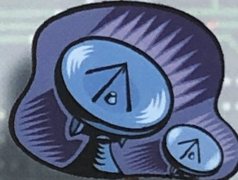
LINUX • JAVA • VISUAL BASIC • DEMOSCENE • VISUAL C++ • PROGRAMACIÓN 3D • SQL

PROGRAMACIÓN ACTUAL

AÑO 1 • NÚMERO 8

ARGENTINA 10$ • CHILE 3000$ • PORTUGAL 1500$

SÓLO 995 ptas



LOS FORMULARIOS WEB

LOS VIRUS DE ARRANQUE

SECCIÓN NIVEL BÁSICO

Manejo de cadenas en **ENSAMBLADOR**, Entrada y salida en **CURSO DE C**, Punteros y listas en **CURSO DE PROGRAMACIÓN**

SECCIÓN BAJO NIVEL

Recorte de polígonos en **ENTORNOS 3D**, Métodos de compresión en **PROGRAMACIÓN GRÁFICA**, Formato **PCX** en **DEMOSCENE**

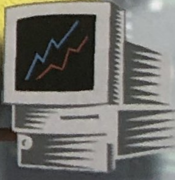
SECCIÓN LAS EMPRESAS DEMANDAN

Formularios en **VISUAL BASIC 5.0**. Ventanas en **VISUAL C 5.0**. Disparadores en **BASES DE DATOS**. Criptografía en **COMERCIO ELECTRÓNICO**

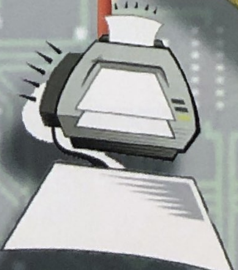
SECCIÓN CAMPUS ACTUAL

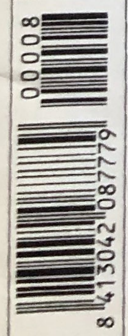
Redes domésticas en **LINUX ACTUAL**, Tipos abstractos en **PROGRAMACIÓN EN ADA**, **TCP/IP** en **REDES LOCALES**

CÓMO MONTAR UNA RED CON LINUX



Prensa Técnic@





PROTOCOLOS TCP/IP

EN NUESTRO CD ROM

MICROSOFT DIRECTX 5.0 AL COMPLETO

ADEMÁS UTILIDADES DE PROGRAMACIÓN, WINDOWS E INTERNET

TRIAL DE FILEMAKER PRO 3.0

DEMOS DE LA EUSKAL, ASSEMBLY, Y MUCHO MÁS...

Introducción a los formatos gráficos

En la anterior entrega comentamos la imagen como mapa de bits y, ahora, vamos a tratar de almacenarla en el disco con un determinado formato para que el tamaño del fichero resultante sea menor o igual que la imagen real en la videomemoria.

En este artículo se van a explicar dos métodos de compresión para almacenar los formatos gráficos en disco. El primero de ellos es el *RAW* o formato crudo, y el otro método es la compresión *RLE* (del inglés *Run Lenght Encode*) que es el utilizado en los ficheros PCX(ver sección Demoscene Actual en página 39). La mayoría de los programas comerciales de diseño de imágenes soportan, como mínimo, éste último formato y pueden servir de ayuda a la hora de diseñar el programa.

Hasta ahora para presentar una imagen en la pantalla el programador estaba forzado a generar la imagen (por ejemplo, con los comandos de dibujo de primitivas), pero los resultados obtenidos no son de gran calidad. Sin embargo, en este momento utilizaremos imágenes que pueden ser generadas desde un programa de edición que soporte este tipo de compresión. Con esto se consigue que los programas tengan unos gráficos más vistosos y presentables, e incluso se pueden utilizar programas de diseño de imágenes 3D, así como la imagen renderizada resultante.

