Programacíon Gráfica

Por el equipo de C&V News

Segunda parte

Introducción

Computer Associates empaquetó varias utilidades nuevas en el CA-Clipper 5.3 a fin de incrementar las posibilidades del mismo. Entre éstas el BLinker, linkeador de modo real; el Exospace, linkeador de modo protegido; la tecnología de RDD de Commix y FlexFile, y por último la librería gráfica Light Lib Graphics de DFL. Muchos desarrolladores esperaban ansiosamente las posibilidades gráficas que este paquete incluye.

En este artículo, el autor investigará como puede utilizarse el modo gráfico en las aplicaciones y de esa forma crear gráficos propios sin necesidad de otros "add-ons".

Activar el modo gráfico

Antes de poder lograr la creación de gráficos en CA-Clipper 5.3, es necesario entender cómo se utilizan las funciones de graficación.

Generalmente los desarrolladores quieren aplicar el modo gráfico, pero sólo si esto fuera posible con la menor cantidad de retoques al código. Después de todo, ¿quién quiere recodificar una aplicación entera?

Compilando y Linkeando

Por suerte las funciones gráficas de CA-Clipper 5.3 son fáciles de utilizar dentro de los programas. Con un mínimo agregado:

```
#include "LLIBG.CH" // Archivo de cabecera
```

al inicio del fuente, luego modificando los archivos de linkeo y adicionando el archivo LLIBG.LIB en la sección de librerías o en la línea de comando del linkeador:

```
EXOSPACE FI MiTest.OBJ LIB LLIBG
```

Veamos a continuación un simple ejemplo que nos permitirá ver lo sencillo que es utilizar el modo gráfico del CA-Clipper 5.3

Un ejemplo sencillo

Examinemos la aplicación (algo insignificante pero muy útil)

```
// Ejemplo simple que ilustra sobre el uso del modo gráfico
Function Main()
```

- ```
  local cText := "Esto es simple"
  scroll() // Limpiar la pantalla
  @ 02, 01 SAY cText
  @ 04, 01 SAY Memory( 0 )
  @ 05, 01 SAY Memory( 1 )
  @ 06, 01 SAY Memory( 2 )
  alert( "Presione una tecla para continuar" )
  ```

```
Return ( NIL )
```

Ahora veamos como cambia la aplicación para el modo gráfico (Las líneas cambiadas están en itálica) :

```
#include "LLIBG.CH"
// Ejemplo simple que ilustra sobre el uso del modo gráfico
Function Main()
```

- ```
  local cText := "Esto es simple"
  ```

```
      SET VIDEO MODE TO LLG_VIDEO_VGA_640_480_16
      scroll() // Limpiar la pantalla
      @ 02, 01 SAY cText
      @ 04, 01 SAY Memory( 0 )
      @ 05, 01 SAY Memory( 1 )
      @ 06, 01 SAY Memory( 2 )
      alert( "Presione una tecla para continuar" )

Return ( NIL )
```

Si se recompila y se relinkea la aplicación, ésta correrá en modo gráfico. Felicitaciones.

Funciones de bajo nivel

Seguiré detallando todo lo que se necesita para crear gráficos en CA-Clipper 5.3. A continuación explicaré aquello que denomino Funciones de Bajo Nivel para Dibujo y Texto.

El sistema de coordenadas

Hasta ahora los desarrolladores CA-Clipper tenían sólo un sistema de coordenadas, 80 columnas por 25 o 50 líneas o renglones. La posición inicial de la pantalla era dada por la coordenada (0,0) que marca la punta superior izquierda, mientras que la coordenada (24,79) ó (49, 79) denota la punta inferior derecha de la misma.

Cuando se trabaja en modo gráfico, se habla de pixels (elementos de imagen o un simple punto en la pantalla). La versión de Light Lib Graphics incluída en el paquete permite resoluciones de 640x480 pixels y 16 colores, esto es gráficos VGA Standard.

Debido a esto el sistema de coordenadas tiene la primer posición en el (0, 0) y la última en (639, 479).

