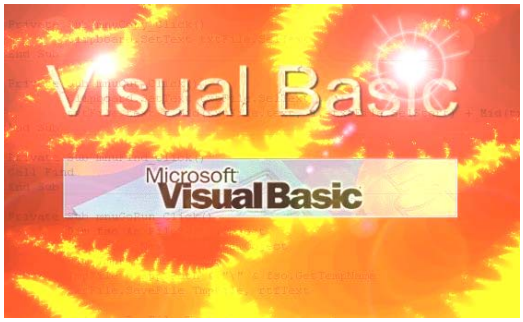
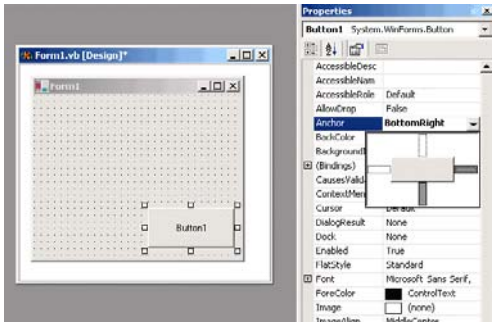
CURSO

Curso Completo de Visual Basic 6.0





Escuela Superior de Ingenieros Industriales
UNIVERSIDAD DE NAVARRA
Javier García de Jalón · José Ignacio Rodríguez
Alfonso Brazález · Patxi Funes · Eduardo Carrasco · Jesús Calleja

6.6 EVENTOS Y PROPIEDADES RELACIONADAS CON GRÁFICOS

6.6.1 El evento Paint

El evento ***Paint*** se ejecuta cuando un objeto -de tipo ***form*** o ***pictureBox***- se hace visible. Su finalidad es que el resultado de los ***métodos gráficos*** y del método ***print*** aparezcan en el objeto correspondiente. Hay que tener en cuenta que si se introducen métodos gráficos en el procedimiento ***form_load*** su resultado no aparece al hacerse visible el formulario (es como si se dibujara sobre el formulario antes de que éste existiera). Para que el resultados de ***print*** y de los métodos gráficos aparezcan al hacerse visible el formulario, deben introducirse en el procedimiento ***paint_form***.

También los controles ***pictureBox*** tienen evento ***paint***, que se ejecuta al hacerse visibles.

El evento ***Paint*** tiene mucha importancia en relación con el refresco de los gráficos y con la velocidad de ejecución de los mismos. En los apartados siguientes se completará la explicación de este tema.

6.6.2 La propiedad DrawMode

Esta es una propiedad bastante importante y difícil de manejar, sobre todo si se quieren realizar cierto tipo de acciones con los métodos gráficos.

La opción por defecto es la nº 13: ***Copy Pen***.

La propiedad ***DrawMode*** controla cómo se dibujan los controles ***line*** y ***shape***, así como los resultados de los métodos gráficos ***pset***, ***line*** y ***circle***. La opción por defecto hace que cada elemento gráfico se dibuje con el color correspondiente (por defecto el ***foreColor***) sobre lo dibujado anteriormente. En ocasiones esto no es lo más adecuado pues, por ejemplo, si se superponen dos figuras del mismo color o si se dibuja con el ***backColor***, los gráficos resultan indistinguibles.

Para entender cómo funciona ***DrawMode*** es necesario tener claros los conceptos de ***color complementario*** y ***combinación de dos colores***. El color complementario de un color es el color que sumado con él da el ***blanco*** (&HFFFFFF&). Por ejemplo, el color complementario del ***rojo*** (&H0000FF&) es el ***cyan*** (&HFFFF00&).