Lo que pretendemos introducir en este artículo es la manera de obtener ficheros binarios en nuestro disco duro, conteniendo el mapa de bits de una imagen para una posterior utilización en cualquier programa mediante su apertura, carga y volcado en VideoRAM. Para esto, podemos hacer dos cosas: bien volcar el bitmap directamente en el fichero, tal y como está en videomemoria (con lo que una imagen de 320x200 ocuparía en 13h 320*200=64.000 bytes) o comprimir este mapa de bits con algún método de compresión para reducir el espacio en disco, ya sea el método LZW (ficheros GIF), el RLE (ficheros PCX), compresión fractal, etc. En este caso vamos a ver el más sencillo, el *Run Length Encoding* o RLE de los ficheros PCX de ZSoft.

También se puede utilizar la imagen comprimida para almacenar los sprites que se usen en los programas. Después, sólo habrá que descomprimir la imagen en una zona de memoria y, sabiendo el desplazamiento de cada sprite, se podrán realizar animaciones, etc. De esta forma no es necesario que los dibujos estén en el ejecutable (en forma de arrays, etc.).

EL FORMATO RAW

Básicamente, este formato se compone de la cabecera, la paleta y el cuerpo o bitmap de la imagen, aunque esta estructura no es estándar. Cada usuario puede hacer variantes cambiando los datos de la cabecera de la forma que más le favorezca para su propósito, o incluso prescindir de ella si se conocen de antemano los datos de la imagen. Por ejemplo, si todas las imágenes que se utilicen son del mismo tamaño, se puede hacer que los bucles del programa que dibujen en la pantalla tengan un valor fijo (el ancho y alto de las imágenes utilizadas).

Sin embargo, no está de más incluir, al menos, el tamaño de la imagen, de forma que se puedan utilizar imágenes de distintas extensiones con un pequeño cambio en el código, puesto que sólo se verán modificados los bucles que se dediquen a dibujar en la pantalla.

A la hora de dibujar la imagen se debe de tener en cuenta que no se usa el mismo método que con los bitmaps o los sprites. Para éstos había que conocer su posición (x, y) en la pantalla con la que, posteriormente, se calculaba el desplazamiento en la memoria de vídeo. Esto se desarrollaba porque la imagen era bastante más pequeña que la resolución de la pantalla por lo que se debía dar su posición. Sin embargo, ahora las imágenes que dibujaremos pueden ser tan grandes como la pantalla o incluso mayores que ésta. Para almacenarlas se puede utilizar la linealidad de la memoria de vídeo. La memoria está organizada por bancos consecutivos, como se vio en el artículo anterior, por lo que, dependiendo del tamaño de la imagen, se utilizará un determinado número de bancos. Por ejemplo, en 13h sólo necesitamos utilizar el primer banco, porque se usan tan sólo 64.000 bytes (320x200 bytes). Si habláramos de modos SVGA con resoluciones como por ejemplo 800x600, necesitaríamos 480.000 y como la memoria está dividida en segmentos de 65.536 bytes la tarjeta las maneja por medio de los bancos.

Sin embargo, sólo será necesario cambiar de bancos cuando el tamaño de la imagen sobrepase los primeros 65536 bytes del primer banco. Si el tamaño de la imagen es inferior no habrá que cambiarlo porque todos los datos de la imagen están en él. Un ejemplo de esto podría ser una imagen de 320x200 y 256 colores, puesto que cada imagen ocupa en la vídeo ram 320 de ancho por 200 de alto con 1 byte por pixel (al ser de 256 colores)= 64000 bytes. En este ejemplo la imagen cabe en un solo segmento de memoria (es inferior a 65536 bytes), por lo que no hará falta cambiar de bancos. Así, si se quisiera almacenar la imagen en el disco, se guardan 64000 bytes del cuerpo de la imagen(que están en el primer banco), los 768 bytes de la paleta más la longitud de la cabecera, que variará dependiendo del número de campos que se pongan. En este artículo, la cabecera que se empleará tendrá sólo el ancho y el alto en campos de enteros (2 bytes), por lo que el fichero tendrá un tamaño total de 64772 bytes.