El siguiente ejemplo dibujará una hermosa diagonal multicolor en la pantalla.

```
#include "INKEY.CH"
#include "LLIBG.CH"
Function Main()
```

- ```
  local nX, nY
  SET VIDEO MODE TO LLG_VIDEO_VGA_640_480_16
  SET EVENTMASK TO INKEY_ALL
  scroll()
  for nX := 0 TO 639
  for nY := 0 TO 479
  GPUTPIXEL( nX, nY, ( nX+nY ) % 16, LLG_MODE_SET )
  Next
  Next
  alert( "Trabajo terminado!!!" )
  ```

```
Return ( NIL )
```

Si se piensa utilizar diferentes modos de video en las aplicaciones sería conveniente declarar constantes para los valores fijos:

```
#define GMAXCOL 639
#define GMAXROW 479
```

Dibujando un punto

Como vimos en el ejemplo anterior, la función GPUTPIXEL() es la responsable de dibujar un punto en la pantalla. Los valores de las coordenadas X e Y, el color y el modo de dibujo deben ser especificados.

El próximo ejemplo muestra lo que se llama "una invasión de insectos".

```
#include "INKEY.CH"
#include "LLIBG.CH"
#define GMAXCOL 639
#define GMAXROW 479
Function Main()
```

- ```
  local nX, nY, nLoop
  SET VIDEO MODE TO LLG_VIDEO_VGA_640_480_16
  SET EVENTMASK TO INKEY_ALL
  scroll()
  for nLoop := 1 TO 100000
  ```

```
            // Generamos las coordenadas en
            // forma aleatoria...o Casi
            nX := ( seconds() * nX ) % GMAXCOL
            nY := ( seconds() * nY ) % GMAXROW
            GPUTPIXEL( nX, nY, ( nX+nY ) % 16, LLG_MODE_SET )
        next
        alert( "Trabajo terminado!!!" )

Return ( NIL )
```

Ya vimos como dibujar puntos, veamos como dibujar líneas.

Dibujando líneas

Bueno, esto lo haremos con GLINE(), que tiene la misma sintaxis que GPUTPIXEL() excepto por 2 parámetros adicionales. Estos son una coordenada X y una Y que marcan el punto de finalización de la línea.

Veamos el ejemplo realizado en base al anterior con algunos cambios resaltados en itálica:

```
#include "INKEY.CH"
#include "LLIBG.CH"
#define GMAXCOL 639
#define GMAXROW 479
Function Main()
```

- ```
  local nX, nY, nLoop
  local nX2, nY2
  SET VIDEO MODE TO LLG_VIDEO_VGA_640_480_16
  SET EVENTMASK TO INKEY_ALL
  scroll()
  for nLoop := 1 TO 100000
      // Generamos las coordenadas en
      // forma aleatoria...o Casi
      nX := ( seconds() * nX ) % GMAXCOL
      nY := ( seconds() * nY ) % GMAXROW
      nX2 := ( seconds() * nX ) % GMAXCOL
      nY2 := ( seconds() * nY ) % GMAXROW
      GLINE(nX,nY,nX2,nY2, (nX+nY)%16, LLG_MODE_SET )
  next
  alert( "Trabajo terminado!!!" )

Return (NIL)
  ```

También se puede cambiar el bucle para que se realice de 1 a 10.000 ( salvo que se armen la líneas!!! ).

Luego de esto, ejecutamos la aplicación. Puesto que los valores de coordenadas no son exactamente aleatorios, probablemente se obtendrán algunos comportamientos extraños (tal como una línea persistente a la izquierda de la pantalla), pero el ejemplo demostrará igualmente lo fácil que es dibujar líneas en el modo gráfico.

## Dibujando un rectángulo

Un rectángulo es esencialmente parecido a una línea, excepto que en éste hay 4 líneas que considerar. Afortunadamente para nosotros, aunque no tengamos que dibujar las 4 líneas individualmente, sólo necesitamos especificar la coordenada de inicio y la del final, la función GRECT() hará el resto.

Como un premio adicional, tenemos aquí un nuevo parámetro para utilizar, con él se marca el estilo del rectángulo. Si se especifica LLG_FILL, entonces se dibuja uno relleno, pero si especificamos LLG_FRAME entonces obtendremos una caja vacía.

