```
10 PMODE 4,1:SCREEN 1,0:PCLS
20 XS=0:XE=100
30 YS=0:YE=100
40 YI=(YE-YS)/(XE-XS)
50 FOR X=XS TO XE
60 PSET(X,Y)
70 Y=Y+YI
80 NEXT X
90 GOTO 90
```

Aunque el PSET puede utilizarse de esta forma para dibujar líneas rectas, de hecho es mucho más fácil utilizar la orden LINE. Esta es una orden muy versátil que tan sólo requiere que se le definan los puntos inicial y final de la línea mediante sus coordenadas X e Y, y que se especifique el color de fondo y el color principal mediante PSET o PRESET.

LINE (X1,Y1) – (X2,Y2), PSET

dibujará una línea desde X1,Y1 hasta X2,Y2 en el color principal y

LINE (X1,Y1) – (X2,Y2), PRESET

dibujará una línea entre las mismas coordenadas en el color de fondo, lo que de hecho quiere decir que la borrará.

Aunque la rutina del bucle anterior mediante el PSET funciona, es mucho más lenta. Otra ventaja importante de utilizar LINE es que el mejor camino para la línea elegida en función de los puntos disponibles en la pantalla, se calcula automáticamente sin ninguna acción por parte del usuario, por lo tanto la rutina se reduce a:

```
10 PMODE 4,1:SCREEN 1,0:PCLS
20 LINE(0,0)-(100,100),PSET
90 GOTO 90
```

Esto reduce el tiempo necesario en un factor de alrededor de 2Ø, de Ø,86 segundos a Ø,Ø4 segundos. En la práctica esto significa que las líneas aparecen instantáneamente.

La orden LINE suele utilizarse para conectar una serie de puntos para formar una gráfica, u otras figuras complejas. Los puntos pueden ser calculados por el programa o estar guardados en sentencias DATA. En este ejemplo los datos se leen y se guardan en tablas que son a continuación utilizadas por la orden LINE. De forma general la línea se dibuja desde el último punto (X(N-1), Y (N-1)) hasta el punto siguiente (X(N), Y(N)) aunque debe hacerse un arreglo especial para la primera línea ya que no hay último punto. Ya que Q se pone a Ø en la línea 6Ø y se pone a 1 en la línea 9Ø, la primera línea de hecho será de longitud Ø y dibujada en X(N),Y(N).

```
10 PMODE 4,1:SCREEN 1,0:PCLS
20 DIM X(6),Y(6)
30 FOR N=1 TO 6
40 READ X(N),Y(N)
50 NEXT N
60 Q=0
70 FOR N=1 TO 6
80 LINE(X(N-Q),Y(N-Q))-(X(N),Y(N
)),PSET
90 Q=1
100 NEXT N
110 DATA 1,1,10,10,255,14,190,60
,12,80,45,180
120 GOTO 120
```

Si se quiere dibujar una línea en un color distinto de los colores principal o de fondo actuales, entonces estos parámetros deberán redefinirse mediante la orden COLOR. En este ejemplo las coordenadas

iniciales y finales X,Y y el color principal (P) se eligen aleatoriamente, y a continuación se utiliza la orden COLOR para asegurar que la línea se dibuja con el color elegido.

```
10 PMODE 3,1:SCREEN 1,0:PCLS
20 FOR N=1 TO 20
30 X1=RND(256)-1:Y1=RND(192)-1
40 X2=RND(256)-1:Y2=RND(192)-1
50 C=RND(3)+1
60 COLOR C,1
70 LINE(X1,Y1)-(X2,Y2),PSET
100 NEXT N
110 GOTO 110
```

La orden LINE también puede utilizarse fácilmente para generar cajetines rectangulares añadiendo el sufijo B y especificando las coordenadas del extremo superior izquierdo y del extremo inferior derecho.

LINE (X1,Y1) – (X2,Y2), PSET, B

si únicamente añadimos este sufijo a la línea 7Ø del último programa se generarán cajetines de distintos colores en lugar de dibujar únicamente líneas.

```
70 LINE(X1,Y1)-(X2,Y2),PSET,B
```

Como es de esperar PRESET borrará el cajetín.

Hay dos formas de rellenar el área encerrada por el rectángulo. El método más sencillo es utilizar el sufijo BF que automáticamente rellenará el rectángulo con el color principal.

```
70 LINE(X1,Y1)-(X2,Y2),PSET,BF
```

Esto siempre rellenará el rectángulo con el mismo color que el que se utiliza para la línea que lo delimita. Si se quiere que el perímetro y el área encerrada difieran en color, puede dibujarse un rectángulo rellenado, ligeramente inferior que el anterior y después dibujar un rectángulo vacío alrededor de éste, con un color distinto. Obsérvese que deberán añadirse o sustraerse dos unidades en las coordenadas X ya que estamos en PMODE 3 y los puntos se seleccionan por parejas.

```
10 PMODE 3,1:SCREEN 1,0:PCLS
20 FOR N=1 TO 20
30 X1=RND(256)-1:Y1=RND(192)-1
40 X2=RND(256)-1:Y2=RND(192)-1
```

```
45 IF X1>X2 THEN X=X1:X1=X2:X2=X
46 IF Y1>Y2 THEN Y=Y1:Y1=Y2:Y2=Y
50 C=RND(2)+1
60 COLOR C,1
70 LINE(X1+2,Y1+1)-(X2-2,Y2-1),P
SET,BF
80 COLOR 4,1
90 LINE(X1,Y1)-(X2,Y2),PSET,B
100 NEXT N
110 GOTO 110
```

La orden PAINT

Un método alternativo para rellenar los rectángulos es utilizar la orden PAINT que rellenará cualquier área especificada con cualquier color permitido. Deben especificarse las coordenadas iniciales, seguidas del código del color que debe utilizarse para pintarlo, y el parámetro final es el color del límite, que le dice al sistema cuándo debe detenerse y dejar de pintar.

Por ejemplo:

```
PAINT (128,96),4,2
```

Significa empezar en el punto central de la pantalla (128,96) y colocar todos los puntos del color 4 (rojo) hasta que se llegue a puntos que son de color 2 (amarillo) o si no hay ningún lugar donde seguir pintando, entonces detenerse.

```
70 LINE(X1,Y1)-(X2,Y2),PSET,BF
80 PAINT(X1+2,Y1+1),4,2
```

Comparando la velocidad para generar un rectángulo coloreado en (5Ø, 5Ø) – (15Ø, 15Ø) mediante los dos métodos vemos que LINE...BF es más de dos veces más rápido que PAINT (Ø,8 segundos frente a 1,7), pero en contra de esto hay que tener en cuenta la mayor facilidad al especificar el color en PAINT y el hecho de que rellenará áreas irregulares tan fácilmente como rectángulos. Por ejemplo pruebe:

```
10 PMODE 3,1:SCREEN 1,0:PCLS
20 LINE(10,10)-(80,20),PSET
30 LINE(80,20)-(95,60),PSET
40 LINE(95,60)-(25,90),PSET
50 LINE(25,90)-(10,10),PSET
60 PAINT(12,12),2,4
```


Cualquier punto dentro del área elegida puede utilizarse como posición inicial, aunque un punto inicial cerca de una esquina consigue un efecto más logrado, ya que de otra forma el rellenado no siempre parece que proceda de una forma lógica.

Suelen aparecer un par de problemas cuando se utiliza la orden PAINT:

1) No sucede nada.

Recuerde que si el punto inicial ya está del color del límite, entonces la orden terminará tan pronto como empiece. Así esta orden no tendría ningún efecto:

```
60 PAINT(12,12),2,1
```

2) El pintado se desborda hacia lugares donde no se quería pintar.

a) Recuerde que esta pintura es muy corrosiva y que se desbordará fácilmente a través de cualquier agujero que haya en el límite por pequeño que sea. Haga esta pequeñísima modificación en la línea 5Ø y mire y observe las desastrosas consecuencias. Una vez que empieza a pintar no puede detenerse, incluso la tecla BREAK no tiene ningún efecto.

```
50 LINE(25,87)-(10,10),PSET
```

Este problema potencial a veces puede convertirse en una ventaja si se quiere rellenar una serie de áreas adyacentes, pero independientes, con el mismo color, si deliberadamente se puede formar algún tipo de enlace en lugares estratégicos.

```
10 PMODE 3,1:SCREEN 1,0:PCLS
20 LINE(10,10)-(100,40),PSET,B
40 LINE(30,40)-(80,80),PSET,B
60 PAINT(12,12),2,4
```

Este programa generará dos rectángulos adyacentes, pero tan sólo se pintará el rectángulo superior. Para rellenar los dos al mismo tiempo debemos generar un punto de enlace.

```
50 PSET(40,40,3)
```

b) Recuerde que tan sólo puede especificarse un color límite, por lo que la orden PAINT no puede utilizarse para rellenar un área que está limitada por colores distintos. Si se cambia el color mientras se están formando los dos rectángulos, las líneas serán de distinto color (el primer rectángulo rojo y el segundo azul).

```
30 COLOR 3,1
```

Si ahora intenta rellenar estos rectángulos, todo, excepto tres paredes del primer rectángulo, quedará de color amarillo.

4. Círculos

CIRCLE es una orden muy versátil que puede utilizarse para producir muchos tipos distintos de curvas. En su forma más sencilla necesita únicamente tres parámetros X, Y y R:

CIRCLE (X,Y),R

X e Y son las coordenadas en pantalla del centro del círculo, y R es el radio en puntos de pantalla (en una matriz de 256 por 192).

Mediante un bucle FOR-NEXT pueden producirse una serie de círculos concéntricos (fig. 4.1) de radio creciente.

```
20 PMODE 4,1:SCREEN 1,0:PCLS
40 FOR R=0 TO 90 STEP 5
50 CIRCLE (128,96),R
60 NEXT R
200 GOTO 200
```

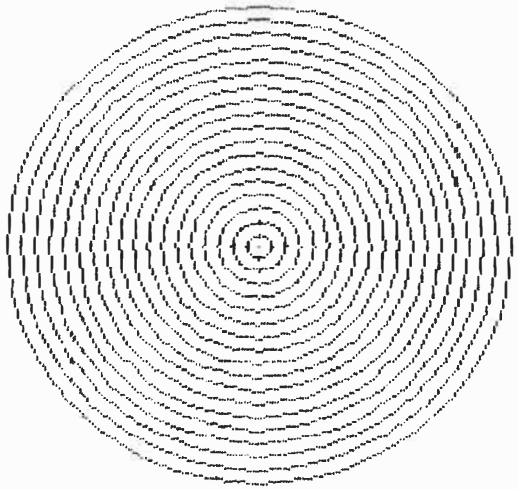


Fig. 4.1 Circulos concéntricos.

Si se dibujan círculos muy grandes, éstos saldrán de la pantalla y aparecerán achatados (fig. 4.2). Ya que el número de puntos en el eje Y es 192, el círculo más grande que puede dibujarse sin que quede distorsionado tendrá un radio de 96 puntos de pantalla.

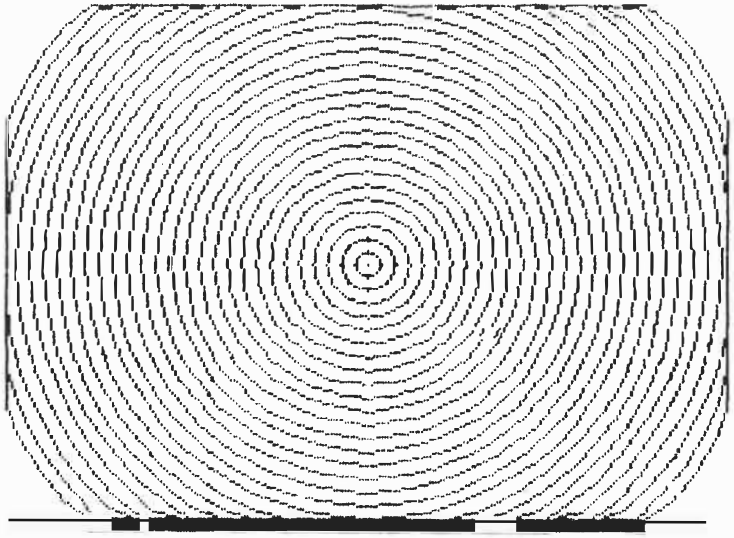


Fig. 4.2 Círculos que salen fuera del área de la pantalla.

Si el STEP en el bucle FOR-NEXT es negativo, los círculos irán disminuyendo de tamaño.

```
40 FOR R=0 TO 90
```

Quizás espere que con un STEP de 1 (por defecto) nos daría un círculo completamente relleno, pero en la práctica no sucede así (fig. 4.3) y algunas áreas quedan en blanco.

```
40 FOR R=90 TO 0 STEP -5
```

El motivo de estos espacios en blanco son las aproximaciones que deben hacerse en los cálculos matemáticos de la circunferencia del círculo, para que encaje en los puntos disponibles de la pantalla. Un examen más preciso revela que ninguno de los círculos es completamente redondo, si no que está hecho de una combinación de peque-

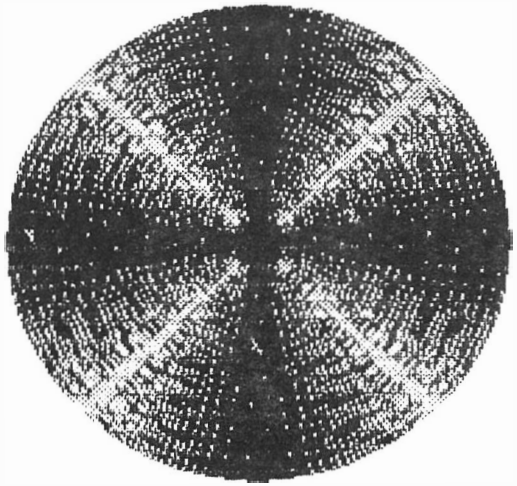


Fig. 4.3 Círculos «rellenos».

ñas líneas rectas. La precisión con que esté realizado el círculo dependerá de la resolución utilizada. Muchos de nuestros ejemplos utilizan los PMODE 3 y 4, pero vale la pena comparar la visualización en los distintos PMODE (fig. 4.4).

```
10 CLS:INPUT"PMODE";P
20 PMODE P,1:SCREEN 1,0:PCLS
40 FOR R=0 TO 90 STEP 10
50 CIRCLE (128,96),R
60 NEXT R
70 I$=INKEY$:IF I$="" THEN 70 EL
SE 10
```

Si no se especifica lo contrario, entonces el color principal actual será el utilizado para dibujar el círculo, pero también pueden definirse colores particulares mediante el cuarto parámetro C.

CIRCLE(X,Y),R,C

A veces, esto suele utilizarse para borrar selectivamente un círculo, dibujándolo de nuevo con el color de fondo.

```
20 PMODE 4,1:SCREEN 1,0:PCLS
30 FOR C=1 TO 0 STEP -1
40 FOR R=0 TO 90 STEP 10
50 CIRCLE (128,96),R,C
60 NEXT R
70 NEXT C
200 GOTO 200
```

Tal como está este programa, dibujará círculos verdes de radio creciente y después círculos negros de radio creciente (fig. 4.5) dejando finalmente la pantalla en blanco.

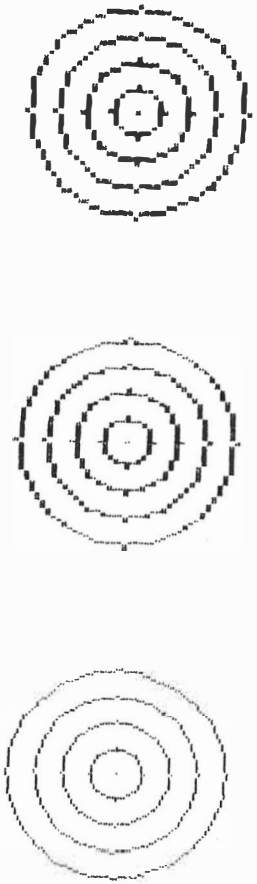


Fig. 4.4 Visualización con distintos PMODE (ø,2,4 empezando desde arriba).

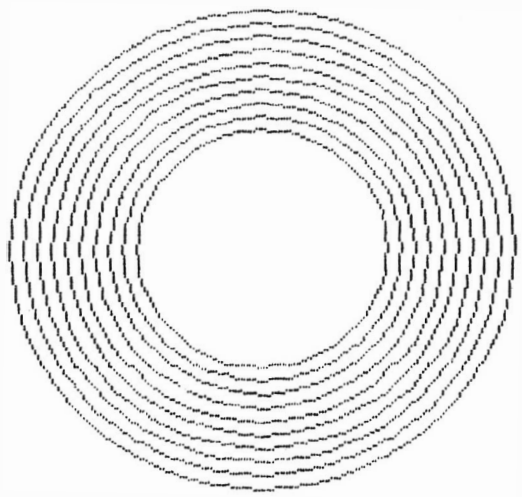


Fig. 4.5 Borrado de círculos.

Si la posición del bućle FOR C...NEXT C se mueve dentro del bucle del radio, se generarán círculos parpadeantes. Borre las líneas 3Ø y 7Ø y reescríbalas como la 45 y 55.

```
40 FOR R=0 TO 90 STEP 10
45 FOR C=1 TO 0 STEP -1
50 CIRCLE (128,96),R,C
55 NEXT C
60 NEXT R
```

También puede invertirse el STEP y borrar los círculos desde fuera hacia dentro.

```
30 C=1
40 FOR R=0 TO 90 STEP 10
50 CIRCLE (128,96),R,C
60 NEXT R
70 C=0
80 FOR R=90 TO 0 STEP -10
90 CIRCLE (128,96),R,C
100 NEXT R
```

El color puede elegirse aleatoriamente (fig. 4.6):

```
20 PMODE 3,1:SCREEN 1,0:PCLS
40 FOR R=0 TO 90 STEP 5
50 CIRCLE (128,96),R,RND(4)
60 NEXT R
200 GOTO 200
```

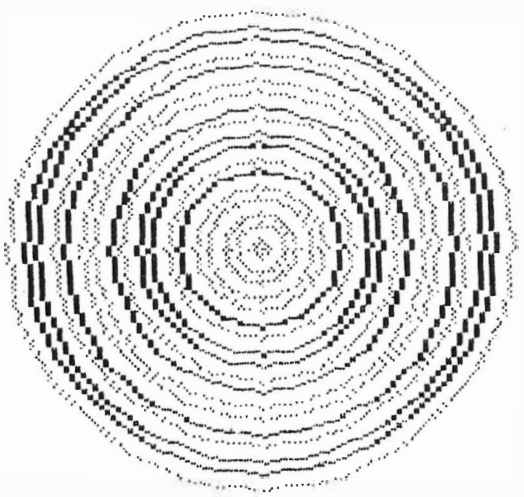



Fig. 4.6 Círculos coloreados.

o puede calcularse de alguna manera. Por ejemplo, podríamos relacionar el color con el radio del círculo (recordando siempre que el resultado debe ser un color válido).

```
40 FOR R=20 TO 80 STEP 5
50 CIRCLE (128,96),R,R/20
```

Elipses

Aunque esta orden se llame CIRCLE, también se puede utilizar fácilmente para dibujar elipses, cambiando el siguiente parámetro, HA, que es la relación altura/anchura.

CIRCLE(X,Y),R,C,HA

La relación altura/anchura es sencillamente la altura del círculo dividido por su anchura (fig. 4.7). Para un círculo real, el valor es 1. Si el dibujo es corto con relación a su anchura, HA será menor que 1, y si es más alto, HA será mayor que 1.

Cuando se cambie cualquiera de estos últimos parámetros, siempre hay que tener cuidado de incluir también los parámetros anteriores o reinará el caos. En realidad, no hay que colocar números, pero al

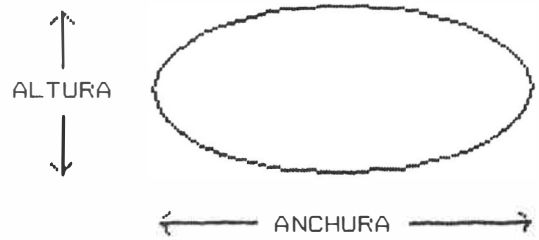


Fig. 4.7 Relación H/A de la elipse.

menos se deben incluir las comas que indican dónde empiezan y terminan los parámetros.

CIRCLE(128,96),90,1,0,5
o CIRCLE(128,96),90,,0,5

En todos estos ejemplos hemos escrito los números no enteros completos para mayor claridad, pero en la práctica pueden abreviarse ya que el Ø de antes de la coma no es necesario.

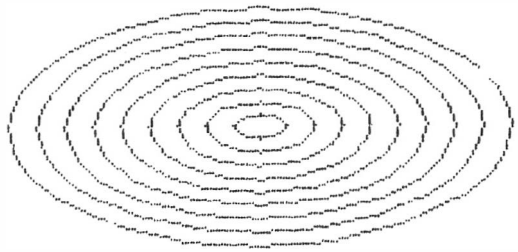


Fig. 4.8 Elipses concéntricas.

Una serie de elipses concéntricas con la misma HA pueden generarse tan fácilmente como los círculos (fig. 4.8).

```
20 PMODE 4,1:SCREEN 1,0:PCLS
40 FOR R=10 TO 90 STEP 10
50 CIRCLE (128,96),R,1,0.5
60 NEXT R
200 GOTO 200
```

Si el radio se mantiene constante, pero se varía la HA de Ø a 1, se formarán una serie de elipses aplastadas de igual diámetro (fig. 4.9).

```
40 FOR HW=0 TO 1 STEP 0.1
50 CIRCLE (128,96),90,1,HW
60 NEXT HW
```

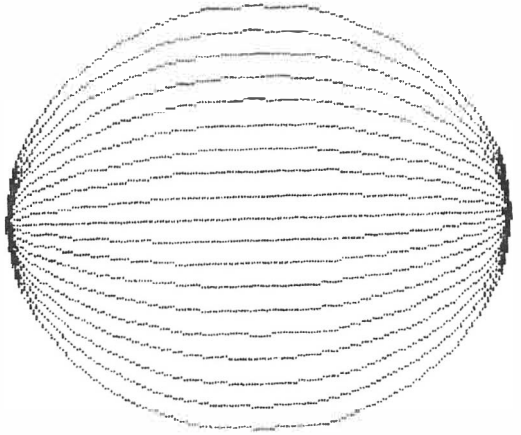



Fig. 4.9 Cambio de la relación H/A desde Ø hasta 1.

Si la HA se sigue incrementando aparecerá una distorsión vertical. La figura 4.10 muestra las siguientes 1Ø elipses con la HA de 1 a 2, en incrementos de Ø,1. Aunque HA puede tener cualquier valor hasta 255, los valores más grandes no se utilizan ya que únicamente dan líneas verticales.

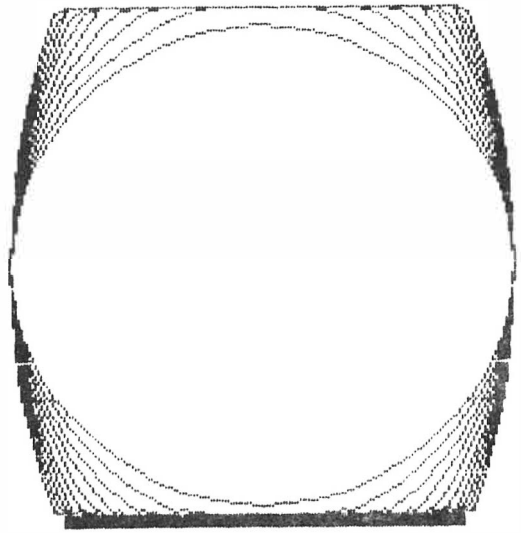


Fig. 4.10 Cambio de la relación H/A desde 1 hasta 2.

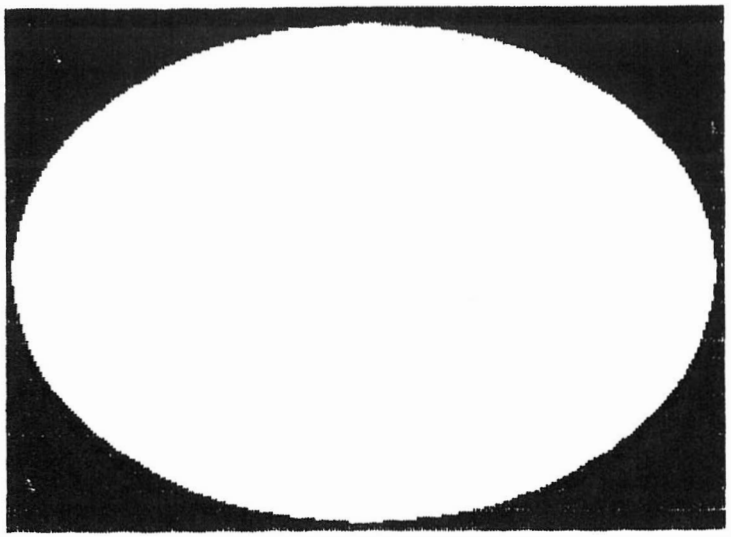


Fig. 4.11 Efecto «viñeta».

La relación HA para un televisor normal es 4/3 y la pantalla de visualización tiene 256 por 192 puntos, por lo que la mayor elipse que puede acomodarse tiene un radio de 128 y una relación de HA de Ø,75 (fig. 4.11). Este tipo de dibujo sobre las esquinas de la pantalla puede formar una bonita viñeta para enmarcar algún dibujo antiguo.

```
40 FOR R=125 TO 180 STEP 3
50 CIRCLE (128,96),R,1,0.75
60 NEXT R
```

Arcos

Hasta ahora siempre hemos dibujado círculos completos, pero también podemos dibujar partes limitadas (ARCos) de círculos (y elipses). Los dos últimos parámetros posibles definen el principio (P) y el final (F) del círculo.

CIRCLE(X,Y),R,C,HA,P,F

La orden CIRCLE siempre dibuja siguiendo la dirección de las agujas del reloj y a partir de la posición de las 3 en punto. La posición correspondiente a las 3 en punto se define como Ø y los otros puntos corresponden a valores crecientes entre Ø y 1.

Si empezamos en Ø,5 y terminamos en 1 se formará la parte superior de un círculo (fig. 4.12).

```
40 FOR R=10 TO 90 STEP 10
50 CIRCLE (128,96),R,1,1,0.5,1
60 NEXT R
```

Empezando en Ø,5 y terminando Ø,75 nos dará el cuadrante superior izquierdo (fig. 4.13) y de forma similar P = Ø,75 y F = 1 nos da el cuadrante superior derecho (fig. 4.14).

```
50 CIRCLE (128,96),R,1,1,0.5,0.7
5
```

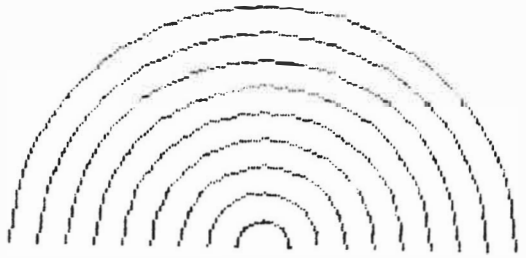


Fig. 4.12 Semicírculos.

Fig. 4.13 Cuadrante superior izquierdo.

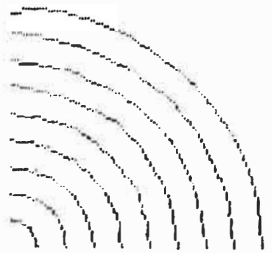


Fig. 4.14 Cuadrante superior derecho.

Diferencias más pequeñas entre P y F nos dibujarán segmentos más pequeños y que pueden empezar y terminar en cualquier punto (fig. 4.15). De la misma forma también se pueden dibujar trozos de elipse.

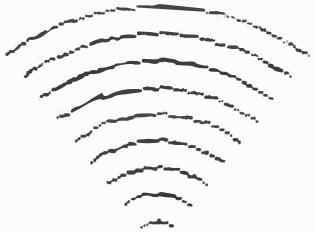


Fig. 4.15 Segmento.

Se puede dibujar un arco sin distorsión sobre la pantalla incluso en el caso en que la totalidad del círculo del que proviene este arco fuese tan grande que no cupiese en la pantalla, siempre que el centro de este círculo sea un punto válido de la pantalla (fig. 4.16).

```
40 FOR R=20 TO 240 STEP 20
50 CIRCLE (0,0),R,1,0.75,0,0.25
60 NEXT R
```

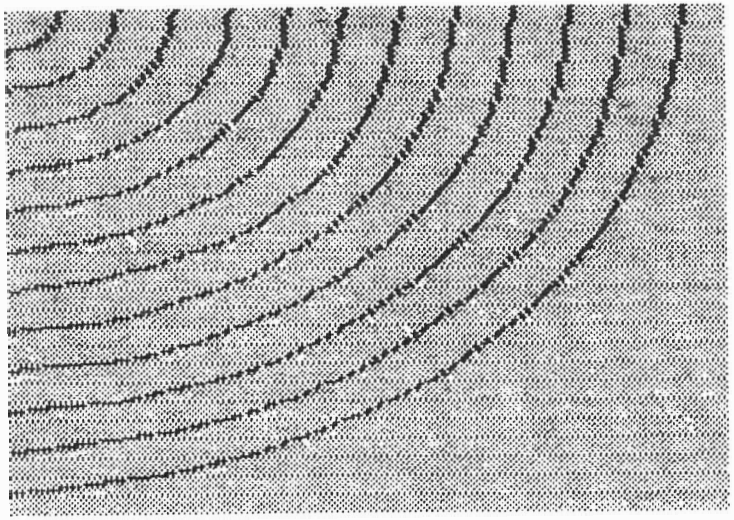


Fig. 4.16 Arcos de círculo cuyo centro está en el límite de la pantalla.

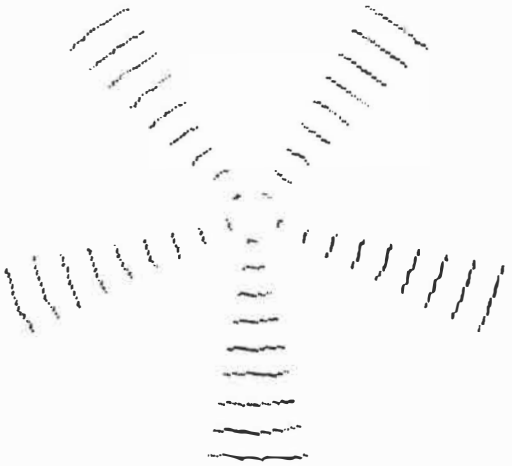


Fig. 4.17 Cinco segmentos.

Si se quiere dibujar más de un segmento de un círculo, puede cambiarse el punto inicial P mediante un bucle FOR-NEXT y expresar el final (F) como P más la anchura deseada del segmento (fig. 4.17).

```
45 FOR S=0 TO 1 STEP .2
50 CIRCLE (128,96),R,1,1,S,S+0.0
5
55 NEXT S
```

El primer segmento será dibujado desde Ø a Ø,Ø5, el segundo desde Ø,2 a Ø,25, el tercero desde Ø,4 a Ø,45, el cuarto desde Ø,6 a Ø,65, el quinto desde Ø.8 a Ø,85.

En los modos de cuatro colores, una técnica similar que incorpora el número del color en los cálculos, puede utilizarse para generar segmentos de distintos colores (fig. 4.18). El código del color deberá dividirse para obtener un valor adecuado.

```
20 PMODE 3,1:SCREEN 1,0:PCLS1
40 FOR R=10 TO 90
45 FOR S=0 TO 1 STEP .2
46 FOR C=2 TO 4 STEP 2
50 CIRCLE (128,96),R,C,1,S+C/4,S
+C/4+0.1
54 NEXT C
55 NEXT S
60 NEXT R
200 GOTO 200
```

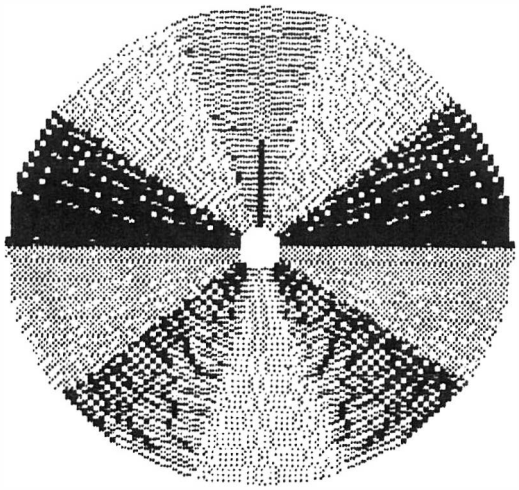


Fig. 4.18 Segmentos coloreados.

En lo que se refiere a principio y final de los arcos, el bucle más interior da valores de 2/4=Ø,5 o 4/4=1 y el bucle medio se incrementa en Ø,2. El bucle más interno dibuja pares de arcos. El primer arco se dibuja en el color 2 desde Ø,5 a Ø,6 y el segundo en color 4 desde 1 a 1.1, etc. (tabla 4.1).

Tabla 4.1

COLOREADO DE SEGMENTOS DE CIRCULOS

Segmento	S	C/4	S+C/4	S+C/4+0.1
1 amar.	0	0.5	0.5	0.6
2 rojo	0	1.0	1.0	1.1
3 amar.	0.2	0.5	0.7	0.8
4 rojo	0.2	1.0	1.2	1.3
5 amar.	0.4	0.5	0.9	1.0
6 rojo	0.4	1.0	1.4	1.5
7 amar.	0.6	0.5	1.1	1.2
8 rojo	0.6	1.0	1.6	1.7 etc.

Obsérvese que los segmentos amarillos empiezan todos a partir de los puntos impares y los segmentos rojos en los pares y que si el valor es mayor que 1 entonces el 1 no tiene ningún efecto.

Espirales

También pueden construirse espirales a partir de pequeños arcos, si se incluyen los incrementos de diámetro adecuado (fig. 4.19).

```
20 PMODE 4,1:SCREEN 1,0:PCLS
40 FOR R=10 TO 90
45 FOR S=0 TO 1 STEP 0.05
50 CIRCLE (128,96),R,1,1,S,S+0.0
5
54 R=R+1
55 NEXT S
60 NEXT R
200 GOTO 200
```

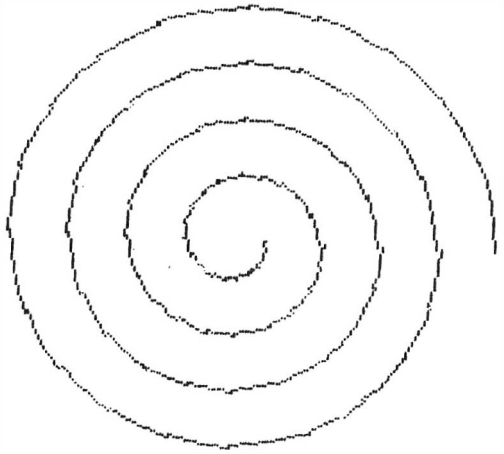


Fig. 4.19 Espiral.

Un par de cambios menores en los parámetros pueden tener importantes efectos como resultado (fig. 4.20).

```
50 CIRCLE(128,96),R,1,.4,S,S+.4
```

Fig. 4.20 Espiral elíptica.

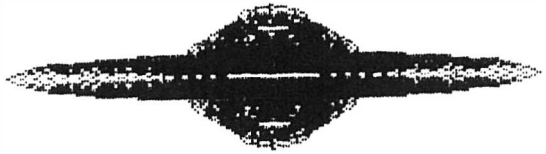


Fig. 4.21 Platillo volante

Finalmente, una demostración de que manejando de forma adecuada programas tan sencillos pueden obtenerse efectos sorprendentes. Observe en la figura 4.21 y vea lo sencillo que es construir un platillo volante a partir de una sencilla elipse.

```
10 ST=.1:FI=1.5
20 PMODE 4,1:SCREEN 1,0:PCLS
30 FOR R=20 TO 90 STEP 5
40 FOR HW=ST TO FI STEP .1
50 CIRCLE (128,96),R,1,HW
60 NEXT HW
70 FI=FI-.3
80 NEXT R
```


5. La orden DRAW

La orden DRAW es una de las características gráficas más versátiles disponibles en el BASIC, y tanto la sintaxis como las aplicaciones son muchas y variadas. La orden DRAW (al igual que PLAY) siempre actúa sobre una cadena o texto y quizás al principio le confunda un poco ya que no hay ni más ni menos que 15 órdenes distintas dentro del DRAW (tabla 5.1). Sin embargo, estas subórdenes pueden dividirse en tres grupos principales, en función de si causan movimiento, un cambio en el modo o tienen otras acciones.

Para poder ver en acción todas las subórdenes de DRAW, en primer lugar debemos seleccionar la visualización en alta resolución (entre de momento la línea 2Ø sin preocuparse y olvídela hasta más adelante).

```
10 PMODE 4,1:SCREEN 1,0:PCLS
20 DRAW"S48"
1000 GOTO 1000
```

Aunque no puede verse nada en la pantalla, se ha posicionado un cursor invisible en el centro de la misma (coordenadas 128,96) y esto será aparente cuando se añada una orden DRAW.

```
30 DRAW"U"
```

Aparecerá ahora una pequeña línea que apunta hacia arriba en el centro de la pantalla, y si se modifica el texto poniendo «URDL» (arriba, derecha, abajo, izquierda) se formará un pequeño cuadrado (fig. 5.1).

```
30 DRAW"URDL"
```

Fig. 5.1 Cuadrado.

Tabla 5.1

ORDENES DE DIBUJO

MOVIMIENTO

vertical

U	arriba (0 grados)
D	abajo (180 grados)

horizontal

L	izquierda (270 grados)
R	derecha (360 grados)

diagonal

E	a 45 grados
F	a 135 grados
G	a 225 grados
H	a 315 grados

absoluto

M	dibuja una línea hasta las coordenadas especificadas
BM	movimiento en blanco(se mueve hasta las nuevas coordenadas sin dibujar)

MODO

A	cambia el ángulo
C	cambia el color
S	cambia la escala

OTROS

N	no actualiza las coordenadas del último dibujo
X	ejecuta una sub-orden

Obsérvese que cada nueva línea se dibuja a partir del punto donde terminó la última, por lo que el cuadrado está desplazado hacia la parte superior derecha de la pantalla. Los puntos y comas entre estas órdenes son opcionales, y por lo general se suprimen para ahorrar espacio en el texto. Si el número aparece a continuación de una de estas letras, definirá cuántas veces deberá repetirse esta orden particular. Así U4 dibujará una línea el doble de larga que U2 y cuatro veces más larga que U. Si doblamos las órdenes U y D en el texto, entonces generaremos un rectángulo alargado verticalmente en lugar del cuadrado (fig. 5.2).

```
30 DRAW"U2RD2L"
```

Fig. 5.2 Rectángulo.

E, F, G y H funcionan exactamente de la misma forma, pero dan las cuatro posibles posiciones diagonales a 45 grados. Puede construirse cualquier combinación de estas órdenes para formar cualquier tipo de dibujo. Por ejemplo, este texto dibuja una letra A (fig. 5.3).

```
30 DRAW"U5ER2FD5U3L4"
```

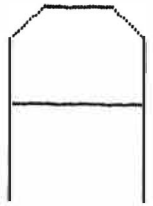


Fig. 5.3 Dibujo de un carácter.

No es necesario que el texto se defina inmediatamente después de la palabra DRAW ya que esta orden puede actuar también sobre textos ya existentes.

```
30 A$="U5ER2FD5U3L4"
40 DRAW A$
```

Escala

Ahora vamos a revelar el secreto de la línea 2Ø que utiliza la característica «escala», de gran utilidad, para multiplicar efectivamente la totalidad del texto por un factor. Si no se especifica la escala, entonces el factor es 1 y la U, por ejemplo, dibujará una línea con una longitud de 1 punto. Por otra parte, ya que seleccionamos una escala de 48 en la línea 2Ø, la U en realidad dibuja una línea de 48 puntos de largo.

Por regla general, los puntos se colocan de distintas formas de acuerdo con los modos de resolución seleccionados (PMODE). Hemos seleccionado PMODE 4, pero también debería observarse el efecto de utilizar modos de resolución más bajos con dos y cuatro colores, y recordar que el sistema quizá no distinga dos coordenadas como puntos distintos en los modos de resolución más bajos.

Fig. 5.4 Cuadrados escalonados.

La escala puede modificarse dentro de un programa, siempre que se tenga en cuenta que la orden DRAW actúa únicamente sobre textos y no sobre variables sencillas. Para incluir una variable sencilla, primero deberá convertirse a una variable de texto mediante la función STR$. Para demostrar esto ejecute esta rutina, la cual generará una serie de rectángulos que irán aumentando en tamaño (fig. 5.4).

```
20 FOR S=4 TO 56 STEP 8
30 DRAW"S"+STR$(S)
40 DRAW"URDL"
50 NEXT S
```

El valor máximo de S es 62, por lo que para conseguir aumentos mayores de tamaño deberán añadirse números después de las órdenes de movimiento (recuerde que quedarán multiplicados por S para producir un efecto acumulativo). Todos los anteriores cuadrados empezarán desde el mismo punto, ya que también terminaban en el mismo punto, pero si se sustituye la línea 4Ø por el texto correspondiente al dibujo de la letra A, aparecerá el desastre. El primer problema es que el dibujo no termina en la posición inicial, mientras que el segundo problema es que las letras alcanzan la parte superior de la pantalla y quedan distorsionadas (fig. 5.5). Estas dificultades pueden resolverse añadiendo «D3» al final del texto, de forma que el dibujo termina ahora en el punto inicial, lo que además deja espacio suficiente para las letras más grandes.

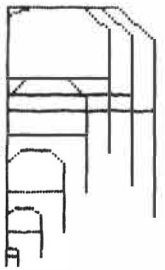


Fig. 5.5 Distorsión causada por el movimiento hacia arriba que se produce cada vez que se dibuja la letra.

Los textos que se utilizan para la orden DRAW pueden juntarse al igual que cualquier otro texto, por lo que las líneas 3Ø y 4Ø podrían combinarse en una sola:

```
30 DRAW"S"+STR$(S)+"U5ER2FD5U3L4
D3"
```

Color

Hasta ahora hemos utilizado únicamente un modo con dos colores, pero ahora vamos a cambiar a PMODE 3 para ver el funcionamiento de la orden de color C, que funciona de forma similar a la S.

```
10 PMODE 3,1:SCREEN 1,0:PCLS
20 FOR C=1 TO 4
30 DRAW"C"+STR$(C)+"S24U5ER2FD5U
3L4D3"
40 A$=INKEY$:IF A$="" THEN 40
50 NEXT C
60 GOTO 20
```

Ahora, cada vez que se pulse una tecla, la letra será redibujada con un color distinto. Ya que el color 1 es el color de fondo, esto significa que a veces se borrará la letra. El color por defecto es el color principal actual, y una vez que se especifique un color de esta manera, será utilizado hasta que se cambie de nuevo.

Ángulo

La orden final dentro del grupo de modo es Angulo, que permite cambiar la dirección de movimiento de cualquier orden que vaya a

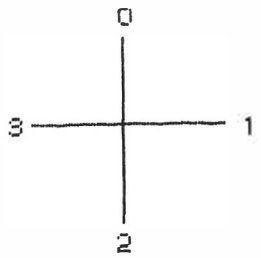


Fig. 5.6 Números de ángulo.

continuación, en incrementos de 9Ø grados, produciendo el giro del dibujo. Los incrementos están definidos por los números del Ø al 3, donde Ø es vertical, 1 es 9Ø grados, 2 es 18Ø grados y 3 es 27Ø grados (fig. 5.6).

```
20 FOR A=0 TO 3
30 DRAW"A"+STR$(A)+"S24U5ER2FD5U
3L4D3"
50 NEXT A
```

Ahora se dibujará la letra en las cuatro posibles direcciones (fig. 5.7). Como antes, hay que tener en cuenta que el ángulo definido será utilizado por cualquier orden DRAW que vaya a continuación, hasta que se cambie.

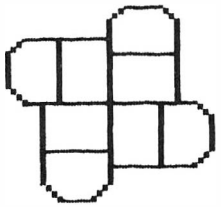


Fig. 5.7 Cambio del ángulo.

No-actualizar

Normalmente cada nueva orden DRAW empieza a partir del último punto dibujado, pero a veces tiene utilidad el poder dibujar un número de líneas a partir del mismo punto, por lo que existe la orden de No-actualizar, N. Si la letra N se coloca antes de cualquier otra orden, entonces la posición del cursor no cambiará durante este movimiento.

Fig. 5.8 Estrella producida por un movimiento sin actualización.

Esta orden se aplica a cada una de las órdenes de esta rutina para producir una estrella que radia desde el punto central (fig. 5.8).

```
20 DRAW"S48NUNENRNFNDNGNLNH"
30 GOTO 30
```

También es de utilidad para producir estructuras ramificadas cuando se utiliza de forma selectiva (fig. 5.9).

```
20 DRAW"R20ND15R15ND10R10ND5"
```

Fig. 5.9 Utilización selectiva del movimiento sin actualización.

Movimiento y movimiento en blanco

Además de las órdenes de movimiento que hemos visto hasta ahora, también tenemos la orden M (movimiento) y BM (movimiento en blanco) que son ligeramente distintas en el hecho de que especifican las nuevas coordenadas en lugar de únicamente la dirección y la distancia. La única diferencia entre M y BM es que M dibuja una línea hasta las nuevas coordenadas, pero BM tan sólo mueve el cursor sin dibujar nada. Hasta ahora nos hemos contentado con empezar todos nuestros dibujos a partir del punto por defecto, que es la posición central de la pantalla, pero esto puede modificarse fácilmente añadiendo BM x,y al principio de nuestro texto. Las coordenadas x e y pueden definirse tanto en modo absoluto como en modo relativo. Si sólo se añaden números se interpretan como absolutos. Así, BM 2Ø,2Ø moverá el dibujo a la parte superior izquierda de la pantalla (fig. 5-10).

```
20 DRAW"BM20,20R20ND15R15ND10R10
ND5"
```

Para indicar un movimiento relativo hay que colocar un + o un − antes del número, cuando el cálculo sea relativo a la posición actual.

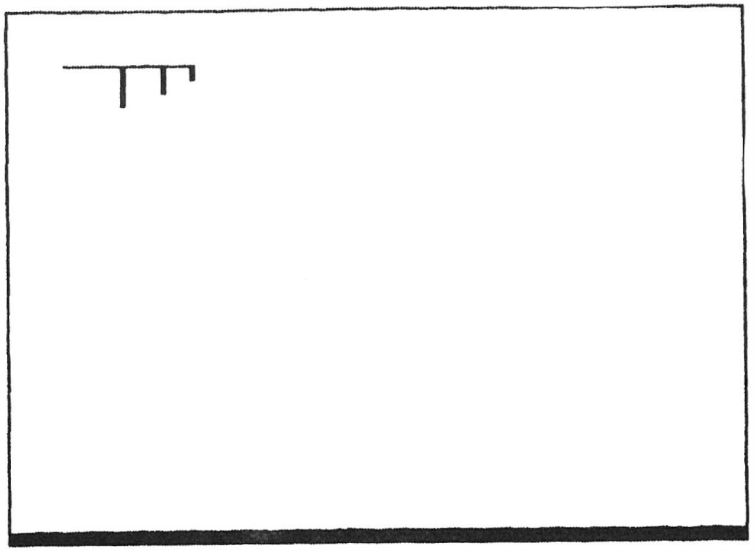


Fig. 5.10 Movimiento en blanco hasta 2Ø,2Ø.

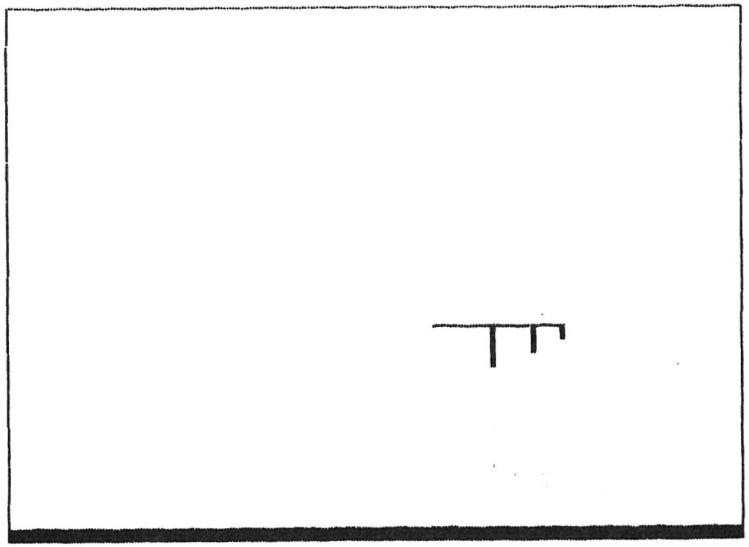


Fig. 5.11 Movimiento en blanco relativo hasta 2Ø, 2Ø.