El ***color complementario*** se puede obtener mediante la simple resta del color blanco menos el color original. Por su parte la ***combinación de dos colores*** es el color que resulta de aplicar el operador lógico ***Or***: el color resultante tiene sus *bits* a 1 si alguno o los dos de los colores originales tiene a 1 el *bit* correspondiente. La explicación de los distintos valores de la propiedad ***DrawMode*** que se obtiene del ***Help*** es la siguiente:

Constant	Setting	Description
VbBlackness	1	Blackness.
VbNotMergePen	2	Not Merge Pen— Inverse of setting 15 (Merge Pen).
VbMaskNotPen	3	Mask Not Pen — Combination of the colors common to the background color and the inverse of the pen.
VbNotCopyPen	4	Not Copy Pen — Inverse of setting 13 (Copy Pen).
VbMaskPenNot	5	Mask Pen Not — Combination of the colors common to both the pen and the inverse of the display.
VbInvert	6	Invert — Inverse of the display color.
VbXorPen	7	Xor Pen — Combination of the colors in the pen and in the display color, but not in both.
VbNotMaskPen	8	Not Mask Pen — Inverse of setting 9 (Mask Pen).
VbMaskPen	9	Mask Pen — Combination of the colors common to both the pen and the display.
VbNotXorPen	10	Not Xor Pen — Inverse of setting 7 (Xor Pen).
VbNop	11	Nop — No operation — output remains unchanged. In effect, this setting turns drawing off.
VbMergeNotPen	12	Merge Not Pen — Combination of the display color and the inverse of the pen color.
VbCopyPen	13	Copy Pen (Default) — Color specified by the **ForeColor** property.
VbMergePenNot	14	Merge Pen Not — Combination of the pen color and the inverse of the display color.
VbMergePen	15	Merge Pen — Combination of the pen color and the display color.
VbWhiteness	16	Whiteness.

El explicar más a fondo las distintas aplicaciones de esta propiedad esta fuera del alcance de este manual introductorio.

6.6.3 Planos de dibujo (Layers)

Visual Basic 6.0 considera ***tres planos superpuestos*** (layers): el plano frontal, el plano intermedio y el plano de fondo. Es importante saber en qué plano se introduce cada elemento gráfico para entender cuándo unos elementos se superpondrán a otros en la pantalla. En principio, los tres planos se utilizan del siguiente modo:

1. En el ***plano frontal*** (***Front***) se dibujan *todos los controles*, excepto los controles gráficos y las labels.

2. En el ***plano intermedio*** se representan los *controles gráficos* y *labels*.

3. En el ***plano de fondo*** se representa el *color de fondo* y el resultado de los *métodos gráficos*.

Estas reglas tienen excepciones que dependen de la propiedad ***AutoRedraw***, de la propiedad ***ClipControl*** y de si los métodos gráficos se utilizan o no asociados al evento ***paint***.

6.6.4 La propiedad AutoRedraw

Esta propiedad tiene una gran importancia. En principio, todas las aplicaciones de ***Windows*** permiten superponer ventanas y/u otros elementos gráficos, recuperando completamente el contenido de cualquier ventana cuando ésta se selecciona de nuevo y viene a primer plano (es la ventana activa). A esto se llama ***redibujar*** (*redraw*) la ventana. Cualquier aplicación que se desarrolle en ***Visual Basic 6.0*** debe ser capaz de redibujarse correctamente, pero para ello el programador debe conocer algo de la propiedad ***AutoRedraw***.

Por defecto, ***Visual Basic 6.0*** redibuja siempre los controles que aparecen en un formulario.
Esto no sucede sin embargo con el resultado de los ***métodos gráficos*** y de ***print***. Para que la salida de estos métodos se redibuje es necesario adoptar uno de los dos métodos siguientes:

1. Si en el ***form*** o ***pictureBox*** la propiedad ***AutoRedraw*** está en ***false***:

   - Si los ***métodos gráficos*** y ***print*** están en el procedimiento correspondiente al evento ***paint*** se redibujan en el *plano de fondo* (los métodos vuelven a ejecutarse, por lo que el proceso puede ser lento en ciertos casos).

   - Si los ***métodos gráficos*** y ***print*** están fuera del evento ***paint*** no se redibujan.

2. Si en el ***form*** o ***pictureBox*** la propiedad ***AutoRedraw*** está en ***true***:

- Si los ***métodos gráficos*** y ***print*** están en el evento ***paint*** se ignoran.
- Si los ***métodos gráficos*** y ***print*** están fuera del evento ***paint*** se redibujan guardando en memoria una copia de la zona de pantalla a refrescar. Este es la forma más rápida de conseguir que los gráficos y el texto se redibujen. Tiene el inconveniente de necesitar más memoria.