Reutilizando el código del fuente anterior y adicionando algunas líneas obtendremos :

```
#include "INKEY.CH"
#include "LLIBG.CH"
#define GMAXCOL 639
#define GMAXROW 479
Function Main()
```

- ```
  local nX, nY, nLoop
  local nX2, nY2
  local lPares
  SET VIDEO MODE TO LLG_VIDEO_VGA_640_480_16
  SET EVENTMASK TO INKEY_ALL
  scroll()
  for nLoop := 1 TO 100000
  ```

```
            lPares := ( ( nLoop % 2 ) == 0 )
            // Generamos las coordenadas en
            // forma aleatoria...o Casi
            nX := ( seconds() * nX ) % GMAXCOL
            nY := ( seconds() * nY ) % GMAXROW
            nX2 := ( seconds() * nX ) % GMAXCOL
            nY2 := ( seconds() * nY ) % GMAXROW
            GRECT(nX,nY,nX2,nY2,;
            iif(lPares, LLG_FILL, LLG_FRAME ),;
                  (nX+nY)%16, LLG_MODE_SET )
      next
      alert( "Trabajo terminado!!!" )

Return ( NIL )
```

Compile y linkee la aplicación para observar los resultados.

Dibujando un polígono

Desafortunadamente la función GRECT() no permite especificar un valor de color distinto para el relleno y el borde. En la función GPOLIYGON() se adicionan un par de habilidades extras:

- Conjunto de vértices : Coordenadas que conectan las líneas que forman los lados. Por ejemplo, especificando {{0,100}, {100,100},{100,200},{200,200}} se dibujará un rectángulo.
- Colores : Posibilidad de especificar diferentes colores de fondo y línea.

Para un mejor aprovechamiento de esta función, practique cambiando los valores del array.

Dibujando un círculo

Los círculos son por naturaleza bastante difíciles de implementar. Imaginemos que tenemos que dibujar cada uno de los puntos individuales que lo forman. De nuevo, somos algo afortunados en tener la función GELLIPSE() provista con el paquete.

Los parámetros de la función son:

```
GELLIPSE(<nXc>,<nYc>,<nRadioX>,<nRadioY>,;
         [<nÁnguloInic>], [<nÁnguloFin>],[<nEstilo>],;
         [<nColor>], [<nModo>],;
         [<nColorContorno>], [<nAltura3D>]) --> NIL
```

Si <nRadioX> == <nRadioY> obtendremos un círculo. De otro modo, se dibujará una elipse. Si especificamos <nÁnguloInic> y <nÁnguloFin> entonces se dibujará sólo un sector de la figura (los valores por defecto son 0 y 360 respectivamente). Si se desa un efecto 3D, entonces deberemos especificar <nAltura3D> en pixels.

Gráficos de torta

El siguiente ejemplo dibuja un gráfico de torta con varias sectores :

```
#include "INKEY.CH"
#include "LLIBG.CH"
Function Main()
```

- ```
  local nDegree := 1
  local nRadius := 100
  SET VIDEO MODE TO LLG_VIDEO_VGA_640_480_16
  scroll()
  do While nDegree <= 360
      GELLIPSE( 320, 240, nRadius, nRadius, nDegree,;
                nDegree + 30, LLG_FILL, nDegree%14+1,;
                LLG_MODE_SET, 15, 10 )
      nDegree += 30
  enddo
  inkey( 5 )
  alert( "Trabajo Terminado!!!!" )

Return (NIL)
```

## Escribiendo texto

Muy bien!!. Sin embargo todavía nos falta declarar algunas cosas en pantalla, ¿o las tenemos?.
```

La función ALERT() escribió texto en la pantalla, y hemos usado el comando @...SAY, ambas cosas aceptables y recomendables. Pero antes de entusiasmarnos demasiado, veamos el siguiente ejemplo:

```
SET VIDEO MODE TO LLG_VIDEO_VGA_640_480_16
SetCursor( 0 )
GRECT( 0,0,100,100,LLG_FILL, 4, LLG_MODE_SET)
@ 02, 01 SAY "HOLA!"
```

¿ Qué piensa qué sucederá ?

Si pensó en una caja roja con la palabra "HOLA!" en su interior, todo está bien. Pero hay un único problema, el fondo del texto es por defecto: negro, entonces no encaja bien con el fondo de la caja roja. Por lo tanto podemos colocar lo siguiente :

```
@ 02, 01 SAY "HOLA!" COLOR "W+/R"
```

Esto nos dará el resultado deseado, pero qué pasaría entonces si la caja tuviese un fondo amarillo o un color totalmente nuevo (asumiendo que podamos trabajar con 256 colores)? El procedimiento anterior fallará rotundamente.

Para solucionarlo podemos probar con lo siguiente :