Ahora, BM + 20, +20 moverá el dibujo hacia la parte inferior derecha de la pantalla, a partir del centro (fig. 5.11). La regla que hay que recordar es que arriba e izquierda son siempre negativos.

```
20 DRAW"BM+20,+20R20ND15R15ND10R
10ND5"
```

La orden de movimiento M se utiliza sobre todo para dibujar líneas relativamente largas hasta puntos predefinidos (de forma parecida a LINE). Se pueden poner variables en las órdenes de M y BM, aunque es un poco confuso, ya que cada variable debe convertirse independientemente a un texto y éstos deben estar separados por comas. En el siguiente programa las coordenadas X e Y son variables para formar una serie de líneas de distinta longitud (fig. 5.12).

```
20 Y=50
30 FOR X=0 TO 250 STEP 10
40 DRAW"NM"+STR$(X)+","+STR$(Y)
50 Y=Y+2
60 NEXT X
70 GOTO 70
```

Fig. 5.12

Ejecutar subtexto

La última orden es la X que llama a un subtexto que ya ha sido definido con anterioridad. La sintaxis es:

XA$;

y es importante señalar que en este caso el punto y coma no es opcional (¡incluso aunque esté al final de una línea!). El principal interés de esta orden está en programas complejos donde determinadas secuencias de la orden DRAW tienen que utilizarse con frecuencia. Como sencilla demostración vamos a formar una pequeña parte de una escalera en A$ y después construiremos trozos más largos, añadiendo textos (pero recuerde que la longitud final del texto no puede

ser mayor que 255). Cada texto se ejecuta a continuación para obtener escaleras de distinta longitud, y ejecutándose B$ dos veces para obtener la escalera más larga (fig. 5.13).

```
10 CLEAR 1000:PMODE 4,1:SCREEN 1
,0:PCLS
20 A$="U10D5R20D5U10BM-20,-0"
30 B$=A$+A$
40 C$=B$+B$
50 D$=C$+C$
60 DRAW"BM10,180XA$;"
70 DRAW"BM60,180XB$;"
80 DRAW"BM110,180XC$;"
90 DRAW"BM160,180XD$;"
100 DRAW"BM210,180XD$;XD$;"
110 GOTO 110
```

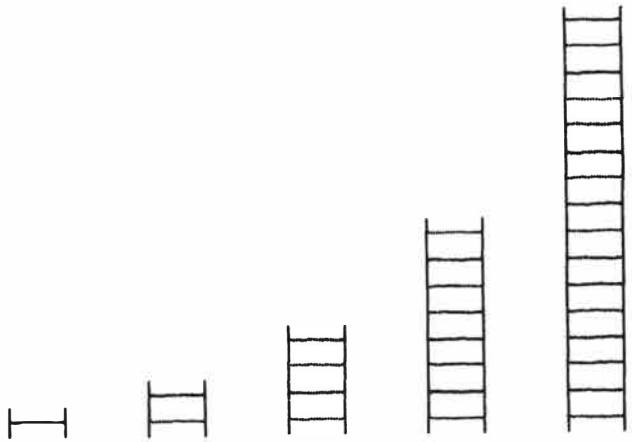


Fig. 5.13 Escaleras.

6. Combinación de las órdenes gráficas

Un dibujo determinado puede construirse mediante la combinación de cualquiera o la totalidad de las órdenes de dibujo gráfico descritas hasta ahora. Para dar una demostración final de cómo hacerlo hemos pedido ayuda a nuestro amigo el PIC-MAN (fig. 6.1). Quizá debiéramos explicar que es distinto de su pariente PI-MAN y que en realidad no es ni un autómata ni tiene ninguna aspiración política (de aquí que aparezca en blanco y negro), y que a diferencia del PAC-MAN no tiene miedo a los fantasmas ni tiene un insaciable apetito de píldoras de poder. En lugar de eso y deliberadamente, se ha construido a partir de una gran variedad de órdenes gráficas para demostrarle cómo pueden combinarse la mayoría de las órdenes de dibujo en alta resolución en un único programa.

Fig. 6.1 PIC-MAN.

Empezaremos seleccionando el PMODE 4 para que podamos tener la más alta resolución posible y, por lo tanto, que puedan añadirse gran cantidad de detalles. SCREEN 1,Ø nos da blanco sobre fondo negro.

```
10 PMODE 4,1:SCREEN1,0:PCLS
```

En su forma más sencilla, la orden CIRCLE necesita únicamente dos parámetros: las coordenadas X e Y del centro y el diámetro del círculo, con lo que habremos conseguido un bonito par de pequeños

ojos redondos (fig. 6.2). Recuerde que las coordenadas se especifican siempre respecto a una matriz de 256 × 192, con independencia del PMODE que se esté utilizando. Cuando se esté planeando un dibujo, puede utilizarse papel gráfico u hojas de dibujo más sofisticadas, pero por lo general un planteo a base de intentar y corregir sobre la pantalla suele ser más rápido cuando hay mucho detalle que colocar. No hay necesidad de especificar nada más, ya que los valores por defecto nos darán un círculo completo en el color principal.

```
40 CIRCLE(79,48),2
50 CIRCLE(84,48),2
```

Fig. 6.2 Ojos.

En realidad las cabezas no son redondas, sino en forma de huevo (especialmente si es usted un micromaníaco), por lo tanto necesitaremos formar una elipse distorsionada en sentido vertical mediante CIRCLE. Será la relación altura/anchura (HA) lo que nos permitirá incluir esta distorsión, pero hay que tener en cuenta que éste deberá ser el CUARTO parámetro. Es muy fácil olvidar que el sistema tan sólo podrá advertir que se trata del cuarto parámetro si puede ver otros tres parámetros antes que él, y que por lo tanto deberá incluirse también el tercer parámetro (color). Aunque de hecho hemos colocado el número 1 para seleccionar el color blanco, el ordenador también reconocerá una coma como el valor por defecto, por lo tanto cualquiera de las siguientes líneas tiene el mismo efecto. En este programa hemos incluido deliberadamente todos los valores actuales para que sea más fácil de leer. La relación HA es mayor que 1 para que la distorsión sea vertical en lugar de horizontal (fig. 6.3).

```
30 CIRCLE(82,50),8,1,1.5
```

o

```
30 CIRCLE(82,50),8,,1.5
```

Fig. 6.3 Cabeza.

Otra característica de CIRCLE es la posibilidad para formar sólo ciertos arcos del círculo completo, utilizando los parámetros cinco y seis para fijar el principio y el final. El PIC-MAN está sonriendo, por lo

que su boca está formada por la mitad inferior de un círculo que se dibuja desde Ø (3 en punto) hasta Ø.5 (9 en punto) (fig. 6.4).

```
70 CIRCLE(82,55),3,1,1,0,.5
```

Fig. 6.4 Boca.

La expresión más sencilla de LINE va únicamente de un punto a otro, como en el caso de la nariz y PSET en lugar de PRESET significa que se utiliza el color blanco (el color principal) (fig. 6.5).

```
60 LINE(82,52)-(82,54),PSET
```

Fig. 6.5 Nariz.

Aunque sus orejas puedan parecer redondeadas, son demasiado pequeñas para formarlas con la orden CIRCLE, por lo que son sencillos cuadrados formados especificando el extremo superior izquierdo y el inferior derecho y añadiendo una B al final de la orden LINE. El cuello se hace de la misma forma (fig. 6.6).

```
80 LINE(73,46)-(74,49),PSET,B
90 LINE(90,46)-(91,49),PSET,B
100 LINE(80,61)-(84,63),PSET,B
```

Fig. 6.6 Orejas y cuello.

Ahora que ya tenemos un cuello, podemos añadir los hombros mediante una combinación de las ideas anteriores de CIRCLE para obtener la mitad superior de una elipse distorsionada horizontalmente (fig. 6.7).

```
110 CIRCLE(82,72),18,1,.5,.5,1
```

Fig. 6.7 Hombros.

Podríamos haber continuado utilizando la orden LINE para dibujar el resto del cuerpo, pero DRAW es más versátil ya que pueden dibujarse al mismo tiempo una serie de líneas en direcciones distintas. Lo primero que haremos será la parte superior del cuerpo (fig. 6.8).

```
120 DRAW"BM64,72D28R6U24R2D20R20
U20R2D24R6U28"
```

Fig. 6.8 Brazos y tronco.

y después la parte inferior (fig. 6.9)

```
130 DRAW"BM72,97D30R8U20R4D20R8U
30"
```

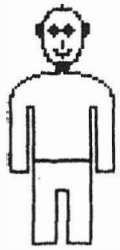


Fig. 6.9 Piernas.

Observe la utilización de movimientos en blanco (BM) para establecer la posición inicial y asegúrese de que sigue las instrucciones completas para ver cómo lo va haciendo. Es mejor planear cuidadosamente el camino a seguir, de forma que sea lo más compacto posible. Debe recordarse siempre que la siguiente orden DRAW suele empezar a partir del último punto dibujado, incluso aunque esto se haya hecho hace más de una hora (siempre que no se utilice el RUN). Por lo tanto, si el dibujo empieza a perderse en su programa, vuelva atrás y compruebe qué es lo que se dibujó en último lugar.

La orden DRAW puede utilizarse para hacer cualquier tipo de dibujo y en otro aspecto donde puede ser de mucha utilidad es en el

colocar texto en la pantalla de alta resolución. Las letras que forman el título PIC-MAN, están dibujadas de esta forma (para más detalle sobre esta técnica véase más adelante) (fig. 6.10).

```
20 DRAW"BM150,100S8U6R3FDGL3BM+8
,+3R2LU6LR2BM+5,+6HU4ER2FHL2GD4F
R2EBM+2,-2R4BM+2,+3U6F2E2D6BM+4,
+0U5ER2FD5U3L4BM+8,+3U6DF4DU6S4"
```

Fig. 6.10 Título.

Es obvio que al PIC-MAN le faltan las botas, las cuales se construirán por las mitades superiores de dos círculos y con gruesas suelas formadas por rectángulos rellenados del color principal (fig. 6.11).

```
140 CIRCLE(76,132),5,1,1,.5,1
150 CIRCLE(88,132),5,1,1,.5,1
160 LINE(71,132)-(81,134),PSET,B
F
170 LINE(83,132)-(93,134),PSET,B
F
```

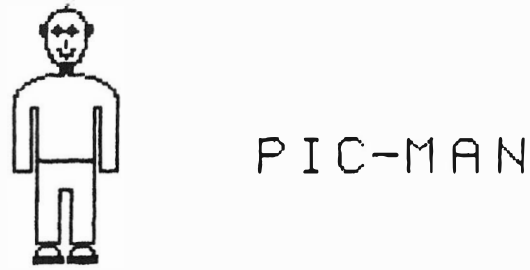


Fig. 6.11 Botas.

Para que parezca más sólido, hemos pintado sus pantalones mediante la orden PAINT (fig. 6.12). La orden PAINT rellenará un área del primer color especificado hasta que alcance al segundo color es-

pecificado y las principales dificultades del usuario es asegurarse que se establecen las coordenadas correctas y que no hay agujeros a través de los cuales la pintura pueda verterse. Intente alterar las coordenadas en la línea 18Ø y observe lo que sucede.

```
180 PAINT(74,100),1,1
```

Fig. 6.12 Coloreado de los pantalones.

La orden DRAW actúa siempre sobre un texto, pero este texto puede haberse definido con anterioridad como un subtexto y puede utilizarse repetidamente mediante la orden X. Ya que tiene dos manos idénticas, para dibujarlas las hemos definido antes en M$. M$ utiliza también el parámetro de no actualización N. Normalmente cada nueva orden DRAW continúa a partir de donde terminó la última línea dibujada, pero si se coloca N delante de una orden, entonces la línea siguiente se dibuja a partir del mismo lugar que la actual. Siga la secuencia con cuidado para ver cómo se forma cada dedo (los pulgares no pueden verse, no es que sea deforme).

```
190 M$="ND5R2ND5R2ND5R2ND5"
```

Para colocar las manos en sus posiciones correctas tan sólo necesitamos seleccionar una nueva posición inicial en la pantalla y después ejecutar M$ colocándola entre «X» y «;» (fig. 6.13).

```
200 DRAW"BM64,100XM$;"
210 DRAW"BM94,100XM$;"
```

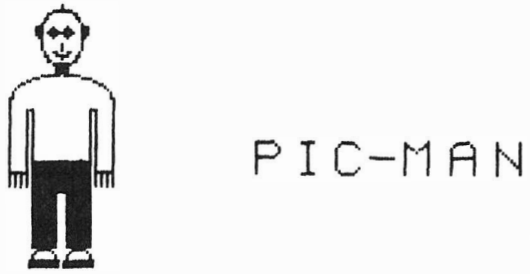


Fig. 6.13 Manos.

El PIC-MAN es más bien presumido y ha decidido ponerse una pajarita definida en A$. Observe que esto se dibuja a partir del centro, utilizando algunas de las órdenes diagonales (F y G) y que es deliberadamente asimétrica. Un movimiento en blanco relativo se utiliza para separar el trazo final del resto del dibujo (fig. 6.14). Esto tiene la ventaja de que no hay que calcular la posición actual, sino únicamente el desplazamiento desde la posición actual como + y − un número de puntos de pantalla. Por regla general no es esencial que se empiece a realizar un dibujo a partir del centro, pero en este caso el PIC-MAN quiere demostrarle que en realidad se trata de una pajarita revolucionaria que crece, por lo que necesita un punto central a partir del que trabajar.

```
220 A$="BM82,69F3U6G6U6F3BM+1,+0
R2":S=4
```

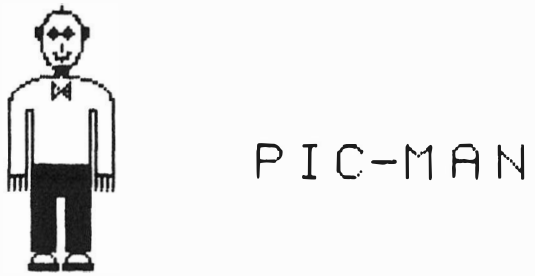


Fig. 6.14 Completo.

El parámetro de escala S establece el tamaño del texto que va a ser dibujado, el parámetro ángulo «A» le permite cambiar la dirección del dibujo en incrementos de 9Ø grados, y el parámetro color «C» le permite cambiar el color del dibujo. Se puede utilizar una variable para cambiar cualquiera de estos parámetros, siempre que se convierta esta variable a una variable de texto mediante STR$. Todas estas ideas se han combinado en esta pequeña secuencia en la que la corbata se dibuja en el color 1 y después en el color Ø (es decir, se dibuja y se borra), en todas las direcciones posibles, y en 1Ø tamaños distintos que van aumentando. Se incluye el sonido para que las cosas vayan más lentas y el movimiento puede seguirse claramente en la pantalla (fig. 6.15).

```
230 DRAW"S"+STR$(S)
240 FOR N=0 TO 3:DRAW"A"+STR$(N)
250 FOR M=1 TO 0 STEP-1:DRAW"C"+
STR$(M)+A$
260 SOUND 255,1:NEXT M,N
270 S=S+1:IF S<10 THEN 230
```

Fig. 6.15 Corbatín giratorio.

Naturalmente, tras la arrogancia vendrá la caída y esta corbata giratoria parece muy peligrosa, por lo que no sería muy sorprendente que estallase. Las explosiones son características muy frecuentes en los programas de ordenador, por lo que ésta es una rutina muy general. Se dibujan una serie de círculos concéntricos en expansión, utilizando la variable X para establecer el diámetro, y se integra un sonido con cada expansión del círculo (fig. 6.16). Se utiliza la orden PLAY en lugar de SOUND, ya que permite la utilización de una duración mucho más corta si el tempo (T) y la longitud de nota (L) se colocan a su valor más alto (255).

```
280 FOR X=1 TO 59 STEP 2:CIRCLE(
82,69),X,1:PLAY"T255L255CD":NEXT
 X
```

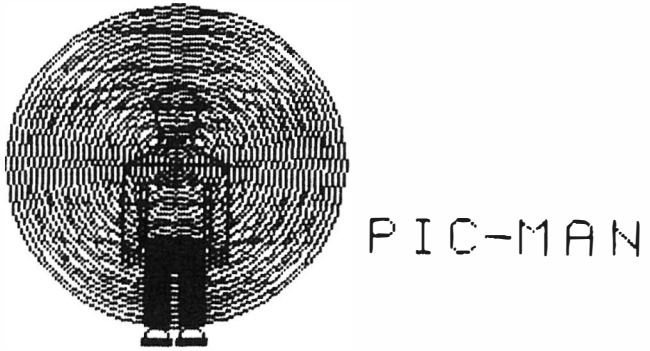


Fig. 6.16 Explosión.

Una vez que ha pasado su punto culminante, la explosión va desapareciendo ya que los círculos se dibujan ahora en orden inverso mediante STEP-1, y tan sólo quedarán algunos fragmentos y las botas

del PIC-MAN (fig. 6.17). Obsérvese que la integración de gráficos y sonidos es más completa aquí, ya que X también varía el volumen y el tempo de la orden PLAY.

```
290 FOR X=59 TO 1 STEP-1:CIRCLE(
82,69),X,0:PLAY"L255V"+STR$(INT(
X/2))+"T"+STR$(X*4)+"DC":NEXT X
```

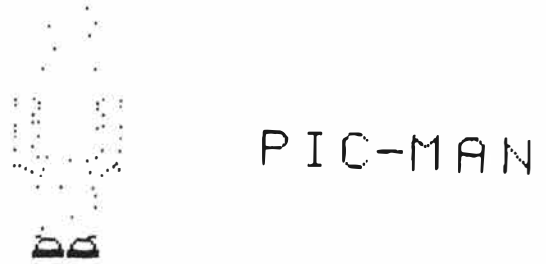


Fig. 6.17 Restos.

7. Movimiento en la pantalla

Algunos tipos de movimientos sobre la pantalla suelen ser un requerimiento muy frecuente en los programas gráficos, pero pueden tomar formas muy distintas. El tipo de movimiento más sencillo trata con un único punto que se mueve de una posición a otra.

Pantalla de textos

Las posiciones de visualización en la pantalla de textos están ordenadas en 16 líneas de 32 caracteres, numeradas secuencialmente desde el Ø al 511, y esto es lo que se tiene en cuenta en esta rutina que moverá un bloque negro (CHR$ (128)) en cualquiera de las cuatro direcciones.

```
10 CLS:P=240
20 A$=CHR$(128)
30 CLS
40 PRINT@ P,A$;
60 I$=INKEY$:IF I$="" THEN 60 EL
SE I=ASC(I$)
70 IF I=8 OR I=21 THEN P=P-1
80 IF I=9 OR I=93 THEN P=P+1
90 IF I=94 OR I=95 THEN P=P-32
100 IF I=10 OR I=91 THEN P=P+32
110 IF P<0 THEN P=0 ELSE IF P>51
0 THEN P=510
150 GOTO 30
```

Obsérvese que se comprueban los códigos ASCII para los caracteres normales y los que se generan con la tecla SHIFT, y que las comprobaciones de límite de la línea 11Ø evitan que el programa se pierda si se intenta salir de la pantalla. La última posición de la pantalla (511) no se utiliza, ya que el visualizar algo en esta posición produce un desplazamiento automático de toda la pantalla.

Ya que la rutina vuelve al CLS de la línea 3Ø cada vez, el carácter que estaba en la antigua posición queda borrado automáticamente. Si

se quiere ser más selectivo y borrar tan sólo el carácter que estaba en la última posición, deberá almacenarse la antigua posición como una nueva variable (UP) y después borrar este punto a medida que se va moviendo, mediante PRINT@UP,B$. A$ y B$ pueden inicializarse con cualquiera de los caracteres gráficos o alfanuméricos. En el ejemplo que viene a continuación, (CHR$(143)) se utiliza para borrar el antiguo punto y dejarlo de color verde.

```
10 CLS:P=240:UP=P
20 A$=CHR$(128):B$=CHR$(143)
30 PRINT@UP,B$
50 UP=P
```

Movimientos no destructivos

En programas más complejos quizá se necesite mover constantemente la pantalla, sin alterarla, por lo que entonces deberá guardarse una copia de lo que hay en la nueva posición de visualización. Puede utilizarse el PEEK para obtener lo que hay en la nueva posición, pero desafortunadamente los valores dados por PEEK no son siempre los mismos que los códigos ASCII. Por lo tanto deberá utilizarse una rutina para calcular el código ASCII adecuado o bien guardar el valor obtenido por el PEEK como una variable sencilla y después colocarlo de nuevo con un POKE en lugar de utilizar la orden PRINT para visualizar un texto. [La pantalla de texto empieza en la posición 1Ø24, por lo que POKE(1Ø24+X) es lo mismo que PRINT@X.] Ya que tan sólo se utiliza una variable, deberá borrarse la antigua posición antes de que se entre la nueva mediante el PEEK. Obsérvese que ahora debe borrarse la línea 3Ø y que se ha añadido un mensaje en la línea 1Ø para que pueda verse cómo funciona la sustitución en la visualización. PE también debe inicializarse al valor entrado en la posición inicial.

```
10 CLS:PRINT@256,"ESTO ES UNA PR
UEBA":P=240:UP=P:PE=PEEK(1024+P)
20 A$=CHR$(128)
30 (borrada)
130 POKE(1024+UP),PE
140 PE=PEEK(1024+P)
150 GOTO 40
```

El cursor de textos

Cuando se utiliza este tipo de rutina para mover el cursor sobre el texto, suele invertirse la visualización del carácter que está en la posi-

ción del cursor, para que este carácter siga siendo visible. Ya que los códigos ASCII para las minúsculas (visualización invertida) son todas 32 unidades mayores que los correspondientes a las letras mayúsculas (normal) tan sólo necesitaremos añadir 32 al valor obtenido por el PEEK.

```
40 PRINT@ P,CHR$(PE+32);
```

Gráficos

Esencialmente pueden utilizarse rutinas similares para tratar con los pixels individuales en los gráficos, tanto de baja como de alta resolución. En ambos casos, calcular los movimientos es incluso más sencillo ya que la pantalla está distribuida mediante coordenadas X e Y. En baja resolución, la pantalla tiene 64 (Ø-63) por 32 (Ø-31), la orden SET convertirá un pixel a cualquier color (C1) y RESET lo volverá de nuevo al color de fondo (negro).

```
10 CLS0:X1=32:Y1=16:X2=X1:Y2=Y1
20 C1=4
30 RESET(X2,Y2)
40 SET(X1,Y1,C1)
50 X2=X1:Y2=Y1
60 I$=INKEY$:IF I$="" THEN 60 EL
SE I=ASC(I$)
70 IF I=8 OR I=21 THEN X1=X1-1
80 IF I=9 OR I=93 THEN X1=X1+1
90 IF I=94 OR I=95 THEN Y1=Y1-1
100 IF I=10 OR I=91 THEN Y1=Y1+1
110 IF X1<0 THEN X1=0 ELSE IF X1
>63 THEN X1=63
120 IF Y1<0 THEN Y1=0 ELSE IF Y1
>31 THEN Y1=31
150 GOTO 30
```

Las órdenes equivalentes en alta resolución PSET y PRESET funcionan de la misma forma que SET y RESET, pero sobre una matriz de 256 por 192.

```
10 PMODE 3,1:SCREEN 1,0:PCLS:X1=
128:Y1=96:X2=X1:Y2=Y1
30 PRESET(X2,Y2)
40 PSET(X1,Y1,C1)
110 IF X1<0 THEN X1=0 ELSE IF X1
>255 THEN X1=255
```



```
120 IF Y1<0 THEN Y1=0 ELSE IF Y1
>191 THEN Y1=191
150 GOTO 30
```

Desafortunadamente, aunque la orden POINT puede utilizarse para comprobar el color de un pixel individual en baja resolución, este valor no puede incorporarse dentro de una orden SET, para poder reproducir la visualización original después del movimiento, ya que Ø (negro) no es un color válido. Por otra parte, el equivalente en alta resolución PPOINT sí que dará un color que puede utilizarse en PSET para reproducir el estado anterior del punto donde está el cursor.

```
20 C1=4:C2=PPOINT(X1,Y1)
30 (borrada)
130 PSET(X2,Y2,C2)
140 C2=PPOINT(X1,Y1)
150 GOTO 40
```

El cursor se moverá ahora sin destruir nada sobre la pantalla, incluso en el caso de que el color de fondo cambie. Esto puede comprobarse cambiando el PCLS en la línea 1Ø a un color distinto o de forma más clara, aumentando esta línea para que nos dé un dibujo coloreado sobre el que moverse.

```
15 CIRCLE(128,96),10,2:PAINT(128
,96),2,2:CIRCLE(128,96),20,3
```

El único problema ahora es que el cursor en rojo es invisible cuando está sobre un fondo rojo. Esto puede solventarse comprobando que el color del cursor (C1) sea distinto del color de fondo (C2), cambiando el color del cursor si fuera necesario.

```
140 C2=PPOINT(X1,Y1):IF C2<>C1 T
HEN 40 ELSE IF C1<3 THEN C1=C1+2
 ELSE C1=C1-2
```

Cursor parpadeante

En una visualización muy compleja quizá sea difícil ver el cursor, por lo que a veces suele iluminarse y apagarse para dar un efecto de parpadeo. Esto se consigue sencillamente modificando la línea de comprobación de tecla, de forma que si no se pulsa ninguna tecla el cursor se borra y se dibuja de nuevo. Por ejemplo:

```
60 I$=INKEY$:IF I$="" THEN PRESE
T(X1,Y1):GOTO 40 ELSE I=ASC(I$)
```

La velocidad de parpadeo puede disminuirse insertando un bucle de espera.

```
60 I$=INKEY$:IF I$="" THEN PRESE
T(X1,Y1):FOR T=1 TO 10:NEXT T:GO
TO 40 ELSE I=ASC(I$)
```

Control del movimiento

Si se quiere viajar más de prisa, en cada decisión puede moverse más de un pixel (incrementar X1 e Y1 en más de 1), aunque esto quizá dé o no un control menos preciso. Recuerde que en PMODE 4 cada coordenada se refiere a un único punto en la pantalla, pero en los PMODE más bajos los puntos están agrupados, de forma que el iluminar más de una coordenada puede producir el mismo efecto. Por ejemplo, en PMODE Ø deberán incrementarse X1 e Y1 al menos en incrementos de 2, y en PMODE 3 deberá incrementarse X1 en incrementos dobles de los utilizados para Y1.

El utilizar INKEY$ para un movimiento continuo puede ser tedioso, ya que la rutina almacenada en ROM para la eliminación de rebotes en el teclado, requiere que levante el dedo cada vez y suelte la tecla. Sin embargo, este problema puede esquivarse fácilmente utilizando esta subrutina donde se utilizaría normalmente el INKEY$. Espera que se pulse una tecla y entonces autorrepite hasta que la tecla se suelta. PEEK (135) nos da el código ASCII de la tecla pulsada.

```
60 ON (PEEK(337)<255)+1 GOTO 60
65 I=PEEK(135)
```

Cuando se utilizan palancas para controlar directamente la posición en la pantalla deben utilizarse valores de escala apropiados para JOYSTK. Por ejemplo, en baja resolución el valor JOYSTK(1) para la coordenada Y debe dividirse por dos para que dé un valor entre Ø y 31.

```
1000 X1=JOYSTK(0):Y1=JOYSTK(1)/2
```

Por otra parte, en PMODE 4, JOYSTK(Ø) debe multiplicarse por 4 (para que dé entre Ø-255) y JOYSTK(1) por 3 para que dé un valor entre (Ø-191).

```
1000 X1=JOYSTK(0)*4:Y1=JOYSTK(1)
*3
```

Aunque este escalado para la alta resolución nos da números dentro del rango adecuado, hay que recordar que no podrán alcanzarse todos los puntos individuales, ya que el movimiento se hace ahora en incrementos de 4 y 3.

También pueden utilizarse las palancas para comprobar la dirección del movimiento en lugar de su posición absoluta y en este caso pueden alcanzarse fácilmente todos los puntos. Los valores instantáneos de la palanca se leen en variables temporales JØ y J1 que se utilizan para determinar la dirección.

```
1000 J0=JOYSTK(0):J1=JOYSTK(1)
1020 IF J0=<20 THEN X1=X1-1 ELSE
 IF J0=>50 THEN X1=X1+1
1030 IF J1=<20 THEN Y1=Y1-1 ELSE
 IF J1=>50 THEN Y1=Y1+1
```

Cuando la palanca está en el centro, ambos valores, JØ y J1, serán 32, de forma que los límites superior e inferior, que serán 2Ø y 5Ø, han sido arbitrariamente elegidos, pero la sensibilidad (distancia que debe moverse la palanca antes de que se produzca un efecto) puede alterarse variando estos valores límites. Los valores cercanos a 32 dan una respuesta más rápida, pero entonces sería más difícil prevenir movimientos no deseados. Ya que las coordenadas X e Y se consideran separadamente, pueden generarse movimientos en diagonal. Si se necesita un estado de espera, puede incluirse una comprobación de la posición central para ambos ejes.

```
1010 IF J0>20 AND J0<50 AND J1>2
0 AND J1<50 THEN 1000
```

La posición de la palanca puede utilizarse también para controlar la velocidad del movimiento si el tamaño del STEP está relacionado con la distancia de la palanca respecto a su posición central. En esta rutina se dan tres tamaños distintos del STEP (1, 2 y 5 unidades). Las posiciones más extremas deben siempre comprobarse en primer lugar.

```
1020 IF J0<10 THEN X1=X1-5 ELSE
IF J0<15 THEN X1=X1-2 ELSE IF J0
<20 THEN X1=X1-3 ELSE IF J0>60 T
HEN X1=X1+5 ELSE IF J0>50 THEN X
1=X1+2 ELSE IF J0>40 THEN X1=X1+
1
```

Movimiento de más de un punto

Hasta ahora hemos analizado la situación más sencilla donde tan sólo se mueve un único punto, pero con frecuencia necesitaremos

mover una serie de puntos relacionados entre sí, que constituyan un dibujo determinado. Analizaremos en primer lugar este caso para la baja resolución.

Iluminación directa de los puntos

La forma más sencilla para realizar el dibujo elegido es iluminar (SET) con el color deseado cada uno de los puntos de la pantalla elegidos, que están definidos por sus coordenadas X e Y. A menos que el número de puntos que deben iluminarse sea muy pequeño, estas coordenadas se guardan por lo general en sentencias DATA. Estos datos pueden entonces leerse (READ) e iluminar los puntos correspondientes.

```
30 CLS0
40 C1=1
50 FOR N=1 TO 16:READ X,Y:SET(X,
Y,C1):NEXT N
80 DATA 0,0,0,1,0,2,0,31,0,1,1,1
,2,1,3,2,0,2,1,2,2,2,3,3,0,3,1,3
,2,3,3,4,0,4,1,4,2,4,3
```

Obsérvese que la pantalla debe borrarse y dejarse de color negro (CLSØ) y si los datos deben utilizarse más de una vez deberá añadirse la orden RESTORE al final de la línea 5Ø.

```
20 FOR R=1 TO 100
50 FOR N=1 TO 16:READ X,Y:SET(X,
Y,C1):NEXT N:RESTORE
60 NEXT R
```

La visualización irá parpadeando a medida que se vayan iluminando los puntos y la pantalla se borre, lo cual se realizará cien veces.

Utilización de tablas

Una modificación del planteo anterior es el leer los datos y guardarlos en tablas X e Y y después utilizar los elementos de la tabla en la orden SET.

```
1 GOSUB 90
50 FOR N=1 TO 16:SET(X(N),Y(N),C
1):NEXT N
75 END
90 DIM X(16),Y(16)
100 FOR N=1 TO 16:READ X(N),Y(N)
:NEXT N:RETURN
```


A primera vista esto parece más complicado, pero este método principalmente tiene dos ventajas. La primera ventaja es un pequeño descenso en el tiempo empleado para visualizar el dibujo sobre la pantalla. Esto puede demostrarse contando el tiempo empleado por un bucle que se ejecute cien veces.

```
10 TIMER=0
70 PRINT TIMER/50
```

Los valores comparativos del TIMER nos dan 17,8 segundos para la lectura directa y 16,8 segundos utilizando el método de la tabla.

Un problema con las sentencias DATA es que tan sólo pueden leerse secuencialmente, por lo que la segunda ventaja y la más importante de transferir los valores a una tabla es que cualquier parte del dibujo puede utilizarse independientemente, de una forma más fácil. Este proceso se analiza después con más detalle, pero la idea general puede verse añadiendo un STEP 2 en la línea 5Ø de forma que sólo se iluminan puntos alternados.

```
50 FOR N=1 TO 16 STEP 2:SET(X(N)
,Y(N),C1):NEXT N
```

Utilización de los caracteres gráficos

Como se recordará de la primera explicación sobre los gráficos de baja resolución, existen restricciones en la forma en que pueden iluminarse los puntos en distintos colores y cualquier estructura de puntos iluminados puede también representarse por ciertos caracteres gráficos del Dragon. Aunque la utilización de este método requiere una planificación cuidadosa, con un planteo inicial sobre papel cuadriculado para determinar los caracteres apropiados, tiene cierto número de ventajas. Cada carácter gráfico es equivalente a cuatro pixels, por lo que el número de valores que deben entrarse queda dividido por cuatro. Los caracteres gráficos pueden visualizarse mediante la orden PRINT, por lo que ya no es necesaria la larga, complicada y a menudo fuente de errores que constituye la sentencia DATA. Las combinaciones imposibles de puntos serán evidentes y existe también un considerable incremento de la velocidad de realización del dibujo. La acción más sencilla es el colocar directamente los caracteres dentro de la orden PRINT.

```
50 FOR N=0 TO 1:PRINT@ 32*N,CHR$
(143);CHR$(143);:NEXT N
```

Esta nueva línea 5Ø produce exactamente la misma visualización que la antigua, mediante un bucle que visualiza dos caracteres en líneas adyacentes. Sin embargo, es mucho más rápida (5,1 segundos); tan sólo cerca del 30 % del tiempo empleado para iluminar cada punto individualmente.

La velocidad todavía puede aumentarse más si el dibujo se guarda como una variable de texto.

```
1 A$=CHR$(143)+CHR$(143)
50 FOR N=0 TO 1:PRINT@ 32*N,A$;:
NEXT N
```

Este pequeño cambio tiene efectos profundos (3,2 segundos) ya que el tiempo desciende hasta sólo el 18 % del original. Si se dobla el número de puntos a iluminar, prácticamente doblará el tiempo empleado (34,1 segundos para la lectura directa o 3Ø,2 segundos con una tabla). Mientras que el incrementar la longitud de A$ hasta 1Ø o incluso 2Ø caracteres tiene un efecto mucho menor (3,7 segundos y 4,2 segundos respectivamente).

Donde se requiera movimiento del dibujo, por lo general será necesario el redibujarlos rápidamente, y el método que utiliza caracteres será evidentemente mejor que el planteo que utiliza la orden SET Si se requiere todavía más velocidad incluya un POKE&HFFD7,Ø (si su máquina lo acepta) lo que reducirá todos los tiempos en un 25 % más.

Movimiento referido

La forma más sencilla de definir la posición en la pantalla del dibujo es mediante desplazamientos a partir de la posición inicial. Para un sistema de coordenadas X,Y llamaremos a estas XO (para el desplazamiento en el eje X) e YO (para el desplazamiento en el eje Y). Para determinar las nuevas coordenadas tan sólo tendremos que sumar o restar XO e YO de la posición actual, tal como se describió anteriormente en la sección de movimiento general, y después añadir estos desplazamientos a la línea que contiene la orden SET.

```
50 FOR N=1 TO 16:SET(X(N)+XO,Y(N
)+YO,C1):NEXT N
```

Cuando se utilicen caracteres gráficos estamos de nuevo en la distribución secuencial Ø-511, por lo que deberemos colocar incrementos de 32 en el eje Y, ya sea en la línea de comprobación de tecla o en la línea de PRINT@.

Modificación de los límites

Cuando debe iluminarse más de una posición en la pantalla al mismo tiempo debemos modificar nuestros límites para asegurarnos de que hay espacio suficiente para la totalidad del dibujo. Si utilizamos la parte superior izquierda de nuestra área como punto de referencia, tendremos que restar la anchura del dibujo del límite del eje X y la altura del límite del eje Y. Por ejemplo, un dibujo de 4 por 4 pixels en la pantalla de baja resolución no puede moverse más allá de unas coordenadas X,Y de valor (63-4), (31-4). Si volvemos a la rutina del cursor veremos que como proceso general el tamaño podría definirse como XS e YS y entonces restar estos valores de los límites absolutos.

```
10 CLS0:X1=32:Y1=16:X2=X1:Y2=Y1:
XS=4:YS=4
110 IF X1<0 THEN X1=0 ELSE IF X1
>(63-XS) THEN X1=(63-XS)
120 IF Y1<0 THEN Y1=0 ELSE IF Y1
>(31-YS) THEN Y1=(31-YS)
```

Nave espacial

Vamos a poner todos estos principios del movimiento de un dibujo en baja resolución, juntos en un programa que mueve un dibujo más complicado, el de una nave espacial (fig. 7.1) alrededor de la pantalla, en lugar de un sencillo rectángulo.

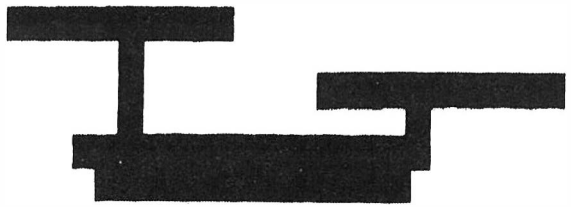


Fig. 7.1 Nave espacial.

La primera idea es el diseñar primero el dibujo en una retícula correspondiente a la baja resolución, poner todas las coordenadas X,Y de los pixels en sentencias DATA, leerlas y colocarlas en tablas, y establecer las coordenadas de la orden SET en función de las teclas de movimiento del cursor. La transferencia de los datos a las tablas

está colocada en una subrutina al final del programa para que no estorbe. A la vuelta de este proceso de inicialización, se iluminarán todos los puntos mediante la línea 3Ø y el programa esperará que se pulse una tecla. La rutina de tecla de cursor utilizada aquí emplea la orden PEEK que actúa sobre ciertas posiciones de memoria reservadas para detectar qué tecla se está pulsando y tiene la ventaja sobre INKEY$ de que es autorrepetitiva.

Tan pronto como se pulse una tecla (PEEK(337)<255) se borrará cada uno de los puntos (RESET) por turno, debido a la línea 5Ø. Obsérvese que esto debe hacerse ANTES de que se actualicen las posiciones de la pantalla.

Cuando se especifiquen los límites no hay que olvidarse de tener en cuenta la anchura y la altura actual del dibujo. Si se examinan los datos, la coordenada X más baja es 5 y la más alta 29, y de forma similar los límites de las coordenadas Y son 1 y 6. Por lo tanto, el tamaño total del dibujo es de 25 por 6 pixels. El dibujo empieza por la parte superior izquierda de coordenadas (5,1) en lugar de las coordenadas iniciales más obvias Ø,Ø, por lo tanto el límite por la derecha de X deberá ser (longitud del eje X (64)−anchura del dibujo (25)−distancia original hasta la izquierda (5))=34, el límite inferior de la Y (longitud del eje Y (32)−altura del dibujo (6)−distancia original hasta la parte superior (1))=25, y los límites de X por la izquierda y de Y por arriba son (Ø−distancia original hasta la izquierda (5))=−5 y (Ø−distancia original hasta arriba (1))=−1, respectivamente.

```
10 GOSUB 1000
20 CLSØ
30 FOR N=1 TO 55:SET(X(N)+XO,Y(N
)+YO,C):NEXT N
40 IF PEEK(337)=255 THEN 40
50 FOR N=1 TO 55:RESET(X(N)+XO,Y
(N)+YO):NEXT N
60 IF PEEK(341)=223 THEN YO=YO-1
70 IF PEEK(342)=223 THEN YO=YO+1
80 IF PEEK(343)=223 THEN XO=XO-1
90 IF PEEK(344)=223 THEN XO=XO+1
100 IF XO<5 THEN XO=5 ELSE IF XO
>34 THEN XO=34
110 IF YO<1 THEN YO=1 ELSE IF YO
>25 THEN YO=25
120 GOTO 30
1000 DIM X(55),Y(55)
1010 FOR N=1 TO 55:READ X(N),Y(N
):NEXT N:RETURN
5000 DATA 5,1,6,1,7,1,8,1,9,1,10
,1,11,1,12,1,13,1,14,1,10,2,10,3
,10,4,8,5,9,5,10,5,11,5,12,5,13,
5,14,5,15,5,16,5,17,5,18,5,19,5,
20,5,21,5,22,5,23,5,9,6,10,6,11,
6,12,6,13,6,14,6,15,6,16,6,17,6,
18,6,19,6,20,6,21,6,22,6
5010 DATA 23,4,19,3,20,3,21,3,22
,3,23,3,24,3,25,3,26,3,27,3,28,3
,29,3
```

Cuando ejecute esta rutina descubrirá que funciona, pero que es muy lenta, empleando cerca de 1,2 segundos para cada actualización de la posición. Da la sensación que la nave espacial está perdida en el espacio, ya que emplea más de 45 segundos en cruzar la pantalla. Evidentemente hay que hacer algo para acelerar las cosas, por lo tanto ¿por qué no sustituir todos los RESET de los puntos iluminados en la línea 5Ø por un sencillo CLSØ? No es sorprendente comprobar que, efectivamente, esto divide por 2 el tiempo empleado para cada actualización, que será de Ø,6 segundos, ya que sólo hay que hacer la mitad del trabajo. Aunque CLSØ borrará cualquier texto que haya en la pantalla, recuerde que ambos, CLS y PRINT, son prácticamente instantáneos, por lo que será más rápido borrar la pantalla completa y volver a visualizar el texto que se necesite, en lugar de utilizar el RESET.

Tal como está, la pantalla se borrará antes de las comprobaciones de tecla y de los cálculos, pero se mejoraría el efecto si borramos la línea 5Ø y saltamos al CLSØ de la línea 2Ø de forma que la visualización original se mantuviese hasta que la nueva estuviese a punto de visualizarse:

```
120 GOTO 20
```

Para intentar que las cosas se acelerasen podríamos utilizar PRINT CHR$ en lugar de SET. En la figura 7.2 se muestra una copia del dibujo de la nave espacial transferida a una orden PRINT@. Las posiciones individuales y códigos de carácter de la orden PRINT@ podrían tratarse como datos y colocarse en tablas al igual que las coordenadas de la orden SET, y entonces visualizarse a partir de los elementos de la tabla con un desplazamiento. ¡Asegúrese de que incluye un punto y coma después del PRINT!

```
30 FOR N=1 TO 29:PRINT@ X(N)+XO,
CHR$(Y(N));:NEXT N
1010 FOR N=1 TO 29:READ X(N),Y(N
):NEXT N:RETURN
```

```
1020 DATA 2,129,3,131,4,131,5,13
1,6,131,7,130,37,138,41,129,42,1
31,43,131,44,131,45,131,46,131,6
8,131,69,139,70,131,71,131,72,13
1,73,131,74,131,75,135,100,132,1
01,140,102,140,103,140,104,140,1
05,140,106,140,107,136
```

Aunque esto dobla de nuevo la velocidad (Ø,3 segundos) sigue pareciendo una deriva a impulsos ya que se emplea mucho tiempo en recobrar los valores de la tabla durante cada actualización. Si se tiene en cuenta la velocidad de movimiento, ahora es más del doble que antes recuerde que cada movimiento le lleva el doble de lejos (un carácter tiene dos pixels de alto por dos pixels de ancho). Para obtener realmente una alta velocidad de movimiento vamos a tener que deshacernos de estas tablas y visualizar textos directamente.

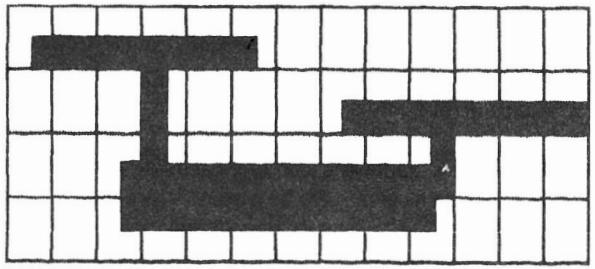


Fig. 7.2 Nave espacial formada por caracteres.

El dibujo tiene cuatro unidades de carácter de altura por lo que necesitaremos cuatro variables de texto (A$-D$), éstas están definidas en las líneas 1ØØØ-1Ø3Ø, y se visualizan en la posición correcta en relación con los otros, mediante los números apropiados para la posición inicial en las órdenes PRINT@ (Ø, 35, 66, 98). Obsérvese que B$ tiene tres caracteres negros (128) incluidos, ya que es más sencillo que dividir el texto en dos.

```
30 PRINT@ 0+PO,A$;:PRINT@ 35+PO,
B$;:PRINT@ 66+PO,C$;:PRINT@ 98+P
O,D$;
1000 A$=CHR$(129)+STRING$(4,131)
+CHR$(130)
1010 B$=CHR$(138)+STRING$(3,128)
```



```
+CHR$(129)+STRING$(5,131)
1020 C$=CHR$(131)+CHR$(139)+STRI
NG$(5,131)+CHR$(135)
1030 D$=CHR$(132)+STRING$(6,140)
+CHR$(136)
1040 RETURN
```

Descubrirá que el tiempo de actualización se ha reducido ahora de forma dramática hasta Ø,Ø4 segundos, lo que es treinta veces más rápido que nuestro programa original. La rutina de tecla de cursor podría alterarse de forma que existiese únicamente un desplazamiento (PO) y éste debe entonces cambiarse en incrementos de 32 para el movimiento vertical. Si se coloca la nave espacial en la parte inferior derecha de la pantalla verá que la posición límite de PRINT@ es 4Ø3.

```
60 IF PEEK(341)=223 THEN PO=PO-3
2
70 IF PEEK(342)=223 THEN PO=PO+3
2
80 IF PEEK(343)=223 THEN PO=PO-1
90 IF PEEK(344)=223 THEN PO=PO+1
100 IF PO<0 THEN PO=0 ELSE IF PO
>403 THEN PO=403
120 GOTO 20
```

El único problema que resta es que la nave volverá hacia atrás cuando alcance el borde de la pantalla, ya que las posiciones PRINT@ están colocadas secuencialmente en lugar de ser coordenadas de X,Y. Para resolver esto necesitaremos analizar los movimientos del cursor en términos de línea (L) y fila (F) en lugar de visualizar sencillamente la posición mediante PRINT@, y calcular PO como (L*32)+F. No hay necesidad de dejar la comprobación de la posición absoluta de la orden PRINT, ya que un valor por encima de 4Ø3 ahora no puede ser alcanzado, ya que F sería mayor que 19 y L que 11.

```
60 IF PEEK(341)=223 THEN L=L-1
70 IF PEEK(342)=223 THEN L=L+1
75 IF L<0 THEN L=0 ELSE IF L>11
THEN L=11
80 IF PEEK(343)=223 THEN F=F-1
90 IF PEEK(344)=223 THEN F=F+1
95 IF F<0 THEN F=0 ELSE IF F>19
THEN F=19
100 PO=(L*32)+F
120 GOTO 20
```

Una forma más fácil de tratar con CHR$

Como ejemplo damos una rutina que da el dibujo de un dragón utilizando gran cantidad de caracteres gráficos de baja resolución (fig. 7.3), pero que requiere que usted entre las siguientes líneas para formar el dibujo.

```
4000 CLS0:PRINT@ 41,CHR$(241);CH
R$(243);CHR$(243);CHR$(159);CHR$
(243);CHR$(243);CHR$(242);STRING
$(4,128);CHR$(247);STRING$(5,61)
;CHR$(248);
4005 PRINT@ 68,STRING$(5,127);
4010 PRINT@ 73,CHR$(255);STRING$
(5,61);CHR$(255);CHR$(242);STRIN
G$(2,128);CHR$(247);CHR$(61);CHR
$(255);CHR$(254);STRING$(2,252);
CHR$(248);
4020 PRINT@ 106,CHR$(253);STRING
$(3,255);CHR$(61);CHR$(255);CHR$
(250);CHR$(128);CHR$(245);CHR$(6
1);CHR$(255);CHR$(254);
4030 PRINT@ 141,CHR$(247);CHR$(6
1);CHR$(254);CHR$(128);CHR$(241)
;CHR$(61);CHR$(255);CHR$(254);
4040 PRINT@ 172,CHR$(245);CHR$(2
55);STRING$(14,61);CHR$(255);CHR
$(242);
4050 PRINT@ 205,CHR$(253);STRING
$(2,255);CHR$(61);STRING$(10,255
);CHR$(61);CHR$(255);CHR$(250);
4060 PRINT@ 238,CHR$(241);CHR$(6
1);CHR$(255);CHR$(254);STRING$(6
,128);CHR$(247);CHR$(255);CHR$(6
1);CHR$(255);CHR$(248);
4070 PRINT@ 269,CHR$(241);CHR$(6
1);CHR$(255);CHR$(254);STRING$(6
,128);CHR$(241);CHR$(255);CHR$(6
1);CHR$(255);CHR$(254);
4080 GOTO 4080
```

Como ha podido ver esto es un poco pesado, muy vulnerable a errores de escritura, y más bien difícil de editar, por lo que intentaremos encontrar una forma más fácil de trata con series largas de caracteres.

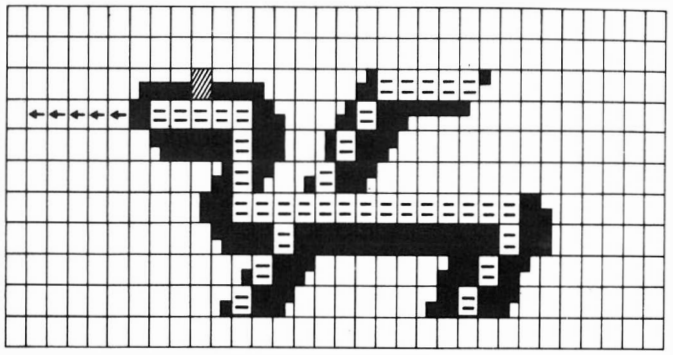


Fig. 7.3 Dragón.

CHR$	241	243	159	242	128	...
TABLA	1	2	3	4	5	

Fig. 7.4 Representación de los caracteres en una tabla.

Si observa las líneas anteriores verá que, en total, hemos utilizado únicamente 14 caracteres distintos, por lo que podríamos guardar sus códigos como datos y después leerlos y guardarlos en una tabla.

```
10 DATA 241,243,159,242,128,247,
61,248,127,255,254,252,250,245,2
53
20 CLEAR 1000:DIM A(15):CLS0
30 FOR CH=1 TO 15:READ A(CH):NEX
T CH
```

Cada elemento de la tabla describe uno de los caracteres (fig. 7.4) y ahora podemos describir cada línea de la figura por una serie de números en lugar de tener que escribir repetidamente CHR$ (...), con tal de que tengamos una rutina de decodificación que pueda convertir nuestra serie de números en códigos CHR$. Por ejemplo, la primera línea sería ahora:

```
110 A$=" 5 5 5 5 5 1 2 2 3 2 2 4
 5 5 5 5 6 7 7 7 7 7 8":GOSUB 10
00
```

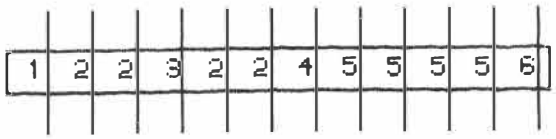


Fig. 7.5 Descomposición de la cadena.