En cuanto al cuerpo de la imagen, este formato es el más sencillo de todos porque no se utiliza ningún método de compresión para la imagen, simplemente se almacena en el disco de manera lineal. Es decir, se almacenan de forma consecutiva los datos de la imagen y de la paleta en el fichero y,

opcionalmente, la cabecera que normalmente se coloca al principio.
Con este método se puede saber el tamaño del fichero crudo resultante, conociendo el ancho y el alto de la imagen, el número de colores y la longitud de la cabecera y la paleta. Los ficheros *RAW* (a veces también llamados SCR, CLP o similar) se basan, pues, simplemente en el almacenamiento de los mapas de bits en archivos binarios.

Muchos programas de edición de imágenes soportan el formato PCX

FORMATO PCX

EL formato de ficheros con extensión .PCX contiene uno de los métodos de compresión más sencillos, el *Run Length Encoding* o RLE basado en la repetición de datos consecutivos, como posteriormente comentaremos.
Por tanto, para explicar este tipo de

LISTADO 1. RUTINA DE DESCOMPRESION DE UN PCX

```
int a, x, y;
dword pos;
byte dato1, dato2;

 pos = 0;
 fseek (fichero, 128, SEEK_SET);
 /* Saltar cabecera */

for (a=0;a<Ytotal;a++)
{
 for (x=0;x<cabecera.BytesPerLine;)
 {
   fread (&dato1, sizeof (dato1), 1, fichero);
   if (dato1 > 192  )
   {  /* Byte comprimido */
  fread (&dato2, sizeof (dato2), 1, fichero);
  for (y=0;y<(dato1 - 192);y++,pos++, x++)
      PutPixel ( pos, dato2 );
   }
   else
   {
  PutPixel ( pos, dato1);
  pos++; x++;
   }
 }
 pos += ancho-cabecera.BytesPerLine ;
 /* Si el ancho de imagen es menor */
 /* que el ancho de pantalla, se */
 /* añade al offset el trozo que falta */
 /* para completar el ancho de la pantalla */
}
```

compresión se va a explicar el formato del PCX y cómo debe utilizarse la información contenida en él.
En primer lugar veremos el formato de la cabecera, puesto que tendremos que leerla e interpretar sus datos para visualizar la imagen correctamente.
La cabecera de un .PCX se compone de los campos que aparecen en la tabla 1. Éstos, si bien se explicarán más adelante, van a tratarse en profundidad en el apartado de compresión pues, al comprimir la imagen, se tienen que generar todos los campos de la cabecera correctamente.
De momento, para la descompresión los datos que más interesa saber son la ventana, con la que se averigua el tamaño de la imagen, el campo bits_por_píxel que indica la cantidad de colores de la imagen, y el campo bytes_por_línea que es la cantidad de bytes que hay por cada línea y plano del dibujo. Esto quiere decir que si la imagen se compone de 4 planos (como ocurre con las imágenes de 16 colores) y además tiene 80 bytes por línea, significa que el ancho real de la imagen en bytes es de 320 píxeles (80*4). Este ejemplo sería el de una imagen de 640x??? y 16 colores, puesto que cada byte de los 320 totales contiene dos píxeles de la pantalla, por lo que 320 * 2 = 640. Para las imágenes de 256 colores, el número de planos es de 1 y el número de bits por píxel el de 8, puesto que un byte corresponde a un píxel en la pantalla.
Para simplificar el entendimiento de la descompresión y la compresión vamos a basarnos en los modos lineales como en 320x200 y los modos SVGA por bancos.
El campo ventana viene dado por cuatro enteros que determinan la posición X e Y, máxima y mínima, por lo que se puede averiguar el tamaño de la imagen simplemente restando las cantidades mínimas a las máximas (por ejemplo, Xmax-Xmin). De esta forma, se puede adaptar la resolución de la pantalla al tamaño de la imagen para visualizar la imagen entera.
Otro campo importante es el que nos indica la cantidad de colores que tiene la imagen, ya que habrá que leer este campo para adaptar el modo de la pantalla. Para esto se deben tener en cuenta este campo junto con el campo "Ventana" visto anteriormente. Los "bits_por_píxel" indican el número de bits que hacen falta para representar un píxel de pantalla. De esta forma, si en este campo hay un 8, significa que un píxel se representa por 8 bits y, por lo tanto, cada píxel puede tener 256 combinaciones distintas o lo que es lo mismo, la imagen es de 256 colores. Así, cada byte leído corresponde a un píxel de la imagen.
El campo Bytes_por_línea indica la longitud en