```
GWRITEAT( 30, 35, "HOLA!", 15 )
```

Note que las coordenadas se expresarán aquí en formato gráfico, lo que nos permitirá tener un mejor control de la posición del texto en la pantalla. Esto es un beneficio, pero a la vez una contra, dado que se requiere un esfuerzo adicional para posicionar la línea de texto en el lugar correcto.

Finalmente, también es posible especificar un array conteniendo al información del font. Más adelante trataremos esto.

Colores y paletas

Como dijimos anteriormente, es posible trabajar con más colores (si se dispone de la versión full de la Light Lib).

En CA-Clipper los programadores están acostumbrados a los valores presentes en el modo texto, "B" para azul, "W" para blanco, etc. En modo gráfico no es así.

El los días de la CGA, los colores se limitaban a un pequeño conjunto predefinido. EGA expandió el rango de los colores mediante el uso de "paletas". Esto es como en una paleta de pintor, donde hay varios colores que se utilizan al mismo tiempo.

Ahora bien, en el modo VGA tenemos 16 colores diferentes que pueden utilizarse al mismo tiempo. En VESA este número se extiende hasta 256. Para ilustrar este punto veamos el siguiente ejemplo:

```
#include "LLIBG.CH"
Function Main()
```

- ```
  local nLoop
  SET VIDEO MODE TO LLG_VIDEO_VGA_640_480_16
  scroll()
  for nLoop := 0 TO 15
      // Selecciona tonalidades de ROJO
      GSETPAL( nLoop,nLoop*16+1,0,0 )
      GRECT( 0,nLoop*30,639,(nLoop+1)*30,;
              LLG_FILL,nLoop,LLG_MODE_SET )
  next
  inkey(0)
  ```

```
Return ( NIL )
```

El programa dibujará varios rectángulos en distintas tonalidades de ROJO, desde el más cargado (cercano al negro) arriba, hasta el más brillante abajo. Utilizando diferentes tonos de cada color, se pueden obtener efectos de sombra que le darán ese toque profesional tan deseado a sus aplicaciones.

¿Cómo es posible esto? Simplemente todos los colores en la computadora se realizan por combinaciones de ROJO, VERDE y AZUL. Por ejemplo, las siguientes son algunas de las combinaciones RGB más utilizadas:

```
NEGRO  := { 0, 0, 0 }
BLANCO := { 255,255,255 }
ROJO   := {255,0,0}
VERDE  := { 0,255,0}
AZUL   := {0,0,255}
```
```

Recuerde que existen cientos de cientos de combinaciones posibles, pero con la librería de CA-Clipper 5.3 sólo pueden colocarse 16 diferentes al mismo tiempo.

Bitmaps e Iconos

¿ No es cierto que siempre se han deseado mostrar imágenes sobre la pantalla? Ya sea un logo como fondo o una fotografía de un empleado, son varios los usos que le podemos dar a los bitmaps.

Así como con las funciones anteriores, es bastante fácil mostrar bitmaps en la pantalla.

¿Y qué sobre los íconos? Estos también son fáciles de manipular.

Cargando un bitmap o ícono

El primer paso para mostrar una imagen es cargarla a una variable de memoria por medio de la función GBMPLOAD(). El siguiente código muestra como puede hacerse esto:

```
aAdd( aBMP, GBMPLOAD("\WINDOWS\WINLOGO.BMP") )
```

Aquí, el valor devuelto por GBMPLOAD() se almacena en un array, de esta manera es fácil manipular varias imágenes.

Mostrando un Bitmap

El siguiente ejemplo demuestra como los bitmaps e íconos se muestran en la pantalla en posiciones por defecto:

```
#include "LLIBG.CH"
Function Main()
```

- ```
  local aBMP := {}
  local aICON := {}
  SET VIDEOMODE TO LLG_VIDEO_VGA_ 640_480_16
  aAdd( aBMP, GBMPLOAD( "\WINDOWS\WINLOGO.BMP")
  aAdd( aICON, GBMPLOAD( "\CLIP53\BIN\DBU.ICO" )
  GBMPDISP( aBMP[1] )
  inkey(0)
  GBMPDISP( aICON[1] )
  inkey(0)
  aBMP := NIL
  aICON := NIL
  ```

```
Return ( NIL )
```

Aquí se utiliza un array separado para bitmaps e íconos. No hace falta y podría utilizarse el mismo.
Si se desea, puede especificarse una posición de origen para la imagen en la función GBMPDISP().
Es muy importante que el programa se encuentre trabajando en modo gráfico antes de intentar mostrar una imagen. De lo contrario el programa dará un GPF .