Ya que tenemos números mayores que 9 deberemos entrar los números más bajos con un espacio delante, de forma que podamos descomponer siempre el texto en bloques de dos (líneas 1ØØØ-1Ø1Ø) (fig. 7.5) y después utilizar la función VAL para obtener el valor del texto resultante (B$), y así convertirlo en el número correspondiente. Entonces la línea 1Ø3Ø busca en la tabla A y visualiza el carácter correspondiente a este número según el elemento de la tabla que tiene este subíndice. Cuando todo el texto se ha decodificado, la posición de visualización se mueve a la siguiente línea y se hace el RETURN.

```
1000 FOR C=1 TO LEN(A$) STEP 2
1010 B$=MID$(A$,C,2)
1020 B=VAL(B$)
1030 PRINT CHR$(A(B));
1050 NEXT C
1060 PRINT:RETURN
```

Las líneas 12Ø-18Ø describen el resto del dibujo y si ejecuta esto verá cómo se crea el dragón, aunque un poco lentamente (3,7 segundos). La línea 3ØØ llama a una subrutina que detiene el programa hasta que se pulse una tecla (1Ø7Ø). Para hacer las cosas más sencillas, en una tabla posterior todos los números de cada A$ empiezan en la misma fila de la pantalla, incluso aunque esto signifique que deben insertarse blancos al principio.

```
120 A$=" 9 9 9 9 910 7 7 7 7 710
 4 5 5 6 710111212 8":GOSUB 1000
130 A$=" 5 5 5 5 5 515101010 710
13 514 71011":GOSUB 1000
140 A$=" 5 5 5 5 5 5 5 5 5 6 711
 5 1 71011":GOSUB 1000
150 A$=" 5 5 5 5 5 5 5 51410 7 7
 7 7 7 7 7 7 7 7 7 7 710 4":GO
SUB 1000
160 A$=" 5 5 5 5 5 5 5 5 5151010
 710101010101010101010 71013":GO
SUB 1000
170 A$=" 5 5 5 5 5 5 5 5 5 5 1 7
```



```
1011 5 5 5 5 5 5 610 710 8":GOSU
B 1000
180 A$=" 5 5 5 5 5 5 5 5 5 1 710
11 5 5 5 5 5 5 110 71011":GOSUB
1000

300 GOSUB 1070
1070 IF PEEK(337)=255 THEN 1070
ELSE RETURN
```

Si se comete un error al entrar cada línea, es mucho más sencillo de corregir, pero este tipo de operación para acelerar las cosas no es de mucha utilidad. La siguiente etapa es el hacer otra tabla (L$) que guarde la imagen de cada línea del dibujo una vez que se ha formado. Cada nuevo carácter se añade al final de L$(n) (línea 1Ø4Ø), y se incrementa L para avanzar hasta el siguiente elemento de L$, para guardar cada nueva línea.

```
20 CLEAR 1000:DIM A(15):DIM L$(8
):CLS0
1040 L$(L)=L$(L)+CHR$(A(B))
1060 L=L+1:PRINT:B$="":RETURN
```

Ahora podemos reproducir el dibujo a partir de la tabla L$ de forma muy sencilla y mucho más rápida que antes. Una comprobación revela que ahora necesita Ø,Ø8 segundos, que es tan sólo el 2 % que en el caso anterior.

```
310 CLS0
320 FOR L=0 TO 7:PRINT L$(L):NEX
T L
400 GOSUB 1070
```

Si utilizamos la orden PRINT@ con respecto al número de línea (P*32), en lugar de sólo la orden PRINT, a continuación de nuestro texto podemos poner un punto y coma para reservar el fondo negro (por esto nos aseguramos al principio de que todas las líneas empezaran en la misma fila de la pantalla).

```
410 CLS0
420 FOR P=0 TO 7:PRINT@(P*32),L$
(P);:NEXT P
500 GOSUB 1070
```

Tal como está, estamos visualizando el dibujo en la parte superior izquierda, pero podríamos fácilmente desplazarla, añadiendo otra va-

riable (PO). Un valor de 198 colocará la figura en la parte inferior derecha de la pantalla.

```
510 CLSO
520 PO=198
530 FOR P=0 TO 7:PRINT@(P*32)+PO
,L$(P);:PRINT CHR$(128);:NEXT P
540 GOSUB 1070
```

El siguiente desarrollo lógico es conseguir que la figura se mueva, y esto puede hacerse sencillamente cambiando el desplazamiento. El cambio debe ser negativo ya que los dragones nunca van hacia atrás.

```
540 PO=PO-1:GOSUB 1070:GOTO 530
```

Nuestro dragón se moverá ahora hacia arriba sobre la pantalla, dividiéndose en los bordes, hasta que el programa se pierda indicando un FC ERROR cuando la posición de visualización sea negativa. Para evitar que siga subiendo podríamos utilizar un bucle para el desplazamiento que vaya de 31 a Ø, aunque seguiría dividiéndose en los bordes, por lo que necesitaremos un CLSØ al final de cada ciclo, lo que producirá que la visualización parpadee.

```
540 PO=PO-1:GOSUB 1070
600 GOSUB 1070
610 CLSO
620 FOR PO=31 TO 0 STEP-1
630 FOR P=0 TO 7:PRINT@ (P*32)+P
O,L$(P);:PRINT CHR$(128);:NEXT P
640 NEXT PO
650 CLSO
660 GOTO 620
```

Desplazamiento horizontal de la pantalla

Es posible producir una visualización mejor, que se desplace sin volver a aparecer por el otro lado de la pantalla, si añadimos algunas secciones negras (STRING$ (3Ø,128)) a ambos lados de nuestro dragón, y sólo se muestra una sección MID$ del texto total que vaya aumentando cada vez.

```
660 GOSUB 1070
700 FOR L=0 TO 7:L$(L)=STRING$(3
0,128)+L$(L)+STRING$(30,128):NEX
T L
```



```
710 CLSO
730 FOR P=1 TO 54:FOR L=0 TO 7:P
RINT@ (L*32),MID$(L$(L),P,30);:P
RINT CHR$(128);:NEXT L:NEXT P
740 GOTO 730
```

No sólo se consigue que el dragón se mueva más suavemente sino que nunca vuelve a aparecer antes de que haya desaparecido por completo. Si quiere ser realmente inteligente, puede hacer que secciones de la pantalla se desplacen en direcciones opuestas al mismo tiempo, construyendo otro texto y modificando la forma en que se descompone el texto, aunque esto, como es obvio, implica un aumento de tiempo.

```
720 X$=STRING$(45,128)+"YA ESTA
AQUI EL DRAGON"+STRING$(10,128)
730 FOR P=1 TO 64:FOR L=0 TO 7:P
RINT@ (L*32)+96,MID$(L$(L),P,30)
;:PRINT CHR$(128);:PRINT@ 0,MID$
(X$,65-P,30);:PRINT@ 384,MID$(X$
,65-P,30);:NEXT L:NEXT P
```

Naturalmente, este tipo de desplazamiento horizontal puede utilizarse para producir cualquier tipo de dibujo, y es particularmente adecuado para visualizaciones a gran escala y para anuncios.

8. Copia de la pantalla

Ahora que ya hemos visto las órdenes que nos permiten crear gráficos en la pantalla y cómo pueden moverse a través de ella, vamos a considerar otras características que están relacionadas con el conseguir copias de la pantalla.

PCOPY

La orden en alta resolución, PCOPY, nos permite hacer copias instantáneas de páginas gráficas completas, una cada vez. Todo lo que hay que hacer es especificar la página fuente (de donde) y la objeto (a donde). Por ejemplo:

```
PCOPY 1 TO 2
```

hará una copia de la página gráfica 1, en la página gráfica 2. Recuerde que ya que esta orden genera una copia, la página fuente todavía sigue teniendo su contenido original, por lo que se puede seguir copiando la misma cosa.

En el PMODE Ø tan sólo se utiliza una página gráfica para la totalidad de la pantalla, por lo que PCOPY puede cambiar la pantalla completa de una sola vez. En el siguiente ejemplo se visualiza primero la página 1 y se borra, dejándola en negro. Cuando se pulsa una tecla se visualiza la página 2, se borra y se deja en negro. Cuando se vuelve a pulsar una tecla, la página 2 se visualiza y se deja en verde. Cuando de nuevo se pulsa una tecla, la página 1 se copia en la página 2 (la página que se está visualizando en aquel momento) por lo que se cambia a negro.

```
10 PMODE 0,1:SCREEN 1,0:PCLS
20 GOSUB 1000
30 PMODE 0,2:SCREEN 1,0:PCLS1
40 GOSUB 1000
50 PCOPY 1 TO 2
60 GOSUB 1000
999 STOP
1000 I$=INKEY$:IF I$="" THEN 100
0 ELSE RETURN
```


Naturalmente, podría conseguirse el mismo efecto que en este ejemplo con un sencillo PCLS, pero una vez que tengamos realmente gráficos en las páginas, la orden tiene más utilidad, sobre todo para producir dibujos animados. Ya que están disponibles ocho páginas gráficas, y el PMODE Ø sólo utiliza una cada vez, podríamos generar siete pantallas distintas que contuviesen círculos de tamaño distinto, en las páginas 2 a 8 (fig. 8.1) y copiarlas mediante PCOPY en la página que se está visualizando (1) alternativamente.

```
10 PCLEAR 8:PMODE 0,1:PCLS
20 FOR N=2 TO 8
30 PMODE 0,N:PCLS
40 CIRCLE(128,96),N*10
50 NEXT N
60 PMODE 0,1:SCREEN 1,0
70 FOR N=2 TO 8
80 PCOPY N TO 1
90 NEXT N
100 GOTO 60
```

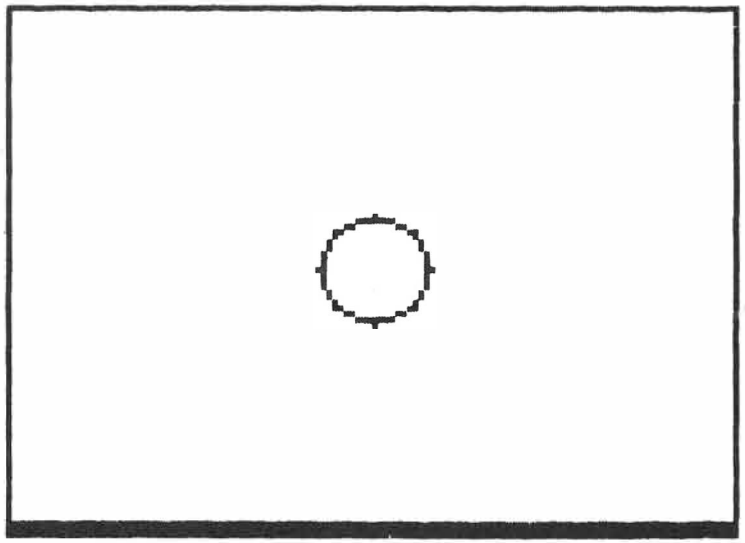
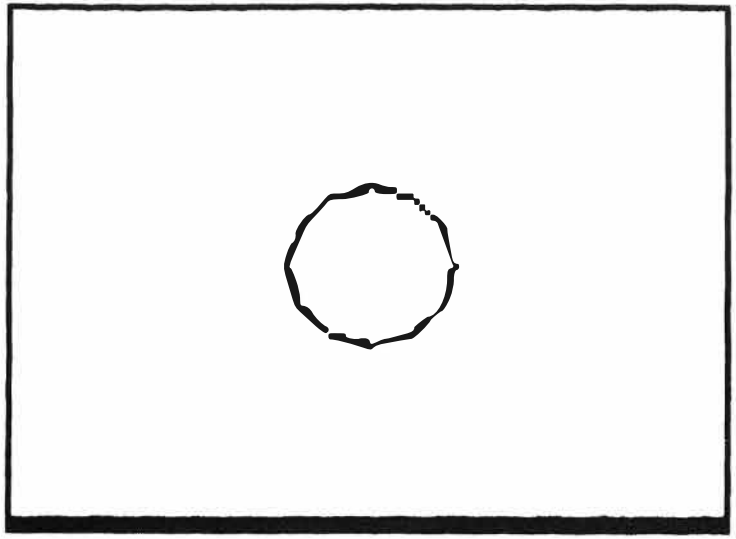


Fig. 8.1



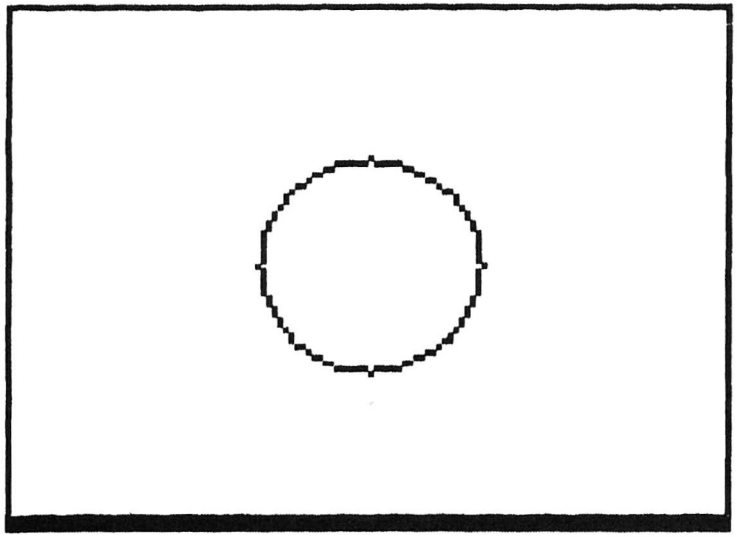
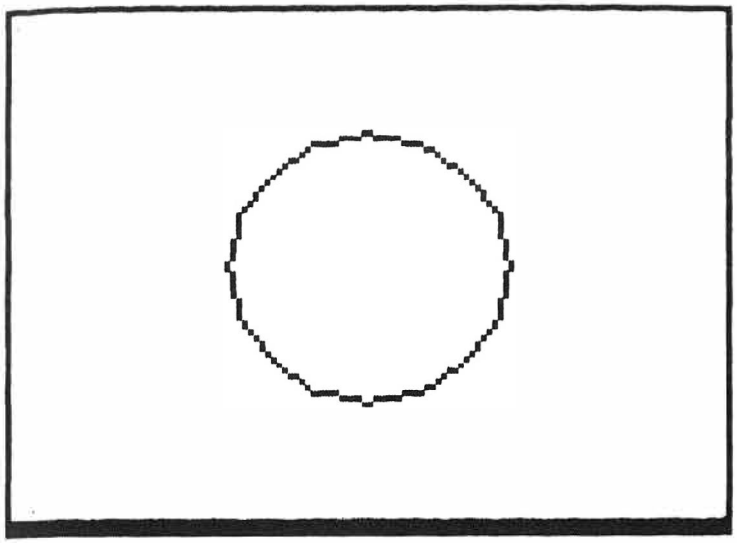


Fig. 8.1



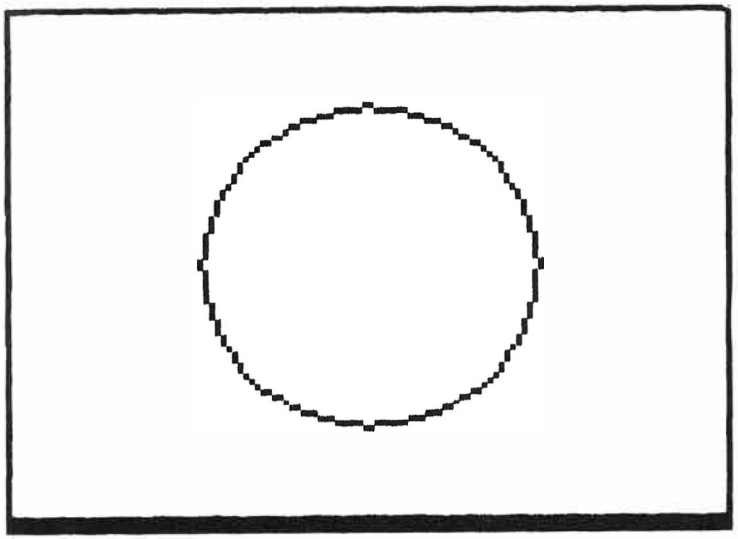
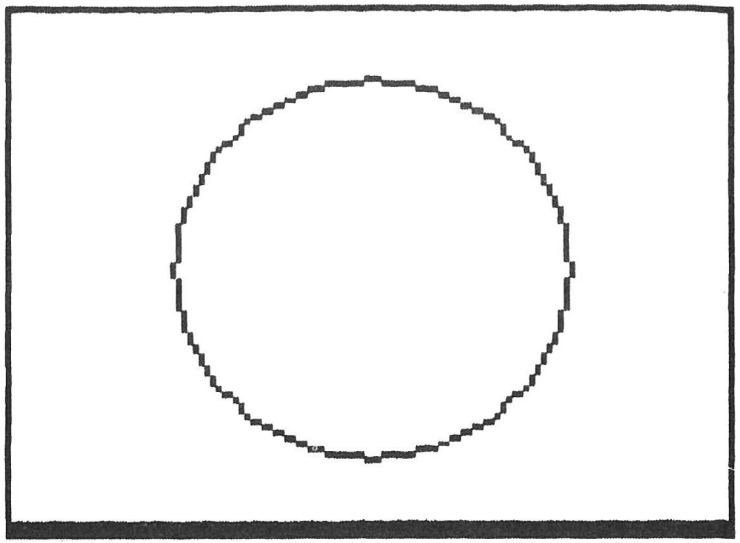


Fig. 8.1



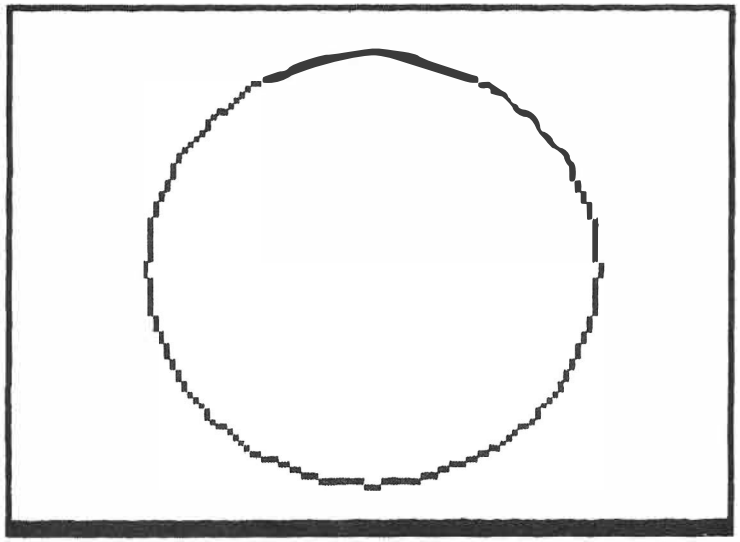


Fig. 8.1

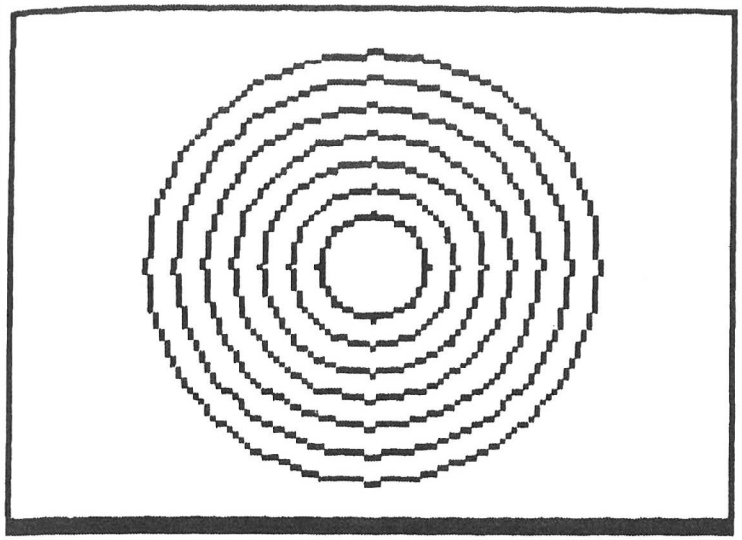


Fig. 8.1

Obsérvese que hay un retraso inicial mientras el programa dibuja por vez primera la serie de círculos en las páginas 2 a 8. Estos círculos no se ven hasta que se copien en la página 1, ya que la única orden SCREEN es la que va a continuación de PMODE Ø,1. El cambiar la visualización muy rápidamente, hace que aparezcan casi todos los círculos en la pantalla al mismo tiempo y se parezca a un remolino.

Un efecto similar al producido por PCOPY, también podría producirse cambiando la página en la orden PMODE y seguirlo cada vez por la orden SCREEN para cambiar la página que se está visualizando.

```
60 FOR N=1 TO 8
70 PMODE 0,N
80 SCREEN 1,0
90 NEXT N
```

En los PMODE más altos se necesitan más páginas para cada pantalla, por lo que cada orden PCOPY copiará únicamente una parte de la pantalla. Esto significa que deberá utilizarse más de una orden PCOPY para cambiar la totalidad de la pantalla, y que podrá guardarse un número menor de dibujos distintos. Si se modifica el programa para utilizar PMODE 1, verá que el número de dibujos alternativos se reduce de 7 a 3. Pero por otra parte se dispone de cuatro colores, y ahora los círculos pueden dibujarse en tres colores distintos.

```
10 PCLEAR 8:PMODE 1,1:PCLS
20 FOR N=3 TO 7 STEP 2
```

```
30 PMODE 1,N:PCLS
40 CIRCLE(128,96),N*10,(N/2)+1
50 NEXT N
60 PMODE 1,1:SCREEN 1,0
70 FOR N=3 TO 7 STEP 2
80 PCOPY N TO 1:PCOPY N+1 TO 2
90 NEXT N
100 GOTO 60
```

Si encuentra difícil observar que los círculos son de colores distintos, puede añadir un bucle de espera que haga el proceso más lento, caso de pulsar una tecla.

```
85 I$=INKEY$:IF I$<>"" THEN FOR
T=1 TO 1000:NEXT T
```

No es necesario copiar la pantalla completa a la vez y si quita el segundo PCOPY, en la línea 8Ø, tan sólo cambiará la parte superior de la pantalla, con lo que producirá un ciclo de semicírculos.

```
80 PCOPY N TO 1
```

Para hacer esto más evidente, cambie el PCLS original en las páginas 1 y 2 por PCLS3.

```
10 PCLEAR 8:PMODE 1,1:PCLS3
```

Cuando se realizan dibujos en un modo y después se visualizan en otro, o tan sólo se copia parte de la pantalla, las cosas pueden volverse confusas. Si cambia a PMODE3, en la línea 6Ø, quizás encuentre difícil explicar el dibujo resultante (fig. 8.2).

```
60 PMODE 3,1:SCREEN 1,0
```

En la parte superior obtendrá una serie de medias elipses de color y parpadeantes, debajo de esto una banda azul y en la parte inferior de la pantalla una elipse amarilla constante. Todo esto es en realidad bastante lógico si lo analiza paso a paso. El PMODE3 utiliza cuatro páginas, y ya que empezamos por la página 1 podemos ver las páginas 1 a 4. La banda azul corresponde a la página 2, que era la mitad inferior de las primeras dos páginas, que se volvieron azules por la acción de PCLS3 de la línea 1Ø. La elipse amarilla de la parte inferior fue dibujada en las páginas 3 y 4 en PMODE1, y esta elipse, y las de las páginas 5 a 8, se van copiando de nuevo en la página 1, en el cuarto superior de la pantalla. Los círculos se han convertido en elipses, ya que cada página tiene ahora la mitad de la altura que tenía en

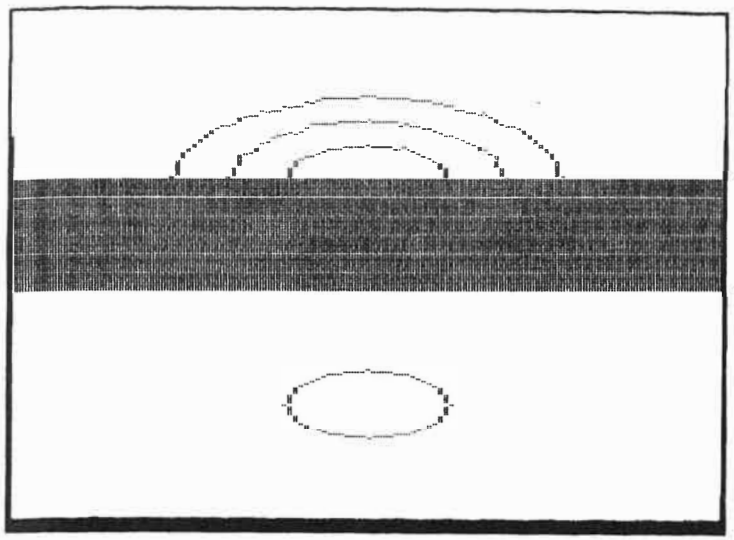


Fig. 8.2 Visualización en PMODE 3.

PMODE1. El copiar únicamente una parte de la pantalla puede utilizarse para producir un efecto «ventana», donde sólo se cambian ciertas secciones de la pantalla.

Como conclusión más importante a recordar acerca de PCOPY es que es muy rápida, pero que únicamente puede cambiar páginas completas a la vez.

Las órdenes GET y PUT

Las órdenes GET y PUT son un par de órdenes complementarias de gran valor, que permiten guardar áreas individuales de la visualización y después reproducirlas en cualquier sitio de la pantalla. Esto, evidentemente, puede ser de gran ayuda cuando se quieran mover dibujos complejos. Sin embargo, no se encuentran en la mayoría de las versiones del BASIC y suelen evitarse por parte del principiante ya que parecen complejas a primera vista y su valor y forma de operación no se explican con claridad en el manual del Dragon.

La orden GET toma un área de la visualización, y guarda el estado de cada pixel contenido en esta área. La orden PUT puede entonces reproducir esta área en cualquier parte de la pantalla. El proceso es parecido a crear una fotocopia de un área y colocarla de nuevo en lugares distintos (obsérvese que el área original no queda afectada

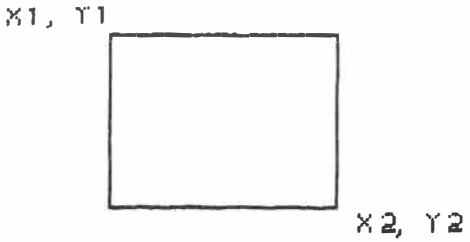


Fig. 8.3 Coordenadas para la orden GET.

por el proceso de copia). El tamaño del área copiada por la orden GET se especifica mediante las coordenadas superior izquierda e inferior derecha del área (fig. 8.3).

Tamaño de la tabla

La información del estado de los pixels que queda copiada por la orden GET debe guardarse en una tabla, y naturalmente deberá DIMensionarse en principio una tabla de tamaño adecuado. El método para calcular el tamaño de la tabla que se describe en el manual del Dragon es muy inexacto y las cifras que producen están muy por encima de las necesarias. Dice que hay que declarar una tabla bidimensional utilizando la altura y la anchura del área en unidades de pixel como dimensiones. De esta forma un área de 5Ø por 5Ø pixels necesitaría una tabla de 25ØØ elementos. Entonces, ya que cada elemento de la tabla necesita 5 bytes, esta tabla ocuparía 12 5ØØ bytes de memoria. Si esto fuese verdad entonces las órdenes GET y PUT serían de una utilización muy limitada, ya que sólo habría espacio suficiente en memoria para copias de áreas bastante pequeñas. Si calculamos el tamaño de tabla necesaria para copiar la totalidad del área de la pantalla que tiene 192 × 256 pixels, el resultado sería de 49 152 elementos, equivalente a la terrible cantidad de un cuarto de megabyte de memoria.

El error de los cálculos anteriores proviene del hecho de que se supone que el estado de cada pixel se guarda como un número en un elemento separado de la tabla. Afortunadamente, esto no es verdad y la información correspondiente a más de un punto está de hecho guardada en cada elemento de la tabla. La idea de una tabla aquí realmente es un poco confusa e improcedente, ya que en este caso el DIMensionar la tabla se utiliza sólo por el sistema para reservar un bloque de memoria en una determinada situación. La forma en que se utiliza entonces esta área no tiene ninguna relación con la manipulación normal de tablas, ya que cada byte es rellenado sencillamente de forma secuencial.

La cantidad de memoria que en realidad se necesita para guardar la información depende del PMODE seleccionado, pero primero vamos a considerar la situación en el modo de resolución más alto, PMODE(4). En este caso existen únicamente dos posibles condiciones para cada punto de la pantalla, es decir, iluminado o apagado. Ya que las situaciones de iluminado o apagado pueden indicarse mediante un único bit, un byte podrá almacenar la condición correspondiente a ocho pixels. De esta manera, en lugar de que cada pixel necesitase cinco bytes, ahora en un byte podrán guardarse ocho pixels, con lo que se ahorra memoria en un factor de 4Ø. Ahora nuestra área de 5Ø×5Ø necesitará únicamente 313 bytes lo que es equivalente a una tabla de 63 elementos (6144 bytes). Ya que los bytes reservados para la tabla se utilizan secuencialmente, no hay ninguna necesidad de utilizar una tabla multidimensional.

El único pequeño problema de hacer la división por un factor de 4Ø de una vez es que sólo pueden utilizarse octetos completos y elementos de la tabla completos, con lo que pueden aparecer pequeños errores debidos al redondeo. Además, también existe una pequeña cantidad para la formación de la tabla. Con áreas pequeñas, el dividir por 4Ø y redondear es siempre efectivo, pero a medida que las áreas son mayores hay que hacer algunas concesiones. La manera más sencilla de asegurarse de que se ha reservado suficiente memoria es el dividir el área en unidades de pixel por 4Ø y después añadir un 10 % a este cálculo para imprevistos. Si surgen problemas, o se quiere utilizar el mínimo absoluto de memoria, intente ejecutar el programa con tablas ligeramente mayores o menores.

El tamaño de las tablas necesarias para otros PMODE puede calcularse de la misma forma. Recuerde siempre que al dividir el número

Tabla 8.1

CALCULO DEL TAMAÑO DE LA TABLA NECESARIA PARA LA ORDEN GET

PMODE	pixels/byte	divisor (aprox.)
0	32	160
1	16	80
2	16	80
3	8	40
4	8	40

de pixels individuales, dividirá también el número de bytes necesarios, pero que un modo con cuatro colores necesita doble número de bytes para codificar el color del pixel (tabla 8.1).

Seleccionemos una pantalla en alta resolución y DIMensionemos una tabla de tamaño apropiado para un área de 5Ø×5Ø. El tamaño de la tabla calculado fue de 63, pero ambos métodos (el del «10 %» y el «probar y corregir») muestran que se necesitan 69 elementos.

```
10 PMODE 4,1:SCREEN 1,0:PCLS
20 DIM A(69)
1000 GOTO 1000
```

GET

En su forma más sencilla, la orden GET tan sólo necesita incluir las coordenadas del punto superior izquierdo e inferior derecho del área «fuente» y el nombre de la tabla a rellenar.

GET(X1,Y1)–(X2,Y2), nombre tabla

```
100 GET(0,0)-(49,49),A
```

Cuando ejecute este programa parecerá que no sucede nada, aunque de hecho se haya realizado una copia de la esquina superior izquierda de la pantalla en la tabla A. Una característica opcional de la orden GET es el sufijo G, que proporciona un almacenaje con un detalle gráfico completo.

GET(X1,Y1)–(X2,Y2), nombre tabla, detalle gráfico

Este parámetro asegura que todas las variaciones de la orden PUT (véase más adelante) funcionarán siempre de manera adecuada, pero hace que las cosas sean más lentas. El tiempo necesario para obtener un área depende también de su tamaño, y la tabla 8.2 da algunos tiempos comparativos para distintos tamaños de área, con el parámetro G y sin él. Como era de suponer, el doblar el tamaño del área dobla el tiempo necesario, pero el incremento de tiempo al añadir la G es del orden de seis veces más, lo que es un valor muy significante. La mayoría de las veces la G no es esencial y el consejo será obviamente el suprimirla siempre que la velocidad sea importante. El tiempo necesario para obtener (GET) la totalidad de la pantalla en PMODE4 es de Ø,6 segundos (–G) y de 4,3 segundos (+G), lo que puede utilizarse para compararlo con la velocidad de PCLS y PCOPY que utilizan Ø,1 segundos para cambiar la misma área.

Hasta ahora sólo hemos obtenido áreas en blanco de la pantalla, por lo tanto vamos a realizar un rectángulo sólido dentro del área para ver lo que sucede.

```
30 LINE(10,10)-(40,40),PSET,BF
```

Tabla 8.2

VELOCIDAD DE LA ORDEN GET CON Y SIN DETALLE GRAFICO COMPLETO
(tiempo en segundos)

	área (pixels)		
	50 x 50	50 x 100	100 x 100
–G	0.04	0.08	0.14
+G	0.24	0.44	0.88

PUT

La forma de la orden PUT es similar a la del GET respecto a sus requerimientos mínimos, que son las coordenadas de extremo superior izquierdo e inferior derecho del área «objetivo», más el nombre de la tabla que contiene la información que debe colocarse.

PUT(X1,Y1)–(X2,Y2),A

```
200 PUT(0,50)-(49,99),A
```

Si ejecuta este programa verá que se dibuja un rectángulo de color y que después este dibujo se repite rápidamente en el área adyacente (fig. 8.4). Es interesante observar que el rectángulo original emplea Ø,22 segundos en ser dibujado, mientras que la totalidad del área se coloca de nuevo en tan sólo Ø,Ø4 segundos. Esto es una sencilla demostración de que puede ser más rápido obtener (GET) y colocar (PUT) un dibujo, que el dibujarlo directamente y esto es una ventaja importante de estas órdenes.

Cuando se utiliza la orden PUT como lo hemos hecho antes, entonces todos los puntos del área objetivo se colocarán en la misma condición que los puntos correspondientes del área fuente, de forma que cualquier cosa que estuviera anteriormente en el área objetivo queda completamente borrado. Hay que tener cuidado en comprobar que el tamaño del área donde se coloca (PUT) sea el mismo que el del

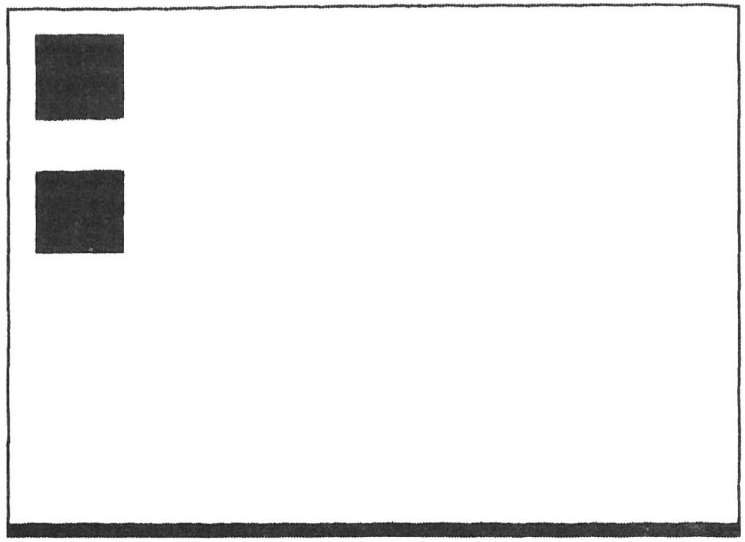


Fig. 8.4 PUT.

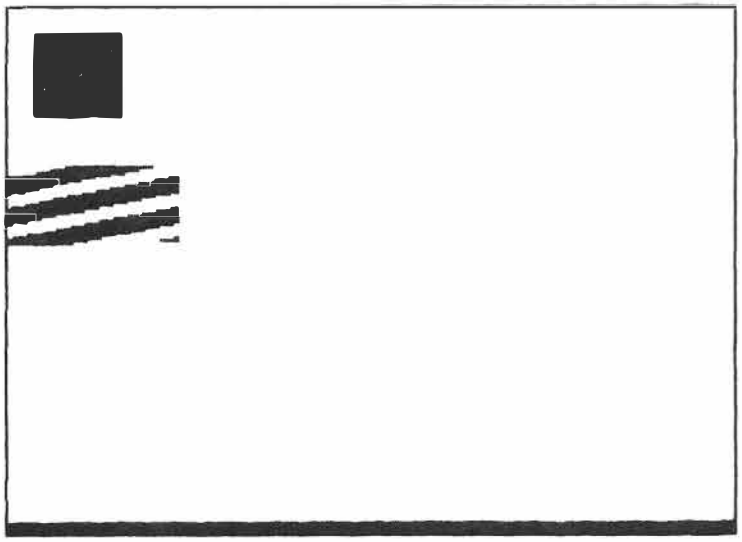


Fig. 8.5 La orden PUT con coordenadas incorrectas.

área de donde se obtiene (GET) o el copiado no se realizará correctamente. La figura 8.5 muestra el efecto de un error en las coordenadas y un efecto similar se producirá si se obtiene con detalle gráfico (G) y se coloca como se ha descrito antes, sin especificar cualquiera de las acciones opcionales.

Acciones

También es posible añadir una serie de acciones opcionales como parámetros, al final de la orden PUT.

PUT(X1,Y1)–(X2,Y2), nombre tabla, acción

Existen cinco opciones distintas posibles, y éstas tan sólo producen resultados correctos si se han guardado con detalle gráfico completo, mediante PUT...G.

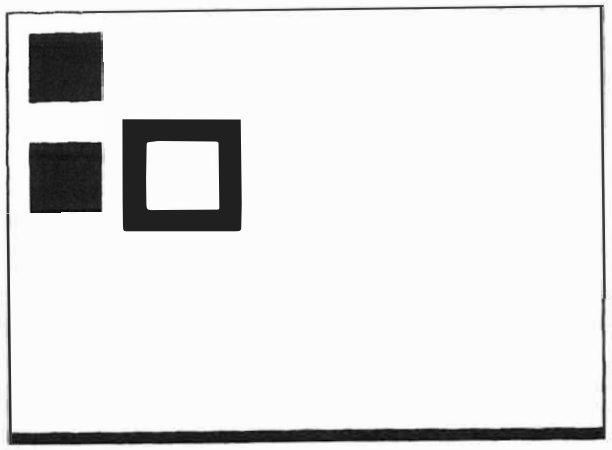
Los dos primeros destruirán cualquier cosa que estuviera en el área objetivo y las sustituirán con una copia del área fuente. Para ver esto en acción modifique la anterior orden GET para que guarde un detalle gráfico completo, añada un PSET al PUT de la línea 2ØØ, y añada a continuación una orden PUT... PRESET.

```
100 GET(0,0)-(49,49),A,G
200 PUT(0,50)-(49,99),A,PSET
210 PUT(50,50)-(99,99),A,PRESET
```

El PSET tiene exactamente el mismo efecto que si no se especificase ninguna acción, produciendo una copia positiva en la cual todos los puntos del área objetivo están colocados de la misma forma que los del área fuente. La opción PRESET es la inversa de PSET y produce el «negativo» del área fuente, iluminando todos los puntos que estaban apagados en la tabla fuente y apagando todos los puntos que estaban iluminados en la tabla fuente (fig. 8.6). Ya que las dos borran el área objeto, no aparece ninguna superposición si se colocan de nuevo copias con coordenadas sobrepuestas (fig. 8.7).

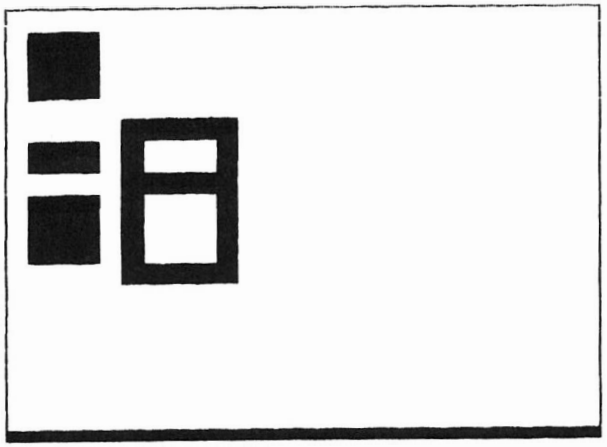
```
220 PUT(0,75)-(49,124),A,PSET
230 PUT(50,75)-(99,124),A,PRESET
```

Observará que la velocidad de operación de la orden PUT ha disminuido marcadamente cuando se le ha especificado una acción. Si comprueba el tiempo, como antes, encontrará que ahora necesita Ø.28 segundos para colocar un área de 5Ø×5Ø, en lugar de los Ø,Ø4 segundos necesarios cuando no se había elegido ninguna opción.



PSET PRESET

Fig. 8.6 PUT.

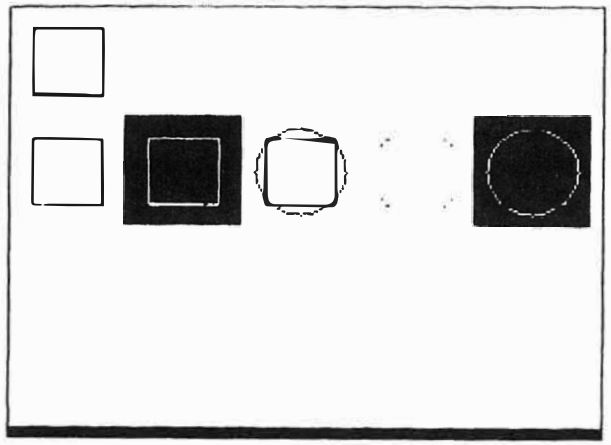


PSET PRESET

Fig. 8.7 La orden PUT con sobreposición.

Las otras tres opciones realizan algún tipo de comparación lógica entre los puntos de las áreas fuente y objeto. Para ver como funcionan, cambiaremos nuestro rectángulo por una versión vacía, añadiremos una serie de círculos a la pantalla, borraremos la línea 23Ø y después colocaremos (PUT) el rectángulo sobre estos círculos con todas las opciones distintas (fig. 8.8).


```
30 LINE(10,10)-(40,40),PSET,B
40 CIRCLE(25,75),20
50 CIRCLE(75,75),20
60 CIRCLE(125,75),20
70 CIRCLE(175,75),20
80 CIRCLE(225,75),20
100 GET(0,0)-(49,49),A,G
200 PUT(0,50)-(49,99),A,PSET
210 PUT(50,50)-(99,99),A,PRESET
220 PUT(100,50)-(149,99),A,OR
240 PUT(150,50)-(199,99),A,AND
250 PUT(200,50)-(249,99),A,NOT
```



PSET PRESET OR AND NOT

Fig. 8.8 Acciones alternativas de la orden PUT.

- PSET tapa por completo el círculo y genera únicamente el rectángulo.
- PRESET da el inverso de PSET.
- OR sobrepone las áreas fuente y objeto y deja iluminados todos los puntos que lo estaban en cualquiera de las dos áreas, dando como resultado el círculo y el rectángulo superpuestos.
- AND apaga todos los puntos que no estaban iluminados en las dos áreas fuente y objeto de forma que tan sólo permanecen iluminadas las cuatro pequeñas áreas de sobreposición entre ambas.

- NOT invierte todos los puntos del área objeto, con independencia de lo que hubiera en el área fuente. Por lo tanto, genera una copia invertida del contenido original del área objeto (el círculo). Obsérvese que el NOT no rellena una tabla, aunque de hecho debe especificarse una para indicar el tamaño del área.

El valor de estas opciones de comparación será más aparente si analizamos algunas aplicaciones.

Movimiento

La forma más sencilla de producir movimiento es utilizar la orden GET sin detalle gráfico y después mediante la orden PUT colocando el dibujo en coordenadas que están indexadas por variables, sin ninguna opción. Si añade esta rutina de teclas de cursor con autorrepetición (líneas 3ØØ-34Ø) y después la ejecuta en PMODE4, descubrirá que las teclas de las flechas hacia arriba y hacia abajo funcionan efectivamente a lo largo del eje Y, pero que el programa se pierde si se intenta mover a izquierda y derecha a lo largo del eje X.

```
10 PMODE 4,1:SCREEN 1,0:PCLS
20 DIM A(69)
30 LINE(10,10)-(40,40),PSET,B
40 XI=1
50 YI=1
100 GET(0,0)-(49,49),A
200 PUT(X+0,Y+50)-(X+49,Y+99),A
300 IF PEEK(337)=255 THEN 300
310 IF PEEK(341)=223 THEN Y=Y-YI
:GOTO 200
320 IF PEEK(342)=223 THEN Y=Y+YI
:GOTO 200
330 IF PEEK(343)=223 THEN X=X-XI
:GOTO 200
340 IF PEEK(344)=223 THEN X=X+XI
:GOTO 200
350 GOTO 200
```

No queda ningún rastro, ya que el step es tan pequeño que el área visible queda borrada a cada movimiento (fig. 8.9). Si se cambian las coordenadas del cuadrado original por las extremas de los límites del área que se está cogiendo, mediante la orden GET, lo que está sucediendo es más obvio (fig. 8.1Ø).

```
30 LINE(0,0)-(49,49),PSET,B
```

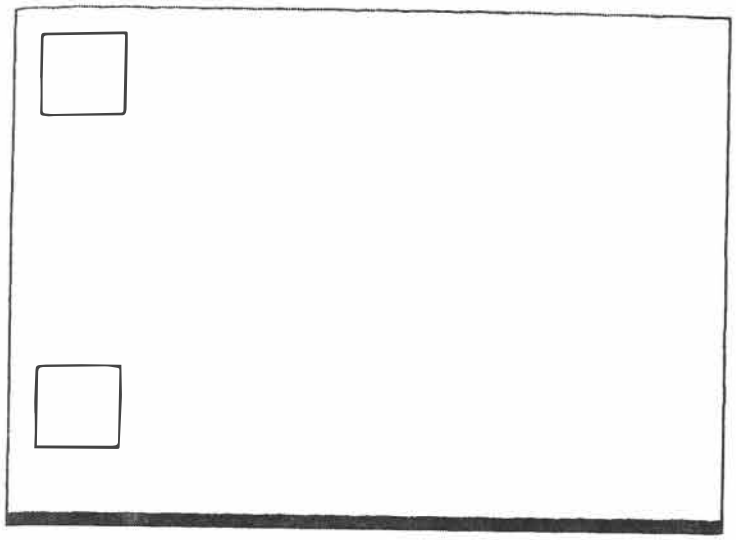



Fig. 8.9 Si el STEP es pequeño, no deja rastro.

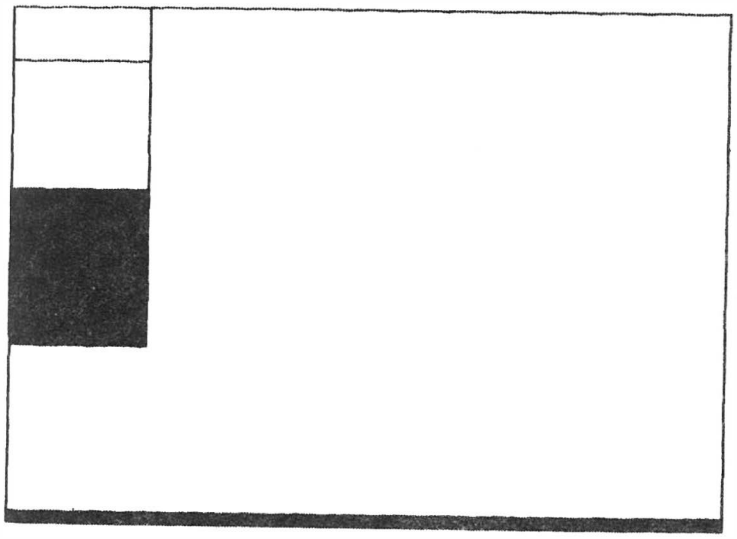


Fig. 8.10 Si no hay margen, deja rastro.

Cuando el incremento de YI es 1, entonces un bloque sólido seguirá el movimiento, pero al aumentar YI (línea 5Ø) irá produciendo una sobreposición cada vez menor, hasta que YI sea mayor que 5Ø y no haya ninguna sobreposición.

Si se utiliza la orden GET con detalle gráfico y a continuación el PUT con PSET, el programa funciona adecuadamente a lo largo de ambos ejes, pero el movimiento es mucho más lento (Ø,28 segundos /ciclo en lugar de Ø,Ø4 segundos).

```
100 GET(0,0)-(49,49),A,G
200 PUT(X+0,Y+50)-(X+49,Y+99),A,
PSET
```

El ejecutar el programa original que no guarda todos los detalles gráficos en los distintos PMODE produce algunos resultados extraños, aunque el movimiento vertical sigue sin ser un problema. El programa no se pierde si se mueve horizontalmente, pero realiza el movimiento a golpes y a distinta velocidad. Una pequeña experiencia revela que en los PMODE 4, 3 y 1 existe la posibilidad del movimiento horizontal rápido, pero relativamente brusco, siempre que se contente con cambiar los incrementos del eje X a incrementos de 8.

```
40 XI=8
```

Por lo tanto, la forma más rápida de actualizar cada movimiento es evitar las opciones, aunque esto dejará siempre un rastro, a menos que haya un espacio en blanco alrededor del área que se obtiene (GET), que sea al menos de la misma longitud que el paso correspondiente a cada movimiento. Las únicas desventajas de la constitución de esta aureola en blanco son que el área que debe moverse debe ser mayor que el tamaño real del dibujo y que dos dibujos no podrán colocarse por completo de lado sin que uno de ellos quede parcialmente borrado (fig. 8.11).