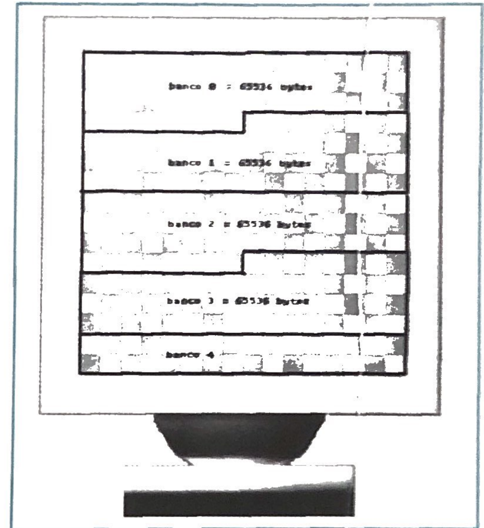
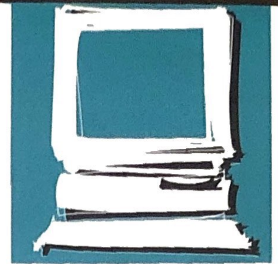


FIGURA 1.

bytes de una línea horizontal. Esto es útil porque los ficheros originarios de Zsoft están comprimidos línea a línea en vez de comprimir el bloque completo de la imagen; es decir, se coge un ScanLine del bitmap, se comprime o descomprime y se almacena, y así hasta el final de la imagen o fichero. De esta forma se evita que al descomprimir, si la imagen es más estrecha que la resolución de la pantalla, se descompriman los primeros colores de la línea siguiente a continuación de esta primera línea. Esto se verá con más detalle en el apartado de descompresión de los ficheros PCX.
Antes de realizar ninguna operación es interesante averiguar si el fichero es realmente un PCX. Para esto, hay unos campos de identificación, como el primer byte que debe ser un 10 (0ah) y es el identificativo de todo PCX. Además, también se puede saber con qué versión se ha creado este fichero, ya que hasta la versión 5 no se admitían imágenes de 24 bits por píxel (16 millones de colores). Si queremos, pues, que un programa de dibujo no pueda cargar ficheros PCX que utilicemos en alguno de nuestros programas, o no queremos que sean reconocidos, podemos cambiar en el fichero el valor de este byte (cambiar el 0ah por otro valor) para que éste no pueda ser identificado y cargado.

El formato crudo o RAW no utiliza métodos de compresión con el cuerpo de la imagen

Otro detalle importante es que la paleta en los PCX varía de lugar dependiendo del número de colores. En el caso de tener 16 colores, la paleta se almacena en el campo Colormap de la cabecera. Este campo tiene un tamaño de 48 bytes, correspondientes a los 16 colores y a las tripletas RGB de cada color (16*3=48).



Sin embargo, si la imagen es de 256 colores la paleta está en el final del fichero, después del cuerpo de la imagen y de un byte que indica si la paleta está al final. Este valor es el 12 decimal. Esto se debe a que la paleta tiene una longitud de 768 bytes (256 * 3 RGB) y no cabe en la cabecera, por lo que se decidió ponerla al final.
Las imágenes de más de 256 colores no tienen paleta, en cambio, se almacena la imagen como si fuera de 8 bits por píxel y 3 planos, siendo cada plano la componente correspondiente (R, G y B).