Consideraciones de memoria

Desde que los Bitmaps y los Íconos son almacenados en el VMM , no es buena idea guardar fondos o imágenes en la RAM, a menos que se cuente con la memoria suficiente. Las variables que contienen a los bitmaps o íconos (estén o no siendo mostrados por pantalla) tienen participación activa en la memoria.

Esta memoria se libera cuando la variable es descartada. En el caso de variables locales (altamente recomendadas), esto sucede cuando la función termina de ejecutarse. Para el caso de otro tipo de variables o para liberar memoria en el transcurso de una función, asignar NIL a la variable es lo aconsejado.

FONTS

Una última opción que solicitan los usuarios es la habilidad de mostrar texto en varios estilos. Aunque ya estaba disponible el modo TMR (text mode redefinition), sin embargo, la implementación de fonts proporcionales en un programa CA-Clipper ha sido un tema muy difícil de hacer, por lo menos hasta ahora.

Tipos de fonts disponibles

Existen dos tipos de fonts disponibles en CA-Clipper 5.3, estos son:

- FND : Son fonts que se cargan en la memoria de la VGA. Se encuentran limitados a un número determinado de estilos. El mayor beneficio de estos es su flexibilidad. Los fonts FND pueden ser utilizados por los comandos @...SAY como por la función GWRITEAT(); la salida estándar de CA-Clipper es mucho más rápida. Por defecto, estos son los que se utilizan al momento de pasar a modo gráfico.
```

- FNT : Son los fonts bitmap de Windows que se cargan de una archivo en disco a la memoria. Por lo tanto, es más difícil manejarse con ellos. Sin embargo, están disponibles muchos estilos (como también muchos conjuntos de símbolos). Recuerde que los fonts FNT son de naturaleza proporcional y por lo tanto no podrán ser manejados por el comando @..SAY, pero sí por GWRITEAT().

En pocas palabras, la elección dependerá de sus necesidades.

Cargando un font

Cargar un font en memoria es bastante simple. La función GFNTLOAD() retorna una array con la información del font. Esta función acepta un único parámetro que es el nombre del font FND o FNT.

Cambiando el Font Activo

La llamada a GFNTSET() permitirá colocar un font por defecto para ser utilizado, pero GWRITEAT() también posee un parámetro de font. Por analogía es posible utilizar SETCOLOR() para cambiar el color por defecto y también se puede utilizar la cláusula COLOR del comando @...SAY.

Un ejemplo del uso de los fonts sería:

```
#include "LLIBG.CH"
Function Main()
```

- ```
  local aFonts := {}
  SET VIDEOMODE TO LLG_VIDEO_VGA_640_480_16
  aAdd( aFonts, GFNTLOAD("ONE.FNT") )
  aAdd( aFonts, GFNTLOAD("TWO.FNT") )
  GWRITEAT( 100,100,"Este es el primer font",,,aFonts[1] )
  GWRITEAT( 200,200,"Este es el segundo font",,,aFonts[2] )
  inkey(0)
  aEval( aFonts, {|x| GFNTERASE( x ) } )
  ```

```
Return ( NIL )
```

Borrar los fonts de memoria

Al contrario de los que ocurre con las imágenes, el VMM no puede eliminar de memoria los fonts cargados. Por lo tanto el programador tendrá la responsabilidad de llamar a la función GFNTERASE() para liberar explícitamente la memoria destinada a un font. La omisión de esto ocacionará que un sector de memoria quede inaccesible para la aplicación.

Poniendo todo junto

Si bien la sección anterior no tuvo mucho que ver con la graficación, es tiempo de reunir los trozos y comenzar a generar los gráficos de negocios con CA-Clipper. Para esto comenzaremos viendo los gráficos de torta para posteriormente hacer lo propio con los de líneas y barras.

Gráficos de torta

En ejemplos anteriores hemos visto como puede fácilmente hacerse un gráfico de este tipo. La esencia lógica es :