Un planteo alternativo es utilizar una tabla vacía para ir borrando las antiguas posiciones a medida que se van moviendo. La tabla en blanco debe tener el mismo tamaño que la tabla «real», aunque sorprendentemente no hay que utilizar la orden GET para rellenarla. El método más sencillo es el utilizar el PUT con cada tabla a su turno, aunque esto produce una visualización violentamente parpadeante.

```
20 DIM A(69),B(69)
200 PUT(X+0,Y+50)-(X+49,Y+99),A
210 PUT(X+0,Y+50)-(X+49,Y+99),B
```

Un planteo más racional es el borrar tan sólo el área antigua si se realiza un movimiento. Podemos hacer esto colocando la línea encar-

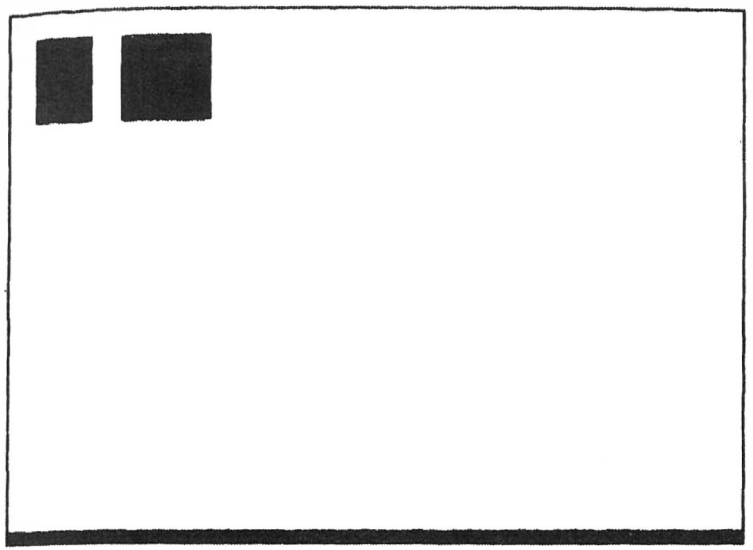


Fig. 8.11 Borrado parcial si las figuras están de lado.

gada de borrar en una subrutina que nada más se llama cuando se realice un movimiento (PEEK(337)<255).

```
210 (borrada)
300 IF PEEK(337)=255 THEN 300 EL
SE GOSUB 500
500 PUT(X+0,Y+50)-(X+49,Y+99),B:
RETURN
```

Una alternativa al planteo de la subrutina es el guardar las antiguas coordenadas y colocar (PUT) una tabla en blanco de nuevo en ellas. Cualquiera que sea el método utilizado, lo más importante que hay que recordar es considerar que el flujo del programa es importante si se quieren evitar órdenes PUT innecesarias y así minimizar el tiempo de ejecución.

Sobreposición

La opción OR es de extremado valor ya que permite aparentar el movimiento de un dibujo encima o a través de otro, sencillamente sobreponiéndolos. Como es lógico, no pueden sobreponerse dos áreas en movimiento utilizando el PUT...OR con las dos tablas, ya que no se borrarían las antiguas posiciones en ninguno de los casos. El método

más sencillo es utilizar el PUT...PSET para la primera y después PUT...OR en la segunda, de forma que el primer PSET borre la totalidad del área.

La velocidad de ejecución disminuye con el tamaño del área, de forma que aunque la sobreposición de áreas pequeñas es casi instantánea, el progreso de la totalidad de la secuencia PUT puede verse con áreas mayores.

Añada la opción PSET a la orden PUT que mueve la tabla A, dibuje un círculo, colóquelo mediante la orden GET en otra tabla (B), y después colóquela de nuevo con OR.

```
20 DIM A(69),B(69)
60 CIRCLE(75,25),20
100 GET(0,0)-(49,49),A,G
110 GET(50,0)-(99,49),B,G
200 PUT(X+0,Y+50)-(X+49,Y+99),A,
PSET
210 PUT(50,50)-(99,99),B,OR
300 IF PEEK(337)=255 THEN 300
```

El cuadrado en movimiento puede colocarse ahora en cualquier posición, encima del círculo estático.

Borrado selectivo

El NOT y el AND no parecen por sí mismos muy interesantes, pero una combinación de NOT y AND puede utilizarse para generar una rutina de borrado selectivo. La secuencia lógica es la siguiente:

1) Colocar el primer dibujo en la tabla A (GET).
2) Invertir el primer dibujo con PUT...A, NOT.
3) Colocar la versión invertida del primer dibujo en una nueva tabla B.
4) Sobreponer el primero y segundo dibujo con PUT...A,OR.
5) Sobreponer la versión invertida del primer dibujo sobre la combinación del primero y segundo dibujo con PUT...B, AND.

```
20 DIM A(69),B(69)
60 CIRCLE(125,75),20
100 GET(0,0)-(49,49),A,G
110 PUT(0,0)-(49,49),A,NOT
120 GET(0,0)-(49,49),B,G
200 PUT(100,50)-(149,99),A,OR
210 PUT(100,50)-(149,99),B,AND
220 GOTO 220
```


El cuadrado aparece sobrepuesto al círculo mediante la OR (fig. 8.12) y entonces queda borrado por completo de una forma selectiva mediante el AND y su inverso, pero únicamente se borran aquellos puntos del círculo que eran comunes con el primer dibujo (fig. 8.13).

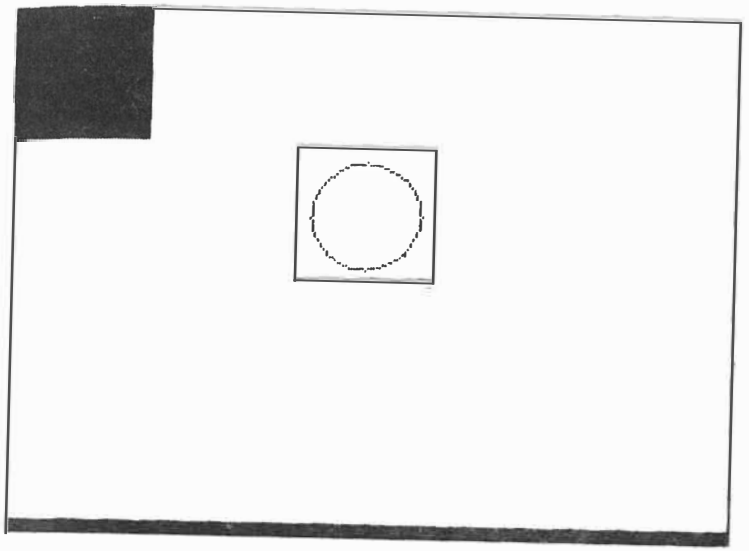


Fig. 8.12 Cuadrado sobre un círculo.

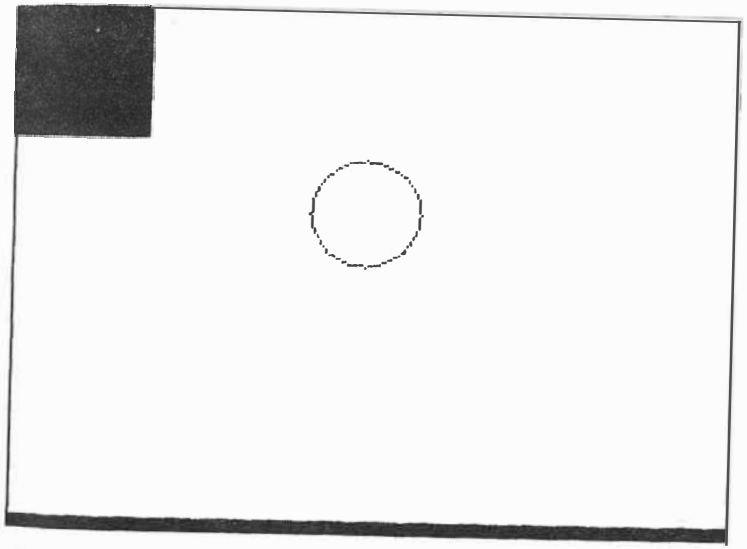


Fig. 8.13 Borrado selectivo del cuadrado.

Como guardar la pantalla en la memoria

A veces puede ser de utilidad el poder guardar el contenido de la pantalla de forma que pueda reproducirse más tarde. Por ejemplo, quizá se quiera guardar un dibujo parcialmente completo en un momento determinado, de forma que puedan hacerse modificaciones posteriores y comprobaciones sin riesgo de resultados desastrosos. La forma más evidente de hacerlo es copiar (PCOPY) la página actual en otra u otras páginas gráficas. El número de páginas completas que pueden guardarse dependen del PMODE. Si se selecciona el número máximo de páginas gráficas (mediante PCLEAR 8) entonces en PMODE 3 y 4 tan sólo puede guardarse una copia, en PMODE 1 y 2 tres copias y en PMODE Ø siete copias.

Naturalmente, pueden guardarse ciertas páginas de la pantalla cuando se utiliza más de una. Aunque no pueden reservarse más de ocho páginas de memoria para gráficos, existe la posibilidad de hacer más copias y guardarlas en otro lugar de la memoria. La forma más sencilla de hacerlo es mediante GET y PUT de la totalidad del área, colocándola en una tabla, como se ha descrito antes. Esto utiliza la misma cantidad de memoria que la página gráfica equivalente, pero esta memoria está en el área de variables y colocada encima del programa en lugar de debajo de él. El utilizar GET y PUT tiene las ventajas de que pueden especificarse acciones distintas y que cualquier parte de la pantalla puede guardarse y reproducirse en cualquier posición, pero la desventaja es que es más lenta que PCOPY.

Copia permanente

Cualquier estado de la pantalla guardado en memoria se perderá cuando se desconecte el ordenador, pero las copias permanentes pueden guardarse fácilmente en cinta como un archivo en código máquina, mediante CSAVEM. Esta orden necesita incluir las direcciones inicial y final del área a copiar, y el número de bytes que van a copiarse (la diferencia entre estas direcciones).

CSAVEM «nombre», (principio), (final), (número de bytes)

Estos valores variarán en función del PMODE que esté funcionando y en función del número de página en que empiece (tabla 8.3).

Por ejemplo, si tenemos un dibujo en PMODE 3 que empieza en la página 1, podrá guardarse en cualquier momento, deteniendo el programa mediante un BREAK y tecleando directamente esta orden:

CSAVEM«DIBUJO»,1536,7679,6144

Tabla 8.3

VALORES PARA LA ORDEN CSAVEM SEGUN EL PMODE

PMODE	INICIO*	FINAL*	NO. DE BYTES
0	1536	3071	1536
1	1536	4607	3072
2	1536	4607	3072
3	1536	7679	6144
4	1536	7679	6144

*sumar 1536 para cada posición inicial de página a partir de la página 1.

CSAVEM también puede incluirse en un programa, de forma que pueden definirse variables al principio y colocarlas en una subrutina.

```
10 N$="DIBUJO":S=1536:F=7679:B=6
144

  ......GOSUB 10000

10000 CSAVEM N$,S,F,B:RETURN
```

Para recuperar el dibujo se carga directamente mediante CLOADM como orden directa o en una línea de programa. CLOADM no tiene que especificar un nombre o alguna dirección y por sí mismo cargará el siguiente archivo que encuentre, de nuevo en su posición original.

CLOADM (cargar el próximo archivo en las páginas originales)

Si se le da un nombre de archivo lo buscará y lo cargará.

CLOADM«dibujo» (carga el archivo llamado «dibujo» en las páginas originales)

La ventaja de colocar CLOADM en una línea de programa es que puede seleccionarse primeramente la pantalla de alta resolución y después ver como se va llenando la pantalla.

```
10 PMODE 3,1:SCREEN 1,0
20 CLOADM
1000 GOTO 1000
```

Si se quiere cargar el archivo de nuevo, pero en páginas distintas, puede especificarse un «desplazamiento». Cada página tiene 1536 bytes de largo, por lo que un desplazamiento de 1536 moverá el dibujo una página hacia arriba.

```
20 CLOADM"DIBUJO",1536
```

Si se ejecuta lo anterior, cargará de nuevo el dibujo en las páginas 2 a 5 en lugar de 1 a 4. Esto funcionará correctamente al principio, pero después su programa desaparecerá cuando cargue código máquina sobre él. No olvide que hay que reservar primero el número suficiente de páginas gráficas, mediante PCLEAR 5 en este caso.

A primera vista, quizás haya pensado que las tablas creadas por la orden GET pueden guardarse como archivos ASCII en la cinta, mediante PRINT−1. Sin embargo, en realidad esto no es posible, ya que la orden GET no rellena la tabla en la forma normal sino que escribe encima de todos los elementos indicadores. En la práctica esto no tiene importanca, ya que la pantalla puede guardarse mediante CSAVEM y después obtener de nuevo la información y guardarla mediante un GET en las tablas, cuando hayan sido cargadas de nuevo mediante CLOADM.

Copia dura

El contenido de la pantalla en alta resolución puede imprimirse en la mayoría de las impresoras con características gráficas, utilizando tan sólo órdenes BASIC. De hecho, todos los dibujos que representan imágenes en pantalla en este libro se produjeron de esta forma. Los detalles precisos del programa necesario variarán de una impresora a otra, ya que los métodos para seleccionar los modos gráficos son muy variables. Por lo tanto, no podemos dar detalles completos, pero explicaremos los principios generales que deben aplicarse al inicializar la impresora.

En primer lugar debe seleccionarse el modo gráfico de la impresora (véase el manual de la impresora). No se puede transferir el contenido de cada byte de la página a la impresora secuencialmente, ya que los bytes están ordenados de izquierda a derecha, pero la impresora funciona mediante segmentos verticales (fig. 8.14). Por lo tanto, deberá traducir las coordenadas de pantalla a secciones verticales. En el byte que se transfiere a la impresora, cada bit que esté a 1 generará un punto impreso y cada bit que esté a Ø producirá un blanco. Afortunadamente, el estado de cada pixel individual de la pantalla puede encontrarse mediante PPOINT(X,Y). La secuencia general de operaciones será por lo tanto leer en la pantalla en secciones verticales y colocar el bit a 1 si PPOINT no es Ø. El poner el bit a 1 puede hacerse añadiendo el número adecuado (1-128) al byte, mediante un test lógico con PPOINT.

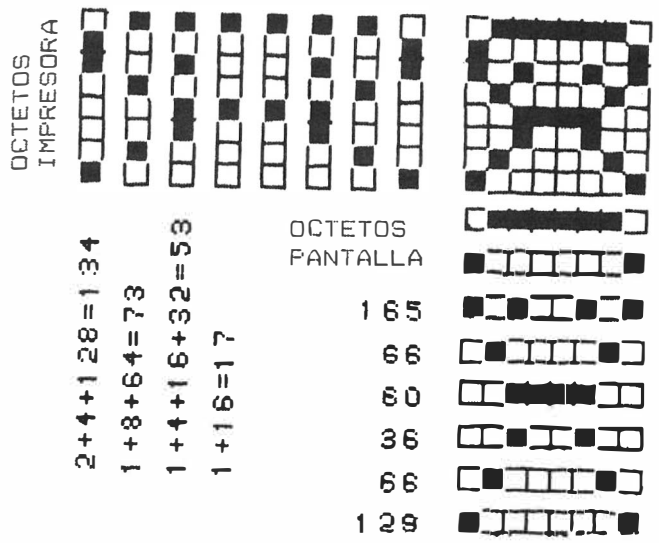


Fig. 8.14 Comparación de los octetos de la pantalla y de la impresora.

La mayoría de las impresoras leen ocho puntos verticales a la vez y esta distribución colocará el primer byte de la impresora, que se toma de la parte superior izquierda de la pantalla.

```
10 X=0
100 Y=0
120 A=PPOINT(X,Y)*1+PPOINT(X,Y+1
)*2+PPOINT(X,Y+2)*4+PPOINT(X,Y+3
)*8+PPOINT(X,Y+4)*16+PPOINT(X,Y+
5)*32+PPOINT(X,Y+6)*64+PPOINT(X,
Y+7)*128
140 PRINT#-2,CHR$(A);
```

El carácter que representa este byte se transfiere al bucle de la impresora mediante PRINT#−2,CHR$(A), pero esto no se imprimirá hasta que se envíe un retorno de carro (CHR$(13)) o el buffer esté lleno. Para imprimir la primera fila completa debemos incrementar X desde Ø a 255, y enviar un CHR$(13) al final.

```
110 FOR X=0 TO 255
150 NEXT X
160 PRINT#-2,CHR$(13);
```

Para movernos hacia abajo en la pantalla deberemos incrementar Y de 8 en 8.

```
100 FOR Y=0 TO 191 STEP 8
170 NEXT Y
```

La Seikosha GP1ØØA es anormal en el hecho de que lee únicamente siete bits a la vez y en que el octavo bit debe estar siempre a 1, por lo que Y debe incrementarse de 7 en 7 y el último bit ponerse siempre a 1.

```
100 FOR Y=0 TO 191 STEP 7
120 A=PPOINT(X,Y)*1+PPOINT(X,Y+1
)*2+PPOINT(X,Y+2)*4+PPOINT(X,Y+3
)*8+PPOINT(X,Y+4)*16+PPOINT(X,Y+
5)*32+PPOINT(X,Y+6)*64+128
```

Si se quiere visualizar una parte de la pantalla nada más pueden colocarse límites adecuados para X e Y.

```
100 FOR Y=20 TO 100
110 FOR X=60 TO 90
```

Esta rutina imprimirá la pantalla tal como está, es decir, que cualquier bit que esté iluminado será impreso. Esto significa que en un modo de dos colores, el verde o el marrón serán impresos y el negro ignorado. En un modo con cuatro colores, el color con el número más alto vendrá indicado por la impresión de los dos bits de un par, y el color de código más bajo estará representado, porque ninguno de los dos bits quedará impreso. Los colores con códigos intermedios producirán que se imprima el bit de la derecha o el de la izquierda, produciendo franjas en forma de cebra inclinadas a derecha o a izquierda (fig. 8.15). A veces puede ser de utilidad que se pueda invertir la visualización y esto puede hacerse mediante un cambio en el test lógico, de forma que se utilice ABS(PPOINT(X,Y+n)−1) en lugar de PPOINT(X,Y+n). También se pueden desarrollar rutinas más complejas para los modos con cuatro colores en las que se clasifican los colores en función del valor obtenido por PPOINT y producen dibujos más distintivos (fig. 8.16).

Fig. 8.15 Bandas en «cebra» generadas cuando el dibujo está creado en PMODE 3 e impreso en PMODE 4.

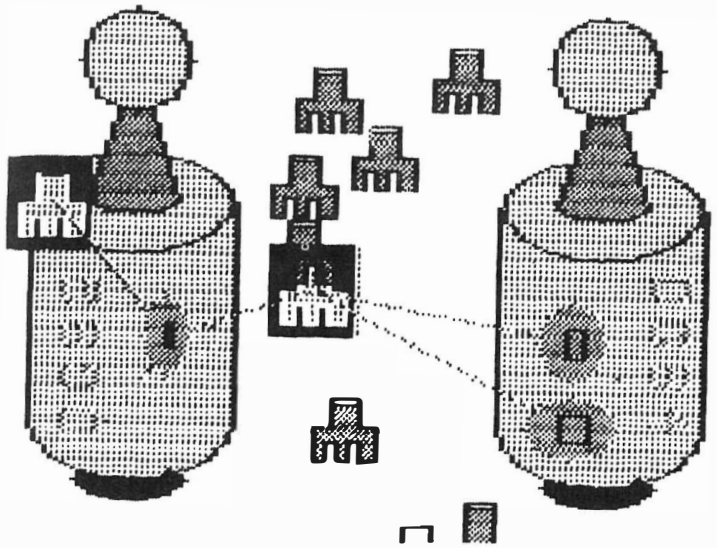


Fig. 8.16 Impresión en «cuatro colores».

9. Presentación gráfica de los datos

Diagramas de barras

Los gráficos en baja resolución tienen un valor limitado para el dibujo de la mayoría de los gráficos, ya que son poco precisos, pero tienen la ventaja de que en el caso de diagramas de barras están fácilmente disponibles los nueve colores (ocho más el negro) y el texto normal. Serán necesarios dos colores para el fondo y los ejes, pero todavía quedan siete colores para indicar cosas distintas en el diagrama.

Para hacer la demostración mediante un diagrama de barras en baja resolución, empezaremos borrando la pantalla y dejándola en fondo verde (CLS1), seleccionaremos el primer color a 2 (amarillo) y formaremos los ejes X e Y, borrando puntos y dejándolos en negro (RESET).

```
10 CLS1:C=2
30 FOR X=30 TO 60:RESET(X,25):NE
XT X
40 FOR Y=4 TO 24:RESET(30,Y):NEX
T Y
```

El eje X puede rotularse mediante una única línea.

```
50 PRINT @ 432,"1 2 3 4 5 6 7"
```

Pero el eje Y requiere un bucle que vaya moviendo la posición de visualización (YP) hacia arriba para cada valor de Y (YV). YV empieza por Ø y se va incrementando en incrementos de 1Ø para cada repetición (396-76) 32/1Ø.

```
70 FOR YP=396 TO 76 STEP-32
80 PRINT @ YP,YV;
90 YV=YV+10:NEXT YP
```

Ahora elegiremos siete valores seudoaleatorios para las distintas barras entre Ø y 1ØØ.

```
110 DIM A(7):FOR N=0 TO 6:A(N)=R
ND(30)+(N*15):NEXT N
```


Los pixels desde el 5 al 24 (2Ø posiciones) deberán representar 1ØØ divisiones del diagrama, por lo que cada bloque será equivalente a cinco unidades. Por lo tanto, los elementos de nuestra tabla deberán convertirse a un número de bloques (BL).

```
170 FOR N=0 TO 6:BL=A(N)/5
200 NEXT N
```

Finalmente, realizamos un bucle a partir del punto del eje X (5) colocando el número requerido de bloques (BL+5) en el punto adecuado de coordenada Y, (29-M), moviendo la coordenada X (N*4+32) a lo largo de cuatro puntos y el color aumentándolo en una unidad (C=C+1) para cada columna completa.

```
180 FOR M=5 TO BL+5:SET(N*4+32,2
9-M,C)
190 NEXT M:C=C+1
210 A$=INKEY$:IF A$="" THEN 210
ELSE RUN
```

Pueden hacerse varias cosas para mejorar la visualización. En primer lugar deberemos darle a la gráfica un título y rotular los ejes.

```
20 PRINT @ 96,"micro-ventas";:PR
INT @ 129,"ilimitadas";
60 PRINT @ 469,"ANYO";
100 PRINT @225,"% BENEFICIO";
```

La lista de números sobre el eje Y es poco uniforme, pero puede formatearse con facilidad mediante PRINT USING «###» y alinearse correctamente.

```
80 PRINT @ YP,"";:PRINT USING"##
#";YV;
```

Por último, el efecto es más interesante si se incluyen algunos bucles de espera para enlentecer los cálculos.

```
185 FOR T=1 TO 100:NEXT T
195 FOR T=1 TO 100:NEXT T
```

Gráficos de línea

Los gráicos de líneas pueden tratarse bien mediante los gráficos en alta resolución (fig. 9.1). Primero tendremos que establecer algu-

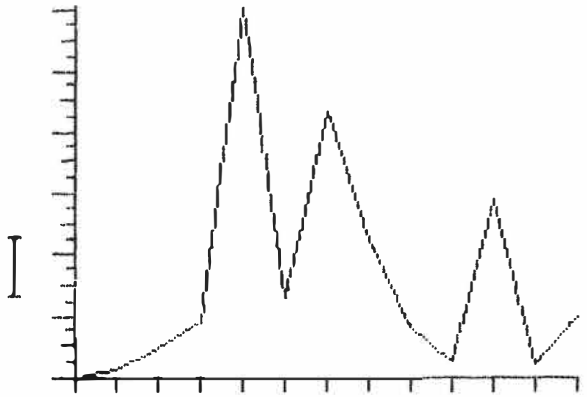


Fig. 9.1 Gráfico de líneas.

nos límites para las coordenadas de pantalla correspondientes al principio y final de los ejes (XP, XF, YP, YF).

```
10 XP=40:XF=220:YP=160:YF=20
```

El tamaño de las divisiones del eje X (XI) depende del número de lecturas que deben incluirse. En este ejemplo generaremos un número aleatorio de lecturas, que como máximo será de cinco, pero es evidente que esto deberá ser entrado normalmente por el usuario mediante INPUT.

```
20 NR=RND(20)+5
30 XI=(XF-XP)/NR
```

Los datos que hay que trazar deben entrarse en una tabla, y en este caso lo haremos de forma aleatoria.

```
40 DIM A(NR)
50 FOR N=1 TO NR
60 A(N)=RND(100)*RND(9)
70 NEXT N
```

Se incluye una comprobación a través de la tabla para buscar el valor más alto y entonces este valor se utiliza para calcular la escala en incrementos del 10 % del valor máximo.

```
80 FOR N=1 TO NR
90 IF A(N)>HI THEN HI=A(N)
100 NEXT N
110 FE=1
120 NH=HI*FE:IF NH>(YP-YF) THEN
FE=FE-0.01:GOTO 120
```


Las divisiones del eje Y se fijan a continuación a cincuenta veces el factor de escala actual (FE).

```
130 YI=50*FE
```

Ahora que los datos ya están preparados, se selecciona la pantalla en alta resolución, en blanco sobre negro, mediante COLOR Ø,1, y se dibujan los ejes X e Y.

```
140 PMODE 4,1:SCREEN 1,0:PCLS1:C
OLOR 0,1
150 LINE(XP,YP)-(XP,YF),PSET
160 LINE(XP,YP)-(XF,YP),PSET
```

Las marcas de la escala se colocan sobre el eje X como una serie de líneas cortas, separadas XI unidades.

```
170 FOR N=XP TO XF STEP XI
180 LINE(N,YP)-(N,YP+5),PSET
190 NEXT N
```

Las señales sobre el eje Y son más complejas ya que utilizan tres longitudes distintas de línea para indicar cada cuarto de división principal. Se colocan a distancias crecientes mediante incrementos del tamaño del STEP en factores de 2. El primer tipo de señal es el más frecuente y tiene 3 puntos de largo, el segundo tipo tiene 5 puntos de largo, y el último 8 puntos de largo. Las señales se sobreponen una a otra pero este método es rápido y mucho más sencillo que la alternativa de un proceso de clasificación.

```
200 FOR N=YP TO YF STEP-YI/2
210 LINE(XP,N)-(XP-3,N),PSET
220 NEXT N
230 FOR N=YP TO YF STEP-YI
240 LINE(XP,N)-(XP-5,N),PSET
250 NEXT N
260 FOR N=YP TO YF STEP-YI*2
270 LINE(XP,N)-(XP-8,N),PSET
280 NEXT N
```

Ya que no se ha incluido ningún texto en este programa, será de utilidad el tener una línea de escala que nos dé una indicación visual del factor de escala utilizado. Esto se forma haciendo un movimiento en blanco hacia la izquierda del eje Y y dibujando (DRAW) una marca de escala allí, con un factor de escala (S) que sea cuarenta veces el factor de escala aplicado a los datos.

```
290 DRAW"BM"+STR$(XP-20)+","+STR
$(YP-20)+"S"+STR$(INT(FE*40))+"B
M+0,-5L2RU10LR2S4"
```

Ahora tendremos que hacer un movimiento en blanco hasta las coordenadas iniciales Ø,Ø suponiendo que el gráfico empieza en el origen (en caso contrario tendrá que hacerse un movimiento en blanco hasta la primera posición X y la primera posición Y del primer elemento de la tabla.

```
300 DRAW"BM"+STR$(XP)+","+STR$(Y
P)
```

Las líneas de la gráfica se construyen a continuación dibujando una línea hasta las coordenadas definidas por la siguiente posición X, Y calculadas a partir del contenido del elemento de la tabla correspondiente, multiplicado por el factor de escala.

```
310 FOR N=XP TO XF STEP XI
320 DRAW"M"+STR$(INT(N))+","+STR
$(INT(YP-(A((N-XP)/XI)*FE)))
330 NEXT N
340 RUN
```

Si añade la línea final y ejecuta el programa verá que van apareciendo continuamente una serie de gráficos generados en forma aleatoria.

Mapas de contorno

Un desarrollo del gráfico de líneas es un mapa topográfico (fig. 9.2) que une los puntos que tienen el mismo valor. Este valor generalmente es la altura (el ejemplo es la vista desde nuestra ventana) pero también podrían ser líneas isobaras en un gráfico del tiempo. Primero seleccionamos la pantalla y dibujamos un cuadrado alrededor del mapa.

```
10 PMODE 4,1:SCREEN 1,0:PCLS
20 LINE(10,10)-(140,160),PSET,B
```

El secreto para hacer aquí un programa sencillo es entrar las coordenadas de la información de forma adecuada. Los datos se toman a partir del mapa para pares de coordenadas X e Y, que dan la posición del siguiente punto con el mismo valor, es decir, el siguiente punto que hay que dibujar (fig. 9.3). Un par de ceros indican que una

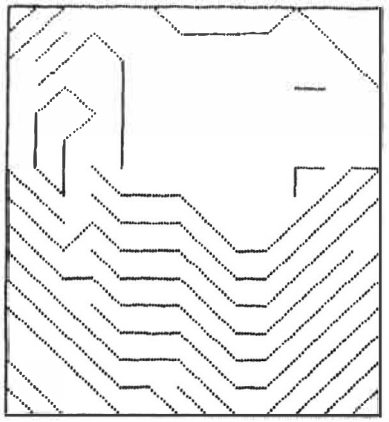


Fig. 9.2 Mapa de contorno.

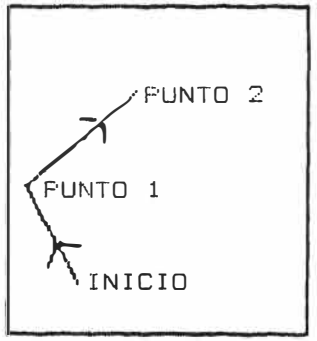


Fig. 9.3 Conexión de puntos en un mapa de contorno.

línea determinada ha terminado y que las siguientes coordenadas definirán el principio de una nueva línea.

```
110 DATA 1,1,1,1,0,0,1,2,2,1,0,0
,1,4,1,3,2,2,3,1,0,0,1,5,2,4,3,3
,4,2,5,1,6,1,0,0,1,6,2,5,3,4,4,3
,5,2,6,2,7,1,0,0,7,2,8,1,0,0,1,7
,2,6,3,5,4,4,5,3,6,3,7,3,8,2,9,1
,10,1,11,2,12,3,13,4,14,5,0,0,11
,1,12,2,13,3,14,4,0,0,12,1,13,2,
14,3,0,0,13,1,14,2,0,0
```

```
120 DATA 14,1,0,0,4,5,5,4,6,4,7,
4,8,3,9,2,10,2,11,3,12,4,13,5,14
,6,0,0,1,8,2,7,3,6,4,6,5,5,6,5,7
,5,8,4,9,3,10,3,11,4,12,5,13,6,1
4,7,0,0,1,9,2,8,3,7,4,8,5,7,6,7,
7,7,8,6,9,5,10,5,11,6,12,7,13,8,
14,9,0,0,1,10,2,9,3,8,0,0,4,9,5,
8,6,8,7,8,8,7,9,6,10,6,11,7
130 DATA 12,8,13,9,14,10,0,0,4,1
0,5,9,6,9,7,9,8,8,9,7,10,7,11,8,
12,9,13,10,14,10,14,11,14,12,14,
13,13,14,12,15,11,16,10,15,9,15,
8,15,7,15,6,16,5,16,4,16,3,16,2,
15,1,14,1,13,0,0,4,7,5,6,6,6,7,6
,8,5,9,4,10,4,11,5,12,6,13,7,0,0
,3,9,2,10,2,11,2,12,3,13
140 DATA 4,12,3,11,3,10,3,9,0,0,
1,11,1,12,0,0,1,15,2,16,0,0,1,16
,0,0,7,16,8,16,0,0,9,16,10,16,0,
0,12,16,0,0,14,16,0,0,14,15,0,0,
14,14,0,0,2,14,3,15,0,0,2,13,3,1
4,4,15,5,14,5,13,5,12,5,11,5,10,
0,0,11,9,11,10,12,10,0,0,11,13,1
2,13,0,0,1,16
```

Primero se lee la posición actual (READ), que hace un movimiento en blanco hasta este punto, y el puntero se coloca de nuevo al principio de los datos (RESTORE).

```
30 READ X,Y:DRAW"BM"+STR$(X*10)+
","+STR$(170-(Y*10)):RESTORE
```

A continuación los 175 pares de puntos se leen por turno. Si X e Y no son cero entonces se hace un movimiento (se dibuja una línea) hasta las coordenadas calculadas por X e Y multiplicados por 1Ø, y a continuación se leen los siguientes puntos (READ). Ya que se ha reinicializado el puntero de datos (RESTORE), la primera línea tendrá una longitud cero ya que va del primer punto a sí mismo. Si se lee un cero entonces se leerán los siguientes valores de X e Y, y se hará un movimiento en blanco hasta estas coordenadas, y a continuación se leerán los siguientes valores.

```
40 FOR N=1 TO 175
50 READ X,Y
60 IF X=0 THEN READ X,Y:DRAW"BM"
+STR$(X*10)+","+STR$(170-(Y*10))
:NEXT N:GOTO 90
70 DRAW"M"+STR$(X*10)+","+STR$(1
70-(Y*10))
80 NEXT N
90 GOTO 90
```


Diagramas de sectores

Los diagramas circulares de sectores, en los cuales el tamaño de los sectores es la indicación de la cantidad, pueden generarse fácilmente mediante la alta resolución. En primer lugar tendremos que seleccionar un modo de alta resolución con cuatro colores adecuado, inicializar una tabla para guardar nuestros valores, y colocar el primer color a 1. Los diagramas de sectores por lo general están divididos en un número bastante pequeño de sectores por lo que tomaremos una serie de siete números aleatorios. Obsérvese que el total (T) también debe calcularse.

```
10 PMODE 1,1:SCREEN 1,0:PCLS
20 DIM SE(7):C=1
30 FOR N=1 TO 7:SE(N)=RND(10):T=
T+SE(N):NEXT N
```

Ahora podemos dibujar una línea circular, y después una serie de arcos de radio creciente y distinta longitud para indicar los sectores (fig. 9.4). Tal como se explicaba antes, los arcos no quedarán llenos

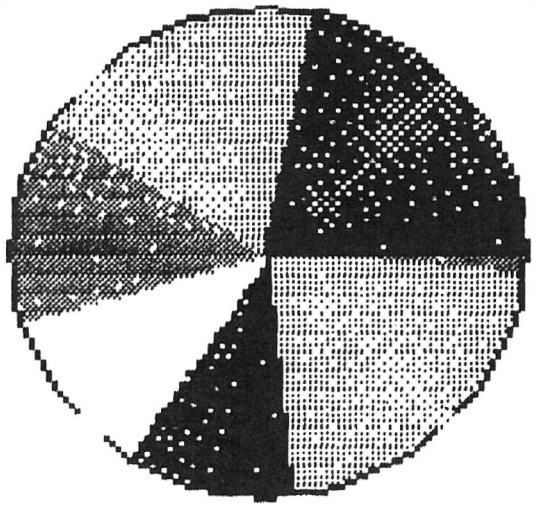


Fig. 9.4 Diagrama de sectores con siete sectores, que empiezan a partir de cero (90°).

por completo, pero siguen siendo bastante efectivos y muy sencillos de construir. El rellenado es más completo en PMODE 1 que en PMODE 3.

```
40 CIRCLE(128,96),90
50 FOR S=1 TO 7
60 FI=PR+(SE(S)/T)
70 FOR R=1 TO 88
80 CIRCLE(128,96),R,C,1,PR,FI
90 NEXT R
100 PR=FI:C=C+1:IF C>4 THEN C=1
110 NEXT S
120 A$=INKEY$:IF A$="" THEN 120
ELSE RUN
```

El valor inicial por defecto (PR) será Ø, por lo que el primer arco se moverá en sentido directo a partir de las 3 en punto. El final (FI) se calcula en unidades apropiadas dividiendo el valor del sector en la tabla (SE(S)) por el total (T) y añadiendo esto al valor inicial. Después de que se haya trazado cada arco, el nuevo valor inicial (PR) se coloca en el antiguo punto final (FI), y el color se incrementa en una unidad. Si el color es mayor que 4 se reinicializa a 1. El arco más largo es ligeramente inferior (R=88) que el radio del círculo (R=9Ø) de forma que el sector formado con el color de fondo no borre la línea exterior.

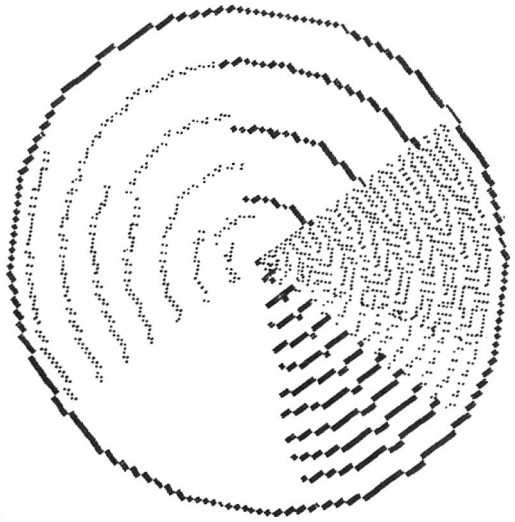


Fig. 9.5 Diagrama de sectores con siete sectores que empiezan a partir de Ø.75, con el tamaño del step relacionado con el número del sector.

Si se quiere que los sectores empiecen a partir de las 12 en punto deberá inicializarse PR a Ø.75 al principio (si se generan valores mayores que 1 la parte entera es ignorada).

```
20 DIM SE(7):C=1:PR=0.75
```

Ya que sólo se dispone de cuatro colores, a primera vista parece difícil el indicar más de cuatro sectores distintos, pero una forma de asegurar la diferencia entre sectores del mismo color es el cambiar el tamaño de step de los arcos si deben dibujarse más de cuatro sectores y también es de utilidad el relacionar el tamaño del step con el número del color (fig. 9.5).

```
20 DIM SE(7):C=1:SP=2:PR=0.75
60 FI=PR+(SE(S)/T):IF S>4 THEN S
P=6
70 FOR R=1 TO 88 STEP SP*C
```

10. Tres dimensiones

La presentación de una vista tridimensional de un objeto es una forma muy efectiva de hacer que parezca más sólido. El aspecto importante acerca de la representación tridimensional es que líneas que se suponen que van hacia lo lejos deberán dibujarse más pequeñas. Por ejemplo, un programa con un bucle que dibuje una serie de cuadrados cada vez mayores y que se desplazan ligeramente cada vez, da la impresión de un cono cuadrado (fig. 10.1).

```
10 PMODE 4,1:SCREEN 1,0:PCLS
20 YO=0:XO=5:R=5
30 FOR I=10 TO 100 STEP 2
40 LINE(XO,YO)-(XO+R,YO+R),PSET,
B
50 YO=YO+2:XO=XO+2:R=R+2
60 NEXT I
520 A$=INKEY$:IF A$="" THEN 520
530 RUN
```

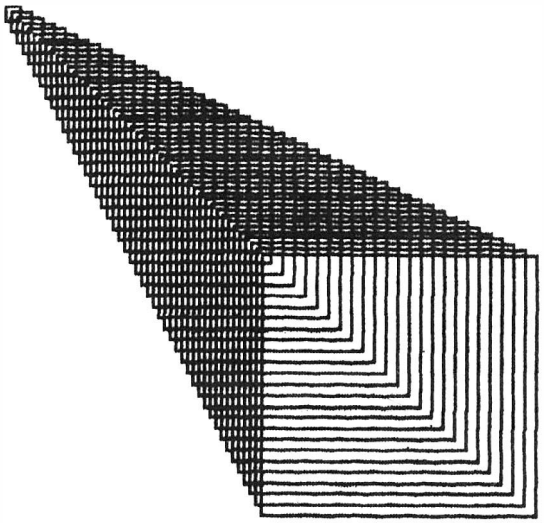


Fig. 10.1 Sección cuadrada en tres dimensiones.

Un efecto incluso más real de un tubo se produce dibujando una serie de círculos desplazados de radio creciente (fig. 10.2).

```
20 YO=50:R=30
30 FOR XO=50 TO 120 STEP 2
40 CIRCLE(XO,YO),R
50 YO=YO+1:R=R+1
```

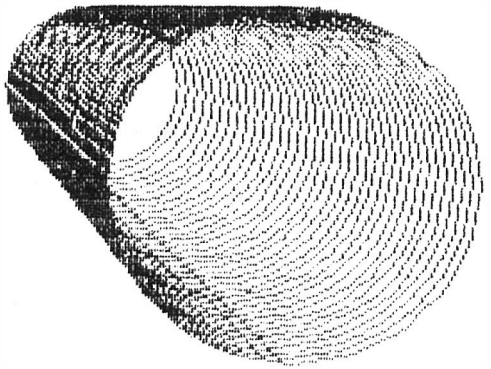


Fig. 10.2 Tubo.

Si establecemos ciertos límites para los ejes X e Y (XP=X inicial, XF=X final, YP=Y inicial, YF=Y final) podemos dibujar un rectángulo mediante una serie de líneas que conecten estos puntos (fig. 10.3).

```
40 XP=100:XF=250
60 YP=160:YF=120
210 PMODE 4,1:SCREEN 1,0:PCLS
230 LINE(XP,YP)-(XF,YP),PSET
240 LINE(XP,YF)-(XF,YF),PSET
260 LINE(XP,YP)-(XP,YF),PSET
270 LINE(XF,YP)-(XF,YF),PSET
```

Fig. 10.3 Líneas conectadas en forma de rectángulo.

Fig. 10.4 Desplazamiento para indicar profundidad.

Para conseguir que esto aparezca como una superficie plana en tres dimensiones deberemos desplazar un lado, cierta distancia) (DI) (fig. 10.4).

```
80 DI=80
240 LINE(XP-DI,YF)-(XF-DI,YF),PS
ET
260 LINE(XP,YP)-(XP-DI,YF),PSET
270 LINE(XF,YP)-(XF-DI,YF),PSET
```

Aunque esto parece más plano, sigue siendo algo incorrecto ya que el lado de atrás tiene la misma longitud que el lado de delante. Se trata de la línea que va desde (XP−DI, YF) hasta (XF−DI, YF) (línea 24Ø) la que necesita acortarse en un extremo y un poco de experimentación nos revelará el valor correcto para este factor de perspectiva (PF). No se olvide que la línea 27Ø también debe modificarse.

```
80 DI=80:PF=25
240 LINE(XP-DI,YF)-(XF-DI-PF,YF)
,PSET
270 LINE(XF,YP)-(XF-DI-PF,YF),PS
ET
```

Si se quiere generar un gráfico tridimensional, puede prescindirse de obtener el factor exactamente correcto de perspectiva, sobre todo si se suprimen las líneas horizontales superiores y las verticales derechas; por lo tanto borre el factor de perspectiva en las líneas 24Ø y 27Ø (fig. 10.5). Necesitaremos algunos datos para dibujarlos, por lo que vamos a generarlos de forma aleatoria.

```
100 DIM A(15,15)
110 FOR N=0 TO 15
120 FOR M=0 TO 15
130 A(N,M)=INT(RND(N*3)+20+RND(M
*3))
140 NEXT M
150 NEXT N
```


Fig. 10.5 Ejes X e Y.

Esto produce 225 números en una tabla bidimensional de 15×15 que utilizaremos para indicar altura. Para generar un gráfico en tres dimensiones que una estos puntos necesitaremos definir las divisiones en los ejes X (XI) e Y (YI), y calcular los incrementos de las coordenadas. El bucle de la X está fuera del bucle Y, por lo que nos moveremos primero de delante hacia atrás.

```
20 XI=20:YI=10
290 FOR N=XP TO XF STEP XI
330 FOR M=YP TO YF STEP-YI
```

El step para YI es negativo ya que queremos trabajar de delante hacia atrás. El elemento de la tabla del eje X correcto (PX) se selecciona dividiendo el valor actual de N menos la posición inicial por el tamaño de las divisiones.

```
300 PX=(N-XP)/XI
```

A continuación deberemos hacer un movimiento en blanco hasta la posición inicial, calcular el siguiente elemento de la tabla Y y mover (dibujar una línea) hasta el punto definido por este elemento (fig. 1Ø.6).

```
310 DRAW"BM"+STR$(N)+","+STR$(YP
)
320 D=0
340 PY=(YP-M)/YI
350 DRAW"M"+STR$(N-D)+","+STR$(M
-A(PX,PY))
```

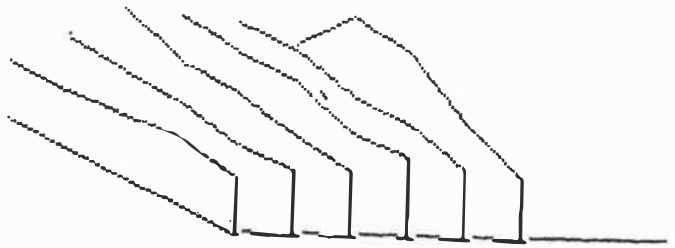


Fig. 10.6 Movimientos de delante hacia atrás.

El elemento del eje X debe tener en cuenta el desplazamiento (D) a la izquierda que debe hacerse. Para el primer punto será Ø, para los puntos siguientes deberá calcularse a partir de YI.

```
370 D=D+(YI*2)
380 NEXT M
390 NEXT N
```

Al final de la primera línea de delante a atrás, la posición X se incrementa y se dibuja la siguiente línea. Las líneas de izquierda a derecha se dibujan de forma similar (fig. 1Ø.7).

```
410 D=0
420 FOR M=YP TO YF STEP-YI
430 PY=(YP-M)/YI
440 DRAW"BM"+STR$(XP-D)+","+STR$
(M)
450 FOR N=XP TO XF STEP XI
460 PX=(N-XP)/XI
470 DRAW"M"+STR$(N-D)+","+STR$(M
-A(PX,PY))
490 NEXT N
500 D=D+(YI*2)
510 NEXT M
```

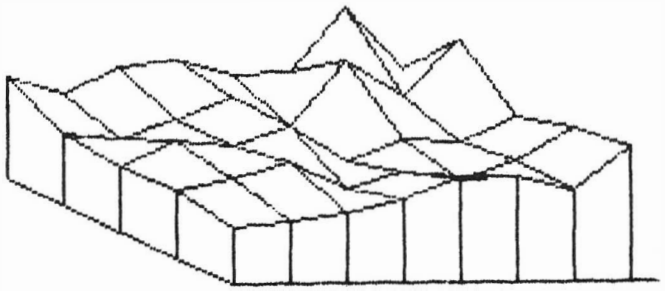


Fig. 10.7 Movimientos de izquierda a derecha.

Si se desea incluir líneas verticales para indicar la altura, pueden añadirse estas líneas que dibujan (DRAW) con N (sin actualización) de forma que se recuerde la última posición (fig. 1Ø.8).

```
360 DRAW"NM"+STR$(N-D)+","+STR$(
M)
480 DRAW"NM"+STR$(N-D)+","+STR$(
M)
```

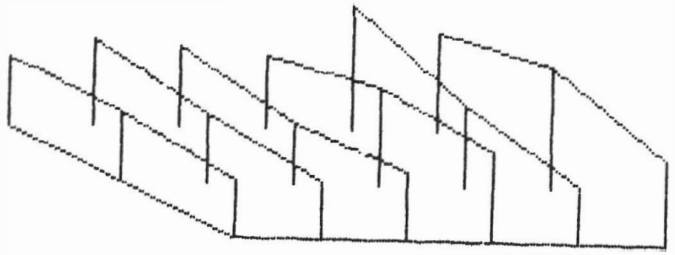



Fig. 10.8 Inclusión de líneas verticales.

También se puede dibujar en tres dimensiones sin mostrar nunca los ejes (fig. 1Ø.9).

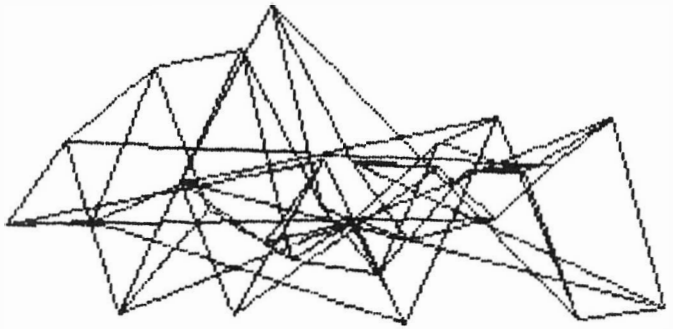


Fig. 10.9 Dibujo con omisión de los ejes.

11. Rotación de figuras

Utilización de órdenes de dibujo en ángulo

La forma más sencilla de rotación se produce directamente mediante la orden del BASIC DRAW en la que se puede especificar el ángulo del Ø al 3, para obtener cuatro copias giradas 9Ø grados. En efecto, esto significa que a cada giro lo que está arriba se lea a la derecha, lo de la derecha abajo, lo de abajo a la izquierda y lo de la izquierda arriba, etc. A primera vista quizá pueda pensar que esto podría conseguirse directamente con los parámetros para dibujar en diagonal, mediante la orden DRAW (E,F,G y H) para obtener las posiciones intermedias, pero no es tan sencillo (fig. 11.1). Esta figura nos muestra dos versiones del mismo dibujo que se diferencian en un ángulo de 45 grados y si analiza cuidadosamente las figuras verá que el número de pixels necesarios para hacer cada una de las secciones del mismo dibujo es en realidad distinto en los dos casos. Esto se debe a que todas estas órdenes mueven un número absoluto de unidades de pixel, pero la hipotenusa de un triángulo isósceles es en realidad casi una vez y media mayor que los otros dos lados. Por lo tanto, tres unidades arriba, abajo, izquierda o derecha representan la misma distancia sobre la pantalla que dos unidades de E, F, G o H. En

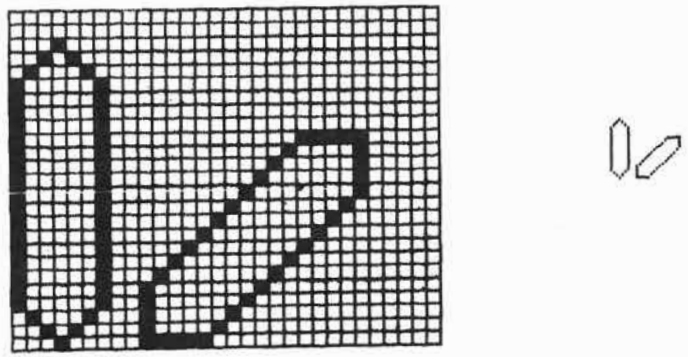


Fig. 11.1 Dibujo con ángulo, mediante la orden DRAW.

las figuras 11.2 y 11.3 se ha dibujado un círculo, con un radio de longitud igual a la del dibujo, alrededor de las posibles figuras realizadas con números iguales de unidades de pixels y con el número de unidades corregida por la diferencia en dirección.

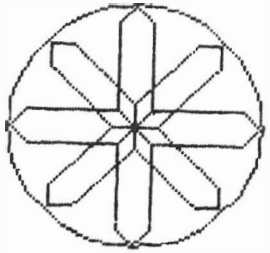


Fig. 11.2 Números iguales.

