UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Controles gráficos de *MetaCard: dibujo e interacción* para el desarrollo de aplicaciones multimedia

Apellidos, nombre	Agustí i Melchor, Manuel (magusti@disca.upv.es)
Departamento	Departamento de Informática de Sistemas y Computadores
Centro	Escola Tècnica Superior d'Enginyeria Informàtica Universitat Politècnica de València

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

1 Resumen

En este artículo y desde la perspectiva de una herramienta de autor multimedia como es *MetaCard [1]*, se va a explorar el uso de los elementos que provee para la utilización de controles y primitivas gráficas.

Se ilustrará, con breves ejemplos, el desarrollo de partes de aplicaciones que utilicen estos recursos multimedia, atendiendo a las características multiplataforma que tiene *MetaCard*.

2 Objetivos

Tras la lectura y experimentación de este documento el alumno será capaz de exponer la forma de uso de las primitivas gráficas disponibles para la realización de aplicaciones multimedia interactivas que provee *MetaCard*,

Para ello, las separaremos en dos grupos que tendrán su propia introducción:

- El dibujado de primitivas básicas como líneas, círculos, rectángulos, etc
- La posibilidad de que los contenidos que se muestran con ellas sean dinámicos, esto es, cambiantes con el tiempo o con la interacción del usuario.

En este documento se exponen pequeños listados de código relacionados con las diferentes órdenes que se exponen para dar pie al usuario a que investigue su uso, ejecutando la aplicación, probando el código y consultando ayuda incluía dentro de la propia herramienta. No espere largas y detalladas explicaciones del código: sea activo, pruébelo.

3 Introducción

A la hora de incluir representaciones visuales en una aplicación, es posible hacer uso de imágenes en formato de mapa de bits o como gráficos vectoriales. Estos segundos son los que vamos a revisar en este documento. *MetaCard* proporciona polígonos (irregulares o regulares), curvas, óvalos y rectángulos (con o sin esquinas redondeadas). Todos ellos son variantes del objeto *graphic*.

Estas primitivas gráficas permiten mostrar información de carácter estático o dinámico, dando lugar a animaciones. También son especialmente interesantes por que ofrecen un grado de libertad a la hora de crearlas, en tanto en cuanto existen definidos unos operadores que permiten representar los objetos geométricos (as primitivas gráficas) más habituales. Por último, no incrementan, generalmente, el uso de recursos de la aplicación; al tiempo que permiten reescalar la representación sin pérdida de calidad, a diferencia de los mapas de bits.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

4 Primitivas de dibujo

En *MetaCard* se pueden definir seis tipos genéricos: polígonos irregulares, curvas, rectángulos, óvalos, rectángulos de esquinas redondeadas y polígonos regulares. Se definen los gráficos vectoriales con las características que permiten editar las diferentes pestañas de la caja de diálogo de propiedades, fig. 1.

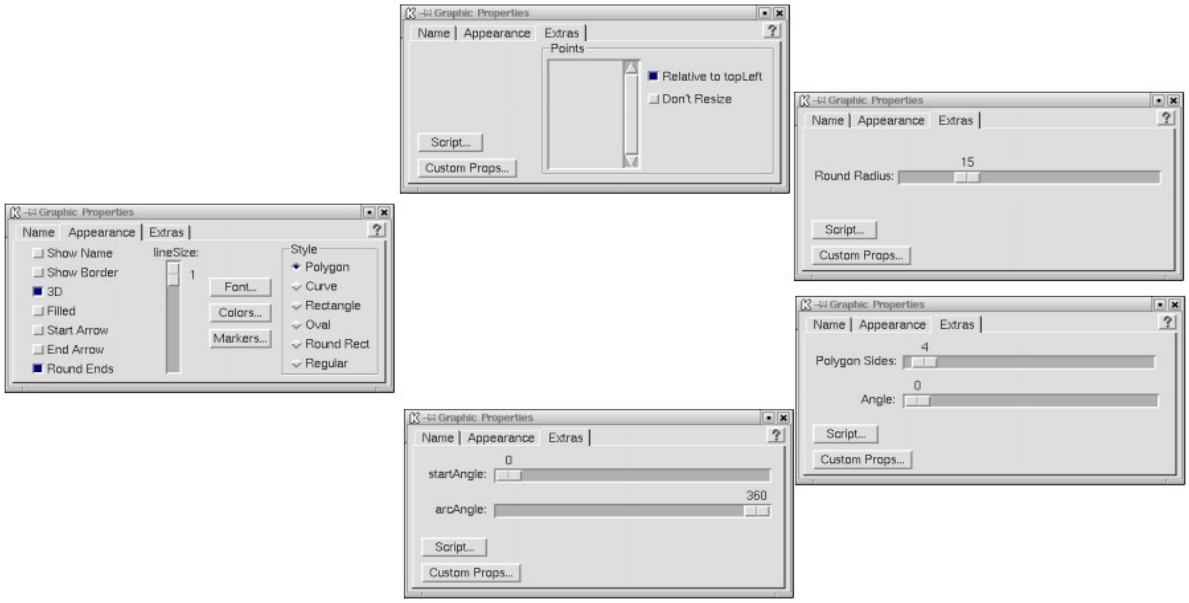


Figura 1: Aspecto y otras propiedades de un objeto tipo "graphic" (gráfico).