```
do While nDegree <= 360
    GELLIPSE( 320, 240, nRadius, nRadius, nDegree, nDegree + 30, LLG_FILL, nDegree%14+1,
LLG_MODE_SET, 15, 10 )
    nDegree += 30
enddo
```

La distinción fundamental entre el gráfico de torta y los otros tipos de gráficos es que, un gráfico correcto está comprendido dentro de una elipse completa (360 grados). Por eso, el gráfico de torta es el más utilizado en aplicaciones donde se requieren comparaciones, por ejemplo determinar la porción de mercado que se muestre en valores absolutos. En estos casos resulta más apropiado que un gráfico de líneas.

Ejemplo

Intentaremos mostrar un gráfico que muestre la porción de mercado de 3 empresas fabricantes de automóviles. Teniendo en cuenta que el mercado total es el 100%, cada 3.6 grado de la torta representa un 1 % del mismo. Por ello, si GM tiene un 45%, FM tiene un 30% y CH un 25%. El gráfico se podría hacer con lo siguiente:

```
gEllipse(320,240,100,100, 0,45 * 3.6, LLG_FILL,1,LLG_MODE_SET, 15,10 )
gEllipse( 320,240,100,100,45 * 3.6, (45+30) * 3.6,LLG_FILL, 2,LLG_MODE_SET, 15,10 )
gEllipse( 320,240,100,100,(45+30) * 3.6 ,(45+30+25) * 3.6,LLG_FILL, 3, LLG_MODE_SET, 15,10 )
```
```

Técnicamente, esto es correcto, pero muy poco práctico!!!

Naturalmente, la solución es utilizar un array de valores para cada gráfico:

```
aGrafico := { { "GM", 45 }, { "FM", 30 }, { "CH", 25 } }
```

Ahora podemos recorrer el array para mostrar cada componente. Primero se calcula el valor total que nos permita crear la elipse completa. El primer elemento de cada array se utilizará luego para mostrar las leyendas.

```
nCount := 0
for nLoop := 1 to len( aGrafico )
    nCount += aGrafico[ nLoop, 2 ]
next
nFactor := 360
nGrados := 0 // Puntero
for nLoop := 1 TO len( aGrafico )
    gElipse( 320,240,100,100, nGrados, aGrafico[ nLoop ] * Factor, LLG_FILL, nLoop,
LLG_MODE_SET, 15, 10 )
    nGrados += aGrafico[ nLoop ] * nFactor
NEXT
```

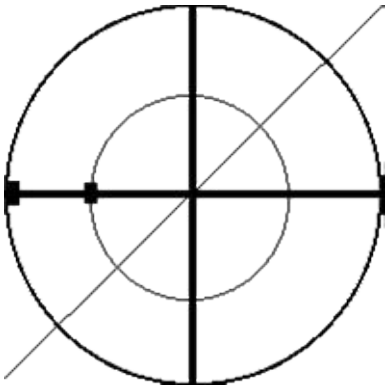
Luego de examinar la función GELLIPSE() podemos ver que estamos especificando 10 pixels de efecto 3D y completamente circular. Para cambiar a un elipse un poco más ancha que alta, simplemente decrementamos el valor del 4º parámetro (<nRadioY>). Finalmente utilizando la directiva LLG_FILL obtenemos las porciones rellenas, podríamos especificar LLG_FRAME para obtener sólo el contorno.

Explotando la torta

Muchos usuarios desean realizar gráficos donde las porciones se hallen separadas o "explotadas".

Dado que cada porción es dibujada individualmente, parece simple determinar un coeficiente de desplazamiento para hacer que aparezcan separadas pero, ¿qué valor de desplazamiento se utilizaría?. Si se toma un valor arbitrario el resultado será que alguna porción se encime con otra. Entonces se deberá calcular el valor apropiado.

El siguiente gráfico muestra el método a utilizar. Consideremos el círculo externo como un gráfico dividido en cuatro porciones iguales (cada una de 90 grados). Para separar la porción superior derecha deberíamos desplazarla sobre la línea punteada (a 45 grados). Entonces el desplazamiento se basaría en :



```
( Angulo Final - Angulo Inicial ) / 2 => ( 90 - 0 ) / 2 = 45 grados
```

El círculo interno, representa el desplazamiento del radio. Todo esto se puede resolver con cálculos simples de trigonometría, que nos permitirán calcular el desplazamiento sobre X e Y, dado que conocemos la hipotenusa (radio del círculo) y el ángulo. El cálculo sería:

```
nX := COS( dToR( nAngulo ) ) * nDespRadial
nY := SIN( dToR( nAngulo ) ) * nDespRadial
```

Dado que CA-Clipper no tiene las funciones SIN() y COS() se utilizan las de la librería CA-Clipper Tools.