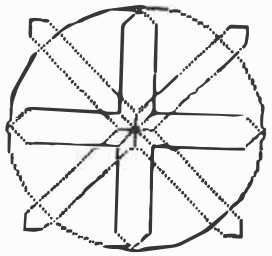


Fig. 11.3 Corregido.

Este programa gira las dos figuras alternadas a través de todas las posiciones posibles, y da el efecto de movimiento al dibujarlas en color principal y después en color de fondo.

```
10 PMODE 4,1:SCREEN 1,0:PCLS
20 A$="H3U16E3F3D16G3"
30 B$="U4E11R4D4G11L4"
40 FOR A=0 TO 3
50 FOR C=1 TO 0 STEP-1
60 DRAW"A"+STR$(A)+"C"+STR$(C)+A
$
70 NEXT C
80 FOR C=1 TO 0 STEP-1
90 DRAW"C"+STR$(C)+B$
100 NEXT C
110 NEXT A
120 GOTO 40
```

Utilizando matemáticas

Para los matemáticos todo es posible (suelen decir), incluso aunque no podamos comprender el cómo y el porqué. Si realmente está interesado en producir rotaciones complejas entonces tendrá que refrescar sus conocimientos de trigonometría y matrices y también encontrar un buen libro sobre el tema. Sin embargo, como introducción veremos cómo gira una figura en dos dimensiones, ya que es bastante fácil. En primer lugar debe inicializarse la pantalla y definir el punto sobre el que va a girarse (XP, YP). Se genera una pequeña cruz para señalar esta posición.

```
30 PMODE 4,1:SCREEN 1,0:PCLS
40 XP=128:YP=96
90 DRAW"BM"+STR$(XP)+","+STR$(YP
)+"U4D2L2R4"
```

Ahora necesitaremos algo que girar, por lo que vamos a formar un triángulo simétrico que señale hacia arriba. Esto se hace conectando cuatro puntos que están definidos por el número de puntos de pantalla que los separan del centro de rotación a lo largo de los ejes X (P1(N)) e Y (P2(N)), siendo definida la dirección a la izquierda y hacia arriba con valores negativos. Colocaremos estos en una tabla, ya que así el programa para el cálculo de cada punto será más claro, y después los colocaremos mediante un PSET.

```
20 DIM P1(4),P2(4)
50 P1(1)=0:P2(1)=0
60 P1(2)=0:P2(2)=-60
70 P1(3)=-30:P2(3)=0
80 P1(4)=30:P2(4)=0
120 FOR N=1 TO 4
150 PSET(XP+P1(N),YP+P2(N))
160 NEXT N
230 GOTO 230
```

Para conectar los puntos entre sí podemos formar líneas (LINEs), ya que deberemos conectarlos por orden del 1 al 2, 2 al 3, 3 al 4 y 4 al 2 (fig. 11.4). Colocaremos esta orden en una sentencia de DATA y lo leeremos a partir de allí.

```
10 DATA 1,2,2,3,3,4,4,2
170 FOR L=1 TO 4
180 READ N1,N2
190 LINE(XP+P1(N1),YP+P2(N1))-(X
P+P1(N2),YP+P2(N2)),PSET
200 NEXT L
210 RESTORE
```

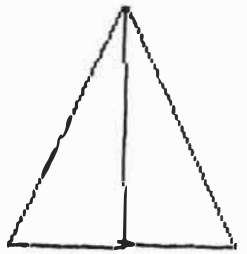



Fig. 11.4 Conexión de líneas para formar dos triángulos simétricos.

Ahora veamos lo que parece lo más difícil, calcular las nuevas posiciones de cada punto cuando se gire un cierto ángulo. Las reglas son:

1) El ángulo debe darse en radianes, por lo que los grados deben convertirse inicialmente.

```
100 FOR AN=0 TO 360 STEP 10
110 A=AN*3.142/180
220 NEXT AN
```

2) Las nuevas posiciones sobre el eje X e Y (NX(N) y NY(N)) para cada antiguo punto se calculan a partir de las siguientes dos fórmulas.

```
130 NX(N)=P1(N)*COS(A)+P2(N)*SIN
(A)
140 NY(N)=-P1(N)*SIN(A)+P2(N)*CO
S(A)
150 PSET(XP+NX(N),YP+NY(N))
190 LINE(XP+NX(N1),YP+NY(N1))-(X
P+NX(N2),YP+NY(N2)),PSET
```

Observe el signo menos en la segunda línea y también recuerde que estos siguen siendo desplazamientos a partir del centro de rotación, por lo que deberemos añadir sus coordenadas para encontrar las posiciones reales sobre la pantalla.

El STEP de ángulo incluido en el programa anterior es de 9Ø grados, por lo que el triángulo se generará en cuatro posiciones alternativas (fig. 11.5). Si se reduce el STEP, el número de posiciones aumenta y el resultado puede llegar a ser muy complejo (fig. 11.6) y también otra forma de generar dibujos.

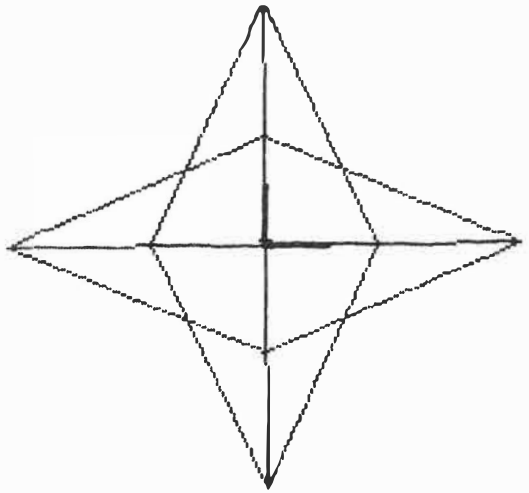


Fig. 11.5 Giro de 360° de un triángulo, en «steps» de 90°.

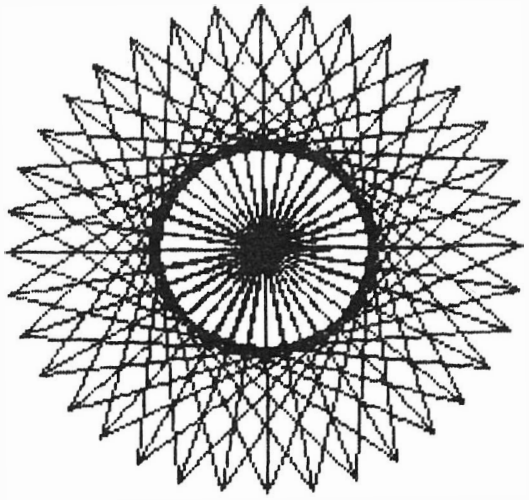


Fig. 11.6 Giro de 360° de un triángulo, en «steps» de 10°.

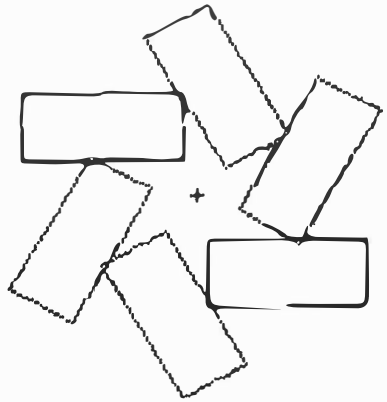


Fig. 11.7 Giro de un rectángulo que está desplazado del centro de giro.

El dibujo que se gire puede tener cualquier forma, y no tiene porqué tocar el centro de rotación. Descubra por sí mismo qué modificaciones deben hacerse a las coordenadas iniciales y a las sentencias DATA para producir el dibujo de la figura 11.7.

12. Acceso instantáneo a las órdenes de alta resolución mediante el teclado

Aunque ya hemos explicado con detalle cómo utilizar cada una de las órdenes gráficas en alta resolución en los programas, todos los esfuerzos tendrían que ser planteados por adelantado. Por otra parte, el dibujar directamente en la pantalla puede ser de mucha utilidad ya que puede cambiarse de idea fácilmente a medida que se está realizando. Si se intenta utilizar la orden INPUT en alta resolución se vuelve de nuevo a la pantalla de textos ya que INPUT detiene el programa.

Por otra parte podría utilizarse INKEY$, que es la forma más sencilla de reconocer las letras que van a utilizarse en la orden DRAW.

```
10 PMODE 4,1:SCREEN 1,0:PCLS
20 A$=INKEY$:IF A$="" THEN 20
30 DRAW A$
40 GOTO 20
```

Ejecute esto y descubrirá que cada vez que pulse una tecla válida (U, D, L, R, E, F, G o H) dibujará hacia arriba, abajo, izquierda, derecha o en una de las cuatro direcciones en diagonal.

Naturalmente, esto tiene un valor práctico limitado ya que tan sólo se puede dibujar en forma sencilla, y ni siquiera puede moverse sin dibujar, por lo que ahora vamos a considerar cómo construir un programa de dibujo directo sofisticado, que le permita manipular todas las órdenes gráficas directamente desde el teclado, mientras se está visualizando la pantalla en alta resolución. Se basa en comprobaciones de tecla utilizando la orden INKEY$ y también la utilización de la orden PEEK sobre las rutinas que recorren el teclado, que proporcionan definiciones de las órdenes gráficas mediante una única tecla, y el uso extensivo de las órdenes GET y PUT.

Selección

La primera tarea a realizar es la selección de la pantalla con el PMODE, SCREEN, y los colores principal y del fondo necesarios.

```
10 CLS:PRINT"PMODE":INPUT P:PRIN
T"CONJUNTO DE COLORES":INPUT CC:
PRINT"COLOR PRINCIPAL":INPUT C1:
PRINT"COLOR DE FONDO":INPUT C2:P
RINT"X PRINCIPIO,FINAL":INPUT XP
,XF:PRINT"Y PRINCIPIO,FINAL":INP
UT YP,YF:GOTO 5000
5000 PMODE P,1:SCREEN 1,CC:PCLS
C2:COLOR C1,C2:DRAW"C"+STR$(C1)
```

Deberán dimensionarse una serie de tablas para guardar las distintas áreas que van a obtenerse mediante las órdenes GET. Se describirán en detalle más adelante. La posición inicial se define por X e Y como el centro de la pantalla (128,96) y el incremento del cursor (IN) se inicializa a cuatro. La X y la Y se mantienen actualizadas en el programa y siempre indican la posición actual sobre la misma. Por último, una lista de las teclas que puedan utilizarse para acceder a las órdenes gráficas está guardada en TV$. Obsérvese que existe un espacio entre W y G, y que dos teclas están definidas por sus códigos CHR$. Se trata de las teclas CLEAR y ENTER que no son caracteres visualizables.

```
5010 DIM CU(10):DIM AC(10):DIM S
C(1229):DIM SA(200):DIM GP(1229)
:DIM PR(10):X=128:Y=96:IN=4:TV$=
"MRCEFTLXD@J1234BPW GOYN"+CHR$(1
2)+CHR$(13)+"IA"
```

Ahora podemos volver a la rutina principal de comprobación de tecla, en la línea 1ØØØ.

```
5020 GOTO 1000
```

Rutina de comprobación de tecla

El movimiento repetido en la misma dirección es más fácil si la tecla tiene autorrepetición. Para obtener la autorrepetición, la rutina principal de comprobación de tecla busca en las posiciones 337 y 135, en lugar de utilizar INKEY$. Las teclas se clasifican comparando el código CHR$ de PEEK(135) con la lista guardada en TV$, utilizando la orden INSTR. El valor de T dependerá de la posición del carácter de la lista, y conducirá a la subrutina correspondiente.

```
1000 IF PEEK(337)=255 THEN 1130
1010 AR=PEEK(135)
1020 A$=CHR$(AR)
1030 T=INSTR(1,TV$,A$)
1040 ON T GOTO 1200,1300,1400,15
00,1600,1700,1800,1900,2000,2100
,2200,2300,2400,2500,2600,2700,2
800,3000,3200,3300,3400,3500,360
0,3700,3800,3900,4000
```

Si la tecla pulsada no está en la lista, el control va a parar a la comprobación de la tecla del cursor. Se realizará una comprobación lógica para las teclas con SHIFT y sin SHIFT y, en consecuencia, las coordenadas X e Y se actualizan según el valor actual del incremento (IN). Se incluyen comprobaciones para asegurar que los límites del área de pantalla definida no se sobrepasan.

```
1050 Y=Y+(IN*((AR=94)-(AR=10)))
1060 Y=Y+(IN*((AR=95)-(AR=91)))
1070 IF Y>YF THEN Y=YF
1080 IF Y<YP THEN Y=YP
1090 X=X+(IN*((AR=8)-(AR=9)))
1100 X=X+(IN*((AR=21)-(AR=93)))
1110 IF X>XF THEN X=XF
1120 IF X<XP THEN X=XP
```

Cursor

El cursor necesita ser no destructivo por completo, ya que al contrario borraría parte del dibujo que hay en la pantalla. Un cursor muy pequeño y no destructivo puede generarse leyendo un pixel con la orden PPOINT y después colocándolo mediante PSET, como ya se describió, pero una forma mejor es utilizar las órdenes GET y PUT sobre un área de la pantalla alrededor de la posición actual. [Esta técnica conservadora (con una c minúscula) mediante las órdenes GET y PUT se utiliza también de forma extensiva en cualquier parte de este programa.] Un cursor puede producirse de cualquier tamaño, pero es conveniente uno de 2 por 2 pixels. Colocamos esto (GET) en la tabla del cursor (CU), con detalle gráfico, y esta tabla se coloca (PUT) de nuevo con PRESET que invierte la visualización en este punto. Después de un corto tiempo de espera, la pantalla original se reproduce de nuevo colocando (PUT) la tabla CU con PSET. El efecto total de esto es la generación de un cursor cuadrado que parpadea rápidamente.

```
1130 GET(X-1,Y-1)-(X+1,Y+1),CU,G
1140 PUT(X-1,Y-1)-(X+1,Y+1),CU,P
RESET
1150 FOR N=1 TO 10:NEXT
1160 PUT(X-1,Y-1)-(X+1,Y+1),CU,P
SET
```

Se resta una unidad de los límites iniciales para X e Y para evitar que el cursor alcance coordenadas negativas ilegales cuando se mueva hasta la parte superior o al extremo izquierdo.

Movimiento y dibujo

Si ahora ejecuta esto, descubrirá que las teclas de flecha sin el SHIFT moverán el cursor sobre la pantalla, pero sin dejar rastro.

Sin embargo, en realidad necesitaremos producir dos posibilidades distintas, el movimiento sin dibujar y el movimiento con dibujo. Si comprobamos la posición 337 y la comparamos con 159 AND 191 podemos descubrir que una de las teclas de cursor ha sido pulsada con la tecla SHIFT, mientras seguimos teniendo la autorrepetición. Si se ha pulsado una tecla sin SHIFT, entonces se realiza un movimiento en blanco hasta las nuevas coordenadas X e Y, pero si se ha pulsado la tecla con el SHIFT, se utiliza la orden MOVE, dibujándose una línea hasta las nuevas coordenadas X e Y. La línea se dibuja en el color principal actual.

```
1170 IF PEEK(337)<>159 AND PEEK(
337)<>191 THEN DRAW"BM"+STR$(X)+
","+STR$(Y):GOTO 1000
1180 DRAW"M"+STR$(X)+","+STR$(Y)
:GOTO 1000
```

Ejecute el programa de nuevo y observe la diferencia entre las teclas con SHIFT y sin SHIFT.

Rutinas de una única tecla

Una serie completa de rutinas gráficas pueden llamarse pulsando una única tecla. Siempre que se pueda, la tecla se utiliza como mnemotécnico de la acción. Las rutinas varían ampliamente en su complejidad, por lo que vamos a empezar con algo muy sencillo.

Incremento del cursor

La distancia que se mueve el cursor en cada ciclo está controlada por el incremento IN que inicialmente se coloca a 4. Las teclas 1 a 4 están designadas para dar 4 valores alternativos de IN, de 1, 2, 4 y 8.

```
2300 IN=1:GOTO 1000
2400 IN=2:GOTO 1000
2500 IN=4:GOTO 1000
2600 IN=8:GOTO 1000
```

Ejecute de nuevo el programa y vea el efecto de pulsar las teclas del 1 al 4 sobre la velocidad de movimiento y dibujo.

Cómo dejar una marca

Aunque las teclas con flechas con la tecla SHIFT pueden utilizarse para dibujar líneas, esto puede llegar a ser un poco pesado, especialmente para distancias largas. Por lo tanto, sería conveniente que pudieran utilizarse órdenes tales como LINE, pero naturalmente estas órdenes requieren las coordenadas de los dos extremos de la línea. Para utilizarlas debemos dejar una marca al principio de la línea y después mover el cursor hasta el punto final. En primer lugar indicaremos que queremos dejar una señal inicial pulsando la barra de ese espacio para alcanzar la subrutina situada en la línea 32ØØ.

```
3200 IF CF=1 THEN PUT(XA-1,YA-1)
-(XA+1,YA+1),AC,PSET:GOTO 1000
3210 XA=X:YA=Y:CF=1
3220 GET(XA-1,YA-1)-(XA+1,YA+1),
AC,G
3230 PUT(XA-1,YA-1)-(XA+1,YA+1),
AC,PRESET
3240 GOTO 1000
```

La primera línea se salta la primera vez, ya que CF tiene el valor por defecto que es Ø, por lo que las antiguas posiciones X e Y se guardan en XA e YA y el flag del cursor (CF) se pone a 1. El contenido de la pantalla en la antigua posición se coge ahora mediante la orden GET y se coloca en la tabla del antiguo cursor (AC) y mediante PUT...PRESET se coloca de forma invertida en el origen. Entonces vuelve a la rutina de comprobación de teclas y el cursor parpadeante puede moverse como antes, hasta alcanzar el punto donde se quiere tomar otra decisión.

Si cambia de idea y decide borrar la marca colocada con anterioridad sin utilizarla, tan sólo hay que pulsar de nuevo la barra de espacio. Ya que CF ahora es 1, se borrará la marca.

Línea

Pulsar la tecla L indica que se quiere dibujar una línea desde la marca (XA, YA) hasta la posición actual del cursor (X, Y). Si no se ha dejado una marca, CF estará a Ø y saltará directamente hacia atrás. También deberá colocarse la visualización anterior en la antigua posición del cursor y poner CF a Ø.

```
1800 IF CF=0 THEN 1000
1810 PUT(XA-1,YA-1)-(XA+1,YA+1),
AC,PSET
1820 LINE(XA,YA)-(X,Y),PSET
1830 CF=0:GOTO 1000
```


Borrado

Si decide que la línea era un error, puede utilizarse la M para borrarla, mediante la orden LINE con PRESET, de la misma forma.

```
1200 IF CF=0 THEN 1000
1210 PUT(XA-1,YA-1)-(XA+1,YA+1),
AC,PSET
1220 LINE(XA,YA)-(X,Y),PRESET
1230 CF=0:GOTO 1000
```

No actualización

A veces es conveniente formar una serie de líneas que radian a partir de un punto central. Para esto, la rutina se consigue a través de la tecla X, e incluso es más sencilla, ya que tan sólo hay que dibujar a partir de la antigua posición del cursor hasta la nueva, pero no hay que borrar la antigua marca.

```
1900 IF CF=0 THEN 1000
1910 LINE(XA,YA)-(X,Y),PSET
1920 GOTO 1000
```

Si se deja una marca, se mueve hasta una serie de posiciones distintas y se pulsa la X en cada punto, se producirán una serie de líneas.

Finalmente, pulse la barra de espacio para borrar la marca en la antigua posición o utilice la L de la línea para la última línea.

Rectángulo y rectángulo coloreado

Ya que ambas cosas, los rectángulos vacíos y con color, se forman añadiendo sufijos a la orden LINE, podrán ser producidos por rutinas similares. Se deja una marca y después el cursor se mueve hasta la esquina diagonalmente opuesta y se pulsa R o F.

```
1300 IF CF=0 THEN 1000
1310 PUT(XA-1,YA-1)-(XA+1,YA+1),
AC,PSET
1320 LINE(XA,YA)-(X,Y),PSET,B
1330 CF=0:GOTO 1000

1600 IF CF=0 THEN 1000
1610 PUT(XA-1,YA-1)-(XA+1,YA+1),
AC,PSET
1620 LINE(XA,YA)-(X,Y),PSET,BF
1630 CF=0:GOTO 1000
```

Círculo

Para producir un círculo necesitaremos definir el centro y el radio. El centro se marca como antes, se mueve el cursor hasta el perímetro del círculo que debe dibujarse y se pulsa la tecla C.

```
1400 IF CF=0 THEN 1000
1410 PUT(XA-1,YA-1)-(XA+1,YA+1),
AC,PSET
1420 R=SQR((ABS(XA-X)^2)+(ABS(YA
-Y)^2))
1430 CIRCLE(XA,YA),R
1440 CF=0:GOTO 1000
```

No importa en qué dirección se marca el radio (R), ya que se calcula como la hipotenusa de un triángulo rectángulo formado por las diferencias ABSolutas de las coordenadas X e Y correspondientes a las posiciones de la marca y del cursor. El círculo quedará dibujado en el color principal actual, pero esto también puede cambiarse mediante una única tecla que dará círculos de distintos colores.

Elipses

Elipses son únicamente variedades de los círculos en lo que se refiere al ordenador, pero en este caso deberá especificarse también la anchura y la altura de forma separada, por lo que serán necesarias algunas modificaciones.

```
1500 IF CF=0 THEN 1000
1510 PUT(XA-1,YA-1)-(XA+1,YA+1),
AC,PSET
1520 A=ABS(XA-X):H=ABS(YA-Y)
1530 CIRCLE(XA,YA),A,,H/A
1540 CF=0:GOTO 1000
```

Para formar una elipse se marca el centro, se mueve hasta un punto que esté a la mitad de la anchura con respecto al eje X y a la mitad de la altura a lo largo del eje Y (fig. 12.1), y después se pulsa la E.

Fig. 12.1 Cursor en posición para marcar la altura y anchura de la elipse.

GET

Cualquier área de la pantalla puede guardarse en una tabla mediante la orden GET activada por la tecla T (tomar). El tamaño de la tabla inicializada originalmente ocupará la totalidad de la pantalla, por lo que no debe haber problemas. El rectángulo de la pantalla que debe tomarse se marca como un rectángulo normal, excepto que debe marcarse la esquina superior izquierda y la inferior derecha por este orden.

```
1700 IF CF=0 THEN 1000
1710 PUT(XA-1,YA-1)-(XA+1,YA+1),
AC,PSET
1720 GET(XA,YA)-(X,Y),GP,G
1730 GX=XA-X:GY=YA-Y:CF=0
1740 SOUND 255,1:GOTO 1000
```

El tamaño del rectángulo en función de las coordenadas X e Y debe guardarse en GX y GY, de forma que pueda colocarse de nuevo correctamente mediante la orden PUT.

La orden GET es muy útil para producir copias de un dibujo sencillo en cualquier parte de la pantalla, o para experimentar con distintas posiciones de un dibujo.

Copia

A medida que los dibujos van siendo más complejos, cada vez tendrá más miedo de cometer un error desastroso e irrevocable. Para prevenir este caso puede incluirse una característica de copia que puede utilizarse en cualquier momento, pulsando el «CLEAR». Esta característica también nos permite el borrar (o ver ¿qué sucede si?) ya que se puede guardar el contenido de la pantalla y después intentar descubrirlo.

```
3700 GET(XP,YP)-(XF,YF),SC
3710 SOUND 255,1:GOTO 1000
```

Esto obtiene una copia de la totalidad del área de trabajo de la pantalla en la tabla SC, da una señal y vuelve. Para colocar de nuevo esta pantalla en cualquier momento pulse la tecla ENTER.

```
3800 PUT(XP,YP)-(XF,YF),SC
3810 SOUND 255,1:GOTO 1000
```

Naturalmente, esta rutina guardará sólo una pantalla, ya que cada vez que utilice este proceso de copia reescribirá encima de la antigua pantalla guardada, pero es de gran valor para un almacena-

miento temporal si usted es nervioso. Coja el hábito de pulsar el CLEAR cuando no sepa qué hacer a continuación, antes de que aparezca el desastre. Si de verdad necesita dos copias de la pantalla completa, puede obtener otra y guardarla en la tabla GP.

PUT

Todas las distintas opciones de la orden PUT pueden utilizarse para reproducir el área obtenida mediante la orden GET y guardada en la tabla GP, sobre cualquier posición de la pantalla y con cualquier acción. Las coordenadas del extremo superior izquierdo son las de la posición del cursor y las otras se calculan a partir del valor guardado del tamaño del área (GX y GY).

La D nos da PSET (Dejar).

```
2000 PUT(X,Y)-(X-GX,Y-GY),GP,PSE
T
2010 SOUND 255,1:GOTO 1000
```

La I produce un PRESET (Invertir).

```
3900 PUT(X,Y)-(X-GX,Y-GY),GP,PRE
SET
3910 SOUND 255,1:GOTO 1000
```

La Y nos da un AND (Y lógico).

```
3500 PUT(X,Y)-(X-GX,Y-GY),GP,AND
3510 SOUND 255,1:GOTO 1000
```

La O da un OR (O lógica).

```
3400 PUT(X,Y)-(X-GX,Y-GY),GP,OR
3410 SOUND 255,1:GOTO 1000
```

La N nos da un NOT

```
3600 PUT(X,Y)-(X-GX,Y-GY),GP,NOT
3610 SOUND 255,1:GOTO 1000
```

Si las coordenadas salen fuera de la pantalla, la reproducción será incorrecta, por lo que es mejor utilizar la rutina de copia antes, si se está cerca de la parte inferior o la parte lateral derecha.

Borrado total

Si quiere abandonar lo que está haciendo y borrarlo todo, pulse la B. Ya que ésta es una acción permanente que no debe utilizarse por accidente, se han colocado ciertos seguros. Una inversión no destructiva de la parte superior de la pantalla será aviso, y la B debe mantenerse pulsada 5 ciclos durante 6 segundos, para que realmente se ejecute el PCLS.

```
2700 GET(0,0)-(255,10),SA,G
2710 PUT(0,0)-(255,10),SA,PRESET
2720 IF CL=0 THEN TIMER=0:CL=CL+
1:ELSE CL=CL+1:IF TIMER>300 THEN
 CL=0:ELSE IF CL=5 THEN PCLS C2:
CL=0
2730 PUT(0,0)-(255,10),SA,PSET
2740 GOTO 1000
```

Cuando se pulsa la tecla B se coge (GET) una banda de la parte superior de la pantalla y se guarda en la tabla SA (señal de aviso), y se coloca mediante PUT...PRESET en forma invertida. La primera vez el TIMER se pone a Ø, el flag de borrado (CL) se incrementa en 1, y la tabla SA se coloca de nuevo mediante PUT...PSET. Si la B sigue pulsada se comprueba el valor del TIMER, comparándolo con 3ØØ, y si se alcanza este valor el flag de borrado se pone a Ø. Si el flag CL ha llegado hasta 5 entonces se produce el borrado (PCUS).

Cambio de colores

Ahora que ya hemos agotado las teclas adecuadas, tendremos que utilizar la W para indicar qué colores utilizaremos como color principal y de fondo. Ya que no se dispone de texto sobre la pantalla, las indicaciones respecto a en qué etapa se está, se dan moviendo bloques invertidos a través de la parte superior de la pantalla. El primer bloque se invierte hacia la izquierda de la pantalla indicando que debe entrarse el color principal en C1$. Ya que necesitamos utilizar aquí el INKEY$ sin repetición, la autorrepetición debe desactivarse mediante POKE 135,Ø. Una vez que se ha entrado el color principal, el bloque original se coloca de nuevo, y la inversión se hace en la derecha de la pantalla, para que se entre el color de fondo en C2$. Finalmente, el bloque se coloca de nuevo y se cambian los colores mediante la orden VAL de C1$ y C2$ y dibujando con (DRAW) «C» + C1$.

```
3000 GET(40,0)-(50,10),PR,G
3010 PUT(40,0)-(50,10),PR,PRESET
3020 FOR N=1 TO 1000:NEXT N
```

```
3030 POKE 135,0
3040 C1$=INKEY$:IF C1$="" THEN 3
040
3050 PUT(40,0)-(50,10),PR,PSET
3060 GET(168,0)-(178,10),PR,G
3070 PUT(168,0)-(178,10),PR,PRES
ET
3080 C2$=INKEY$:IF C2$="" THEN 3
080
3090 PUT(168,0)-(178,10),PR,PSET
3100 C1=VAL(C1$):C2=VAL(C2$):COL
OR C1,C2:DRAW"C"+C1$:GOTO 1000
```

Pintar

Las coordenadas para pintar y los colores se entran de forma similar, pero como la orden PAINT a veces suele causar resultados inesperados, se hace automáticamente una copia, llamando a la rutina de la línea 37ØØ, y el flag de pintar se pone a 1. El cursor se coloca en el punto inicial a partir de donde debe pintarse, y se pulsa la P. Se visualiza un bloque a la izquierda, y se entra el primer color. El segundo bloque indica el segundo color, el del límite, y el bloque final pide la confirmación de su decisión. Si CO$ no es S entonces se abandona la orden PAINT.

```
2800 IF PF=0 THEN PF=1:GOTO 3700
2810 GET(0,0)-(10,10),PR,G
2820 PUT(0,0)-(10,10),PR,PRESET
2830 FOR N=1 TO 1000:NEXT
2840 POKE 135,0
2850 C1$=INKEY$:IF C1$="" THEN 2
850
2860 PUT(0,0)-(10,10),PR,PSET
2870 GET(123,0)-(133,10),PR,G
2880 PUT(123,0)-(133,10),PR,PRES
ET
2890 C2$=INKEY$:IF C2$="" THEN 2
890
2900 PUT(123,0)-(133,10),PR,PSET
2910 GET(245,0)-(255,10),PR,G
2920 PUT(245,0)-(255,10),PR,PRES
ET
2930 CO$=INKEY$:IF CO$="" THEN 2
930
2940 IF CO$<>"S" THEN PUT(245,0)
-(255,10),PR,PSET:GOTO 1000
2950 PAINT(X,Y),VAL(C1$),VAL(C2$
)
2960 PUT(245,0)-(255,10),PR,PSET
2970 PF=0:GOTO 1000
```


Arco

Un arco de un círculo o de una elipse sólo pueden formarse si se definen los puntos inicial y final, pero primero deberán definirse la anchura (A) y la altura (H) tal como se ha descrito para las elipses.

```
4000 IF CF=0 THEN 1000
4010 PUT(XA-1,YA-1)-(XA+1,YA+1),
AC,PSET
4020 A=ABS(XA-X):H=ABS(YA-Y)
4030 FOR N=1 TO 1000:NEXT N
4040 POKE 135,0
4050 GET(10,0)-(15,10),PR,G
4060 PUT(10,0)-(15,10),PR,PRESET
```

Se puede saltar a esta rutina pulsando la A, pero existe un problema práctico para indicar los puntos inicial y final mediante teclas sencillas, ya que las teclas del Ø al 9 permitirían tan sólo 1Ø puntos del arco distintos. Los valores actuales para los puntos final e inicial deben estar comprendidos entre Ø y 1 y la siguiente solución le permite entrar fácilmente números decimales mediante INKEY$. Primero el valor inicial. La línea 4Ø7Ø comprueba INKEY$ y si no está vacío entonces la línea 4Ø8Ø comprueba si IN$ era CHR$ (13) (= ENTER). Si no, IN$ se añade al final de T1$ y se toma otro IN$. Así se van añadiendo a T1$ hasta que se pulse el ENTER. De la misma forma, el punto final se va generando en T2$ a partir de FI$.

```
4070 IN$=INKEY$:IF IN$="" THEN 4
070
4080 IF IN$<>CHR$(13) THEN T1$=T
1$+IN$:GOTO 4070
4090 PUT(10,0)-(15,10),PR,PSET
4100 GET(230,0)-(245,10),PR,G
4110 PUT(230,0)-(245,10),PR,PRES
ET
4120 FI$=INKEY$:IF FI$="" THEN 4
120
4130 IF FI$<>CHR$(13) THEN T2$=T
2$+FI$:GOTO 4120
```

Cuando ambos puntos, el inicial y el final, se han entrado, se dibuja el arco.

```
4140 PUT(230,0)-(245,10),PR,PSET
4150 CIRCLE(XA,YA),A,,H/A,VAL(T1
$),VAL(T2$)
```

Ya que quizá quiera prolongar el arco Q$, espere para ver si pulsa otra vez la A de arco de nuevo. Si es así, T1$ y T2$ se borran, pero la forma del círculo se retiene ya que A y H no se borran.

```
4160 Q$=INKEY$:IF Q$="" THEN 416
0
4170 IF Q$="A" THEN T1$="":T2$="
":GOTO 4030
4180 CF=0:T1$="":T2$="":GOTO 100
0
```

Guardar/Cargar

Una vez que se ha completado el dibujo, probablemente se querrá guardar, por lo que la tecla G nos conduce a la rutina de guardar/cargar que vuelca el contenido de las cuatro primeras páginas gráficas en el cassette, en forma de un archivo en código máquina que después puede cargarse de nuevo.

```
3300 CLS:PRINT"GUARDAR O CARGAR"
:INPUT Q$:IF LEFT$(Q$,1)="G" THE
N 3310 ELSE IF LEFT$(Q$,1)="C" T
HEN 3350 ELSE SCREEN 1,CC:GOTO 1
000
3310 PRINT"GUARDAR":PRINT,,"NOMB
RE DEL ARCHIVO";:INPUT NO$
3320 CSAVEM NO$,1536,7679,6144
3330 SCREEN 1,CC:GOTO 1000
3350 PRINT"CARGAR":PRINT,,"NOMBR
E DEL ARCHIVO";:INPUT NO$
3360 SCREEN 1,CC:CLOADM NO$
3370 GOTO 1000
```

Dibujando con la palanca

Aunque puede moverse sobre la pantalla mediante las teclas del cursor, a veces es más conveniente utilizar esta rutina de la palanca que se llama mediante la J, ya que entonces pueden hacerse movimientos diagonales con más facilidad. La rutina de la tecla del cursor se sustituye por la rutina de la palanca, hasta que una de las teclas del cursor se pulse de nuevo. La palanca se utiliza para controlar la dirección más que la posición absoluta (véase más adelante). Los valores dados por la orden JOYSTK para Ø y 1 se guardan en variables y se utilizan comprobaciones lógicas con respecto a las posiciones límites para actualizar la X y la Y.

```
2200 J0=JOYSTK(0):J1=JOYSTK(1):X
=X+(IN*((J0<=20)-(J0=>50))):Y=Y+
(IN*((J1<=20)-(J1=>50)))
```

A continuación de las comprobaciones de los límites se realiza una réplica de la rutina normal del cursor y después se comprueba si se está pulsando alguna tecla.

```
2210 IF Y>YF THEN Y=YF ELSE IF Y
<YP THEN Y=YP
2220 IF X>XF THEN X=XF ELSE IF X
<XP THEN X=XP
2230 GET(X-1,Y-1)-(X+1,Y+1),CU,G
:PUT(X-1,Y-1)-(X+1,Y+1),CU,PRESE
T:FOR N=1 TO 10:NEXT:PUT(X-1,Y-1
)-(X+1,Y+1),CU,PSET
```

Si se pulsa una tecla el PEEK (337) es menor que 255 y se abandona la rutina de JOYSTK.

```
2250 IF PEEK(337)<255 THEN 1000
```

Si no se está pulsando ninguna tecla, entonces se comprueba el botón de la palanca, comparando PEEK (6528Ø) con 126 y 254. Si no se está pulsando el botón se realiza un movimiento en blanco, pero si se está pulsando se utiliza la orden MOVE y el movimiento no será en blanco.

```
2260 IF PEEK(65280)<>254 AND PEE
K(65280)<>126 THEN DRAW"BM"+STR$
(X)+","+STR$(Y):GOTO 2200 ELSE D
RAW"C"+STR$(C1)+"M"+STR$(X)+","+
STR$(Y):GOTO 2200
2270 GOTO 1130
```

Entrada en el modo de carácter

La última orden de tecla es la @ que deja el modo de dibujo y va a un modo alternativo en el cual se visualizan caracteres preformados. D$ se pone a «Ø» de forma que estos caracteres se dibujan inicialmente de izquierda a derecha (véase más adelante).

```
2100 POKE 135,0:D$="0":GOTO 20
```

Modo carácter

Comprobación de teclas y cursor

El factor de escala para la orden DRAW (S) se inicializa a cuatro veces el incremento del cursor y se comprueba INKEY$. Se forma un cursor mediante las órdenes GET y PUT como antes, pero aquí es una línea en lugar de un rectángulo, ya que el eje X tiene la longitud Ø. El cursor parpadeante se repite hasta que se pulsa una tecla y si ésta es la @, el programa vuelve de nuevo al modo de dibujo.

```
20 S=IN*4:C$=INKEY$:GET(X,Y)-(X+
IN,Y),CU,G:PUT(X,Y)-(X+IN,Y),CU,
PRESET:FOR N=1 TO 10:NEXT N:PUT(
X,Y)-(X+IN,Y),CU,PSET:IF C$="" T
HEN 20 ELSE IF C$="@" THEN 1000
```

Entonces se calcula el código ASCII de la última tecla pulsada y se utiliza en una comparación con las teclas con flecha del movimiento del cursor. Cada vez que se pulsen las teclas del cursor a la izquierda o a la derecha, se moverá una unidad de carácter y cada vez que se pulsen las teclas de movimiento hacia arriba o abajo se moverá una unidad y media. Esto dará el espaciado correcto entre caracteres alfanuméricos y líneas. Una vez que se han comprobado los límites de X e Y se realiza un movimiento en blanco hasta la nueva posición.

```
21 A=ASC(C$):X=X+((S*1.5)*((A=8)
-(A=9))):Y=Y+((S*2)*((A=94)-(A=1
0))):IF Y>YF THEN Y=YF ELSE IF Y
<YP THEN Y=YP
22 IF X>XF THEN X=XF ELSE IF X<X
P THEN X=XP
23 DRAW"BM"+STR$(X)+","+STR$(Y):
 IF A>31 AND A<91 THEN GOSUB 25:
X=X+(S*1.5):GOTO 20:ELSE 20
```


Si el código ASCII de la tecla pulsada no está entre 32 y 9Ø el programa vuelve de nuevo a la línea 2Ø, pero si está dentro de estos límites va a la subrutina de la línea 25. A la vuelta de esta rutina de dibujo del carácter, la posición X se actualiza.

Clasificación de los caracteres

La línea 25 es una rutina que clasifica las teclas comprendidas entre los valores 32 y 9Ø mediante la orden ON GOSUB, en función de sus códigos ASCII. Cada una de estas subrutinas dibuja un carácter distinto y el programa está hecho de forma que los números de línea de estas subrutinas correspondan a los códigos ASCII de la tecla que representan. Por ejemplo, el pulsar la tecla A conduce a la línea 65. La única tecla con un código entre 32 y 9Ø que no tiene una subrutina es la @ (código 64) ya que esta tecla se ha utilizado para volver al modo de dibujo.

```
25 DRAW"C"+STR$(C1)+"A"+D$+"S"+S
TR$(S):ON(ASC(C$)-31)GOSUB 32,33
,34,35,36,37,38,39,40,41,42,43,4
4,45,46,47,48,49,50,51,52,53,54,
55,56,57,58,59,60,61,62,63,64,65
,66,67,68,69,70,71,72,73,74,75,7
6,77,78,79,80,81,82,83,84,85,86,
87,88,89,90:RETURN
```

Subrutinas de carácter

Se puede dibujar cualquier tipo de carácter en las subrutinas, y lo único que hay que tener en cuenta es que al final hay que hacer un movimiento en blanco hasta el punto estándar, correspondiente a la siguiente posición de carácter. Los ejemplos dados (tabla 12.1) incluyen todas las letras mayúsculas y los números junto con algunos otros caracteres especiales. Los caracteres están construidos en una matriz de 5 por 6 (fig. 12.2). Si quieren definirse incluso más caracteres, pueden incluirse las letras minúsculas y duplicar la línea 25 mediante la 26, colocando códigos ASCII más altos. La gran ventaja de utilizar la orden DRAW para producir caracteres es que estos pueden ser de cualquier tamaño y forma y pueden escalarse, colorearse y dibujarse con el ángulo deseado. Son de especial interés las rutinas de acentuación que sustituyen a los caracteres normales de las teclas #, $, % y &.

```
35 DRAW"BM-4,-7EBM+3,+8":RETURN
36 DRAW"BM-4,-7HBM+5,+8":RETURN
```

```
37 DRAW"BM-5,-7EFBM+3,+7":RETURN
38 DRAW"BM-4,-1DGBM+5,-1":RETURN
```

y el símbolo del copyright que sustituye al !

```
33 DRAW"BM+1,+0R3EU4HL3GD4FBM+2,
-2LHERBM+4,+4":RETURN
```

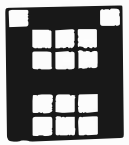


Fig. 12.2 Formación de la letra A en una retícula de 6×5.

El signo mayor que (>) se ha sustituido por un dibujo mayor que debe ser razonablemente familiar a los usuarios del Dragon (fig. 12.3). Ya que éste es mayor que el resto de los caracteres, la posición X se avanza más de lo normal.

```
62 DRAW"BM+2,+0R17BM-4,+0HL6GE2R
4FH2L2GH4BM+6,+3U5BM+2,+6E4":X=X
+(S*3):RETURN
```

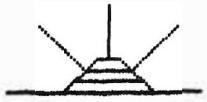


Fig. 12.3 Un carácter alternativo.

Color (C1) escala (S) y ángulo (D)

Si observa de nuevo el principio de la línea 25 verá que cada carácter se dibuja en el color principal actual, y con la escala y ángulo actual. Para cambiar el color principal (C1) o escala (S) debe volverse de nuevo al modo de dibujo y quizá recuerde que D$ se puso a «Ø» antes de entrar en el modo de carácter, por lo que el dibujo aparece de izquierda a derecha. Existen cuatro escalas distintas que producirán

letras de distintos tamaños (fig. 12.4). Obsérvese que la línea 32 establece el color a C2 de forma que dibuja con el color de fondo para producir dos cosas; un espacio y borrar un carácter.

Fig. 12.4 Distintos tamaños de letras.

El ángulo puede actualizarse en el modo de carácter mediante un proceso con dos etapas. Primero se incluye una comprobación de la tecla ENTER (CHR$(13)). Si INKEY$ no es ENTER entonces se excluye la línea 24. Si se pulsa ENTER se comprueba de nuevo INKEY$ y se emite un sonido de aviso hasta que se pulse otra tecla. Las teclas del Ø al 3 podrán utilizarse ahora para cambiar el ángulo del dibujo. Las teclas no válidas se rechazan mediante la orden VAL.

```
23 DRAW"BM"+STR$(X)+","+STR$(Y):
IF A=13 THEN 24 ELSE IF A>31 AND
 A<91 THEN GOSUB 25:X=X+(S*1.5):
GOTO 20:ELSE 20
24 D$=INKEY$:SOUND 1,1:IF D$=""
THEN 24 ELSE IF VAL(D$)>3 THEN D
$="":GOTO 20:ELSE 20
```

Esta característica es de mucha utilidad al rotular diagramas, ya que el texto puede escribirse en las cuatro direcciones (fig. 12.5).

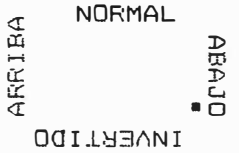


Fig. 12.5 Cambio de ángulo.

Resurrección

Si ha sido lo suficientemente desafortunado para intentar lo imposible, y por lo tanto ha hecho que se pierda el programa, tecleando SCREEN 1, SN: GOTO 1ØØØ probablemente le dejará de nuevo donde estaba antes del último movimiento.

Avisos escritos

Ahora que ya se tiene la posibilidad de producir texto en la pantalla de alta resolución, pueden cambiarse fácilmente los mensajes mediante «bloques», descritos antes, por mensajes realmente escritos. El mensaje que se va a dar se define como M$ y después se descompone en caracteres que serán enviados uno a uno hacia las rutinas de dibujo del carácter. Colocaremos la rutina de descomposición del mensaje en la dirección 6ØØØ y colocaremos D$=«Ø» para asegurar que la orden de ángulo nos dé siempre un texto normal.

```
6000 D$="0":S=4:FOR N=1 TO LEN(M
$):C$=MID$(M$,N,1):GOSUB 25:NEXT
 N:RETURN
```

Como demostración modificaremos la rutina de PAINT (pintar). La manera más fácil de reproducir la pantalla después de los mensajes de aviso es mediante la orden GET, guardando la parte superior de la pantalla en la tabla SA.

```
2810 GET(0,0)-(255,10),SA,G
```

Ahora definimos el primer mensaje como M$, hacemos un movimiento en blanco hasta la posición donde deseamos escribirlo y vamos a la rutina de descomposición.

```
2820 M$="COLOR 1?":DRAW "BM10,10
":GOSUB 6000
```

Cuando ya se ha entrado el valor, la parte superior de la pantalla se coloca de nuevo como antes.

```
2860 PUT(0,0)-(255,10),SA,PSET
```

Los otros mensajes se tratan de la misma forma, excepto en que ya no hay necesidad de guardar la parte superior en SA cada vez, por lo que pueden borrarse algunas líneas.

```
2870 (borrada)
2880 M$="COLOR 2?":DRAW "BM50,10
":GOSUB 6000
2890 C2$=INKEY$:IF C2$="" THEN 2
890
2900 PUT(0,0)-(255,10),SA,PSET
2910 (borrada)
2920 M$="PINTAR S/N?":DRAW "BM10
0,10":GOSUB 6000
```



```
2940 IF CO$<>"S" THEN PUT(0,0)-(
255,10),SA,PSET:GOTO 1000
2960 PUT(0,0)-(255,10),SA,PSET
```

Cualquier otro texto de mensaje puede escribirse en la pantalla de la misma forma.

LISTA DE ORDENES ASOCIADAS A LAS TECLAS

Teclas	Acción
MODO DIBUJO	
teclas cursor	mueven el cursor
teclas cursor shift	dibujo con el cursor
1	incremento cursor=1
2	incremento cursor=2
3	incremento cursor=4
4	incremento cursor=8
barra de espacio	deja una marca
L	línea desde la marca
M	borra línea desde marca
X	línea desde marca sin actualizar
R	rectàngulo
F	rectàngulo relleno
C	circulo
E	elipse
T	orden GET

clear — copia del dibujo

D — orden PUT PSET

I — orden PUT PRESET

Y — orden PUT AND

O — orden PUT OR

N — orden PUT NOT

B — borrar pantalla(debe pulsarse durante 5 seg.)

W — cambio de colores (primer numero es el principal, el segundo es el de fondo)

P — orden PAINT (pulsar 2 veces,entrar el color de relleno,el color del límite, y "S" si es correcto)

A — arco (el primer número es el principio el segundo es el final,pulsar "A" otra vez para continuar este arco)

G — guardar/cargar en cassette

J — joystick (pulsar una tecla para salir de este modo)

@ — entra en modo caracter

MODO CARACTER

teclas cursor — movimiento cursor

enter — cambio ángulo (pulsar 0-3 para seleccionar angulo)

@ — entra modo dibujo

cualquier otra tecla — dibuja carácter

para recobrar el dibujo cuando se pierda:
SCREEN 1,CC:GOTO 1000

Tabla 12.1

CARACTERES DE MUESTRA

```
32 Z=S/4:DRAW"C"+STR$(C2)+"S4D"+
STR$(Z*3):FOR J=1 TO Z*3:FORI=1
TO Z*2:DRAW"U6":NEXTI:DRAW"R":FO
R I=1 TO Z*2:DRAW"D6":NEXT I:DRA
W"R":NEXTJ:FORJ=1 TO Z:DRAW"BM+3
,-"+STR$(Z*3):NEXTJ:DRAW"C"+STR$
(C1)+"S"+STR$(S):RETURN
33 DRAW"BM+1,+0R3EU4HL3GD4FBM+2,
-2LHERBM+4,+4":RETURN
34 DRAW"BM+0,-6DBM+2,+0UBM+4,+6"
:RETURN
35 DRAW"BM-4,-7EBM+3,+8":RETURN
36 DRAW"BM-4,-7HBM+5,+8":RETURN
37 DRAW"BM-5,-7EFBM+3,+7":RETURN
38 DRAW"BM-4,-1DGBM+5,-1":RETURN
39 DRAW"BM+0,-6DBM+4,+5":RETURN
40 DRAW"BM+2,+0HU4EBM+4,+6":RETU
RN
41 DRAW"BM+1,+0EU4HBM+5,+6":RETU
RN
42 DRAW"BM+0,-1E4BM+0,+4H4BM+8,+
5":RETURN
43 DRAW"BM+0,-3R4L2U2D4BM+5,+1":
RETURN
44 DRAW"BM-1,+0DGBM+4,-2":RETURN
45 DRAW"BM+0,-3R4BM+4,+3":RETURN
46 DRAW"BM-1,+0UBM+4,+1":RETURN
47 DRAW"BM+0,-1E4BM+4,+5":RETURN
48 DRAW"BM+0,-1FR2EU4HL2GD4BM+8,
+1":RETURN
49 DRAW"BM+1,+0U6GBM+6,+5":RETUR
N
50 DRAW"BM+4,+0L4UER2EU2HL2GBM+8
,+5":RETURN
51 DRAW"BM+0,-1FR2EUHL2R2EUHL2GB
M+8,+5":RETURN
52 DRAW"BM+3,+0U6G3R4BM+4,+3":RE
TURN
```

```
53 DRAW"BM+0,-1FR2EU2HL3U2R4BM+4
,+6":RETURN
54 DRAW"BM+0,-2ER2FDGL2HU4ER2FBM
+4,+5":RETURN
55 DRAW"BM+2,+0U2E2U2L4BM+8,+6":
RETURN
56 DRAW"BM+1,+0R2EUHL2HUER2FDGL2
GDFBM+7,+0":RETURN
57 DRAW"BM+0,-1FR2EU4HL2GDFR3BM+
4,+3":RETURN
58 DRAW"BM+0,-5DBM+0,+2DBM+4,+1"
:RETURN
59 DRAW"BM+0,-5DBM+0,+2DGBM+5,+0
":RETURN
60 RETURN
61 DRAW"BM+0,-2R4BM+0,-2L4BM+8,+
4":RETURN
62 DRAW"BM+2,+0R17BM-4,+0HL6GE2R
4FH2L2GH4BM+6,+3U5BM+2,+6E4":X=X
+(S*3):RETURN
63 DRAW"BM+2,+0UBM+0,-1UREUHLGBM
+7,+5":RETURN
65 DRAW"U5ER2FD5U3L4BM+8,+3":RET
URN
66 DRAW"U6R3FDGFDGL3U3R3BM+5,+3"
:RETURN
67 DRAW"BM+1,+0HU4ER2FHL2GD4FR2E
BM+4,+1":RETURN
68 DRAW"U6R3FD4GL3BM+8,+0":RETUR
N
69 DRAW"R4L4U3R4L4U3R4BM+4,+6":R
ETURN
70 DRAW"U3R4L4U3R4BM+4,+6":RETUR
N
71 DRAW"BM+1,+0R2EULRDGL2HU4ER2F
BM+4,+5":RETURN
72 DRAW"U6D3R4U3D6BM+4,+0":RETUR
N
73 DRAW"BM+1,+0R2LU6LR2BM+4,+6":
RETURN
74 DRAW"BM+0,-1FR2EU5BM+4,+6":RE
TURN
75 DRAW"U6BM+0,+3RE3G3F3BM+4,+0"
:RETURN
76 DRAW"R4L4U6BM+8,+6":RETURN
77 DRAW"U6F2E2D6BM+4,+0":RETURN
78 DRAW"U6DF4DU6BM+4,+6":RETURN
```



```
79 DRAW"BM+1,+0R2EU4HL2GD4FBM+7,
+0":RETURN
80 DRAW"U6R3FDGL3BM+8,+3":RETURN
81 DRAW"BM+1,+0R2EU4HL2GD4FBM+1,
-2F2BM+4,+0":RETURN
82 DRAW"U6R3FDGL3RF3BM+4,+0":RET
URN
83 DRAW"BM+0,-1FR2EH4ER2FBM+4,+5
":RETURN
84 DRAW"BM+2,+0U6L2R4BM+4,+6":RE
TURN
85 DRAW"BM+0,-6D5FR2EU5BM+4,+6":
RETURN
86 DRAW"BM+0,-6D4F2E2U4BM+4,+6":
RETURN
87 DRAW"BM+0,-6D6E2F2U6BM+4,+6":
RETURN
88 DRAW"UE4UBM+0,+6UH4UBM+8,+6":
RETURN
89 DRAW"BM+2,+0U4H2F2E2BM+4,+6":
RETURN
90 DRAW"R4L4UE4UL4BM+8,+6":RETUR
N
```

13. Las órdenes GET y PUT con caracteres de alta resolución

Aunque las rutinas de dibujo que se han dado antes pueden incorporarse en cualquier programa, en cierta manera son lentas de operación y pueden aparecer problemas si teclea demasiado de prisa. Sin embargo, una vez que se han utilizado para dibujar en la pantalla, pueden guardarse y utilizarlas después de forma más rápida mediante GET y PUT.