No se preocupe si no la ve bien: cree un gráfico vectorial y examine las propiedades del mismo. Algunas de las propiedades son aplicables en función del tipo de objeto que definan: ¡el radio no es aplicable a un rectángulo, pero sí a una circunferencia!.

Veamos algunos ejemplos de subtipos de un gráfico, en concreto el primero y el cuarto siguiendo el orden de aparición en la barra de "*Menu Bar*".

El *Graphic tool (polygon)* se puede utilizar para realizar objetos cerrados o no. Es como la herramienta de "polilínea" que tienen otras aplicaciones, puesto que pueden quedar abiertos los dos extremos.

La fig. 2 muestra cómo se puede realizar una gráfica a partir de dos series de datos numéricos que asociaremos a sus coordenadas de filas de y columnas. Estos valores pueden ser el resultado de un cálculo previo, algo cambiante en tiempo de ejecución. Aquí por simplicidad los he fijado estáticos en campos de texto. La aplicación consiste en una pila que muestra un gráfico ("grafic"), junto con dos campos de texto ("copiaValors" y "coordXY") para contener la información de coordenadas que hay que “pintar”. El código, que se muestra en el listado 1, es para el control de tipo gráfico. Acuérdese de desactivar la propiedad **Relative to topLeft** del gráfico "grafic".

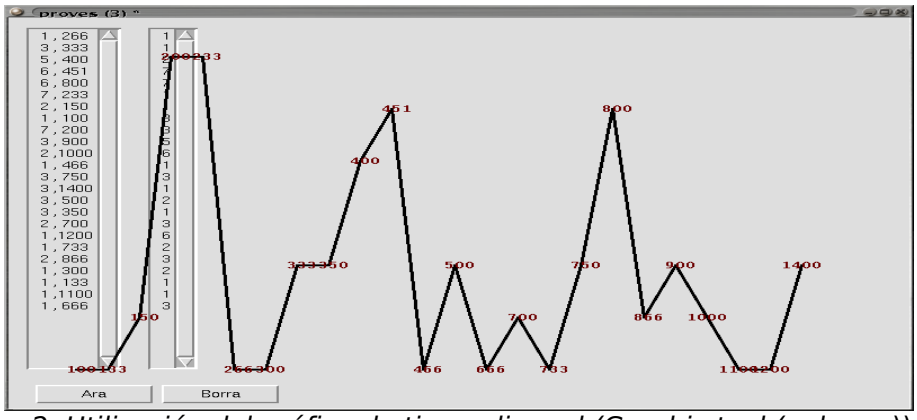


Figura 2: Utilización del gráfico de tipo poligonal (Graphic tool (polygon)).

 # Código del gráfico "grafic"

```
on mouseUp
# Cada cuánto de ancho ponemos un punto? En funció del ancho  de la ventana
  sort lines of me ascending by item 1 of each
  put trunc((the height of this card - 120) / \
                  (the first item of last line of me - the first item of first line of me)  ) \
      into separacioY
  put  (the first item of first line of me) into minValor
# Normalizar alturas al alto de la ventana
  put fld "copiaValors" into me
  put trunc((the width of this card - 120) / the number of lines of me  ) \
          into separacioX
  sort lines of me ascending by item 2 of each
  put empty into fld "coordXY"
  repeat with i = 1 to the number of lines of me
    put 60 + (separacioX * (i – 1)),\
        ((the height of this card - 60) - \
          (separacioY * (the first item of line i of me - minValor)))  &\
        return \
        after fld "coordXY"
  end repeat
# Aplicarlo al gráfico
 put fld "coordXY" into the lines of me
end mouseUp
```

Listado 1: Código de ejemplo de polilínea.

El *Graphic tool (oval)* permite la creación de círculos y, si está activa la propiedad **Filled**, de circunferencias y arcos. Veamos un ejemplo que puede servir para empezar una aplicación o para entretener a nuestro usuario mientras hacemos alguna otra cosa. Al estilo de cuando empieza una secuencia de cine, fig, 3, vamos a mostrar una cuenta descendente y sobre esta una circunferencia que se abre y se cierra. Al finalizar la cuenta, empezaría el verdadero intercambio de información con el usuario.