LA DESCOMPRESION RLE

La compresión de la imagen en el PCX es el llamado RLE, *Run Lenght Encode* (Codificación del tamaño durante la ejecución), en el que trata de comprimir la imagen codificando la cantidad de veces que se repite un mismo color. De esta forma, si en la imagen original se encuentran 20 valores iguales y consecutivos, en el fichero comprimido aparecería un byte indicando el número de veces que se repite y, después, el color en cuestión. Es decir, en vez de almacenarse en el fichero 20 veces el color correspondiente, se almacenarían tan sólo 2 bytes: el 1º, el indicador del número de repeticiones y el 2º, el valor a repetir. A título de ejemplo, la diferencia entre un modo y otro (entre crudo o con compresión) sería:

Crudo-> 0,0,0,etc. (20 ceros)
RLE -> 20,0

Si el valor no se debe repetir en la pantalla no habrá ningún byte antes de éste que señale el numero de veces a repetir. Por lo tanto, se puede decir que en este modelo de compresión hay dos tipos diferenciados de valores: el que se debe colocar en la pantalla (el color) y el que indica compresión (llamado byte de Run, o *RunByte*). En el primer caso, el valor se coloca en la pantalla una sola vez, mientras que en el segundo caso se repetirá en la pantalla el color tantas veces como lo indique el byte del contador.

Para hacer la distinción entre un valor comprimido y otro sin comprimir, se activan los dos últimos bits del valor

En realidad hay un problema, y es que se debe hacer una distinción entre los bytes que tienen compresión de los que no. Es decir, en el ejemplo anterior, si analizamos paso a paso lo que hace el ordenador veremos este problema: en primer lugar, se lee del fichero un valor pero el ordenador no sabe si se trata de un color que se debe poner en la pantalla directamente o consiste en un valor que indica compresión.
Para hacer la distinción entre un valor comprimido y otro sin comprimir, se activan los dos últimos bits del valor. Es decir, si el valor leído tiene activado estos dos bits, se trata de un valor comprimido, y los restantes seis bits indican el número de veces que se repetirá el byte siguiente. De esta forma el número máximo de repeticiones de un mismo valor es el 63, puesto que éste es el número máximo de combinaciones que se pueden conseguir con seis bits.
Por tanto, habrá que desactivarle estos dos últimos bits de mayor peso para que el valor resultante sea el indicado por los primeros seis bits, que es el número de veces que se ha de repetir el siguiente píxel.
Sin embargo, con este sistema no se podría representar en pantalla ningún color superior al 191 porque tendrían estos bits activados y la descompresión lo tomaría como un byte del tipo comprimido. La forma de solucionar este problema es que si se quiere poner un valor superior al 191 se hará con dos bytes: el primero será de tipo comprimido y con el valor del contador a 1 (hay que recordar que este contador son los últimos 6 bits), y el segundo será el color que se desee obtener en la pantalla, que puede ser superior al 191. Si se traducen a bits, un ejemplo posible podría ser éste: el primer byte sería el 193 (11000001b) y el segundo el 210 (11010010b), que es un ejemplo de cómo se pondría en la pantalla el valor 210.
Esto se puede traducir a pseudocódigo para comprobar cómo se realiza la descompresión:

```
Read (fichero, valor1)
if valor1 AND 192 = 192
  then Begin
    read (fichero, valor2)
    for x=0 hasta (valor 1 AND 00111111b)
     poke (videoram, valor2)
  end else
  poke (videoram, valor1)
```

Sea cual sea el número de colores de la imagen, la descompresión se realiza de la misma forma, puesto que se trata de una descompresión byte a byte. Después, sólo hay que interpretar correctamente los valores obtenidos. Esto significa que si un píxel de la pantalla está compuesto por dos bytes (que es el caso de los 64K colores) se deberán coger dos bytes resultantes para componer el píxel de la pantalla.
Un ejemplo de un programa en C se puede ver en el listado 1, hecho a partir del pseudocódigo anterior. Para mayor simplicidad, la rutina está adaptada para imágenes de 256 colores, mientras que para las imágenes de 16 colores hace falta cambiar de planos para lograr la imagen, y esto se verá con más detalle en sucesivos artículos.