Transferencia de caracteres entre programas

Una vez que se han creado los propios caracteres, evidentemente será de utilidad el poder transferirlos entre programas, de forma que no haya que teclearlos de nuevo, sino que se pueda construir una librería completa de conjuntos alternativos. Se puede definir cada carácter como un texto real, cambiando cada línea por:

...A$=«.....»

en lugar de

...DRAW«....» etc.

y después guardarlos en forma de un archivo en código ASCII en una cinta.

Cómo guardar los caracteres en forma de código máquina

Sin embargo, es mucho más sencillo dibujarlos en la parte superior de la pantalla y después guardar esta área haciendo un vaciamiento de memoria en código máquina en una cinta. Como ejemplo colocaremos las letras y números de ejemplos que hemos definido antes, en la parte superior de la pantalla y después los guardaremos. Se borra la pantalla y el conjunto de colores se invierte para obtener caracteres negros sobre un fondo claro, ya que son más fáciles de leer de esta forma. (Si se quieren guardar como blanco sobre negro tan sólo hay que utilizar PCLS en lugar COLOR Ø,1: PCLS1.) Podemos

seleccionar la posición en pantalla mediante un movimiento en blanco y después saltar a la subrutina normal de dibujo de caracteres para obtener las letras y los números en un formato adecuado.

```
10000 PMODE 4,1:SCREEN 1,0:COLOR
 0,1:PCLS 1:D$="0":X=1:FOR C=48
TO 57:DRAW"BM"+STR$(X)+",7":C$=C
HR$(C):GOSUB 25:X=X+7:NEXT C
10010 FOR C=65 TO 90:DRAW"BM"+ST
R$(X)+",7":C$=CHR$(C):GOSUB 25:X
=X+7:NEXT C
10020 CSAVEM "CARAC",1536,1792,2
24
```

La posición inicial para el primer carácter tiene las coordenadas 1,7, y cada carácter subsiguiente estará a una distancia de 7 pixels a la derecha de éste. Los valores de C entre 48 y 57 definen los números y entre 65 y 9Ø las letras.

En PMODE 4 hay 256/8=32 bytes por línea y únicamente hemos utilizado las 7 líneas superiores, por lo que necesitaremos guardar (CSAVEM) 224 bytes para mantener los 36 caracteres. Ya que esto funciona con menos de 7 bytes por carácter, podrá ver que éste es un método muy económico, y también observará que el código máquina se guarda rápidamente.

Carga de caracteres y selección de la pantalla

Los caracteres que se han guardado en una cinta como código máquina mediante la orden CSAVEM pueden recuperarse con facilidad mediante la orden CLOADM y utilizarlos para producir una visibilidad de texto de calidad superior. Para una visibilidad óptima utilice PMODE 4 e invierta el COLOR para obtener letras en negro sobre verde (o marrón).

```
10 PMODE 4,1:SCREEN 1,0:PCLS 1:C
OLOR 0,1:CLOADM
```

Los caracteres aparecerán en línea a lo largo de la parte superior de la pantalla (fig. 13.1).

```
01 23456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Fig. 13.1 Caracteres recargados.

Dimensionado de las tablas

Antes de obtener (GET) todos los caracteres habrá que dimensionar las tablas. Por desgracia esto representa gran cantidad de trabajo repetitivo, ya que no pueden cambiarse los números de línea en el Dragon o cambiar el nombre de una tabla en las órdenes GET y PUT. (De hecho hemos encontrado una forma de solventar este problema, pero ya que no es muy fácil de comprender lo hemos dejado para más adelante. Si quiere llegar a ser realmente bueno con los gráficos, asegúrese de que comprende cómo funciona este método antes de intentar el método alternativo.) El tamaño de cada tabla es de únicamente UN elemento, ya que la matriz de 5 por 7 sólo necesita 35 bits. Cada tabla se nombra como C más el carácter correspondiente y además una tabla en blanco (BL) queda también DIMensionada.

```
20 DIMC0(1):DIMC1(1):DIMC2(1):DI
MC3(1):DIMC4(1):DIMC5(1):DIMC6(1
):DIMC7(1):DIMC8(1):DIMC9(1):DIM
CA(1):DIMCB(1):DIMCC(1):DIMCD(1)
:DIMCE(1):DIMCF(1):DIMCG(1):DIMC
H(1):DIMCI(1):DIMCJ(1):DIMCK(1):
DIMCL(1):DIMCM(1):DIMCN(1)
30 DIM DU(1):DIMCO(1):DIMCP(1):D
IMCQ(1):DIMCR(1):DIMCS(1):DIMCT(
1):DIMCU(1):DIMCV(1):DIMCW(1):DI
MCX(1):DIMCY(1):DIMCZ(1):DIMBL(1)
```

Obtención de los caracteres (GET)

Ahora deberán establecerse variables adecuadas en función del tamaño y del espaciado de los caracteres que deben entrarse. X e Y establecen las coordenadas iniciales. S es la distancia entre caracteres sobre la pantalla, y A y H la altura y anchura de los caracteres.

```
40 X=1:Y=0:S=7:A=5:H=7
```

La totalidad de los caracteres en mayúscula y los números pueden colocarse en la línea superior de la pantalla, por lo tanto tan sólo hay que obtener (GET) cada carácter, incrementando la coordenada X en S cada vez.

```
100 GET(X,Y)-(X+A,Y+H),C0,G:X=X+
S
```



```
110 GET(X,Y)-(X+A,Y+H),C1,G:X=X+
S
120 GET(X,Y)-(X+A,Y+H),C2,G:X=X+
S
130 GET(X,Y)-(X+A,Y+H),C3,G:X=X+
S
140 GET(X,Y)-(X+A,Y+H),C4,G:X=X+
S
150 GET(X,Y)-(X+A,Y+H),C5,G:X=X+
S
160 GET(X,Y)-(X+A,Y+H),C6,G:X=X+
S
170 GET(X,Y)-(X+A,Y+H),C7,G:X=X+
S
180 GET(X,Y)-(X+A,Y+H),C8,G:X=X+
S
190 GET(X,Y)-(X+A,Y+H),C9,G:X=X+
S
200 GET(X,Y)-(X+A,Y+H),CA,G:X=X+
S
210 GET(X,Y)-(X+A,Y+H),CB,G:X=X+
S
220 GET(X,Y)-(X+A,Y+H),CC,G:X=X+
S
230 GET(X,Y)-(X+A,Y+H),CD,G:X=X+
S
240 GET(X,Y)-(X+A,Y+H),CE,G:X=X+
S
250 GET(X,Y)-(X+A,Y+H),CF,G:X=X+
S
260 GET(X,Y)-(X+A,Y+H),CG,G:X=X+
S
270 GET(X,Y)-(X+A,Y+H),CH,G:X=X+
S
280 GET(X,Y)-(X+A,Y+H),CI,G:X=X+
S
290 GET(X,Y)-(X+A,Y+H),CJ,G:X=X+
S
300 GET(X,Y)-(X+A,Y+H),CK,G:X=X+
S
310 GET(X,Y)-(X+A,Y+H),CL,G:X=X+
S
320 GET(X,Y)-(X+A,Y+H),CM,G:X=X+
S
330 GET(X,Y)-(X+A,Y+H),CN,G:X=X+
S
340 GET(X,Y)-(X+A,Y+H),CO,G:X=X+
S
```

```
350 GET(X,Y)-(X+A,Y+H),CP,G:X=X+
S
360 GET(X,Y)-(X+A,Y+H),CQ,G:X=X+
S
370 GET(X,Y)-(X+A,Y+H),CR,G:X=X+
S
380 GET(X,Y)-(X+A,Y+H),CS,G:X=X+
S
390 GET(X,Y)-(X+A,Y+H),CT,G:X=X+
S
400 GET(X,Y)-(X+A,Y+H),CU,G:X=X+
S
410 GET(X,Y)-(X+A,Y+H),CV,G:X=X+
S
420 GET(X,Y)-(X+A,Y+H),CW,G:X=X+
S
430 GET(X,Y)-(X+A,Y+H),CX,G:X=X+
S
440 GET(X,Y)-(X+A,Y+H),CY,G:X=X+
S
450 GET(X,Y)-(X+A,Y+H),CZ,G:X=X+
S
```

Selección del formato de pantalla

Una vez que todos los caracteres están guardados y seguros en sus tablas correspondientes, la pantalla puede borrarse e inicializar nuevas variables que controlen el formato de pantalla. X e Y son las coordenadas iniciales, S el incremento a lo largo del eje X, T el incremento a lo largo del eje Y, y XI, YI, XF e YF los valores límites de las coordenadas para los ejes X e Y. Aunque el tamaño de los caracteres es fijo, la S controla la cantidad de espacio entre caracteres sobre el eje X, y por lo tanto el número de caracteres por línea. T controla la cantidad de espacio entre líneas de caracteres, y por lo tanto el número de líneas que puedan incluirse en la pantalla. Si sólo debe escribirse en una parte de la pantalla, entonces deben modificarse los valores de XI, YI, XF e YF. La combinación de los valores dados produce una matriz de 42 por 24 caracteres (un total de 1ØØ8, casi el doble del número que puede generarse en la pantalla de textos normal del Dragon). Aunque se necesitan varios segundos para llenar inicialmente las tablas, éstas se mantendrán mientras no se utilice la orden RUN. Si se pierde el programa, asegúrese de que lo reinicializa utilizando GOTO.

```
500 PCLS:X=2:Y=0:S=6:R=8:XI=2:XF
=253:YI=0:YF=191
```


El cursor y la comprobación de teclas

T$ se llena con el valor obtenido con INKEY$ y se genera un cursor parpadeante, en este caso mediante un doble PUT de la tabla en blanco, con un NOT para invertir la pantalla dos veces. El primer PUT, NOT invierte el estado de todos los puntos del área, y el segundo PUT, NOT los invierte de nuevo de forma que quedan como al principio. Cuando se pulsa una tecla se obtiene el valor ASCII correspondiente y si éste está por encima o por debajo del código de los caracteres actuales, el programa salta a la línea 15ØØ.

```
510 T$=INKEY$:PUT(X,Y)-(X+A,Y+H)
,BL,NOT:PUT(X,Y)-(X+A,Y+H),BL,NO
T:IF T$="" THEN 510 ELSE T=ASC(T
$):IF T<47 OR T>91 THEN 1500
```

Las otras teclas se clasifican mediante una orden ON GOSUB relativa a sus códigos. Seis de los caracteres que faltan, cuyos códigos están entre los de los números y letras mayúsculas, producen de nuevo un salto a la rutina que contiene INKEY$, pero @ salta hacia otra rutina colocada en la línea 161Ø.

```
520 ON T-47 GOSUB 1000,1010,1020
,1030,1040,1050,1060,1070,1080,1
090,510,510,510,510,510,510,1610
,1100,1110,1120,1130,1140,1150,1
160,1170,1180,1190,1200,1210,122
0,1230,1240,1250,1260,1270,1280,
1290,1300,1310,1320,1330,1340,13
50
```

Colocación (PUT) de los caracteres

Existe una subrutina con PUT, PSET para cada carácter y después el programa vuelve (RETURN). Podrá ahorrarse parte del tecleado si guarda el programa actual (CSAVE), borra todo, excepto las líneas que tienen GET, y después añade de nuevo el fragmento al programa anterior. El proceso para añadir un programa es: primero utilizar la orden PEEK en las posiciones 27 y 28 y después colocar mediante la orden POKE en la posición 25 el número que había en la 27 y en la posición 26 el número que había en la 28 menos 2. Esto coloca el puntero de «inicio del programa en BASIC» por encima del final del programa que queda en la memoria, con lo que puede cargarse de forma segura otra copia encima de ésta. Ahora deberá renume-

rar el programa a partir de las líneas originales y entonces, mediante POKE 25, 3Ø y POKE 26,1 se restablece el puntero de inicio, dejándolo donde estaba, con lo que los dos conjuntos de líneas formarán un único programa. Ahora podrá editar las copias de las líneas con GET para convertirlas en líneas con PUT.

```
1000 PUT(X,Y)-(X+A,Y+H),C0,PSET:
RETURN
1010 PUT(X,Y)-(X+A,Y+H),C1,PSET:
RETURN
1020 PUT(X,Y)-(X+A,Y+H),C2,PSET:
RETURN
1030 PUT(X,Y)-(X+A,Y+H),C3,PSET:
RETURN
1040 PUT(X,Y)-(X+A,Y+H),C4,PSET:
RETURN
1050 PUT(X,Y)-(X+A,Y+H),C5,PSET:
RETURN
1060 PUT(X,Y)-(X+A,Y+H),C6,PSET:
RETURN
1070 PUT(X,Y)-(X+A,Y+H),C7,PSET:
RETURN
1080 PUT(X,Y)-(X+A,Y+H),C8,PSET:
RETURN
1090 PUT(X,Y)-(X+A,Y+H),C9,PSET:
RETURN
1100 PUT(X,Y)-(X+A,Y+H),CA,PSET:
RETURN
1110 PUT(X,Y)-(X+A,Y+H),CB,PSET:
RETURN
1120 PUT(X,Y)-(X+A,Y+H),CC,PSET:
RETURN
1130 PUT(X,Y)-(X+A,Y+H),CD,PSET:
RETURN
1140 PUT(X,Y)-(X+A,Y+H),CE,PSET:
RETURN
1150 PUT(X,Y)-(X+A,Y+H),CF,PSET:
RETURN
1160 PUT(X,Y)-(X+A,Y+H),CG,PSET:
RETURN
1170 PUT(X,Y)-(X+A,Y+H),CH,PSET:
RETURN
1180 PUT(X,Y)-(X+A,Y+H),CI,PSET:
RETURN
1190 PUT(X,Y)-(X+A,Y+H),CJ,PSET:
RETURN
```



```
1200 PUT(X,Y)-(X+A,Y+H),CK,PSET:
RETURN
1210 PUT(X,Y)-(X+A,Y+H),CL,PSET:
RETURN
1220 PUT(X,Y)-(X+A,Y+H),CM,PSET:
RETURN
1230 PUT(X,Y)-(X+A,Y+H),CN,PSET:
RETURN
1240 PUT(X,Y)-(X+A,Y+H),CO,PSET:
RETURN
1250 PUT(X,Y)-(X+A,Y+H),CP,PSET:
RETURN
1260 PUT(X,Y)-(X+A,Y+H),CQ,PSET:
RETURN
1270 PUT(X,Y)-(X+A,Y+H),CR,PSET:
RETURN
1280 PUT(X,Y)-(X+A,Y+H),CS,PSET:
RETURN
1290 PUT(X,Y)-(X+A,Y+H),CT,PSET:
RETURN
1300 PUT(X,Y)-(X+A,Y+H),CU,PSET:
RETURN
1310 PUT(X,Y)-(X+A,Y+H),CV,PSET:
RETURN
1320 PUT(X,Y)-(X+A,Y+H),CW,PSET:
RETURN
1330 PUT(X,Y)-(X+A,Y+H),CX,PSET:
RETURN
1340 PUT(X,Y)-(X+A,Y+H),CY,PSET:
RETURN
1350 PUT(X,Y)-(X+A,Y+H),CZ,PSET:
RETURN
```

Cuando ejecute este programa, sin duda quedará impresionado por la velocidad de las órdenes GET y PUT, que parece que operen instantáneamente, y en verdad con más rapidez de lo que usted pueda teclear. La velocidad es la principal ventaja de éste método de generación de caracteres, pero naturalmente esto debe ser a costa de la imposibilidad de escalar y colorear los caracteres, o de cambiar el ángulo sobre la pantalla. En realidad es un método muy útil para obtener una cantidad razonable de texto sobre la pantalla de una vez. No esté tentado de guardar parte de lo escrito utilizando el GET sin detalle gráfico y el PUT sin acción, ya que hemos encontrado que esta combinación, muy conveniente en teoría, en la práctica no funciona con mucha efectividad. Debería aparecer cierta mejora en la velocidad, respecto al método descrito antes, pero en la práctica hemos

encontrado que en realidad tiene tendencia a perderse, dando misteriosos FC ERROR si se escribe de prisa.

Una vez que se ha colocado un carácter, la posición X se incrementa. Si se ha alcanzado el límite del eje X (X>XF) entonces la coordenada X se reinicializa a la posición inicial, pero la coordenada Y se mueve hacia abajo hasta la siguiente línea. Si se alcanza el final de la pantalla deberá realizarse la acción adecuada.

```
540 X=X+S:IF X>XF THEN X=1:Y=Y+R
:IF Y>YF THEN 2000
550 GOTO 510
2000 STOP
```

Movimiento

La barra de espacio (código 32) produce un movimiento en blanco hacia la derecha, y en esta línea también se coloca de nuevo la tabla en blanco mediante PUT PRESET. Esto produce el efecto de borrar la pantalla en la posición actual, por lo que también se utiliza para borrar. Obsérvese que esta tabla no se ha llenado nunca, por lo que es más adecuado el PRESET que el PSET.

```
1500 IF T=32 THEN PUT(X,Y)-(X+A,
Y+H),BL,PRESET:X=X+S:GOTO 510
```

Las teclas con flecha también pueden utilizarse para moverse sobre la pantalla en las cuatro direcciones.

```
1510 IF T=8 THEN X=X-S:GOTO 1560
1520 IF T=9 THEN X=X+S:GOTO 1560
1530 IF T=94 THEN Y=Y-R:GOTO 156
0
1540 IF T=10 THEN Y=Y+R:GOTO 156
0
1550 GOTO 510
1560 IF X<XP THEN X=XP
1570 IF X>XF THEN X=XP:Y=Y+R
1580 IF Y<YP THEN Y=YP
1590 IF Y>YF THEN Y=Y-R
1600 GOTO 510
```

Otro caso

Si se quieren minúsculas verdaderas, tendrá que cargar más caracteres y duplicar los GET y PUT, pero es fácil que se produzca una


```
ESTO ES UN EJEMPLO DEL TEXTO PRODUCIDO EN
LA PANTALLA DE ALTA RESOLUCION MEDIANTE
GET Y PUT DE LOS CARACTERES GUARDADOS EN
CINTA COMO UN ARCHIVO EN CODIGO MAQUINA DE
LAS PAGINAS GRAFICAS

012345678901234567890123456789012345678901

EN ESTE FORMATO PARTICULAR HAY CUARENTA Y
DOS CARACTERES EN UNA LINEA Y VENTICUATRO
LINEAS EN LA PANTALLA

PUEDEN PRODUCIRSE CARACTERES INVERTIDOS
```

Fig. 13.2 Visualización en 42 × 24.

alternativa con los caracteres invertidos (fig. 13.2). Si quieren conseguir caracteres inversos tan sólo hay que pulsar @ que conduce a la línea 161Ø que pasa de un formato a otro mediante el flag (FL).

```
1610 IF FL=1 THEN FL=0:RETURN EL
SE FL=1:RETURN
```

Ahora se coloca otra línea en la cual se utiliza la tabla en blanco para colocarla sobre el carácter actual con NOT, invirtiendo así todos los puntos de la pantalla.

```
530 IF FL=1 THEN PUT(X,Y)-(X+A,Y
+H+1),BL,NOT
```

Más o menos caracteres

La visualización con 42 por 24 caracteres que acabamos de describir es la mayor que puede tratarse de manera confortable (fig. 13.2). El espacio entre caracteres puede reducirse disminuyendo S a 5, lo que da 51 caracteres por línea (1224 por pantalla), pero esto es llegar a una situación muy límite (fig. 13.3). El aumentar la S hasta 7 reduce

```
012345678901234567890123456789012345678901234567890

EN EL CASO EXTREMO PUEDEN LLEGARSE A INCLUIR HASTA
UN MAXIMO DE CINCUENTA Y UN CARACTERES EN PANTALLA
AUNQUE LA LEGIBILIDAD DISMINUYE BASTANTE SI EL
TAMAÑO DEL STEP SE REDUCE HASTA ESTE PUNTO.
```

Fig. 13.3 Visualización en 51 × 24.

```
012345678901234567890123456789012345

INCLUSO ESTE FORMATO AMPLIAMENTE
ESPACIADO DE TREINTA Y SEIS COLUMNAS
TIENE VENTICUATRO LINEAS EN LA
PANTALLA Y POR LO TANTO UN TOTAL DE
864 CARACTERES POR PANTALLA EN
LUGAR DE LOS 512 QUE NORMALMENTE
TIENE LA PANTALLA DE TEXTO DEL
DRAGON.
```

Fig. 13.4 Visualización en 36 × 24.

el número de caracteres por línea a 36 (fig. 13.4) y si la S es 8 estamos de nuevo con los 32 caracteres por línea, que es lo normal de la pantalla de texto del Dragon, aunque por el hecho de que siga habiendo 24 líneas se tiene un total de 768 caracteres sobre la pantalla en lugar de 512. Sin duda debe existir un compromiso entre la legibilidad y la cantidad y mediante esta técnica puede elegirse en función de los factores externos.

14. Cómo trabajar con una retícula

Aunque es posible realizar dibujos libres sobre la pantalla, cierto tipo de sistema con retícula proporciona una guía muy útil cuando se quiere estar seguro de que el dibujo sigue un formato determinado. Esto es muy importante cuando se quieren definir conjuntos de caracteres o imágenes de dibujos animados. El sistema de retícula que se describe aquí es una derivación mucho más potente del dibujo sobre papel cuadriculado y es una idea que elimina las gomas de borrar y el Tipp-ex.

Elección

Antes de construir una retícula sobre la pantalla habrá que decidir qué PMODE y qué colores se quieren utilizar, así como el tamaño, forma y escala de la retícula escogida.

```
20 CLS:PRINT"PMODE";:INPUT PM:PR
INT"CONJUNTO DE COLORES";:INPUT
CC:PRINT"COLOR PRINCIPAL";:INPUT
 Q1:PRINT"COLOR DE FONDO";:INPUT
 Q2:CLS:PRINT"ANCHURA DE LA RETI
CULA";:INPUT A:PRINT"ALTURA DE L
A RETICULA";:INPUT H:PRINT"ESCAL
A";:INPUT SX
```

Formación de la retícula

Ahora estableceremos la pantalla y definiremos la posición inicial de la retícula XI, YI.

```
30 PMODE PM,1:SCREEN 1,CC:PCLS Q
2
40 XI=10:YI=50:SY=SX
```

Con XI=1Ø e YI=5Ø, la retícula empieza aproximadamente a un cuarto hacia abajo de la parte izquierda de la pantalla y un poco separado del extremo izquierdo. Para que cada elemento de la retícula sea

un cuadrado, el factor de escala para el eje Y (SY) debe ser el mismo que para el eje X (SX).

Las coordenadas finales de la retícula (XF, YF) se calculan multiplicando la anchura de la retícula (A) por el factor de escala del eje X (SX), y la altura de la retícula (H) por el factor de escala del eje Y (SY). A continuación debe hacerse una comprobación para asegurarse que el área calculada cabe en la pantalla.

Si esta comprobación falla, el programa se ejecuta de nuevo. La posición actual en la pantalla (XP, YP) se coloca en el comienzo de la retícula (XI, YI).

```
50 XF=XI+(SX*A):YF=YI+(SY*H):XF=
XI:YP=YI:IF XF>190 OR YF>180 THE
N RUN
```

El límite XF de 19Ø que se ha especificado deja un área libre a la derecha de la pantalla. Para hacer pruebas elijamos PMODE 4,1, Color principal 1, Color de fondo Ø, anchura 1Ø, altura 1Ø, escala 1Ø.

La retícula seleccionada podrá ahora dibujarse mediante una serie de líneas (LINE) entre el principio y el final, separadas en función del factor de escala.

```
60 COLOR Q1,Q2:N=YI:FOR M=XI TO
XF STEP SX:LINE(M,N)-(M,N+(YF-YI
)),PSET:NEXT M:M=XI:FOR N=YI TO
YF STEP SY:LINE(M,N)-(M+(XF-XI),
N),PSET:NEXT N
```

Cursor parpadeante

Necesitaremos un cursor para indicar la posición en la tabla, y ya que éste se forma colocando PUT, primero deberá dimensionarse una tabla en blanco (B).

```
10 DIM B(10)
```

El cursor parpadeante se forma mediante la orden PUT de la tabla en blanco sobre la posición actual de la pantalla, utilizándola dos veces con NOT. El primer NOT invierte este sector de la retícula y el segundo lo invierte de nuevo, produciendo la visualización original.

```
80 FL=5:FOR R=1 TO 2:PUT(XP,YP)-
(XP+SX,YP+SY),B,NOT:FOR Q=1 TO R
^FL:NEXT Q:NEXT R:IF PEEK(337)=2
55 THEN 80 ELSE T=PEEK(135)
```


Mientras no se pulse ninguna tecla (PEEK(337)=255) esta secuencia se irá repitiendo. Para conseguir que el estado del sector de la retícula que está debajo del cursor sea más visible hay un bucle de espera que es función de si estamos en el primero (R=1) o segundo (R=2) NOT. La velocidad del parpadeo también está relacionada con la variable (FL), que tiene profundos efectos, ya que el cálculo del tiempo de espera es exponencial (RFL). Un valor para FL de 5 produce un efecto razonable. Cuando se pulsa una tecla el valor de PEEK(135) se guarda en T.

Movimiento sobre la retícula

El movimiento de las teclas de control del cursor, y los límites se comprueben continuamente, de forma que no sea posible salirse de la retícula.

```
160 IX=((T=8)-(T=9))
170 IY=((T=94)-(T=10))
180 XP=XP+(IX*SX):YP=YP+(IY*SY)
190 IF XP>XF-SX THEN XP=XF-SX:GO
TO 80 ELSE IF XP<XI THEN XP=XI:G
OTO 80
200 IF YP>YF-SY THEN YP=YP-SY:GO
TO 80 ELSE IF YP<YI THEN YP=YI:G
OTO 80
210 GOTO 80
```

Obsérvese que el tamaño del movimiento está relacionado con los factores de escala (SX y SY).

Rellenado de la retícula y correcciones

Para rellenar los sectores de la retícula utilizaremos otra tabla (W) que, en principio, está rellena con el contenido de la pantalla en las coordenadas iniciales.

```
10 DIM W(10):DIM B(10)
70 GET(XI,YI)-(XI+SX,YI+SY),W,G:
POKE 135,0
```

(La orden POKE 135,Ø al final es para cancelar la autorrepetición cuando después se realiza un nuevo dibujo de la retícula.)

El código de tecla escogido para el rellenado es el de la barra de espacio (32), que fue elegido por ser la orden más frecuente.

```
100 IF T=32 THEN PUT(XP,YP)-(XP+
SX,YP+SY),W,PRESET:GOTO 80
```

Cuando se coloca la tabla W, mediante la orden PRESET se invierte la visualización. Entonces el programa vuelve de nuevo a la rutina del cursor.

Para quitar un bloque de la retícula se utiliza el PUT, PSET con la misma tabla si se pulsa la X.

```
110 IF T=88 THEN PUT(XP,YP)-(XP+
SX,YP+SY),W,PSET:GOTO 80
```

Cómo hacer una copia «real»

Las aplicaciones más interesantes de este programa son la creación de caracteres y de dibujos animados, por lo que tendremos de ser capaces de transferir nuestras ideas desde la retícula a la pantalla real. Esto puede realizarse mediante PSET y PRESET de los puntos adecuados y con una copia en miniatura producida a la derecha de la retícula a partir de las coordenadas CX, CY.

```
40 XI=10:YI=50:SY=SX:FL=5:CX=200
:CY=90:XC=CX:YC=CY
```

La posición actual es XC, YC y éstas se actualizan si el movimiento está dentro de la retícula, ya que tan sólo se llega a la línea 21Ø después de un movimiento válido.

```
210 XC=XC+IX:YC=YC+IY:GOTO 80
```

El PSET y PRESET de XC, YC se añaden a las anteriores líneas de llenado y borrado.

```
100 IF T=32 THEN PUT(XP,YP)-(XP+
SX,YP+SY),W,PRESET:PSET(XC,YC):G
OTO 80
110 IF T=88 THEN PUT(XP,YP)-(XP+
SX,YP+SY),W,PSET:PRESET(XC,YC):G
OTO 80
```

Si ahora ejecuta de nuevo este programa verá que todas las acciones llevadas a cabo sobre la retícula se trasladan a una versión más pequeña, a la derecha de la pantalla.

Cómo guardar las copias

Cuando haya construido una copia satisfactoria podrá guardarla en la parte superior de la pantalla pulsando «@». Esto producirá que

se obtenga una copia (GET) de la parte derecha de la pantalla y se guarde en la tabla CH que posteriormente se colocará (PUT) de nuevo en el cuarto superior.

```
10 DIM W(10):DIM B(10):DIM CH(50
0)
40 XI=10:YI=50:SY=SX:FL=5:CX=200
:CY=90:XC=CX:YC=CY:C1=0:C2=0
120 IF T=64 THEN GET(CX,CY)-(CX+
A,CY+H),CH,G:PUT(C1,C2)-(C1+A,C2
+H),CH,PSET:C1=C1+A:GOTO 60
```

Las coordenadas iniciales utilizadas en la orden PUT están predefinidas como C1, C2 y la posición sobre el eje X (C1) se mueve a lo largo del eje un número de unidades equivalente a la anchura de la retícula (A) depués de cada orden PUT.

Cuando esta rutina vuelva a la línea 6Ø volverá a dibujar la retícula, pero no la borrará. Esto es de mucha utilidad si se quiere hacer una serie de dibujos animados (véase más adelante). Cuando se vuelva a dibujar la retícula, los espacios entre bloques desaparecen, de forma que el estado es evidente.

Empezar de nuevo

Si decide que no le gusta el contenido de la retícula, y quiere borrarlo, pulsando la tecla CLEAR obtendrá un borrado parcial de la pantalla.

```
130 IF T=12 THEN FOR P=2 TO 4:PM
ODE 0,P:PCLS 02:NEXT P:PMODE PM,
1:GOTO 60
```

Tan sólo se borran las páginas de la 2 a la 4 ya que PCLS se hace en PMODE Ø. Esta orden se utiliza también si se quiere borrar una retícula después de guardar una copia en la parte superior de la pantalla. Ya que estas copias están en la página 1 no quedan afectadas por la rutina de CLEAR. Si realmente quiere destruir las copias guardadas pulse el 3 para borrar la página 1.

```
140 IF T=51 THEN PMODE 0,1:PCLS
02:PMODE 4,1:POKE 135,0:GOTO 60
```

Si decide que incluso el tamaño de la retícula es incorrecto, entonces pulse la tecla «1» para ejecutar de nuevo el programa.

```
90 IF T=49 THEN RUN
```

Inversión

Existe la posibilidad de obtener una inversión parcial, tanto del contenido de la retícula como de las áreas copiadas. El área de la copia tiene encima la tabla CH colocada mediante PUT, NOT, mientras que la retícula está invertida debido a la utilización repetida de la orden PUT sobre la tabla W con NOT, mientras se mueve hacia abajo el área de pantalla adecuada. Si se repite esta acción de esta forma, puede utilizarse una tabla menor de la que sería necesario de otra manera.

```
150 IF T=73 THEN FOR N=YI-SY TO
YF+SY:PUT(XI-SX,N)-(XF+SX,N),W,N
OT:NEXT N:PUT(CX,CY)-(CX+A,CY+H)
,CH,NOT:GOTO 80
```

Esto permite visualizar y guardar una copia invertida, pero la pantalla debe borrarse antes de continuar.

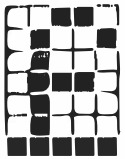
Cómo guardarlas

Cuando haya formado todos los caracteres o figuras que necesite podrá guardar el contenido de la pantalla mediante la orden CSAVEM como se ha descrito antes.

Aplicaciones

Cualquier tipo de figuras pueden crearse mediante este sistema, y la retícula puede ser pequeña o grande. De hecho, esta rutina se ha utilizado para generar material para algunos otros capítulos. Los caracteres de texto se definen fácilmente; por ejemplo la figura 14.1

```
abcdefghijklmnopqrstuvwxyz
```



£

Fig. 14.1 Generación de minúsculas y de caracteres especiales.

muestra un conjunto de letras minúsculas que pueden guardarse, además del signo de la libra. El logotipo del Dragon está formado también en la figura 14.2, y un tractor en la figura 14.3. Dejamos para usted el añadir el resto de los elementos de una granja. La generación de una serie de dibujos animados se describen en otras partes del libro.

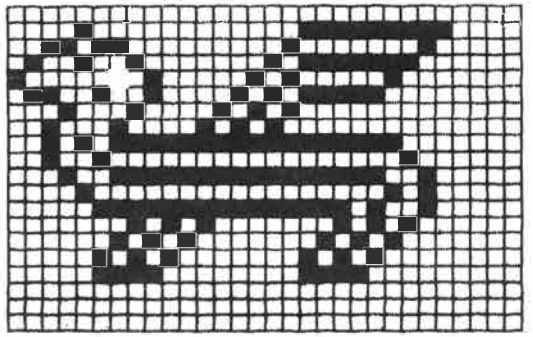


Fig. 14.2 Logotipo del Dragon.

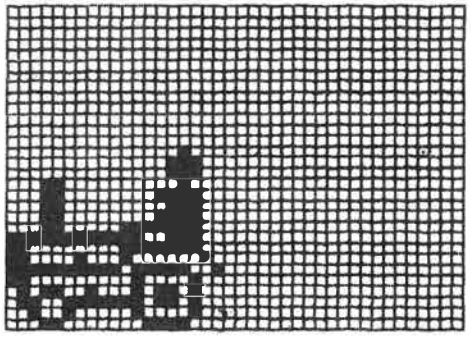


Fig. 14.3 Tractor.

15. Animación

Corredor

El siguiente paso, después del sencillo movimiento de cosas sobre la pantalla, es animar un dibujo, es decir, mover partes del mismo para dar la impresión de que está vivo. Una vez más analizaremos el planteo de iluminar puntos mediante la orden SET y considerar la producción del efecto de una figura que está corriendo. En primer lugar hemos diseñado dos figuras alternativas, la primera estacionaria, mirando hacia adelante, y la segunda corriendo, mirando hacia la derecha (fig. 15.1).

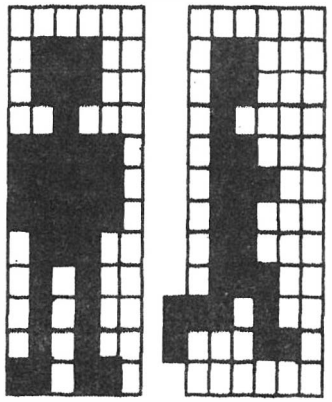


Fig. 15.1 Corredor.

Las coordenadas están en sentencias de DATA y leídas (READ) en tablas, al igual que antes, pero en este caso existen realmente dos conjuntos alternativos de coordenadas. Los primeros 35 puntos (en la línea 5Ø2Ø) constituyen una figura (estática), y los puntos desde el 36 al 59 (en la línea 5Ø3Ø) la otra (en movimiento). Tan sólo se necesitan dos tablas, ya que podemos tomar cualquier punto de una tabla en cualquier momento y no debemos empezar por el principio de los datos cada vez. En las líneas 1ØØØ y 2ØØØ están las subrutinas separadas para dibujar cada imagen.

Si se sigue el orden de los puntos de las sentencias DATA y después se analiza este programa en funcionamiento, verá que los sencillos efectos de animación se consiguen gracias a que los puntos correspondientes a las piernas se iluminan de una forma relativamente lenta y en una secuencia determinada, de forma que una pierna aparece antes que la otra. No existe ningún motivo para convertir estos puntos, que deben iluminarse a su equivalente en CHR$ ya que el aumento en velocidad enmascararía aquí el efecto del movimiento.

La secuencia de operación es la siguiente. Primero se visualiza el título y si no se pulsa ninguna tecla entonces se visualiza la primera figura mediante la subrutina de la línea 1ØØØ. Si se pulsa una tecla, el programa salta a la línea 12Ø que actualiza el desplazamiento en la pantalla (XO), borra la imagen anterior, va a la subrutina que dibuja la segunda imagen (2ØØØ), borra de nuevo la pantalla y vuelve a visualizar el título.

```
10 GOSUB 5000
20 CLS0
30 XO=2:YO=0:C=2
100 PRINT @256,"CORREDOR"
110 IF PEEK(337)=255 THEN GOSUB
1000:GOTO 110
120 XO=XO+1:CLS0:GOSUB 2000:CLS0
:GOTO 100
1000 FOR N=1 TO 35:SET(X(N)+XO,Y
(N)+YO,C):NEXT N:RETURN
2000 FOR N=36 TO 59:SET(X(N)+XO,
Y(N)+YO,C):NEXT N:SOUND 1,1:RETU
RN
5000 DIM X(59),Y(59)
5010 FOR N=1 TO 59:READ X(N),Y(N
):NEXT N:RETURN
5020 DATA 1,1,2,1,3,1,1,2,2,2,3,
2,2,3,0,4,1,4,2,4,3,4,4,4,0,5,1,
5,2,5,3,5,4,5,0,6,1,6,2,6,3,6,4,
6,1,7,2,7,3,7,1,8,3,8,1,9,3,9,1,
10,3,10,0,11,1,11,3,11,4,11
5030 DATA 1,1,1,2,2,1,2,2,1,3,1,
4,1,5,1,6,1,7,2,4,2,5,2,6,2,7,3,
5,1,8,2,8,3,8,3,9,3,10,4,10,1,9,
0,9,-1,9,-1,10
```

Una alternativa del CLSØ es utilizar un único texto de 192 caracteres (BL$) para borrar sólo la parte superior de la pantalla (posiciones de visualización desde Ø a 191).

```
40 BL$=STRING$(192,128)
120 XO=XO+1:PRINT @0,BL$;:GOSUB
2000:PRINT @0,BL$;:GOTO 110
```

Sprinter

El corredor que acabamos de describir parecía moverse debido a la lentitud de las órdenes SET y RESET, pero también es posible utilizar las técnicas descritas para ello, mediante PSET y PRESET, en alta resolución. Sin embargo, puede generarse una animación mucho más precisa en alta resolución, si se utilizan las órdenes GET y PUT, aunque naturalmente será necesario realizar los dibujos sobre los cuales actuarán las órdenes GET y PUT. Las figuras 15.2 y 15.3 muestran dos imágenes del movimiento de un Sprinter que pueden formarse mediante la orden PSET sobre las coordenadas dadas en las sentencias de DATA.

```
10 DATA 4,0,5,0,6,0,7,0,4,1,5,1,
6,1,7,1,4,2,5,2,6,2,7,2,4,3,5,3,
6,3,7,3,4,4,5,4,6,4,7,4,5,5,6,5,
4,6,5,6,6,6,7,6,4,7,5,7,7,7,3,8,
4,8,7,8,2,9,3,9,4,9,7,9,2,10,4,1
0,7,10,8,10,9,10,10,10,11,10,1,1
1,2,11,4,11,7,11,8,11,9,11,10,11
,11,11,12,11,2,12,3,12
20 DATA 4,12,5,12,6,12,7,12,3,13
,4,13,5,13,6,13,7,13,4,14,7,14,4
,15,5,15,6,15,7,15,4,16,5,16,6,1
6,7,16,4,17,5,17,6,17,7,17,8,17,
9,17,10,17,4,18,5,18,6,18,7,18,8
,18,9,18,10,18,5,19,6,19,10,19,1
1,19,5,20,6,20,10,20,11,20,5,21,
6,21,10,21,11,21,5,22,6,22
30 DATA 10,22,11,22,12,22,13,22,
5,23,6,23,10,23,11,23,12,23,13,2
3,5,24,6,24,5,25,6,25,5,26,6,26,
7,26,8,26,5,27,6,27,7,27,8,27
40 DATA 55,0,56,0,57,0,58,0,55,1
,56,1,57,1,58,1,55,2,56,2,57,2,5
```



```
8,2,55,3,56,3,57,3,58,3,55,4,56,
4,57,4,58,4,56,5,57,5,55,6,56,6,
57,6,58,6,55,7,57,7,58,7,54,8,55
,8,57,8,58,8,53,9,54,9,55,9,57,9
,58,9,53,10,55,10,57,10,58,10,59
,10,60,10,61,10,62,10
50 DATA 52,11,53,11,55,11,57,11,
58,11,59,11,60,11,61,11,62,11,63
,11,53,12,54,12,55,12,58,12,54,1
3,55,13,58,13,55,14,58,14,55,15,
56,15,57,15,58,15,55,16,56,16,57
,16,58,16,55,17,56,17,57,17,58,1
7,55,18,56,18,57,18,58,18,59,18,
56,19,57,19,59,19,60,19
60 DATA 56,20,57,20,60,20,61,20,
56,21,57,21,61,21,62,21,51,22,52
,22,53,22,54,22,55,22,56,22,57,2
2,61,22,62,22,51,23,52,23,53,23,
54,23,55,23,56,23,57,23,61,23,62
,23,51,24,52,24,61,24,62,24,51,2
5,52,25,61,25,62,25,63,25,64,25,
61,26,62,26,63,26,64,26
```

```
80 PMODE 4,1:SCREEN 1,0:PCLS
90 FOR N=1 TO 249
100 READ X,Y:PSET(X+10,Y)
110 NEXT N
```

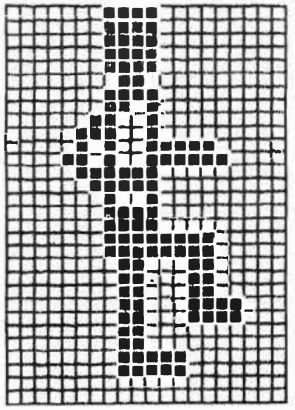


Fig. 15.2 Sprinter. Imagen 1.

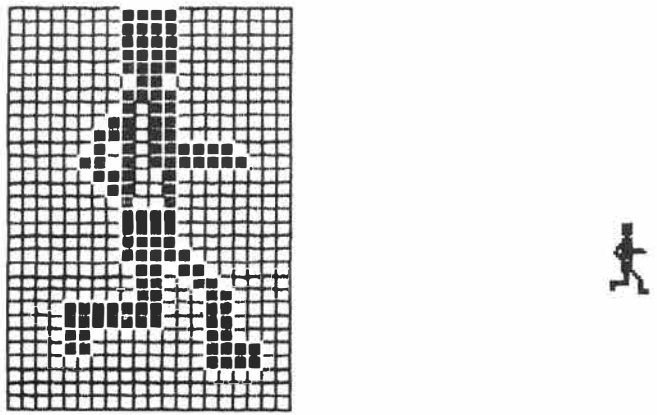


Fig. 15.3 Sprinter. Imagen 2.

Una vez que se han colocado (PSET) las dos imágenes (un trabajo muy lento, pero al menos sólo hay que hacerlo una vez) podrán colocarse (GET) en las tablas F1 y F2, y borrar (PCLS) las figuras que se acaban de realizar a fin de prepararlo todo para la secuencia de animación.

```
70 DIM F1(50):DIMF2(50)
120 GET(5,0)-(30,27),F1,G
130 GET(55,0)-(80,27),F2,G
140 PCLS
```

La secuencia más sencilla consiste en utilizar el PSET sobre cada tabla por turno, de forma que la figura corra sobre la parte situada en la mitad del lado izquierdo de la pantalla.

```
180 PUT(X,100)-(X+25,127),F1,PSE
T
200 PUT(X,100)-(X+25,127),F2,PSE
T
230 GOTO 180
```

Si ahora se incrementa la X en un bucle FOR...NEXT, se conseguirá que corra a través de la pantalla desde la izquierda a la derecha. Obsérvese que cada imagen se muestra a cada incremento de X, antes de que se actualice la X. La figura 15.4 muestra las dos imágenes congeladas alternativamente en un número determinado de posiciones de pantalla.

```
160 FOR X=1 TO 230 STEP 5
220 NEXT X
230 GOTO 140
```


Fig. 15.4 Sprinters congelados.

Se moverá de una forma bastante correcta y relativamente rápida a través de la pantalla, pero ¿qué sucede si existe un fondo visible detrás de él? Añada algunas líneas horizontales para generar un fondo de prueba y ejecútelo de nuevo.

```
150 FOR LI=100 TO 130 STEP 5:LIN
E(0,LI)-(255,LI),PSET:NEXT LI
```

Como puede ver en la figura 15.5, las líneas desaparecen a medida que el hombre corre por encima de ellas, lo que no es de mucha utilidad en un programa real. Lo que podemos hacer es obtener (GET) el fondo antes de colocar la figura y después colocar de nuevo el fondo cuando se haya movido. Tan sólo obtendremos el fondo una vez para ambas imágenes, ya que es el fondo original el que debemos colocar de nuevo. Si reproducimos otra vez el fondo mediante PSET, reaparecen las líneas (fig. 15.6), pero hay mucho parpadeo y mientras se realizan las imágenes, parte de las líneas quedan borradas.

```
70 DIM F1(50):DIMF2(50):DIM FO(2
0)
170 GET(X,100)-(X+25,127),FO,G
210 PUT(X,100)-(X+25,127),FO,PSE
T
```

Fig. 15.5 Borrado del fondo.

Fig. 15.6 Colocación mediante PUT, PSET de la figura sobre el fondo.

Para obtener una sustitución mejor del fondo (fig. 15.7) tendremos que hacer las cosas un poco más complicadas, y aplicar algunas acciones lógicas en nuestras órdenes PUT.