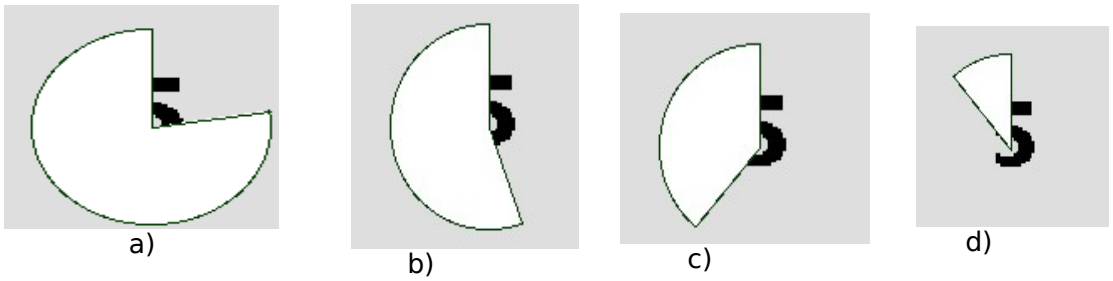


Figura 3: Ejemplo de objeto gráfico de tipo circular (Graphic tool (oval)), mostrando cuatro instantes de la ejecución de esta animación.

```
#Códiog de la tarjeta
on openCard
  put "5" into fld "num"; send "mouseUp" to grc "circul"
end openCard

on continuar
  go to next card
end continuar
```

Listado 2: Primera parte del código de ejemplo de un óvalo.

La tarjeta en cuestión consta únicamente de dos objetos que examinaremos enseguida, de momento su misión es inicializar: el valor inicial de la cuenta atrás y enviar un mensaje de que "empiece el espectáculo". Cuando haya terminado, ya la avisarán para que cambie a otra tarjeta. Aquí el código de "ir a otra tarjeta" se ha hecho lo más sencillo posible para no desviarnos demasiado del manejo de gráficos. Pero observe que se puede hacer cualquier cosa, desde hacer sonar un archivo de audio hasta elegir a qué tarjeta se va, no tiene por qué ser la siguiente. Véase listado 2.

El gráfico tiene como misión hacer un ciclo completo, esto es, ir cerrándose dejando ver lo que hay debajo (que es la cuenta) y se realimenta así mismo reenviándose el mensaje. Mientras *MetaCard* puede ejecutar otro código "en paralelo", listado 3. Terminado el ciclo, envía un mensaje al campo de texto para que este decremente la cuenta y ya decidirá aquel qué sigue.

Quien controla cuando acaba esto es el campo de texto, véase listado 4, así que puede tomar más o menos tiempo y el mecanismo sigue siendo válido. La labor de este control es decrementar la cuenta, mostrarla, señalar al gráfico que haga su trabajo y, cuando sea el momento, decirle entonces a la tarjeta que la cuenta ha terminado y que esta decida cómo, cuándo y a qué tarjeta pasar. El usuario sólo verá un cambio que asumirá como el siguiente contenido.

```
# Gráfico "circul"
on mouseUp
  set startAngle of me to 90
  set arcAngle of me to 360
  send "paso" to me
end mouseUp


on paso
#  put "Ara " & arcAngle of me
  set arcAngle of me to arcAngle of me - 20
  if  arcAngle of me = 0
  then
    send "decrNum" to fld "num"
  else
    send "paso" to me in 10 millisecond
  end if
end paso
```

Listado 3: Segunda parte del código de ejemplo de óvalo.

```
# Campo de texto "num"
on decrNum
#  put "text = " & the text of me
  if the text of me <> 0
  then
    put the text of me - 1 into me
    send "mouseUp" to graphic "circul"
  else
    send "continuar" to this card
  end if
end decrNum
```

Listado 4: Tercera parte del código de ejemplo de óvalo.