El ejemplo total

A continuación veremos un ejemplo donde graficaremos los resultados de una encuesta sobre solicitudes de empleo en el área de informática.

```
// Archivo : PP.PRG
```

```
// Versión : Clipper 5.3a
// Por : Claudio Gabriel Torrillo
// Para : C&V News - ClipSupport Argentina
// ---------------------------------------------------------------------------
// Revisión : 1.00
// Descripción: Ejemplo Graficación - Gráfico de tortas...
// ---------------------------------------------------------------------------

/*
#cp DEBUG: NO, WARNI: SI, ERSIN: NO, GENPP: NO
#cp LINEA: SI, PRGIN: SI, MODUL: SI, QUIET: SI
#cp SALIRA: .\, COMPIL: Clipper
#cp FINCLP: CP
*/

#include "Common.Ch"
#include "InKey.Ch"
#include "LLIBG.CH"

Function Main()

    // Graficación de solicitudes de empleo en el
    // mercado laboral.
    local aGrafico := {}
    SET VIDEOMODE TO LLG_VIDEO_VGA_640_480_16
    scroll()

    // Generar una pantalla gráfica para trabajo...
    _GEscritorio()

    // Generamos un array con la información a graficar

    // Se ha modificado la estructura del array para
    // obtener algunas funcionalidades extras:
    // Elemento 1 : Etiqueta del dato
    // Elemento 2 : Dato numérico a graficar
    // Elemento 3 : Si la porción debe explotarse a no
    // Elemento 4 : Valor numérico de color del dato.

    aGrafico := { { "Otros" , 7, .F., 09 },;
                  { "Informático", 3, .T., 04 } }

    Graficar( "Demanda Laboral por sectores", aGrafico )

    if _GWMsg( "[Intro] para ver detalle sector informático" )
        retu ( nil )
    endif

    // Reestablecer el escritorio....
    _Gescritorio()

    aGrafico := {   { "Adm de redes" , 70, .F., 08 },;
                    { "Analistas" , 170, .F., 03 },;
                    { "Data/Entry Operador", 130, .F., 05 },;
                    { "Diseñador Gráfico" , 100, .F., 06 },;
                    { "Gte /Jefe /Auditor" , 80, .T., 15 },;
                    { "Gte Org. y Métodos" , 50, .F., 09 },;
                    { "Programadores" , 250, .T., 02 },;
                    { "Soporte Técnico", 150, .F., 04 } ;
                }

    Graficar( "Demanda Laboral Informática", aGrafico )
    _GWMsg( "Presione un tecla para finalizar..." )

Return ( NIL )

Function Graficar( cTitulo, aGrafico )

    // Función encargada de generar un gráfico de torta
    // a partir de la información brindada en el array....
    local aInfo
    local nArco, nColor, nFactor, nGrados, nLoop, nRadDes
    local nCentroX := 320
    local nCentroY := 240
    local nCount := 0
    local nDesA, nDesX, nDesY
    // Obtener la suma total de los datos a graficar
    // Necesario para el cálculo del factor....
```

```
      nCount := 0
      aEval( aGrafico, {|x| nCount += X[2] } )
      // Cuantos grados por unidad de medida
      nFactor := ( 360/nCount )
      // Puntero
      nGrados := 0
      // Radio de desplazamiento
      nRadDes := 10
      for nLoop := 1 TO len( aGrafico )
          // Angulo del arco que representa al dato...
          nArco := aGrafico[ nLoop, 2 ] * nFactor
          if aGrafico[ nLoop, 3 ]
              // Angulo por el cual debe desplazarse la porción
              nDesA := (nArco/2) + nGrados
              // Obtención de nuevas coordenadas de centro
              // según el desplazamiento deseado...
              nDesx := DespX( nDesA, nRadDes, nCentroX )
              nDesy := DespY( nDesA, nRadDes, nCentroY )
          else
              nDesX := nCentroX
              nDesY := nCentroY
          endif
          // Determinar el color
          if aGrafico[ nLoop, 4 ] = NIL
              nColor := nLoop