```
180 PUT(X,100)-(X+25,127),F1,OR
190 PUT(X,100)-(X+25,127),FO,AND
200 PUT(X,100)-(X+25,127),F2,OR
210 PUT(X,100)-(X+25,127),FO,AND
```

Fig. 15.7 La respuesta lógica.

Primero colocamos la primera imagen (F1) sobre el fondo (FO) con una OR. Esto nos da el fondo más el dibujo 1, ya que todos los puntos que estaban iluminados en la tabla o en la pantalla también quedan iluminados. A continuación colocamos de nuevo el fondo (FO) mediante un AND, de forma que únicamente los puntos comunes a ambos, la pantalla actual y la pantalla original, continúan iluminados. Esto genera la posición original con lo que podemos entonces colocar el segundo dibujo (F2) mediante una OR y después hacer una AND con el fondo (FO) como hicimos con la figura 1. Obsérvese que es esencial el colocar de nuevo el fondo entre ambos dibujos si se quieren evitar problemas con la Federación Internacional de Atletismo respecto a corredores con tres piernas.

Volando alto

El grado de realismo en un dibujo animado depende de la exactitud de los dibujos, pero también del número de dibujos en la secuencia. Como ejemplo, las figuras 15.8a-e muestran una serie de cuatro dibujos distintos de un pájaro volando. La forma más fácil de generar una serie como ésta, de dibujos relacionados entre ellos, es generarlos sobre un sistema de retícula de pantalla como el que se ha descrito. La primera imagen muestra las alas en su posición más alta y cuando se pulse la @ para transferir el dibujo ya terminado a la parte superior de la pantalla (fig. 15.8b), de nuevo se dibujarán las líneas de la retícula, de manera que los puntos dibujados formen ahora una estructura sólida. Esta característica hace que sea más sencillo construir el dibujo de cada imagen siguiente, ya que puede modificarse fácilmente el dibujo existente, pero seguir viendo qué partes han sido modificadas. Una vez realizadas todas las imágenes, puede guardarse (CSAVEM) la parte superior de la pantalla y al cargarla de nuevo utilizar las órdenes GET y PUT para colocar estos dibujos sin tener que pensar acerca de los puntos iluminados. Sin duda esto es mucho más fácil que el teclear largas sentencias de DATA, pero de hecho sí que tendrá que copiar estos dibujos sobre la retícula de pantalla si quiere que este pájaro vuele. Ahora ya debe ser capaz de modificar el programa del Sprinter (anterior) para obtener las áreas correctas.

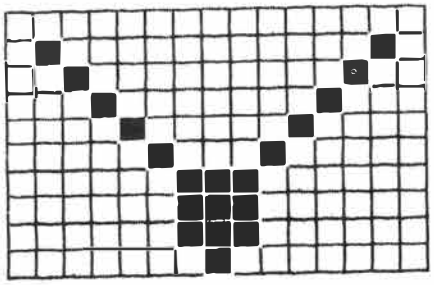


Fig. 15.8a Volando alto.

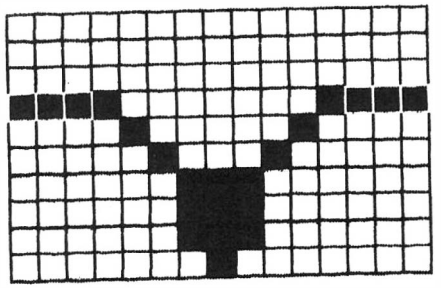


Fig. 15.8b

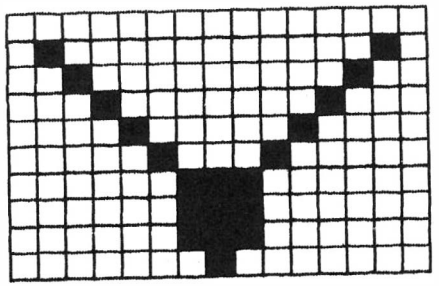


Fig. 15.8c

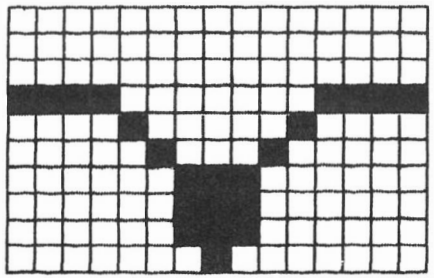


Fig. 15.8d

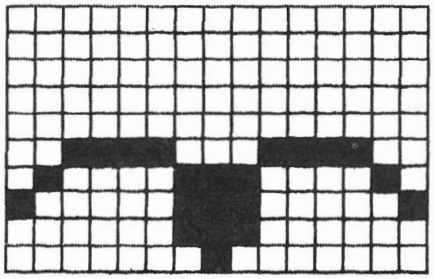


Fig. 15.8e

Oasis

Las únicas desventajas reales de la animación mediante GET y PUT son que no se puede cambiar la escala, el color o el ángulo del dibujo. La orden DRAW le permitirá cambiar estos factores, pero como es mucho más lenta que GET y PUT sólo tiene utilidad en algunas aplicaciones, y los nuevos dibujos se generan mejor en las páginas

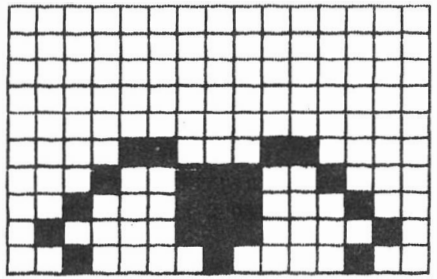


Fig. 15.8f

gráficas que no pueden verse y después se visualizan mediante la orden PCOPY en la pantalla actual. Como ejemplo veremos cómo producir un oasis en el desierto que va haciéndose mayor a medida que nos vamos acercando.

En primer lugar necesitamos borrar (PCLEAR) las ocho páginas gráficas y dejar las cuatro primeras de color amarillo mediante PCLS (color 2) para representar la arena.

```
10 PCLEAR 8:PMODE 3,1:SCREEN 1,0
:PCLS 2
```

La forma más rápida de dejar la parte superior de la pantalla de color azul, para que represente el cielo, es cambiar el PMODE a 1 que utiliza únicamente dos páginas y hacer un PCLS 3. Recuerde que ya que no hay una orden SCREEN seguimos viendo el PMODE 3. Ahora cambiamos de nuevo el PMODE a 3 y hacemos un círculo coloreado que representa el Sol.

```
20 PMODE 1,1:PCLS 3:PMODE 3,1:CI
RCLE(230,30),20,2:PAINT(230,30),
2,2
```

En cada figura, el oasis se genera sobre una pantalla escondida en las páginas 5 a 8. El PCLS 2 en PMODE 3 deja estas páginas de color amarillo y después el PCLS 3 en PMODE 1 genera la mitad superior en azul.

```
50 PMODE 3,5:PCLS 2:PMODE 1,5:PC
LS 3:PMODE 3,5
```

El oasis se genera mediante las órdenes DRAW y PAINT.

```
40 A$="C1LGER2FHLFHGEC4":W$="BM1
28,110C3BM-6,+0FR10EL12BM+6,+0":
FT$="C4U5XA$;BM+4,+5U4XA$;BM-8,+
4U3C1XA$;"
60 DRAW W$:PAINT(128,111),3,3:DR
AW FT$
```

Para poder ver el oasis sobre la pantalla deberemos utilizar PCOPY sobre las últimas tres páginas de la pantalla escondida y copiarlas en las últimas tres páginas de la pantalla que se está visualizando. Ya que la página superior no cambia, no hay ningún interés en copiarla.

```
70 PCOPY 6 TO 2:PCOPY 7 TO 3:PCO
PY 8 TO 4
90 GOTO 90
```

Si ejecuta este programa verá un pequeño oasis a lo lejos (fig. 15.9), pero si añade un incremento en el factor de escala (S) aumentará rápidamente de tamaño.

```
30 FOR S=4 TO 48 STEP 4:DRAW"S"+
STR$(S)
80 SOUND(S*5),1
90 NEXT S
```

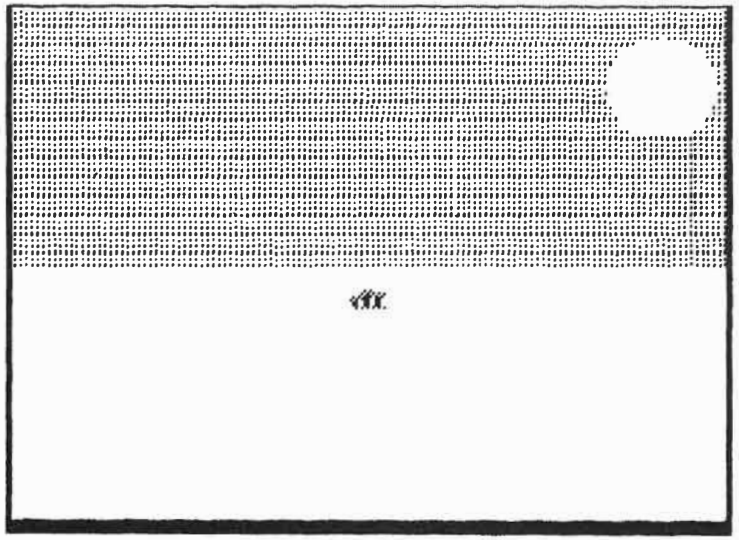


Fig. 15.9a Oasis (animación por variación de la escala).

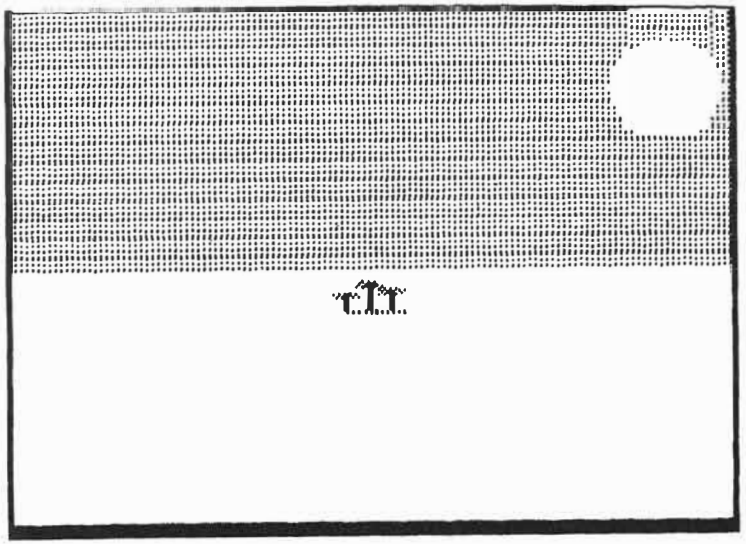


Fig. 15.9b

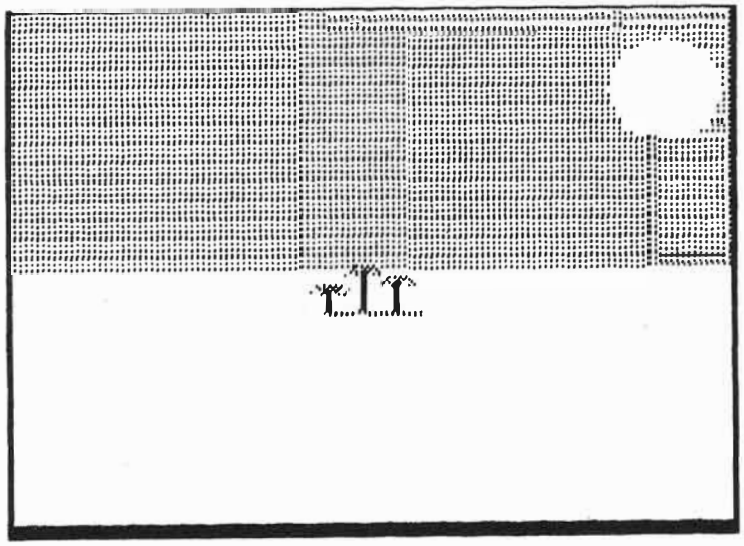


Fig. 15.9c

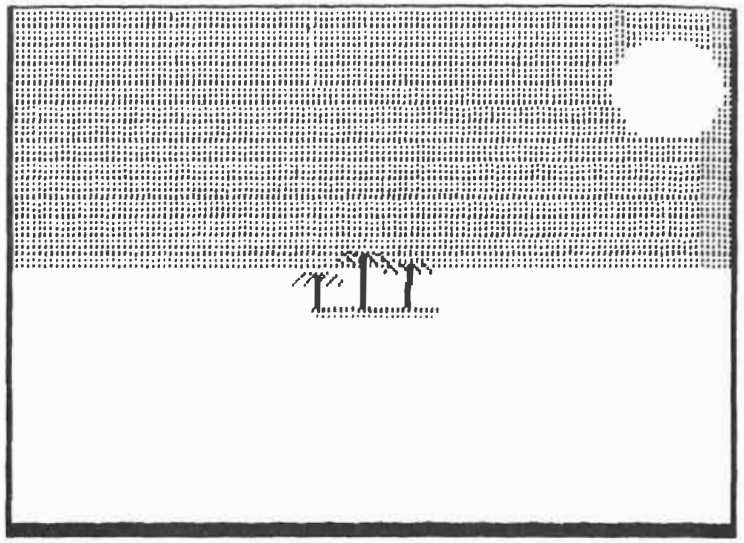


Fig. 15.9d

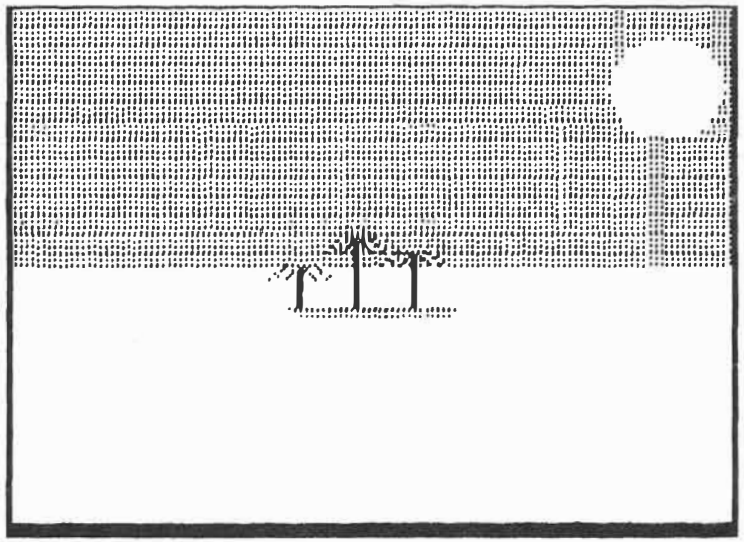


Fig. 15.9e

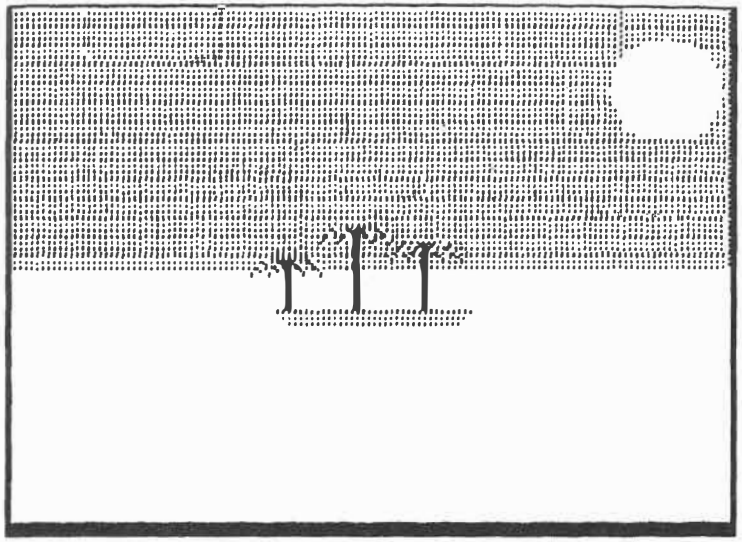


Fig. 15.9f

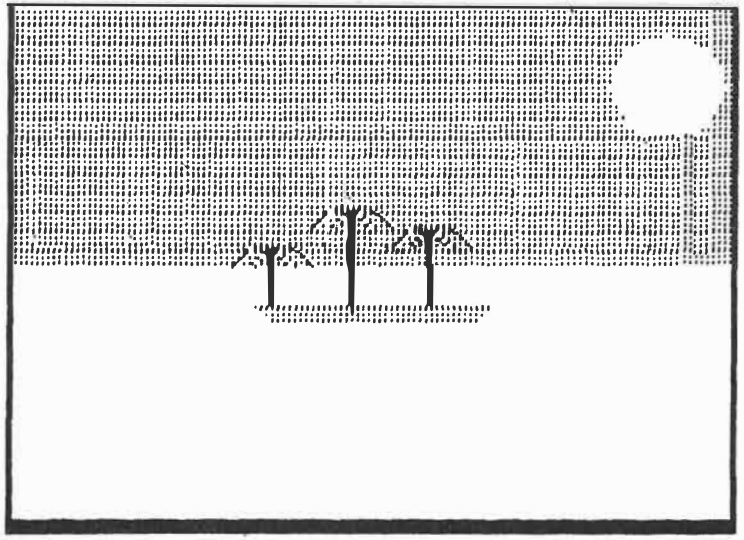


Fig. 15.9g

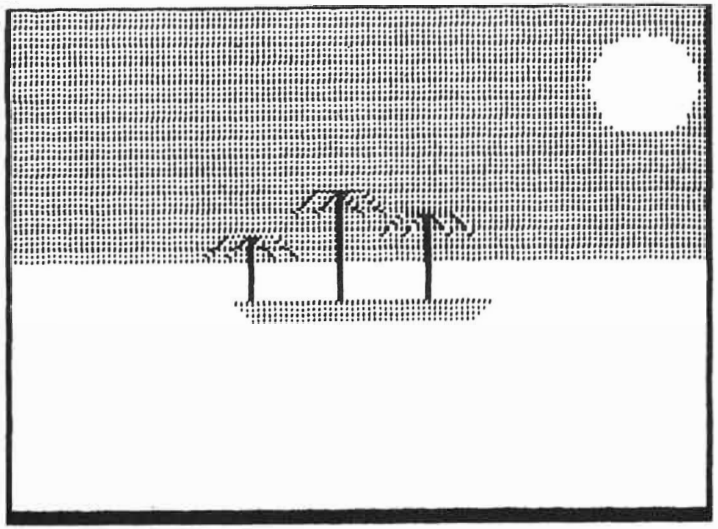


Fig. 15.9h

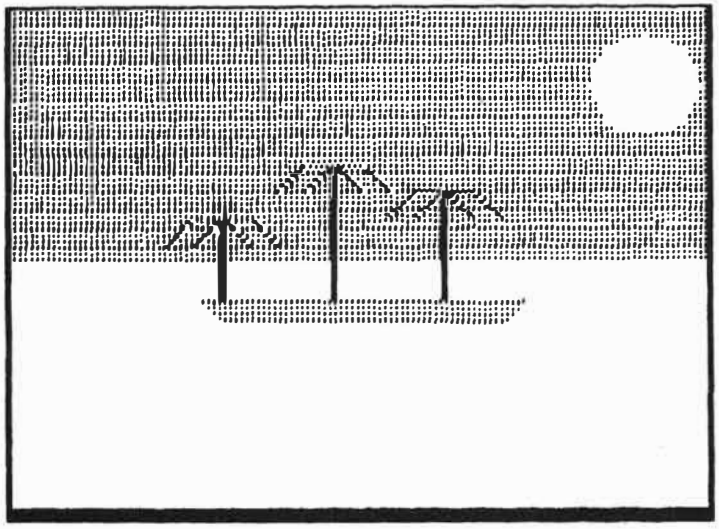


Fig. 15.9i

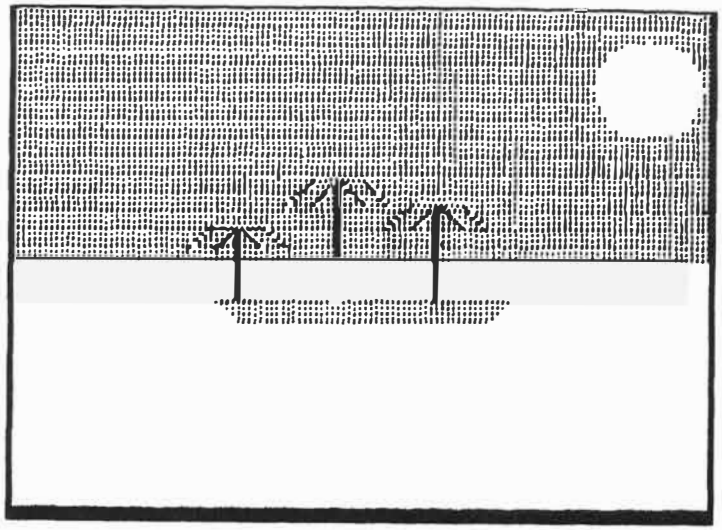


Fig. 15.9j

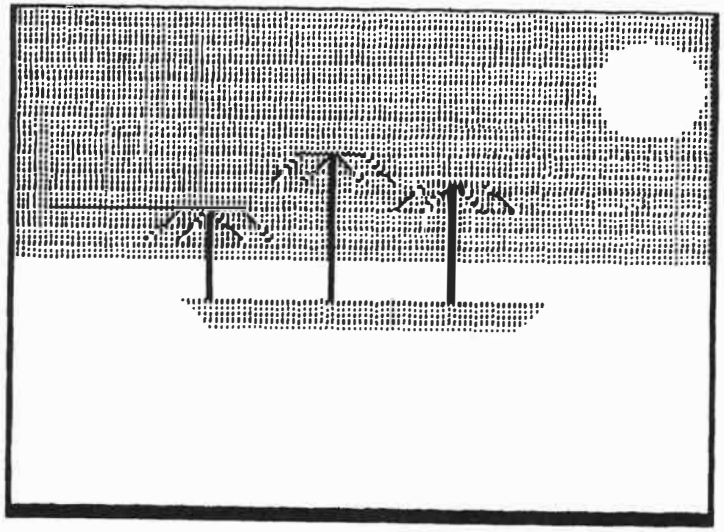


Fig. 15.9k

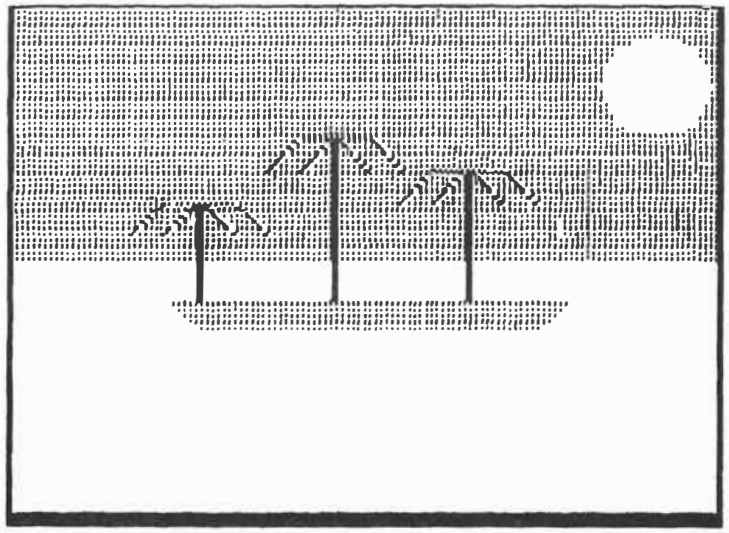


Fig. 15.9l

Naturalmente, los espejismos son muy frecuentes en el desierto, por lo que no deberá sorprenderse mucho si su cabeza empieza a girar y el oasis desaparece en la distancia.

```
100 FOR A=3 TO 0 STEP-1:PMODE 3,
5:PCLS 2:PMODE 1,5:PCLS 3:PMODE
3,5:DRAW"A"+STR$(A)+"S"+STR$(16*
A)+W$+PT$
110 PCOPY 6 TO 2:PCOPY 7 TO 3:PC
OPY 8 TO 4:NEXT A
120 FOR N=255 TO 1 STEP-5:SOUND
N,1:NEXT N
130 RUN
```


16. Síntesis de sonido

Aunque nunca va a conseguir que el Dragon suene como un sintetizador real, es posible demostrar algunas de sus características con este programa. Generalmente se conoce como el sintetizador «MOG», ya que los resultados a veces son un tanto extraños.

Repetición del sonido del teclado

Aunque antes le hemos mostrado cómo ejecutar notas directamente sobre el teclado mediante INKEY$, este método tenía dos desventajas principales. La primera es el hecho de que INKEY$ no realiza una autorrepetición, lo que obliga a levantar el dedo de la tecla antes de que pueda generarse una nueva nota. El segundo problema es que las teclas correspondientes a las notas C D E F G A B no están en lugares muy lógicos para tocar música. Como ya ha visto, el hacer que las teclas se autorrepitan es fácil si se utiliza un PEEK de la posición 135, que contiene el código ASCII de la última tecla pulsada. Si convertimos esto a su representación de carácter mediante CHR$, podemos ejecutarla (PLAY) y un bucle mantendrá la nota hasta que la tecla deje de pulsarse (obsérvese que no se trata del caso de convertir el número en un texto mediante STR$ sino de generar el carácter con CHR$).

```
80 A=PEEK(135)
130 A$=CHR$(A)
180 PLAY A$
190 GOTO 80
```

Si prueba esta rutina descubrirá que no funciona, aunque no es debido a que el planteo sea equivocado. En primer lugar existe el problema del rebote de la tecla ENTER, utilizada para ejecutar el programa, y en segundo lugar la posición 135 retiene el código de la última tecla pulsada. Una pausa inicial eliminará el problema del rebote, y un PEEK en la posición 337 nos dirá si se ha pulsado una tecla.

Añada estas líneas y las teclas A-G se autorrepetirán de manera satisfactoria.

```
60 FOR N=1 TO 100:NEXT N
70 IF PEEK(337)=255 THEN 190
190 GOTO 70
```

La autorrepetición es más bien lenta, por lo tanto cambie el tempo por defecto, colocándolo de manera que acelere las cosas. Un valor de alrededor de T5Ø nos da un efecto razonable.

```
50 PLAY"T50"
```

Tal como está, la nota se detendrá tan pronto como suelte la tecla, pero si se salta a la línea 18Ø en lugar de a la 19Ø, la última nota continuará ejecutándose hasta que se pulse otra tecla.

```
70 IF PEEK(337)=255 THEN 180
```

Reconfiguración del teclado

El pulsar una tecla no válida seguirá produciendo un FC ERROR, pero ahora que podemos repetir una nota de forma continua, o de forma alternativa mantenerla permanente, vamos a pensar una manera de reordenar el teclado. Si compara el teclado del Dragon con el teclado de un piano podrá ver que un conjunto de teclas más adecuado para utilizar sería Z S X D C V G B H N J y M (fig. 16.1), donde Z X C V B N y M representan las notas blancas y S D G H y J las notas negras. Una sencilla forma de convertir estas teclas en las notas adecuadas es utilizar la función INSTR. Primero deberemos colocar una variable de texto (K$) que contenga todas las teclas válidas. Luego tendremos que comparar la tecla pulsada con K$. Si la tecla pulsada no se corresponde con un carácter de K$, entonces INSTR dará Ø. Si, por otra parte, existe una coincidencia con un carácter de A$, entonces INSTR nos dará el número correspondiente a la posición de este carácter en el texto.

```
10 K$="ZSXDCVGBHNJM"
140 C=INSTR(1,K$,A$)
150 IF C>0 THEN A$=STR$(C) ELSE
80
```

Cuando ejecute esto descubrirá que las notas designadas funcionan al igual que la escala CDEFGAB y que las otras notas son ignoradas. Pero ¿cómo se ha conseguido este milagro? Para comprender esto debemos volver al principio. Aunque hasta ahora sólo hemos considerado la ejecución de notas designadas por letras, el Dragon también comprende que los números del 1 al 12 representan las mismas notas (fig. 16.2). Lo que hemos hecho es ordenar las notas en T$ de forma que su posición nos dé el número correspondiente.

Una segunda octava

Si colocamos otra variable de texto que contenga un conjunto distinto de caracteres (fig. 16.3), podemos utilizarlos para generar una octava más alta si añadimos «O+» antes de la nota, pero deberemos colocar «O−» después de la nota para volver a la octava normal.

```
20 L$="T6Y7UI9OOP:@"
150 IF C>0 THEN A$=STR$(C):GOTO
180
160 C=INSTR(1,L$,A$)
170 IF C>0 THEN A$="O+"+STR$(C)+
"O-":ELSE 80
```

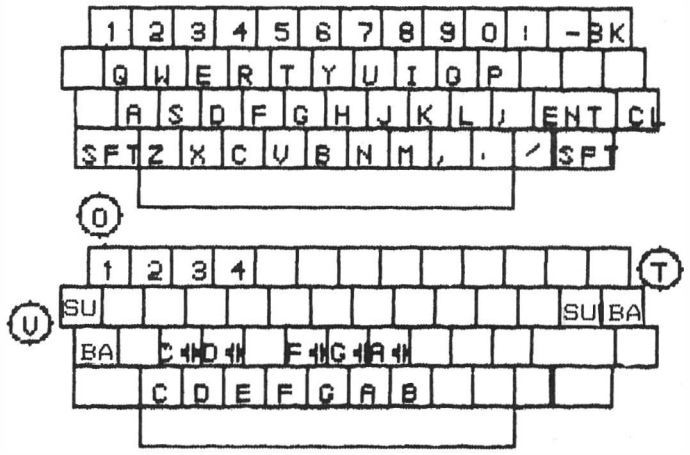


Fig. 16.1 Reconfiguración del teclado.

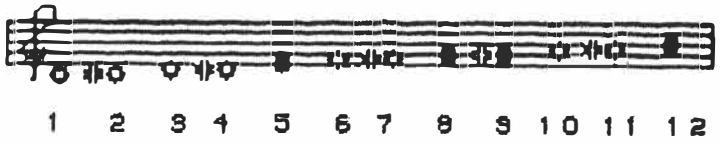


Fig. 16.2 Representación numérica de las notas.

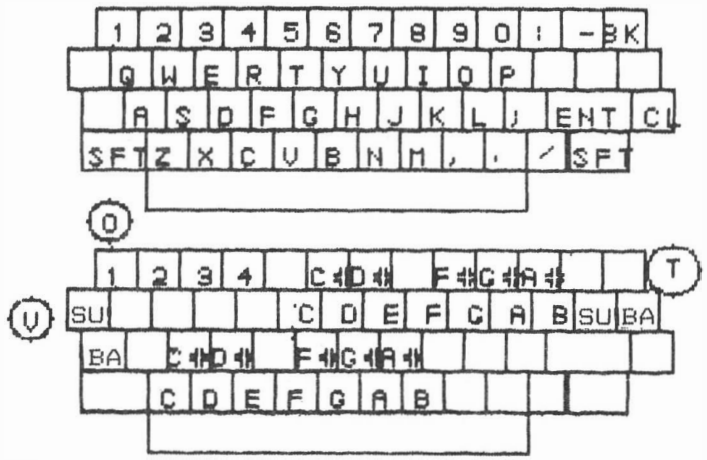


Fig. 16.3 Adición de una segunda octava.

Cambio del tempo

Antes hemos establecido el tempo mediante T5Ø, pero ya que esto no será adecuado para todas las melodías y gustos, ¿por qué no construir una forma de alterar el tempo, subiéndolo y bajándolo mientras se está ejecutando el programa? Podemos enlazar fácilmente T+ y T− con las teclas de cursor a la izquierda y la derecha de forma que la flecha hacia la izquierda disminuya el tempo y la de la derecha lo aumente. Será de más utilidad si el resultado de la acción puede oírse inmediatamente, por lo que continuaremos ejecutando A$ cuando cambie el tempo. Si se pulsa cualquier tecla que no esté designada como una nota, A$ no se actualiza y por lo tanto se seguirá ejecutando la A$ anterior y seguirá repitiéndose la última nota. Cuando se suelte la tecla del cursor, el tempo permanecerá en su valor actual, de forma que pueda utilizarse esta característica para ir probando hasta obtener el resultado deseado.

```
90 IF A=9 THEN PLAY"T+":GOTO 180
100 IF A=8 THEN PLAY"T-":GOTO 18
0
```

Es posible conseguir que esta rutina se pierda si se logra hacer T<1 o T>255, pero en la práctica es poco probable que se alcancen estos límites, por lo que no nos hemos preocupado en incluir ninguna comprobación de los límites.

Control de volumen

Podríamos alterar el volumen de la misma forma que el tempo, pero ya que el rango de valores es mucho menor (1-31) resulta más fácil alcanzar un valor ilegal. Por lo tanto, es mejor utilizar una variable externa y añadirla después de la conversión STR$.

```
30 V=15
110 ON((A=10)-(A=94))+2 GOTO 210
,120,200
180 PLAY A$+"V"+STR$(V)
200 V=V+1:IF V>31 THEN V=31:GOTO
 180:ELSE 180
210 V=V-1:IF V<1 THEN V=1:GOTO 1
80:ELSE 180
```

El volumen se deja en el valor medio al principio (V=15) y se añade PLAY «V»+STR$(V) delante de A$, en la línea 18Ø. Las teclas se clasifican mediante la orden ON GOTO que busca los códigos de las teclas del cursor hacia arriba y hacia abajo. Si no se aprieta ninguna de estas dos teclas, entonces (A=1Ø) y (A=94) son las dos incorrectas y (Ø)−(Ø)=Ø por lo que cuando se añade 2 el resultado es 2, y el programa continúa hacia la línea 12Ø. Cuando se pulsa la flecha hacia arriba (A=1Ø) es verdad y (A=94) es falso, por lo que (−1)−(Ø)+2=1, lo que hace que el programa vaya a la línea 2ØØ, que aumenta el volumen, a menos que V>31. Cuando se pulsa la flecha hacia abajo, (A=1Ø) es falso y (A=94) es verdad, por lo que (Ø)−(−1)+2=3, por lo que va a la línea 21Ø que baja el volumen, a menos que V<1.

Cambio de octavas

Ya que tan sólo hemos definido nuestra octava superior como una octava más alta que nuestra octava inferior, es fácil cambiar ambas octavas de una vez. La octava inicial se coloca a 3 en la línea 4Ø y la línea 12Ø busca los códigos de las teclas del 1 al 4 (49-52) que cambian la octava inferior a este número mediante la ejecución de la orden PLAY «On».

```
40 O=3
120 ON(A-48) GOTO 220,230,240,25
0
220 PLAY"O1":GOTO 70
230 PLAY"O2":GOTO 70
240 PLAY"O3":GOTO 70
250 PLAY"O4":GOTO 70
```

Envolventes

Hasta ahora, la repetición de cada una de las notas suena de la misma forma, aunque la mayoría de los instrumentos musicales de hecho tienen una envolvente característica de sonido. En términos sencillos esto significa que la velocidad con la que el sonido aumenta (ataque) y cae (caída) varía, y una aproximación del control de envolvente puede conseguirse mediante bucles que ejecuten cada nueva tecla cambiando la secuencia de volúmenes. A continuación se dan varios ejemplos y en cada caso la rutina descrita puede utilizarse para sustituir la línea 18Ø.

La situación más sencilla es una caída constante desde el valor máximo.

```
180 FOR E=31 TO 1 STEP-1:PLAY A$
+"V"+STR$(E):NEXT E
```

Ajuste el tempo hasta que se obtenga un efecto razonable y a continuación altere el tamaño del STEP para obtener una caída más rápida.

```
180 FOR E=31 TO 1 STEP-INT(N/5):
PLAY A$+"V"+STR$(E):NEXT E
```

Un efecto menos regular y más interesante puede generarse relacionando el tamaño del step con el volumen actual.

```
180 FOR E=31 TO 1 STEP-3:PLAY A$
+"V"+STR$(E):NEXT E
```

El control del ataque puede introducirse de la misma forma mediante un bucle que se vaya incrementando. Normalmente, el ataque es más rápido que la caída, por lo que los incrementos serán mayores que los decrementos.

```
180 FOR E=1 TO 31 STEP 7:PLAY A$
+"V"+STR$(E):NEXT E:FOR E=31 TO
1 STEP-2:PLAY A$+"V"+STR$(E):NEX
T E
```

Es natural que los valores también pueden entrarse si se desea generar cualquier secuencia determinada.

```
180 PLAY "V7;"+A$+"V14;"+A$+"V28
;"+A$+"V31;"+A$+"V20;"+A$+"V10;"
+A$:PLAY "V9;"+A$+"V8;"+A$+"V7;"
+A$+"V6;"+A$+"V5;"+A$+"V4;"+A$
```


También pueden utilizarse los signos <,>,+ y − junto con **V**. Este aumento y descenso rápido del volumen genera un efecto parecido a un trémolo.

```
180 PLAY"V7;"+A$+"V>;"+A$+"V>;"+
A$+"V<;"+A$+"V<;"+A$+"V7;"+A$
```

Por último, debemos señalar qùe también es posible cambiar el tempo, la longitud de la nota o la octava, para generar interesantes sonidos mediante los mismos métodos, aunque lo dejaremos para que usted experimente con ellos.

17. Editor gráfico de música

Este editor gráfico de música es una excelente demostración de la combinación de las posibilidades gráficas y de sonido del Dragon, ya que le permite entrar una pieza de música, visualizarla según la notación musical estándar sobre la pantalla y después ejecutarla (figura 17.1).

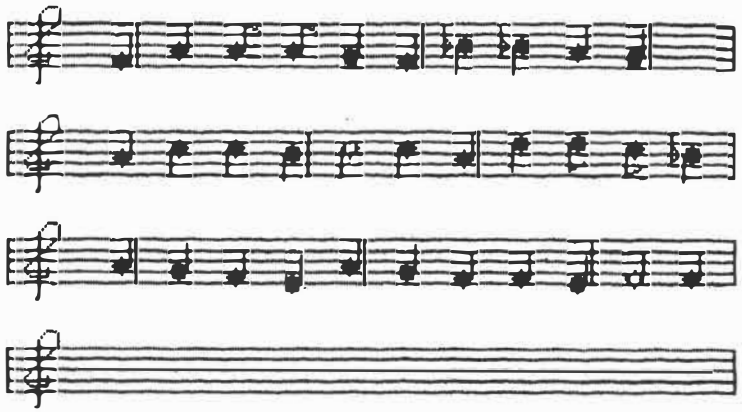


Fig. 17.1 Editor gráfico de música.

Cuando se entre la música necesitamos considerar una serie de factores distintos. Un único carácter del manuscrito nos dice una cosa. La forma del carácter nos dice la longitud de la nota y su posición sobre el pentagrama nos dice qué nota es de la escala y cuál es la octava. También podremos incluir los bemoles y los sostenidos. Se suministran dos modos. En el modo de edición, la posición está indicada por un cursor parpadeante, que se sitúa en la línea del pentagrama que corresponde a la nota actual de la escala. Las teclas de cursor pueden utilizarse para mover éste en cualquier dirección. Las flechas hacia arriba y hacia abajo cambian la nota de la escala, las flechas a la izquierda y a la derecha mueven la posición respecto a la melodía, y las flechas hacia arriba y hacia abajo con el SHIFT nos mueven de una línea a otra. La longitud de la nota requerida se elige pulsando las teclas del 1 al 4. La barra de espacio se utiliza para borrar una nota no deseada.

La melodía se guarda en variables de texto que se descomponen para obtener la información correspondiente al sonido y al gráfico. Cada nota se codifica mediante un bloque de siete caracteres.

Por ejemplo, L12Ø3B–, L 4Ø2C', o L 8Ø2D#

Los tres primeros caracteres definen la longitud de la nota (por ejemplo L12, L 4 o L 8). Nótese el espacio cuando L es menor que 1Ø. Los dos caracteres siguientes especifican la octava, que puede ser Ø2 o Ø3. El sexto carácter es la nota en la escala (A–G) y el último carácter indica si la nota G es bemol (–), natural (') o sostenida (#).

Si se pulsa la «P» en el modo de edición, entonces se pasa al modo PLAY y se ejecutará la melodía entrada hasta el momento y se irá visualizando en la pantalla. También se suministra un método para guardar la melodía.

Inicialización

La primera etapa en el proceso de inicialización implica el borrado de la pantalla, dejándola en negro o en verde, reservar 1Ø ØØØ bytes para variables, e inicializar varias de ellas. La X controla la posición izquierda/derecha en una línea, y la Y la posición total arriba/abajo sobre la pantalla, NO es la posición vertical en el pentagrama, y LI es la línea actual de música (1–4)>. Se inicializa una tabla de cuatro elementos, PA$(n) para guardar las notas entradas en cada línea. Inicialmente estos se rellenan por completo por 255 apóstrofes (') (CHR$(39)). Si se intenta ejecutar (PLAY) un espacio en blanco, a veces se obtiene un FC ERROR, pero el sistema no se queja al ejecutar un CHR$ (39), aunque usted no pueda oírlo. Rellenar las variables de esta forma evita problemas cuando se realice la descomposición de los textos.

```
10 GOTO 690
690 PMODE 4,1:SCREEN 1,0:PCLS 1:
COLOR 0,1:CLEAR 10000:X=40:Y=48:
NO=7:LI=1:DIM PA$(4):FOR N=1 TO
4:PA$(N)=STRING$(255,39):NEXT N
```

Partes gráficas

Primero dibujaremos todas las partes gráficas necesarias y después las colocaremos mediante GET y PUT (fig. 17.2). La imagen

Fig. 17.2 Partes gráficas.

correspondiente a cada parte gráfica debe guardarse en una tabla separada, mediante GET, por lo que se inicializan varias tablas.

```
700 DIM RE(0,10):DIM B1(0,10):DI
M B2(0,10):DIM N1(0,10):DIM N2(0
,10):DIM C1(0,10):DIM C2(0,10):D
IM S1(0,10):DIM S2(0,10):DIM SP(
0,30):DIM BA(0,10):DIM CU(0,10):
DIM SO(0,10):DIM BE(0,10)
```

Ahora podrán dibujarse los signos para las distintas longitudes de notas. Todas ellas tienen un círculo como parte básica, por lo que se dibujan siete de ellos. Esto completa el dibujo de la primera, la redonda.

```
710 FOR N=20 TO 140 STEP 20:CIRC
LE(N,20),3:NEXT N
```

Los otros seis dibujos representan únicamente tres longitudes de nota, ya que la posición de la cola en ellas debe variar de acuerdo con su posición sobre el pentagrama. Primero las que tienen una cola creciente,

```
720 FOR N=40 TO 80 STEP 20:LINE(
N+3,20)-(N+3,10),PSET:NEXT N
```

y después las que tienen una cola decreciente.

```
730 FOR N=100 TO 140 STEP 20:LIN
E(N-3,20)-(N-3,30),PSET:NEXT N
```

Ahora necesitaremos un poco de pintura negra para diferenciar la corchea y la negra de la blanca,

```
740 PAINT(60,20),0,0:PAINT(80,20
),0,0:PAINT(120,20),0,0:PAINT(14
0,20),0,0
```

y finalmente deberemos terminar la cola de la corchea.

```
750 LINE(83,10)-(88,13),PSET:LIN
E(137,30)-(142,27),PSET
```


A continuación, una sección para reponer el dibujo del pentagrama.

```
760 FOR N=0 TO 16 STEP 4:LINE(15
0,N+15)-(170,N+15),PSET:NEXT N
```

seguida por una barra,

```
770 LINE(180,20)-(180,36),PSET
```

un signo de sostenido

```
780 DRAW"BM200,24S4R2U2BM+2,+0;D
2R2BM+0,+2;L2D2BM-2,+0;U2L2"
```

y el signo del bemol.

```
790 DRAW"BM220,20D10E3H3"
```

Ahora colocaremos todos estos dibujos mediante la orden GET en las tablas adecuadas, antes de pasar a la subrutina que dibuja el pentagrama (que está colocado como una subrutina, ya que también se utiliza posteriormente por la rutina de ejecución).

```
800 GET(17,17)-(23,23),RE,G
810 GET(37,10)-(43,23),B1,G
820 GET(57,10)-(63,23),N1,G
830 GET(77,10)-(88,23),C1,G
840 GET(97,17)-(103,30) B2,G
850 GET(117,17)-(123,30),N2,G
860 GET(137,17)-(148,30),C2,G
870 GET(150,0)-(170,56),SP,G
880 GET(180,20)-(180,36),BA,G
890 GET(200,22)-(207,28),SO,G
900 GET(220,20)-(224,30),BE,G
910 GOSUB 920:GOTO 60
```

Dibujo del pentagrama

Las partes gráficas se borran y se construyen cuatro conjuntos de cinco líneas a lo largo de la pantalla (fig. 17.3). El complejo signo de la clave se dibuja fácilmente mediante la orden DRAW, después de un movimiento en blanco adecuado para conseguir la posición inicial.

```
920 PMODE 4,1:SCREEN 1,0:PCLS 1:
CLS 1:COLOR 0,1
930 FOR N=40 TO 160 STEP 40
940 FOR M=0 TO 16 STEP 4
950 LINE(0,N+M)-(255,N+M),PSET
960 NEXT M
970 LINE(0,N)-(255,N+16),PSET,B
980 DRAW"BM10,"+STR$(N+22)+"RU25
E3R3FD4G12DF2R6U3"
990 NEXT N
1000 RETURN
```

Después del RETURN saltará de nuevo al propio programa, en la línea 6Ø.

Fig. 17.3 El pentagrama.

Cursor y comprobación de tecla

Mediante INKEY$, entramos el código de la tecla en A$ y después obtenemos (GET) un cuadrado de la pantalla, alrededor de las coordenadas X, Y y lo guardamos en CU, e inmediatamente lo colocamos (PUT) de nuevo con PRESET. Esto invierte la visualización en esta área de la pantalla. Después de una pequeña pausa, CU se coloca de nuevo con PSET para reproducir la visualización original. Si no se pulsa ninguna tecla, esta secuencia de parpadeo del cursor se repite. Si se pulsa una tecla se hace una comprobación para ver si la

posición actual está demasiado hacia la izquierda (X<4Ø) o hacia la derecha (X>24Ø).

```
60 A$=INKEY$:GET(X-5,Y-5)-(X+5,Y
+5),CU,G:PUT(X-5,Y-5)-(X+5,Y+5),
CU,PRESET:FOR N=1 TO 50:NEXT N:P
UT(X-5,Y-5)-(X+5,Y+5),CU,PSET:IF
 A$="" THEN 60 ELSE IF X<40 OR X
>240 THEN GOTO 80
```

Longitud de la nota

Si la posición es válida, entonces se toma el VALor de la tecla pulsada. Unicamente las teclas numéricas tendrán un VALor, por lo que con esto se consigue la separación de las teclas numéricas respecto a las otras. Las teclas del 1 al 4 se utilizan para indicar la longitud de la nota, desde semibreve hasta corchea, y sólo éstas bifurcarán, en la orden ON GOTO, hacia las líneas que dibujan los caracteres.

```
70 A=VAL(A$):ON A GOTO 210,220,2
30,240
```

La semibreve se trata fácilmente ya que tiene la misma forma, con independencia del lugar que ocupan en el pentagrama. Obsérvese que la tabla se coloca mediante PUT..., AND, en lugar de OR, para producir la sobreposición, ya que la visualización en pantalla está invertida.

```
210 PUT(X-3,Y-3)-(X+3,Y+3),RE,AN
D:GOTO 250
```

Para las otras longitudes de nota, la posición de la nota actual sobre la escala debe comprobarse a fin de determinar si debe alargarse o acortarse la cola. Si la posición del cursor no se ha cambiado, entonces la posición de la nota (NO) seguirá siendo 7.

```
220 IF NO<7 THEN PUT(X-3,Y-10)-(
X+3,Y+3),B1,AND:GOTO 250:ELSE PU
T(X-3,Y-3)-(X+3,Y+10),B2,AND:GOT
O 250
230 IF NO<7 THEN PUT(X-3,Y-10)-(
X+3,Y+3),N1,AND:GOTO 250:ELSE PU
T(X-3,Y-3)-(X+3,Y+10),N2,AND:GOT
O 250
240 IF NO<7 THEN PUT(X-3,Y-10)-(
X+8,Y+3),C1,AND:GOTO 250:ELSE PU
T(X-3,Y-3)-(X+8,Y+10),C2,AND:GOT
O 250
```

Adición a las cadenas de textos

Una vez que la visualización en pantalla esté actualizada, una orden ON GOSUB relacionada con la nota (NO) en la escala, coloca NO$ a la octava correcta y al formato de nota para su ejecución (PLAY). Después del RETURN se genera PL$ añadiendo «L» con un valor igual a cuatro veces el VALor de la tecla pulsada (A*4) y NO$.

```
250 ON NO GOSUB 260,270,280,290,
300,310,320,330,340,350,360,370,
380,390:PL$="L"+RIGHT$(STR$(A*4)
,2)+NO$:GOSUB 400:X=X+20:GOTO 20
260 NO$="02C":RETURN
270 NO$="02D":RETURN
280 NO$="02E":RETURN
290 NO$="02F":RETURN
300 NO$="02G":RETURN
310 NO$="02A":RETURN
320 NO$="02B":RETURN
330 NO$="03C":RETURN
340 NO$="03D":RETURN
350 NO$="03E":RETURN
360 NO$="03F":RETURN
370 NO$="03G":RETURN
380 NO$="03A":RETURN
390 NO$="03B":RETURN
```

Entonces se llama a la subrutina de la línea 4ØØ. Esto producirá la inserción del texto actual (PL$) en el texto total (PA$(LI)). XS se calcula a partir de la posición actual en la pantalla y define la ruptura entre dos notas. SB se utiliza si PL$ es un sostenido o un bemol (véase más adelante).

```
400 XS=(((X-20)/20)*7):PA$(LI)=L
EFT$(PA$(LI),XS-SB)+PL$+MID$(PA$
(LI),XS+8,LEN(PA$(LI))-4):RETURN
```

Finalmente se actualiza la posición en pantalla (X=X+20) y el programa vuelve de nuevo a la línea 2Ø.

Comprobación de los límites

Tras la pulsación de cada tecla se realizan una serie de comprobaciones para asegurar que la nueva posición del cursor está dentro de los límites, y XA (distancia del movimiento actual) se pone a cero.


```
20 IF X+XA<40 THEN X=X-XA ELSE X
=X+XA
30 IF X+XA>250 THEN X=X-XA ELSE
X=X+XA
40 IF X>240 THEN X=240
50 XA=0
```

Otras teclas

Si se pulsa una tecla numérica fuera del rango del 1 al 4, entonces pueden llamarse otras rutinas.

Teclas de cursor

Mediante tests lógicos, el movimiento de las teclas del cursor izquierda/derecha se convierte en incrementos de XA (posición del eje X) y los movimientos de las teclas de cursor arriba/abajo en cambios en NO (posición de la nota en la línea actual).

```
80 A=ASC(A$):XA=(10*((A=8)-(A=9)
)):NO=NO+(((A=10)-(A=94)))
```

Si la posición de la nota cae fuera de los límites, se reinicializa el valor del límite y después la coordenada Y total se calcula a partir de la línea actual (LI) y la nota (NO).

```
90 IF NO<1 THEN NO=NO+1
100 IF NO>14 THEN NO=NO-1
110 Y=(LI*40)+22-(NO*2)
```

«B» = barra vertical

Si se pulsa la tecla «B» se inserta una barra. Esto es puramente decorativo y no se añade al texto.

```
120 IF A$="B" THEN PUT(X-15,(LI*
40))-(X-15,(LI*40)+16),BA,PSET
```

El cursor con la tecla SHIFT

Las teclas de cursor hacia arriba y hacia abajo con la tecla SHIFT producen el movimiento de una línea a otra, siempre que no se sobre-

pasen los límites. La posición inicial se reinicializa hacia el final izquierdo, y se actualiza la coordenada Y total.

```
130 IF A=91 AND LI<4 THEN LI=LI+
1:X=40:Y=(LI*40)+22-(NO*2)
140 IF A=95 AND LI>1 THEN LI=LI-
1:X=40:Y=(LI*40)+22-(NO*2)
```

< barra de espacio > = borrar

Cuando se pulsa la barra de espacio se coloca (PUT) una sección del pentagrama mediante PSET sobre la nota que debe borrarse, con lo que desaparece de la pantalla. Al mismo tiempo la antigua nota se borra de PA$(LI), sustituyéndola por una serie de CHR$ (39).