La cabecera debe ser generada de forma correcta para que la imagen sea reconocida por otros programas

El anterior pseudocódigo sólo descomprime un byte del fichero, sin embargo, para descomprimir toda la imagen se deben seguir los pasos siguientes:
1) Leer la cabecera y tomar los datos más relevantes. Recordar que el ancho real de la imagen es el ancho leído de la cabecera + 1, y lo mismo ocurre con el alto.
2) Ir al final del fichero, retornar 769 bytes y comprobar que el byte en esa posición es el 0Ch (12 decimal).
3) Leer los restantes 768 bytes del fichero, que son los valores correspondientes a la paleta.

LISTADO 2. RUTINA DE DESCOMPRESION DE UN RAW

```
void descomprime (FILE * fichero)
{
 int a, x, y;
 dword pos;
 char valor;
 word ancho=0;

 pos = 0;
 fseek (fichero, 768, SEEK_SET);

 fread (&valor, 1, 1, fichero);
 while (!feof(fichero))
 {
  fread (&valor, 1, 1, fichero);
  if (ancho == cabecera.ancho)
     {
  pos += (ancho_pant - cabecera.ancho)-1;
  ancho = 0;
     } else ancho++;
  PutPixel (pos, valor);
  pos++;
 };
}
```

4) Dividir todos los valores de la paleta por cuatro para que entren en el rango de 0 - 63 del DAC.
5) Colocar la paleta en el DAC.
6) Colocarse el byte nº 129 del fichero. Este es el primer valor después de la cabecera.
7) Declarar las variables temporales que contendrán la posición X e Y actual. Sus valores iniciales serán las coordenadas superior izquierda especificados en la cabecera.
8) Leer un byte. Comprobar si los dos bits de más peso están activados. Se puede hacer de dos formas:
• Hacer un AND con C0h (192 dec) y comprobar si el resultado es C0h (192 dec)
• Comprobando si el byte es >= 192. Si no es elcaso, saltar al paso 11.
9) Se ha detectado compresión. Coger el valor de los restantes 6 bits, haciendo un AND con 3Fh (63 dec). El valor resultante es el contador.
10) Leer otro byte del fichero. Éste es el valor del color de tantos píxeles como lo indique el contador.
11) Colocarse en la posición de la pantalla indicada por las variables temporales y poner el píxel del color leído.
12) Incrementar la posición de las variables temporales, teniendo en cuenta el ancho y alto especificado en la cabecera. Si se ha llegado al ancho máximo, saltar al paso 14.
13) Si se está ejecutando un byte comprimido, decrementar el contador y volver al paso 11 hasta que contador = 0. Si no es un byte comprimido, o el contador está a cero, saltar al paso 8.
14) La imagen está dibujada en la pantalla. Sólo queda cerrar el fichero.

LA COMPRESION RLE

El algoritmo de compresión es igual al de descompresión, pero al revés. Se debe comprimir la imagen línea a línea escribiendo en el fichero la cantidad de píxeles iguales y consecutivos que hay en la imagen.
Al igual que al descomprimir, se deben distinguir los bytes que indiquen que se ha producido la compresión de los que no, y ésto se hace activando los dos bits de más peso del primer byte (que es el que, a la vez, hace de contador) y escribiendo a continuación el byte del color que se repite en la imagen original. Además, se debe escribir en el fichero destino como comprimido cualquier color que sea superior al 192, añadiendo un contador de 1. Es decir, un píxel de color mayor que el 192 se almacenaría como un RunByte de 1 y luego el color (2 bytes para 1 sólo píxel), de ahí que, algunas imágenes comprimidas con RLE (aquellas que tienen muchos colores de valor mayor que 192), ocupen más que el equivalente crudo de la misma.