              aGrafico[ nLoop, 4 ] := nColor
          else
              nColor := aGrafico[ nLoop, 4 ]
          endif
          GEllipse( nDesX, nDesY ,; // Coordenada del centro
                       100, 100 ,; // Radios de la elipse
                        nGrados ,; // Angulo inicial del arco
                nGrados + nArco,; // Angulo final
                       LLG_FILL ,; // Modo de relleno
                         nColor ,; // Color de relleno
                   LLG_MODE_SET ,; // Modo de dibujo
                              0 ,; // Color de borde
                               5 ) // Pixels efecto 3D
          // Correr el puntero de posición de la porción
          // hacia el siguiente ángulo disponible...
          nGrados += aGrafico[ nLoop, 2 ] * nFactor
      NEXT

      _GEtiquetas( aGraficos )

Return ( NIL )

Static Function _GEtiquetas( )
```

- ```
  local aInfo
  // Cargar el font para las etiquetas de los valores
  GFntSet( aInfo := GFntLoad( "rom8pix.fnd" ) )
  // Imprimir los valores
  for nLoop := 1 TO len( aGrafico )
      // Cuadro con referencia de color
      GRect( 05,aInfo[8]*(nLoop+1), 05+aInfo[8], aInfo[8]*(nLoop+2), LLG_FILL, aGrafico
  [nLoop,4], LLG_MODE_SET )
      // Etiqueta de descripción del valor
      GWriteAt( 05+aInfo[8]+03, aInfo[8]*(nLoop+1), aGrafico[nLoop,1], 15, LLG_MODE_SET,
  aInfo )
  next
  // Liberar recursos de font
  GFntErase( aInfo )

  Return ( NIL )

  Static Function DespX( nDesA, nRadDes, nCentro )
  ```

- ```
  // Cálculo de desplazamiento sobre eje X
  local nDesX
  nDesX := nRadDes * COS( DTOR(nDesA) )
  // Dado que siempre graficamos en el 4to cuadrante...
  // (0,0) En coordenada superior derecha.
  // Entonces las coordenadas X son positivas...

  Return ( nCentro + nDesX )
  ```

```
Static Function DespY( nDesA, nRadDes, nCentro )
```

- ```
  // Cálculo de desplazamiento eje Y
  local nDesY
  nDesY := nRadDes * SIN( dTOR(nDesA) )
  // Dado que siempre graficamos en el 4to cuadrante
  // (0,0) En coordenada superior derecha.
  // Entonces las coordenadas Y son negativas....
  ```

```
Return ( nCentro - nDesY )


////////////////////////////////
// Funciones de entorno
////////////////////////////////

Function _GTitulo( cTitulo )
```

- ```
  // Colocar el título del gráfico
  local aInfo := {}
  // Cargar un font para el título.
  GFntSet( aInfo := GFntLoad( "BWAY.FND" ) )
  // Escribir el título centrado en el top de pantalla
  GWriteAt( (640-GWriteAt(0,0,cTitulo,,LLG_MODE_NIL))/2, aInfo[8], cTitulo, 01,
  LLG_MODE_SET, aInfo )
  // Liberar recursos del font.
  GFntErase( aInfo )
  ```

```
Return ( NIL )

Function _Gescritorio()
```

- ```
  // Generar una pantalla gráfica base para el ejemplo
  GFrame( 00,00,639,479,7,15,8,3,3,3,3,LLG_MODE_SET )
  ```

```
Return ( NIL )

Static Function _GMsg( cMsg )
```

- ```
  // Genera una línea de mensaje con el texto especificado...
  local aInfo // Información del font
  local nAltoFrame := 3 // Ancho dibujo FRAME
  GFntSet( aInfo := GFntLoad( "BWAY.FND" ) )
  GFrame( nAltoFrame,480-nAltoFrame*3-aInfo[ 8 ],640-nAltoFrame,479-
  nAltoFrame,7,15,8,3,3,3,3,LLG_MODE_SET,LLG_FRAME )
  GWriteAT( nAltoFrame*3,480-nAltoFrame*2-aInfo[ 8 ],cMsg,01,LLG_MODE_SET )
  ```

```
Return ( NIL )

Function _GWMsg( cMsg )
```

- ```
  // Muestra un mensaje y espera la acción del usuario
  _GMsg( cMsg )
  ```

```
Return ( inkey(0) )
```

Resumen final Como podrán apreciar, luego del ejemplo presentado, es fácil obtener la presentación de gráficos con el nuevo CA-Clipper 5.3. En la próxima entrega desarrollaremos ejemplos donde podremos ver la forma de realizar gráficos de barras y de líneas.
```