```
150 IF A=32 THEN PUT(X-10,(LI*40
)-15)-(X+10,(LI*40)+31),SP,PSET:
PL$="'''''''':GOSUB 400
```

«#» = sostenido

El signo de barras se utiliza para indicar un sostenido y éste se coloca mediante PUT...PSET en lugar de AND, para que sea más claro. El signo # aparece a la izquierda de la actual posición del cursor y ya que SB se coloca a 1, el signo de barras se añade a la nota situada a la izquierda de la posición actual del cursor, sustituyendo el CHR$ (39) en la séptima unidad del bloque.

```
160 IF A$="#" THEN PUT(X-11,Y-3)
-(X-4,Y+3),SO,PSET:PL$="#":SB=1:
GOSUB 400:SB=0
```

«–» = bemol

El signo menos indica un bemol y funciona de la misma forma.

```
170 IF A$="-" THEN PUT(X-7,Y-7)-
(X-3,Y+3),BE,AND:PL$="-":SB=1:GO
SUB 400:SB=0
```


«P» = ejecución (PLAY)

La «P» conduce a la rutina de PLAY que llamará primeramente a la subrutina de la línea 92Ø, que dibuja el manuscrito en blanco.

```
180 IF A$="P" THEN GOSUB 410
410 GOSUB 920
```

Cada línea se considera por turno, colocándose primero la posición inicial (X2) a la coordenada 4Ø.

```
420 FOR PL=1 TO 4:X2=40
```

El texto se descompone a partir de la posición 8 (primer carácter) hasta el final, en bloques de 7, y cada bloque es ejecutado (PLAY).

```
430 FOR X1=8 TO 255 STEP 7
440 PLAYMID$(PA$(PL),X1,7)
```

El final de las notas en una línea se detecta por la presencia de dos bloques consecutivos de CHR$ (39).

```
450 IF MID$(PA$(PL),X1,7)="'''''
'''" THEN FL=FL+1 ELSE FL=0
460 IF FL>2 THEN NEXT PL:RETURN
```

Para reproducir los gráficos, el segmento de texto debe decodificarse. En primer lugar debemos extraer el último carácter que será NO$.

```
470 NO$=MID$(PA$(PL),X1+5,1)
```

NO$ se compara con la escala de notas contenida en VN$, mediante INSTR, colocando en N1 el número correspondiente a la nota de la escala. Ahora podrá calcularse la posición actual de Y1.

```
480 VN$="CDEFGAB":N1=INSTR(1,VN$
,NO$):Y1=(PL*40)+22-(N1*2)
```

La octava puede ser únicamente dos o tres, por lo tanto tan sólo debemos comprobar si el tres está en la quinta posición para saber si debemos aumentar Y1 hasta la octava más alta.

```
490 IF MID$(PA$(PL),X1+4,1)="3"
THEN Y1=Y1-14
```

La longitud de la nota se extrae a partir del segundo y tercer caracteres (LN$) y esto se convierte en un número mediante la orden VAL.

```
500 LN$=MID$(PA$(PL),X1+1,2)
510 LN=VAL(LN$)
```

A continuación dividimos el valor de la longitud de la nota por cuatro para ir a las rutinas que de hecho colocarán (PUT) las notas. Estas son muy similares a las descritas antes.

```
520 ON(LN/4) GOTO 540,550,560,57
0
530 GOTO 580
540 PUT(X2-3,Y1-3)-(X2+3,Y1+3),R
E,AND:GOTO 580
550 IF N1<7 THEN PUT(X2-3,Y1-10)
-(X2+3,Y1+3),B1,AND:GOTO 580:ELS
E PUT(X2-3,Y1-3)-(X2+3,Y1+10),B2
,AND:GOTO 580
560 IF N1<7 THEN PUT(X2-3,Y1-10)
-(X2+3,Y1+3),N1,AND:GOTO580:ELSE
 PUT(X2-3,Y1-3)-(X2+3,Y1+10),N2,
AND:GOTO 580
570 IF N1<7 THEN PUT(X2-3,Y1-10)
-(X2+8,Y1+3),C1,AND:GOTO 580:ELS
E PUT(X2-3,Y1-3)-(X2+8,Y1+10),C2
,AND:GOTO 580
```

Si el último carácter es «#» o «−» entonces el signo correspondiente se coloca en la posición adecuada.

```
580 IF MID$(PA$(PL),X1+6,1)="#"
THEN PUT(X2-11,Y1-3)-(X2-4,Y1+3)
,SO,PSET
590 IF MID$(PA$(PL),X1+6,1)="-"
THEN PUT(X2-7,Y1-7)-(X2-3,Y1+3),
BE,AND
```

La coordenada izquierda/derecha (X2) se incrementa en 2Ø y se toma la siguiente nota.

```
600 X2=X2+20:NEXT X1,PL:RETURN
```

(S)=Guardar/Cargar

La S conduce a la rutina de guardar/cargar que permite guardar los textos en cinta, en forma de archivos en código ASCII y después cargarlos de nuevo para reproducir ambas cosas, el sonido y los gráficos. Después de guardarlo, el cursor vuelve a la parte superior de la pantalla de alta resolución.

```
190 IF A$="S" THEN 610
610 CLS:PRINT @228,"";:INPUT"QUI
ERES CARGAR O GUARDAR";Z$
620 IF LEFT$(Z$,1)="C" THEN 660
ELSE IF LEFT$(Z$,1)<>"G" THEN SC
REEN 1,0:GOTO 20
630 INPUT"NOMBRE DEL ARCHIVO";NA
$
635 FOR LI=1 TO 4
640 OPEN"O",#-1,"L"+STR$(LI)+NA$
:PRINT#-1,PA$(LI):CLOSE#-1:NEXT
LI
650 LI=1:Y=48:X=40:NO=7:SCREEN 1
,0:GOTO 20
```

Después de cargar un texto, la posición del cursor se coloca en la parte superior y se activa automáticamente la rutina de ejecución (PLAY).

```
660 INPUT"NOMBRE DEL ARCHIVO";NA
$
665 FOR LI=1 TO 4
670 OPEN"I",#-1,"L"+STR$(LI)+NA$
:INPUT#-1,PA$(LI):CLOSE#-1:NEXT
LI
680 LI=1:Y=48:X=40:NO=7:A$="P":S
CREEN 1,0:GOTO 180
```

Cualquier otra tecla nos llevará a la línea 2ØØ de donde volverá a la 2Ø.

```
200 GOTO 20
```

18. Más allá del BASIC

Aunque pueden hacerse muchas cosas con los programas en BASIC, siempre quedarán ciertas cosas que le gustaría hacer, pero que descubrirá que son imposibles. A menudo sentirá que no puede mover las cosas con suficiente rapidez, o que sus programas utilizan demasiada memoria. La solución de estos problemas reside en el interior del Dragon, donde deberá utilizar directamente su funcionamiento interno. No hay espacio suficiente en este libro para entrar en detalle sobre este tema complejo, pero le daremos algunos ejemplos e indicaciones para mostrarle las posibilidades, incrementar sus ganas y, quizás, iniciarle en la exploración de esta área fascinante.

Antes de empezar deberá comprender con claridad la forma en que el Dragon ejecuta sus órdenes. Primero debe tener en cuenta que el microprocesador 68Ø9, situado en el interior del Dragon, puede comprender únicamente instrucciones si éstas vienen dadas en forma de números, y que el interpretador de BASIC convierte los programas escritos en pseudoinglés en estos números, de forma que el microprocesador pueda actuar sobre ellos. Ya que el BASIC tiene un lenguaje interpretado, sus líneas se vuelven a leer cada vez que el programa pasa a través de ellas. Si puede alterar estas líneas mientras el programa se está ejecutando, entonces podrá cambiar el propio programa.

Ya hemos visto antes las órdenes PEEK y POKE en otros contextos, pero ahora vamos a analizarlas de otra forma, en la que pueden ser de mucha utilidad. Caso de que lo haya olvidado, vamos a recordarle que el PEEK le dice lo que hay en una determinada posición de memoria y que el POKE colocará cualquier número, desde Ø a 255, en una posición de memoria.

Utilización del POKE en su programa

Recordará que cuando desarrollamos el programa para escribir texto en la pantalla de alta resolución, utilizando las órdenes GET y PUT, tuvimos que escribir líneas separadas de DIM, GET y PUT para cada uno de los caracteres, ya que el interpretador del BASIC no permitía cambiar el nombre de la tabla que se quería utilizar. Naturalmente, esto era muy pesado, pero además todas estas líneas ocupan memoria. Debe existir alguna solución a este problema, por lo tanto ¿por qué no utilizar el PEEK en algunas posiciones de memoria que contienen su programa en BASIC, de forma que pueda ver como está guardado? Colocaremos las líneas clave que contienen el DIM, GET y PUT al final del programa y añadiremos un RETURN a cada una de ellas de forma que podamos llamarlas como subrutinas.

```
100 DIM CZ(10):RETURN
110 GET(G1,E1)-(G2,E2),CZ,G:RETU
RN
120 PUT(P1,U1)-(P2,U2),CZ,NOT:RE
TURN
```

Cómo encontrar el final

Ya que éstas son las últimas líneas del programa, sabemos que deben encontrarse justamente antes del final del mismo, y si miramos en las posiciones 27 y 28 podemos encontrar dónde es esto, ya que estos dos bytes contienen siempre la dirección del final del programa actual. Definiremos estas posiciones como la variable EN.

```
10 EN=PEEK(27)*256+PEEK(28)
```

Para observar nuestro programa deberemos utilizar el PEEK en las posiciones que están antes del puntero. La siguiente línea mirará el contenido de las últimas 69 posiciones de memoria del programa y visualizará la dirección de la posición de memoria (N), el desplazamiento negativo a partir del puntero final (EN-N), el número que contiene esta posición (PEEK(N)) y el carácter correspondiente a este número (CHR$(PEEK(N)).

```
20 FOR N=(EN-68)TO EN:PRINT N;"
   ";(EN-N);"     ";PEEK(N);"
   ";CHR$(PEEK(N)):NEXT N
```

Si ahora ejecuta esto, verá aparecer cuatro columnas cambiantes que serán parecidas a las que se muestran en la tabla 18.1. (Además de los caracteres mostrados en nuestra tabla, también aparecerán algunos caracteres gráficos a la derecha de la pantalla. Estos no son aceptables para nuestra impresora que por lo tanto los ha ignorado.) Si encuentra que los números de las posiciones actuales difieren de los que están visualizados, entonces en su programa habrá un número distinto de espacios, o ha utilizado previamente la orden PCLEAR para reservar un número distinto de páginas gráficas, en lugar del valor por defecto de cuatro. Esto no importa realmente por ahora. Ana-

lice con más detalle la última columna de nuestra tabla, liste el programa en la pantalla, y compare los caracteres con las últimas tres líneas del programa. Deberá ser capaz de reconocer partes del mismo. Por ejemplo, las posiciones 7845 y 7846 contienen C y Z, y la 7841 y 7842 contienen U y 2. La mayor parte del resto del listado aparece como irreconocible, ya que el sistema utiliza ciertos números para definir las órdenes del BASIC en lugar de guardar cada uno de los caracteres de la palabra que constituyen la orden. Los puntos clave en lo que a nosotros se refiere son los correspondientes al nombre de la tabla a utilizar. Si analiza todas las posiciones listadas descubrirá que aparecen así:

7787 C
7788 Z (en la orden DIM)
7817 C
7818 Z (en la orden PUT)
7845 C
7846 Z (en la orden GET)

Tabla 18.1

CONTENIDO DE LAS POSICIONES DE MEMORIA AL FINAL DEL PROGRAMA

direc.	desplaz.	PEEK	CHR$(PEEK)
7787	67	67	C
7788	66	90	Z
7789	65	40	(
7790	64	49	1
7791	63	48	0
7792	62	41	)
7793	61	58	:
7794	60	145	
7795	59	0	
7796	58	30	
7797	57	144	
7798	56	0	
7799	55	110	n
7800	54	179	¦
7801	53	40	(
7802	52	71	G
7803	51	49	1
7804	50	44	,
7805	49	69	E
7806	48	49	1
7807	47	41	)
7808	46	196	Ä
7809	45	40	(
7810	44	71	G
7811	43	50	2
7812	42	44	,
7813	41	69	E
7814	40	50	2
7815	39	41	)
7816	38	44	,
7817	37	67	C
7818	36	90	Z
7819	35	44	,
7820	34	71	G
7821	33	58	:
7822	32	145	
7823	31	0	
7824	30	30	
7825	29	172	¼
7826	28	0	
7827	27	120	x
7828	26	180	´
7829	25	40	(
7830	24	80	P
7831	23	49	1
7832	22	44	,
7833	21	85	U
7834	20	49	1
7835	19	41	)
7836	18	196	Ä
7837	17	40	(
7838	16	80	P
7839	15	50	2
7840	14	44	,
7841	13	85	U
7842	12	50	2
7843	11	41	)
7844	10	44	,
7845	9	67	C
7846	8	90	Z
7847	7	44	,
7848	6	192	À
7849	5	58	:
7850	4	145	
7851	3	0	
7852	2	0	
7853	1	0	
7854	0	69	E

Cambio de los nombres de las tablas

Ahora utilice la orden POKE en la posición 7846 con distintos números (como órdenes directas) y después liste la línea 12Ø.

```
POKE 7846,65
120 PUT(P1,U1)-(P2,U2),CA,NOT:RE
TURN
```

Analícelo con cuidado y verá que el nombre de la tabla utilizado en la orden PUT es ahora CA en lugar de CZ! (65 es el código ASCII de A).

```
POKE 7846,66
```

producirá:

```
120 PUT(P1,U1)-(P2,U2),CB,NOT:RE
TURN
```

También puede hacerse un POKE sobre el mismo lugar, definiéndolo mediante un desplazamiento negativo a partir del puntero que señala el final del programa. El desplazamiento para 7846 es −8.

```
POKE EN-8,67
```

da

```
120 PUT(P1,U1)-(P2,U2),CC,NOT:RE
TURN
```

Los desplazamientos para Z en las líneas que contienen DIM y GET son −66 y −36, respectivamente.

El valor absoluto del puntero de final de programa cambiará si se altera la longitud del programa, pero si se define la posición donde se quiere utilizar el POKE como un desplazamiento a partir de éste, entonces obtendrá el resultado correcto siempre que no se altere el programa después del punto clave. Este es el motivo por el que hemos colocado estas líneas al final del programa. No importa lo que se añada antes de ellas, el desplazamiento a partir del final siempre será correcto. Incluso aunque utilice la orden RENUM para cambiar los números de línea del programa, funcionará. Otro factor muy importante es que el POKE puede hacerse también desde dentro del programa. Añada esta línea, ejecute y liste.

```
30 POKE EN-8,68
```

da

```
120 PUT(P1,U1)-(P2,U2),CD,NOT:RE
TURN
```


Por lo tanto, ahora podemos alcanzar las partes que otros métodos no pueden alcanzar y modificar nuestro programa mientras se está ejecutando, pero ¿cómo podemos aplicar esto en la práctica?

Para qué puede utilizar la orden POKE

El utilizar la orden POKE en su programa es una ocupación que no es particularmente peligrosa. Lo peor que puede suceder es que el sistema se pierda y pierda su programa, por lo que quizá lo mejor es guardar una copia del mismo ahora, si se quiere utilizar la orden POKE aleatoriamente. La forma más directa de aplicar esta idea del POKE para obtener (GET) y colocar (PUT) caracteres, sería el utilizar el carácter correspondiente en el nombre de la tabla. Por desgracia no es todo tan sencillo, ya que muchos de los caracteres tienen otro significado cuando están en una línea de programa. Por ejemplo, los dos puntos (:) se utilizan para separar órdenes.

Piense en lo que normalmente es un nombre válido para una variable y entonces sabrá qué caracteres puede utilizar o no. Se permiten uno o dos caracteres; el primero debe ser una letra mayúscula y el segundo puede ser una letra o un número. Si se utiliza un solo carácter para definir la tabla, entonces estará limitado a 26 tablas distintas. Por otra parte, con una letra seguida por un número tendrá 26*1Ø=26Ø tablas distintas, y con dos letras otras 26*26=676 lo que da un total tan grande que no debe haber ningún problema en hacer tablas suficientes para todos los caracteres necesarios (incluso aunque no haga nada más que diseñar caracteres durante el resto de su vida).

En lugar de entrar en discusiones sobre la mejor forma de codificar cada carácter, a continuación le vamos a dar el listado que contiene una modificación directa que cambia el nombre de la tabla para los caracteres de la A a la Z, como un ejemplo, y dejamos el resto para usted. La reducción en las necesidades de memoria es importante ya que el programa completo cabrá ahora en menos de un K. No olvide que también puede utilizar la orden POKE con la «acción» de la orden PUT de la misma forma, para obtener cambios globales respecto a la forma en que la tabla afecta la pantalla.

```
1 EN=PEEK(27)*256+PEEK(28)
10 PMODE4,1:SCREEN 1,0:PCLS1:COL
OR 0,1:CLOADM"MCHAR2"
15 FOR N=48 TO 57:POKE EN-9,N:GO
SUB 2000:NEXT N
16 FOR N=65 TO 90:POKE EN-9,N:GO
SUB 2000:NEXT N:DIM BL(1)
```

```
40 X=1:S=7:A=5:B=7
50 FOR N=48 TO 57:POKE EN-22,N:G
OSUB 1900:X=X+S8:NEXT N
55 FOR N=65 TO 90:POKE EN-22,N:G
OSUB 1900:X=X+S:NEXT N
500 PCLS:X=2:Y=0:S=6:R=8:XI=2:XF
=253:YI=0:YF=191
510 T$=INKEY$:PUT(X,Y)-(X+A,Y+B)
,BL,NOT:PUT(X,Y)-(X+A,Y+B),BL,NO
T:IF T$="" THEN 510 ELSE IF T$="
@" THEN 1620 ELSE T=ASC(T$):IF T
<48 OR T>57 THEN IF T<65 OR T>91
 THEN 1500
520 POKE EN-50,T:GOSUB 1800
530 IF FL=1 THEN PUT(X,Y)-(X+A,Y
+B+1),BL,NOT
540 X=X+S:IF X>XF THEN X=1:Y=Y+R
:IF Y>YF THEN 1610
550 GOTO 510
1500 IF T=32 THEN PUT(X,Y)-(X+A,
Y+B),BL,PRESET:X=X+Q:GOTO 510
1510 IF T=8 THEN X=X-S:GOTO 1560
1520 IF T=9 THEN X=X+S:GOTO 1560
1530 IF T=94 THEN Y=Y-R:GOTO 156
0
1540 IF T=10 THEN Y=Y+R:GOTO 156
0
1550 GOTO 510
1560 IF X<XP THEN X=XP
1570 IF X>XF THEN X=XP:Y=Y+R
1580 IF Y<YP THEN Y=YP
1590 IF Y>YF THEN Y=Y-R
1600 GOTO 510
1620 IF FL=1 THEN FL=0:GOTO 510
ELSE FL=1:GOTO 510
1800 PUT(X,Y)-(X+A,Y+B),CT,PSET:
RETURN
1900 GET(X,Y)-(X+A,Y+B),CZ,G:RET
URN
2000 DIM CZ(1):RETURN
```

Modos gráficos escondidos

El chip generador de la visualización utilizada en el Dragon es un dispositivo de propósito general que también es capaz de generar otros modos de visualización que no están disponibles con el Micro-

soft Color BASIC. Sin embargo, para que pueda trabajarse en estos modos hay que realizar un control directo sobre el generador de visualización (VDG), lo que significa que la cosa es bastante complicada y deberá aprender cómo funcionan los distintos modos.

Semigráficos

La visualización de gráficos en baja resolución disponible en el BASIC es el modo semigráfico 4, en el cual cada posición correspondiente a un carácter del texto se divide en cuatro elementos (fig. 18.1) y cada posición de carácter utiliza un byte. Este byte consta de tres partes. Su primer bit es 1, lo que indica que se trata de un modo semigráfico, los tres bits siguientes codifican el color y los últimos cuatro bits indican si cada elemento debe estar iluminado o no. Ya que el código de color puede establecerse únicamente para la totalidad de la posición de carácter, no pueden iluminarse elementos distintos con distintos colores.

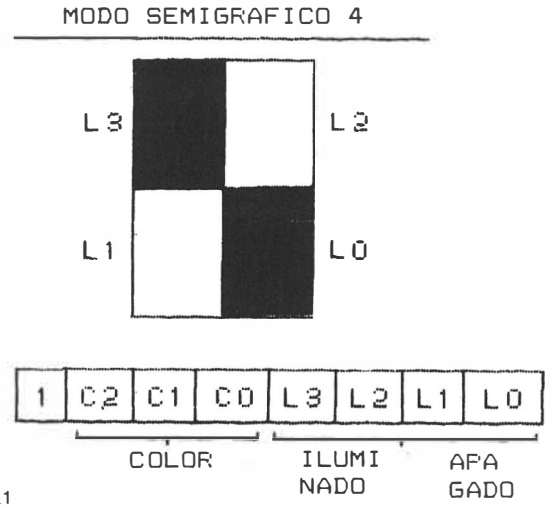


Fig. 18.1

Este modo se selecciona automáticamente por el interpretador de BASIC cuando se está utilizando la pantalla de texto, pero los otros modos semigráficos sólo pueden obtenerse haciendo un POKE sobre ciertas direcciones. Incluso cuando ya se han seleccionado estos modos distintos de esta forma, la visualización sobre la pantalla sólo puede alterarse mediante la orden POKE sobre la memoria de panta-

lla (o utilizando código máquina) y en la práctica esto significa que estos modos son difíciles de utilizar. Los detalles sobre los POKE necesarios se dan en los ejemplos siguientes.

Modo semigráfico número 6

El modo semigráfico 6 divide cada posición de carácter en 6 elementos y utiliza la misma cantidad de memoria que el modo semigráfico 4 (fig. 18.2). El primer bit está a 1 para indicar semigráficos, y tan sólo se utiliza el segundo bit para codificar el color, por lo que nada más pueden indicarse dos colores. El resto de los bits señalan el estado iluminado/apagado y sólo puede utilizarse un color para cada posición de carácter. El conjunto de colores es azul/rojo (si el bit 4 de 65314 está a 1) o magenta/naranja (si el bit 4 de 65314 es Ø). En realidad, este modo no es de mucha utilidad, pero la siguiente demostración indica cómo rellena la pantalla con cada uno de los caracteres gráficos por turno.

```
10 POKE 65314,PEEK(65314)+16
20 POKE 65472,0:POKE 65474,0:POK
E 65476,0
30 GOSUB 100
40 POKE 65314,PEEK(65314)+24
50 GOSUB 100
60 GOTO 60
100 FOR CH=128 TO 255
110 FOR SC=1024 TO 1535
120 POKE SC,CH
130 NEXT SC,CH
140 RETURN
```

Modo semigráfico 8,12 y 24

Los modos semigráficos 8,12 y 24 son más interesantes. Nos dan un control de elementos más pequeños (64*64, 64*96, y 64*192 pixels respectivamente) y permiten utilizar los ocho colores en la misma posición de carácter. También incluye el texto normal en la visualización (algo que naturalmente no puede hacerse con los modos de alta resolución del BASIC). En cada caso la posición de carácter se divide en «N/2» filas de los elementos (fig. 18.3/18.5) y cualquiera de los ocho colores puede especificarse para cada fila. Sin embargo, ya que se utiliza un byte para codificar cada fila se necesita más memoria (2–6 K). En todos estos bytes, el primer bit está a 1, los tres siguientes se

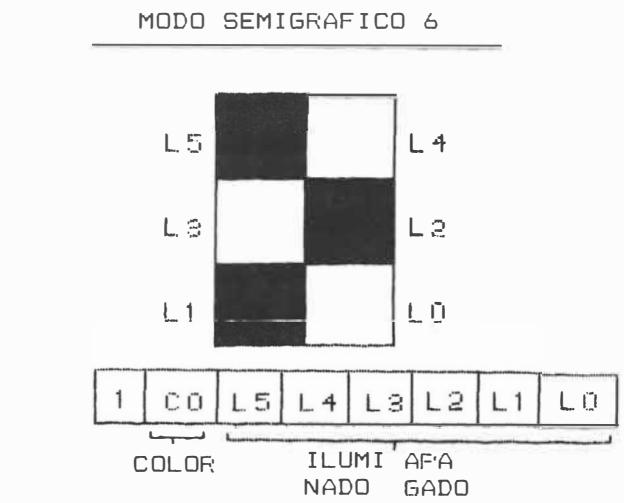


Fig. 18.2

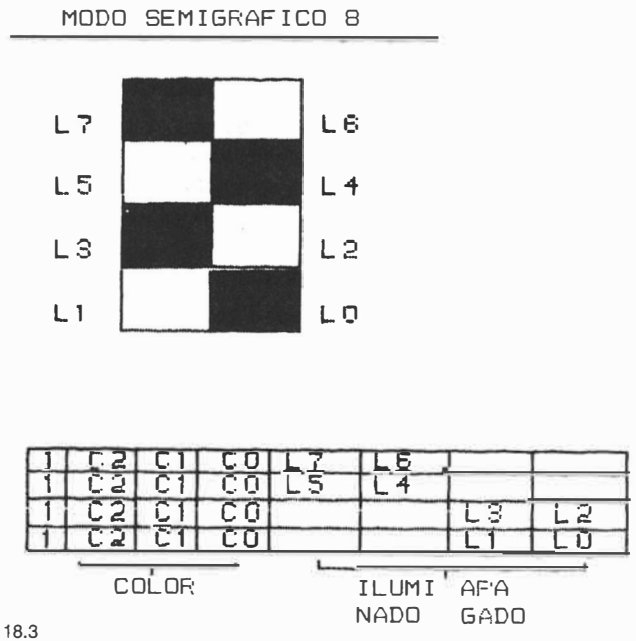


Fig. 18.3

MODO SEMIGRAFICO 12

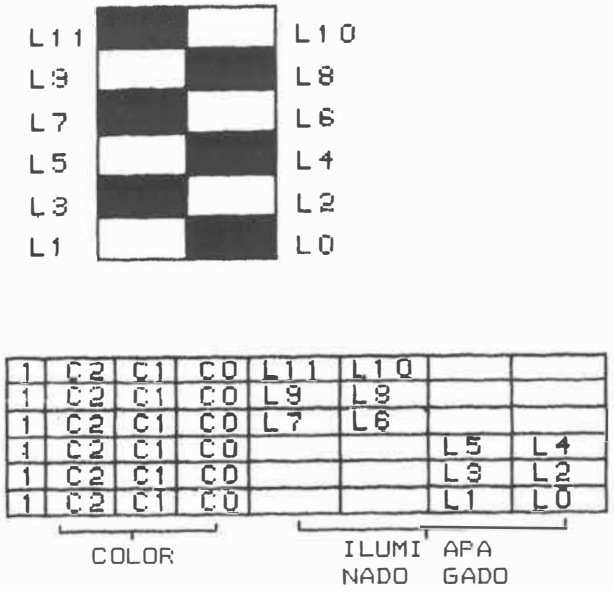


Fig. 18.4

utilizan para codificar el color y los cuatro últimos para indicar el estado de cada elemento. La siguiente demostración ilumina pixels con colores aleatorios en cada momento para mostrar los grados de resolución disponibles.

```
10 CLS8:PMODE 4,1:PCLS 1
20 POKE 65472,0:POKE 65475,1:POK
E 65476,0:H=63:GOSUB 100
30 POKE 65472,0:POKE 65474,0:POK
E 65477,1:H=95:GOSUB 100
40 POKE 65472,0:POKE 65475,1:POK
E 65477,1:H=191:GOSUB 100
50 GOTO 50
100 FOR LN=1 TO H
110 FOR CL=1 TO 31
120 POKE 1024+LN*32+CL,RND(127)+
128
130 NEXT CL,LN
300 CLS 8:PCLS 1:RETURN
```


MODO SEMIGRAFICO 24

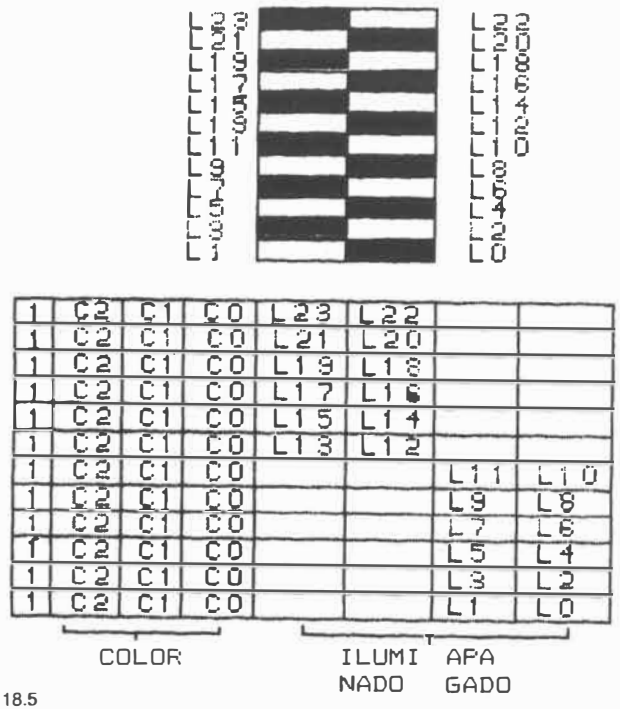


Fig. 18.5

Las órdenes CLS8, PMODE 4,1 y PCLS1 están incluidas como una forma conveniente de borrar la pantalla de textos y las cuatro primeras páginas de la memoria de video antes de empezar, y entre modos, ya que no existe una rutina de borrado de la pantalla incorporada para estos modos no implementados. Si utiliza estos modos podrá obtener ocho colores en alta resolución, pero tendrá que planear muy bien la visualización para obtener una visualización correcta sin la ayuda de todas estas órdenes gráficas de gran utilidad disponibles en el Microsoft Color BASIC.

Inclusión del texto

El texto normal se genera dentro del VDG cuando se recibe un número menor que 128, pero para colocar este texto en la pantalla en estos modos especiales deberá enviar el código del carácter repetida-

mente. El número de repeticiones es el mismo que el número del modo semigráfico, enviando una rebanada horizontal del carácter a cada repetición. Si añade estas modificaciones, tendremos texto en cada uno de los modos. Los números aparecen invertidos en la pantalla ya que los códigos no son los mismos que los códigos ASCII (como ya se ha explicado).

```
10 CLS 8:PMODE 4,1:PCLS 1:M=255:
ST=1024
20 POKE 65472,0:POKE 65475,1:POK
E 65476,0:H=63:R=8:M$="SEMIGRAFI
CO 8":GOSUB 100
30 POKE 65472,0:POKE65474,0:POKE
 65477,1:H=95:R=12:M$="SEMIGRAFI
CO 12":GOSUB 100
40 POKE 65470,0:POKE 65475,1:POK
E 65477,1:H=191:R=24:M$="SEMIGRA
FICO 24":GOSUB 100
50 GOTO 50
100 FOR LN=1 TO H
110 FOR CL=1 TO 31
120 POKE 1024+LN*32+CL,RND(127)+
128
130 NEXT CL,LN
200 CO=0:FOR N=1 TO LEN (M$):M=A
SC(MID$(M$,N,1))
210 FOR CH=1 TO R*16 STEP 32
220 POKE ST+CH+CO,M
230 NEXT CH:CO=CO+1:NEXT N
240 SOUND 1,16
300 CLS 8:PCLS 1:RETURN
```

Gráficos verdaderos

También pueden obtenerse tres modos gráficos verdaderos de más, con los POKE adecuados.

64×64 modo con cuatro colores (un K de memoria de video)
64×128 modo con dos colores (1 K de memoria de video)
64×128 modo con cuatro colores (2 K de memoria de video)

Estos son todos de resolución más baja que los PMODE disponibles con el BASIC y ya que son de interés relativo, hemos omitido los detalles para inicializarlos.

Llamada a subrutinas en código máquina

El código máquina es el lenguaje utilizado por el microprocesador e incluso aunque usted no llegue al extremo de escribir programas completamente en código máquina, puede utilizar subrutinas que ya estén escritas para mejorar sus programas. No podemos aquí entrar en detalles sobre el código del 68Ø9, ya que necesitaríamos un libro completo para ello (si cree que el BASIC es complicado, entonces se dará cuenta muy pronto de que el código máquina es como los antiguos jeroglíficos sumerios en comparación con él). Por lo tanto, sólo explicaremos cómo utilizar las subrutinas en código máquina en sus programas de BASIC, y le daremos algunos ejemplos de rutinas de sonido y de gráficos. Todos los datos están dados en código hexadecimal. Esto quizás haga que las cosas parezcan más complicadas, pero si se quiere entrar en el código máquina hay que acostumbrarse a utilizarlo alguna vez, por lo que quizá debiera empezar ahora.

Reserva de espacio

En primer lugar necesitaremos reservar espacio en memoria para guardar el código máquina que escriba, de forma que no pueda borrarse o ser utilizado por los programas BASIC o las variables. Esto se consigue con la orden CLEAR, que también se utiliza para reservar espacio para textos en los programas de BASIC. Para reservar espacio para el código máquina hay que añadir un segundo parámetro, que limita la dirección más alta que puede utilizar el BASIC. Así:

```
10 CLEAR 200
```

reserva 2ØØ bytes para textos y:

```
10 CLEAR 200,&H6000
```

reserva 2ØØ bytes para textos y el área situada a partir de la dirección &H 6ØØØ para rutinas en código máquina.

Entrada de rutinas en código máquina

Si va a realizar muchos programas en código máquina, entonces debería invertir en un editor/ensamblador, pero mientras tanto este pequeño programa le permitirá entrar los códigos de una forma bastante adecuada. No hay necesidad de escribir «&H» para indicar números hexadecimales, ya que esto se añade automáticamente. (También se incluyen listados en ensamblador para el afortunado.)

```
1000 CLS:PRINT"DIRECCION INICIAL
";:INPUT DI$:DI=VAL("&H"+DI$)
1010 PRINT"EMPIEZA A ENTRAR DATO
S"
1020 PRINT HEX$(DI);:INPUT A$
1030 POKE DI,VAL("&H"+A$)
1040 DI=DI+1
1050 GOTO 1020
```

Una dificultad importante con el código máquina es que no existen rutinas incorporadas para la detección de errores, por lo que si se comete un error al entrar el dato, es fácil que el programa se pierda.

Sonidos sencillos

El sonido se activa cargando un byte en la dirección &HFF23, y la tonalidad que suene depende del valor cargado en &HFF2Ø. La duración depende de un bucle de espera incorporado en el programa. Esta sencilla rutina consigue un único sonido. Cuando haya entrado los números de la segunda columna de la tabla 18.2, a partir de la dirección &H6ØØØ con el programa cargador anterior, podrá ejecutarlo mediante EXEC a partir de la dirección inicial. Tendrían que poderse definir hasta 1Ø rutinas en código máquina separadas en el Dragon, mediante la función USRn, pero debido a un error en la ROM, siempre se llama a la USRØ, con independencia del número especificado. Cuando no se necesita pasar parámetros a la rutina esto no es ningún problema, ya que utilizando la orden EXEC sobre la dirección inicial de la rutina, puede solventarse el problema.

```
100 EXEC&H6000
```

Si este programa se ejecuta en BASIC, producirá un único sonido y después volverá, señalándolo con el OK. Si se añade la línea 11Ø GOTO 2Ø, lo irá repitiendo hasta que se pulse la tecla BREAK. Cuando se necesiten pasar parámetros a una rutina en código máquina, la forma más sencilla de hacerlo es utilizar el EXEC después de haber colocado los valores dentro de la rutina mediante un POKE. El valor de la tonalidad utilizada está guardada en la dirección &H6ØØ9 y la duración en forma de un número de dos bytes en las direcciones &H6ØØ6 y &H6ØØ7, por lo que puede experimentar colocando valores distintos mediante POKE. Por ejemplo:

```
20 POKE &H6007,&HAF
```

Si se siente perezoso para pensar en varios valores, entonces pruebe:

```
20 POKE &H6007,RND(&HFF)
```

aunque le advertimos que sonará un poco como código Morse.

Tabla 18.2

```
DESENSAMBLADO DESDE 6000 HASTA 6015
6000 86 3F      LDA  #3F
6002 B7 FF 23   STA  $FF23
6005 8E 00 FF   LDX  #00FF
6008 C6 5F      LDB  #5F
600A F7 FF 20   STB  $FF20
600D 5C         INCB
600E 26 FA      BNE  600A
6010 30 1F      LEAX -1,X
6012 26 F4      BNE  6008
6014 39         RTS
```

Si añade:

```
30 POKE &H6009,RND(&HFF)
```

sonará un poco más como una orquesta que afina.

Cómo guardar sus rutinas

El área de memoria reservada no se guarda mediante la orden normal del BASIC CSAVE, por lo que habrá que utilizar CSAVEM y tener en cuenta la dirección y longitud del programa. Por ejemplo, esta primera rutina podría guardarse mediante:

```
CSAVEM"sonido",&H6000,&H6014,&H14
```

Efectos sonoros

El código máquina le permite conseguir sonidos más interesantes, ya que éstos pueden cambiar de tonalidad con mucha rapidez. Por ejemplo, el listado de la tabla 18.3 produce un sonido del tipo «PHASER». Se entra a partir de &H61ØØ. La rutina en BASIC que va a continuación, llama a esta rutina siempre que se pulse una tecla, pero coloca valores distintos en ella en función de si se ha pulsado la A o la B, para generar dos sonidos distintos.

```
20 IF PEEK(337)=255 THEN 20 ELSE
 I=PEEK(135)
30 IF I=65 THEN POKE &H6101,&HFF
 ELSE IF I=66 THEN POKE &H6101,&
H3F ELSE 20
40 EXEC&H6100
50 GOTO 20
```

Tabla 18.3

```
DESENSAMBLADO DESDE 6100 HASTA 6113
6100 86 3F      LDA  #3F
6102 B7 FF 23   STA  $FF23
6105 1F 89      TFR  A,B
6107 F7 FF 20   STB  $FF20
610A 5C         INCB
610B 26 FA      BNE  6107
610D 4C         INCA
610E 2A 01      BPL  6111
6110 4F         CLRA
6111 20 F2      BRA  6105
```

Tablas de sonido

A menudo suele ser de utilidad el establecer una secuencia de tonalidades a ejecutar, y que éstas estén organizadas en forma de una «tabla de sonidos», en memoria. El programa de la tabla 18.4 empieza a partir de &H62ØØ y lee los bytes de las tonalidades a partir de la tabla 18.5, que empieza en la &H625Ø y que continúa ejecutando estos sonidos secuencialmente hasta que encuentra un Ø. Utilice el programa cargador para entrar algunos valores en esta tabla y escuche el efecto (contará de todo el espacio disponible hasta la &H64FF). Para acelerar el proceso coloque un valor más pequeño en H62ØB.

Tabla 18.4

```
DESENSAMBLADO DESDE 6200 HASTA 6223
6200 86 3F        LDA  #3F
6202 B7 FF 23     STA  $FF23
6205 10 8E 62 50  LDY  #6250
6209 8E 00 80     LDX  #0080
620C E6 A0        LDB  ,Y+
620E C1 00        CMPB #00
6210 27 13        BEQ  6225
6212 1F 98        TFR  B,A
6214 F7 FF 20     STB  $FF20
6217 5C           INCB
6218 26 FA        BNE  6214
621A 1F 89        TFR  A,B
621C 30 1F        LEAX -1,X
621E 26 F4        BNE  6214
6220 20 E7        BRA  6209
6222 39           RTS
```


Tabla 18.5

```
TABLA DE SONIDO

&H

6250   A3
6251   32
6252   A3
6253   37
6254   84
6255   56
6256   25
6257   89
6258   FF
6259   B5
6260   00
```

Inversión de la pantalla de texto

Los caracteres normales invertidos en la pantalla de texto pueden intercambiarse fácilmente mediante el listado de la tabla 18.6, que hace un EOR (OR exclusivo) de cada carácter de la pantalla de texto con &H4Ø. El programa en BASIC que viene a continuación invertirá la pantalla cada vez que se pulse una tecla, alternando así entre las dos formas.

```
20 I$=INKEY$:IF I$=""THEN 20
30 EXEC&H6500
40 GOTO 20
```

Tabla 18.6

```
DESENSAMBLADO DESDE 6500 HASA 650F
6500 8E 04 00     LDX  #0400
6503 A6 84        LDA  ,X
6505 88 40        EORA #40
6507 A7 80        STA  ,X+
6509 8C 06 00     CMPX #0600
650C 25 F5        BCS  6503
650E 39           RTS
```

Sin duda quedará impresionado por la velocidad de esta rutina, que es prácticamente instantánea. Si sólo quiere invertir una parte de la pantalla, cambie los valores de las direcciones inicial y final, conte-

nidas en dos bytes en &H65Ø1/&H65Ø2 y &H65ØA/&H65ØB, respectivamente. Por ejemplo con un POKE&H65ØA con &HØ5, entonces sólo la mitad superior de la pantalla quedará invertida.

PCLS parcial

La rutina de la tabla 18.7 le permite rellenar ciertos bytes de las pantallas gráficas de alta resolución con cualquier número. Su utilización principal es borrar parte de la pantalla o el colocar un dibujo determinado. La rutina coloca los valores contenidos en &H66Ø1 y &H66Ø3 en bytes consecutivos de la pantalla. Esto es particularmente rápido ya que se hace en un solo movimiento, tratando los registros de ocho bits A y B como un único registro D, de 16 bits. La dirección inicial del área a rellenar está en &H65Ø5/&H66Ø6 y la dirección final en &H66ØA/&H66ØB.

```
20 PMODE 3,1:SCREEN 1,0
30 EXEC&H6600
40 GOTO 40
```

Tabla 18.7

```
DESENSAMBLADO DESDE 6600 HASA 660F
6600 86 00       LDA  #00
6602 C6 55       LDB  #55
6604 8E 06 00    LDX  #0600
6607 ED 81       STD  ,X++
6609 8C 17 FF    CMPX #17FF
660C 25 F9       BCS  6607
660E 39          RTS
```

Si se colocan ceros en &H66Ø1 y &H66Ø3, los tres cuartos superiores de la pantalla quedarán borrados, al igual que si se utilizase PCLS1, y si se coloca un &HFF, el efecto será el equivalente a PCLS4. Si se coloca un Ø en &H66Ø1 y un &HFF en &H66Ø3, el resultado será la obtención de zonas rojas y verdes. Pruebe con otros valores, recordando que cada punto de la pantalla está controlado por un par de bits en PMODE 3.

Tabla 18.8

```
DESENSAMBLADO DESDE 6700 HASTA 6716
6700 FC 67 40      LDD  $6740
6703 10 BE 67 42   LDY  $6742
6707 FE 67 44      LDU  $6744
670A 4C            INCA
670B AE A1         LDX  ,Y++
670D AF C1         STX  ,U++
670F 5A            DECB
6710 26 F9         BNE  670B
6712 4A            DECA
6713 26 F6         BNE  670B
6715 39            RTS
```

Tabla 18.9

```
DESENSAMBLADO DESDE 6800 HASTA 6816
6800 FC 68 40      LDD  $6840
6803 10 BE 68 42   LDY  $6842
6807 FE 68 44      LDU  $6844
680A 4C            INCA
680B AE A3         LDX  ,--Y
680D AF C3         STX  ,--U
680F 5A            DECB
6810 26 F9         BNE  680B
6812 4A            DECA
6813 26 F6         BNE  680B
6815 39            RTS
```

Scrolling

Aunque de forma automática el texto se desplaza hacia arriba en la pantalla de textos, cuando la posición de PRINT alcanza la parte inferior, en la pantalla de alta resolución no existe ninguna rutina de desplazamiento incorporada en el Color BASIC. El listado de la tabla 18.8 suministra un desplazamiento hacia arriba de la pantalla, y la rutina de la tabla 18.9 suministra un efecto similar hacia abajo. El efecto total depende de los valores colocados en las tablas guardadas en &H674Ø y &H684Ø respectivamente. Estos valores pueden guardarse como DATA y colocarse mediante POKE cuando sea necesario. El

ejemplo que viene a continuación da un control más preciso sobre el movimiento hacia arriba y hacia abajo de un círculo flotante, mediante las teclas de cursor hacia arriba y hacia abajo.

```
10 DATA 02,F0,06,10,06,00,02,F0,
0B,EF,0B,FF
20 FOR N=&H6740 TO &H6745:READ A
$:POKE N,VAL("&H"+A$):NEXT N
30 FOR N=&H6840 TO &H6845:READ A
$:POKE N,VAL("&H"+A$):NEXT N
40 PMODE 0,1:SCREEN 1,0:PCLS
50 CIRCLE(128,96),10
60 IF PEEK(337)=255 THEN 60
70 IF PEEK(135)=94 THEN EXEC &H6
700
80 IF PEEK(135)=10 THEN EXEC &H6
800
90 GOTO 60
```


Otros libros sobre MICROINFORMATICA

C. Prigmore	**MICROSOFT BASIC** Curso de autoenseñanza para principiantes
G. Ladevie	**La gestión con BASIC** Comercio y pequeña empresa
G. Guérin	**Microinformática de gestión** Alternativas y utilización
A.P. Mullan	**El ordenador en la Educación Básica** Problemática y metodología
D. Daines	**Las bases de datos en la Educación Básica** Utilización y ejemplos
G.W. Orwig/W.S. Hodges	**Programas educacionales para su ordenador personal**
P. Peilier	**Lenguaje máquina del ZX Spectrum** Subrutinas y trucos
T. Hartnell	**Juegos dinámicos para el ZX Spectrum**
R.G. Hurley	**Los Micro Drives del ZX Spectrum** Utilización y aplicaciones
I. Sinclair	**Introducción al Commodore 64**
I. Sinclair	**Lenguaje máquina del Commodore 64**
S. Money	**Gráficos y sonidos para el Commodore 64**
O. Bishop	**Juegos para el Commodore 64**
B. Lloyd	**Introducción al Dragon**
D. Lawrence	**Programas prácticos para el Dragon**
K. y S. Brain	**Gráficos y sonidos para el Dragon** Incluye subrutinas en código máquina
K. y S. Brain	**Inteligencia artificial en el Dragon**
I. Sinclair	**Lenguaje máquina del Dragon**
M. James/S.M. Gee/K. Ewbank	**Juegos para el Dragon**
V. Apps	**40 juegos educacionales para el Dragon**

Colección «Su ordenador personal»

Así se empieza
Introducción a los ordenadores

Peter Lafferty

204 páginas, de 20 × 14 cm, con más de 100 ilustraciones a dos colores

Indice. Introducción. 1 ¿Qué es un ordenador doméstico? 2 Cómo utilizar su ordenador. 3 ¿Qué puede hacer usted con un ordenador? 4 Cómo escribir sus propios programas. 5 Cómo funcionan los ordenadores. 6 Ampliación del sistema. 7 Elección del ordenador. 8 Hacia el futuro. Apéndice 1. Apéndice 2. Glosario de terminología de ordenadores. Resumen de ordenadores. Bibliografía. Club de usuarios. Indice analítico.

Primeros pasos en BASIC

Susan Curran - Ray Curnow

208 páginas, de 20 × 14 cm, con más de 60 ilustraciones a dos colores

Indice. Introducción. 1 Cómo escribir en la pantalla. 2 Nuestros primeros programas. 3 Introducción de variables. 4 Bucles y ramificaciones. 5 Edición y corrección de errores. 6 Cómo manejar los datos. 7 Cómo escribir programas más largos. 8 Los siguientes pasos. Apéndice 1. Apéndice 2. Apéndice 3. Respuestas a las preguntas. Indice analítico.

El estudiante y el ordenador
Aplicaciones a la enseñanza

Susan Curran - Ray Curnow

168 páginas, de 20 × 14 cm, con más de 40 ilustraciones a dos colores

Indice. Introducción. 1 El ordenador como una ayuda para aprender. 2 Ordenadores para los niños. 3 Programas de recursos. 4 Cómo comprar software. 5 Hardware para la educación. 6 Algunos programas para que usted los pruebe. Apéndice 1. Apéndice 2. Bibliografía. Clubs de usuarios. Indice analítico.

Juegos, imágenes y sonidos

Susan Curran - Ray Curnow

168 páginas, de 20 × 14 cm, con más de 50 ilustraciones a dos colores

Indice. Introducción. 1 Resumen histórico de los juegos por ordenador. 2 Tipos de juegos para ordenador. 3 Gráficos por ordenador. 4 Generación de sonidos mediante su ordenador. 5 Hardware del ordenador. 6 Cómo escribir programas de juegos. 7 Algunos programas que usted puede probar. 8 Compra de programas. Glosario. Indice analítico.

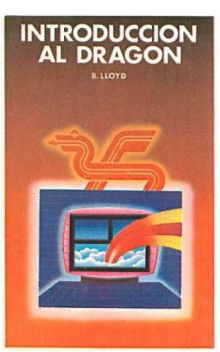
Gráficos y sonidos para el Dragon utiliza un planteo cuidadosamente estructurado para mostrarle cómo desarrollar rutinas en sus propios programas.

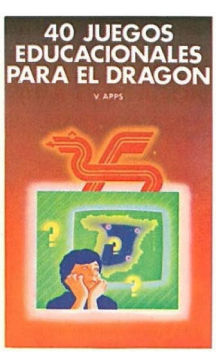
Los aspectos más importantes de las capacidades gráficas y de sonido están analizadas en detalle y completamente ilustradas. El libro empieza a partir de los principios básicos y le lleva a que pueda realizar gráficos de barras, mapas, proyecciones tridimensionales, movimiento, animación, dibujo directo y a cómo imprimir y guardar el contenido de la pantalla, además de muchas otras características. Se examinan con detalle los efectos sonoros complejos, incluyendo la síntesis de sonido a partir del teclado, la visualización gráfica de la música y la integración de gráficos y sonidos.

Además de tratar el funcionamiento y aplicaciones de las órdenes BASIC, el libro explica la organización interna de las características gráficas y de sonido. También le muestra cómo utilizar rutinas en código máquina para mejorar sus programas.

Keith y Steven Brain son padre e hijo y forman un equipo que ya ha publicado el «best-seller» *Dragon 32 Games Master*. Ambos son colaboradores habituales de las revistas *Popular Computing Weekly* y *Dragon User*.

En la misma serie:





Editorial Gustavo Gili, S. A.
Rosellón, 87-89
08029 Barcelona