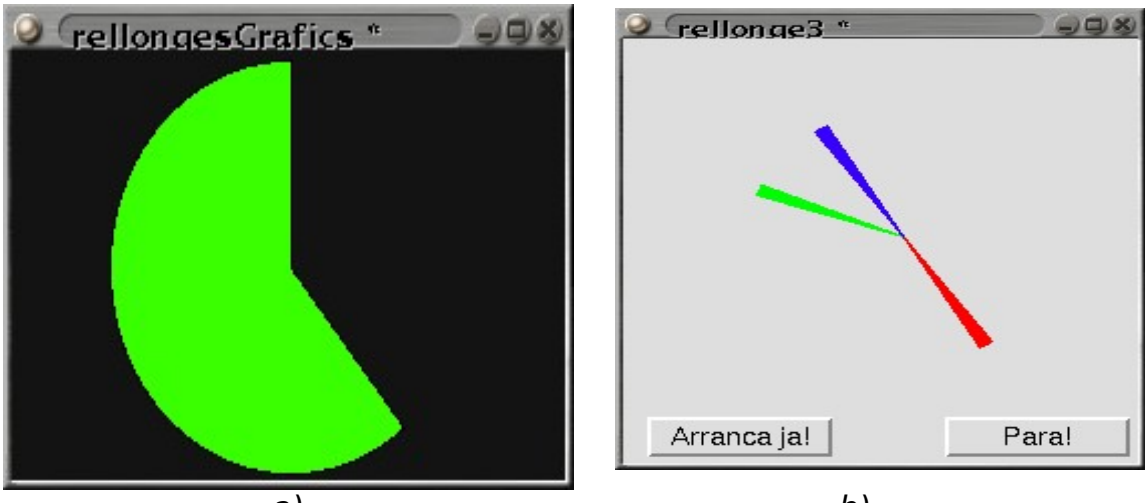
Si quiere decorar un poco más la presentación, puede aprovechar que la pila es la primera en poder entrar en acción para tomar algunas medidas de carácter estético como las del listado 5.

```
# Pila
on preOpenStack
  set the backdrop to black
  move me to the screenLoc
  set the width of me to 256
  set the height of me to 256
  go to card "Inici"
end preOpenStack
```

Listado 5: Cuarta y última parte del código de ejemplo de óvalo.

5 Objetos animados con gráficos vectoriales

Se puede aprovechar un objeto de tipo gráfico para realizar sencillas animaciones por que sus valores se pueden generar en tiempo de ejecución. Veamos un ejemplo que utiliza como primitiva gráfica el óvalo. Este ya ha sido expuesto antes para reducir la necesidad de explicaciones de las primitivas de dibujo empleadas.



a) *b)*

Figura 4: Ejemplos de relojes hechos con óvalos.

¿Qué vamos a realizar? Un reloj. Bueno, uno con una sola varilla “muy grande”, fig. 4a. El caso de un reloj analógico más usual, fig. 4b. es lo mismo con tres “varillas”.

```
# La pila
on openCard
  set the alwaysBuffer of this stack to true
end openCard
local idMensatge, numId

# La tarjeta
local varAux

on mouseUp quinBoto
  if quinBoto = 1
  then
    set the startAngle of graphic "segons" to 90
    set the arcAngle of graphic "segons" to 360
      else
    send "paraLo" to graphic "segons"
  end if
end mouseUp
```

Listado 6: Primera parte del código del reloj de la fig. 4a.

La pila inicializa una propiedad importante en este caso, debido a la variabilidad que estamos introduciendo en el contenido de la ventana: **alwaysBuffer**. La tarjeta permite parar y reiniciar el proceso. El código de ambos se muestra en el listado 6.

La varilla se ha implementado con la primitiva gráfica de un óvalo que se denomina “segons” y el código de este está en el listado 7.

¿Cuál es la diferencia con el reloj con tres varillas de la fig. 4b? Que hay tres objetos gráficos de tipo óvalo: “segons”, “minuts”y “hores” que avanzan a ritmos diferentes, de segundos, minutos y horas respectivamente. Y que se han definido con colores (rojo, verde y azul respectivamente) y una geometría más estilizada queriendo recordar así a las varillas de un reloj analógico.

El cambio más notable es que se replicaría el objeto y código asociados a la única varilla de la fig. 4a y que se tendrían que modificar los tiempos con los que cada varilla ser recuerda, a si misma, el paso del tiempo. Dejo para el lector interesado esta implementación.

```
# Gráfico "segons"
on mouseUp
  send "pasaElTemps" to me in 1 millisecond
end mouseUp

on pasaElTemps
  set the arcAngle of me to (the arcAngle of me - 1)
  if the arcAngle of me <> 0 then
    delete first line of idMensatge
    send "pasaElTemps" to me in 1 second
    put the result & return after idMensatge
  end if
end pasaElTemps

on paraLo
  repeat with numId = the number of lines of idMensatge down to 1
    cancel line numId of idMensatge
  end repeat
end paraLo
```

Listado 7: Segunda parte del código del reloj de la fig. 4a

6 Conclusiones

A lo largo de este documento hemos visto el uso de los controles y primitivas de naturaleza gráfica presentes en MetaCard. También hemos abordado ejemplos de uso de estos elementos animaciones .

La exposición de las opciones disponibles se ha acompañado de ejemplos breves para que el lector pueda continuar el aprendizaje por sus propios medios y necesidades. Se recomienda, encarecidamente, hacer uso de la imaginación y leer la documentación que incorpora *MetaCard.*.

7 Bibliografía

[1] MetaCard <http://www.metacard.com/>.