Comprimido	Descomprimido
10 (menor de 192 = descomprimido)	10
193 200 (contador = 193-192 = 1)	200
230 5	5 5 5 5 .... 37 veces
230 240	240 240 240 .... 37 veces

Tabla 1. Casos posibles al descomprimir PCX.

El algoritmo podría ser éste:
Repite Y veces
Repite X veces
{
- Contar el número de veces que se repite un color, máximo hasta 64 veces o límite de línea.
- Si hay más de uno consecutivo, comprimir al fichero.
- Si sólo es un color aislado
- Si es inferior al nº 192 escribir al fichero sin compresión
- Si es igual o superior, escribir como comprimido (contador a 1)
}
Sin embargo, una de las cosas más importantes de la compresión es generar la cabecera de forma correcta para que los programas comerciales o cualquier programa nuestro, posteriormente, reconozcan el formato e interpreten sus datos de manera correcta, y así poder trabajar con estas imágenes, aunque puede que no interese que los programas detecten el tipo de formato para evitar que las imágenes sean modificadas y, por tanto, de forma indirecta nuestro programa. Un claro ejemplo son los juegos, que para que las imágenes creadas no se distribuyan con facilidad se comprimen con un formato que sólo los creadores conocen, por lo que sólo ellos saben las especificaciones de esta cabecera y, por tanto, cómo interpretar el bitmap.

PCX VS RAW

Una vez que conocemos estos dos formatos para almacenar las imágenes en el disco, lo más conveniente es elegir en cada caso cuál es el que nos interesa más de los dos. Generalmente, será aquel que tenga un tamaño del fichero menor. Sin embargo, aunque el PCX es un formato que intenta que el tamaño en disco de las imágenes sea menor a veces no lo consigue, y el fichero tiende a ocupar más.
Esto ocurre con las imágenes cuyos valores no se repiten consecutivamente y que, además, cada valor es superior a 192. Como ya hemos visto, si en la imagen hay un valor que no se repite y es superior al 192, en el fichero PCX se almacena como dos bytes, el que indica compresión y el valor en cuestión. En este caso un byte original se convierte en dos bytes comprimidos (el doble).

Sea cual sea el número de colores de la imagen la descompresión se realiza de la misma forma

Es el caso de las imágenes tomadas con un escáner porque los valores no se suelen repetir, por lo que, normalmente, la compresión tiende a expandir el tamaño resultante. En estos casos es conveniente utilizar el formato RAW. Así que el lector debe saber cuál es el que conviene utilizar en cada caso. Por supuesto, existen métodos de compresión mucho más complejos, basados en algoritmos de encadenado (o agrupación de subcadenas) como el LZW del formato GIF de Compuserve, o el algoritmo *Inflate/Deflate* usado en el PKZIP y en el nuevo formato gráfico estrella, el PNG, imágenes comprimidas con compresión fractal, etc. La descripción de todos estos formatos gráficos requeriría una sección por sí misma, así que, tras conocer las bases del almacenamiento de imágenes, se seguirán tratando temas relacionados con la programación gráfica.

EN LA PROXIMA ENTREGA

En el próximo artículo se explicarán los registros de la VGA y se verán las posibilidades que nos brindan. Por medio de estos registros podremos acceder a la tarjeta directamente vía hardware y modificar o leer sus parámetros para conseguir un mayor control de todos los procesos gráficos, tal y como se ha venido haciendo hasta ahora (sin su explicación teórica), como por ejemplo, el retrazo (puerto 03dah) o los registros del DAC (puertos 03c7h-03c9h) todos ellos usados hasta ahora y registros VGA. Otros registros permitirán *Scroll Hardware*, *Pixel Panning*, *Split Screen*, etc, pero eso es algo que se verá en el próximo número.