La propiedad ***AutoRedraw*** de los ***forms*** y de las ***pictureBox*** es independiente, por lo que las dos formas anteriores de conseguir que los gráficos se redibujen se pueden utilizar conjuntamente, por ejemplo una en el formulario y otra en las ***pictureBox***.

6.6.5 La propiedad ClipControl

Por defecto esta propiedad de las ***forms*** y ***pixtureBox*** está en ***true***. En este caso los controles están siempre por encima de la salida de los métodos gráficos, por lo que nunca por ejemplo una línea se dibujará sobre un botón o una barra de desplazamiento (los controles están siempre en el plano frontal o en el plano intermedio, según se ha explicado antes).

Cuando la propiedad ***ClipControl*** se pone a ***false*** se produce una doble circunstancia:

- Los métodos gráficos situados en un evento ***paint*** siempre se dibujan en el *plano de fondo* y por tanto respetan los controles.
- Los métodos gráficos situados fuera de un evento ***paint*** se dibujan sobre cualquier elemento que esté en la pantalla, incluidos los controles.

6.7 EJEMPLOS

A continuación se muestra dos ejemplos que hacen uso de algunos de los controles y métodos gráficos explicados previamente.

6.7.1 Ejemplo 6.1: Gráficos y barras de desplazamiento

Este primer programa, cuyo formulario se muestra en la **Figura 6.9**, es un ejemplo sencillo que permite utilizar algunas de las herramientas gráficas de ***Visual Basic***. Para ello se han utilizado dos barras de desplazamiento que, junto a otras dos cajas de texto, modificarán y visualizarán las coordenadas del punto a dibujar.

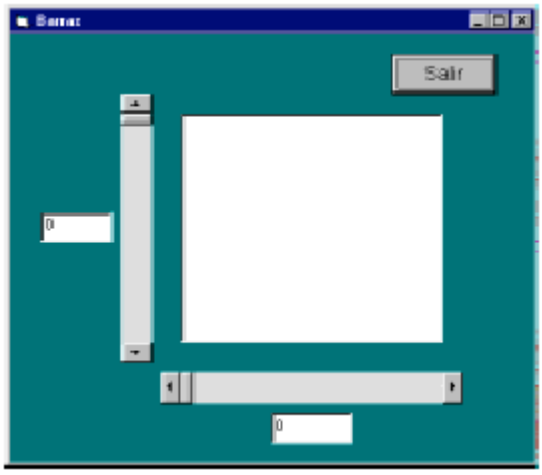


Figura 6.9. Movimiento de un punto con ***Pset***.

La **Tabla 6.**5 muestra los objetos y las propiedades a considerar en este ejemplo.

Control	Propiedad	Valor	Control	Propiedad	Valor
Hscrollbar	Name	HScroll1	TextBox	Name	txtCaja2
	LargeChange	5		Text	0
	Max	100	TextBox	Name	txtCaja3
	Min	0		Text	0
	SmallChange	1	PictureBox	Name	PctBox
VScrollbar	Name	VScroll1		BackColor	&H00FFFFFF&
	LargeChange	5	CommandButton	Name	Command1
	Max	100		Caption	Salir
	Min	0			
	SmallChange	1			

Tabla 6.5. Controles y propiedades del Ejemplo 6.2.

Se presenta a continuación el código del programa:

```
Private Sub Command1_Click()
        End
End Sub

Private Sub Form_Load()
        pctBox.Scale (0, 0)-(100, 100) 'Se escala la picture box
        pctBox.DrawWidth = 5
End Sub

Private Sub HScroll1_Change()
        txtCaja3.Text = Format(HScroll1.Value) 'Leemos valor de la barra hor.
        pctBox.PSet (HScroll1.Value, VScroll1.Value), vbRed 'Dibuja punto
End Sub




Private Sub txtCaja2_KeyPress(KeyAscii As Integer)
        Dim valor As Integer
```

```
        valor = Val(txtCaja2.Text)
        If KeyAscii = 13 Then
                If valor <= VScroll1.Max And valor >= VScroll1.Min Then
                   VScroll1.Value = valor 'Asignamos valor del textbox a barra
                ElseIf valor > VScroll1.Max Then
                   VScroll1.Value = VScroll1.Max
                Else
                   VScroll1.Value = VScroll1.Min
                End If
        End If
End Sub


Private Sub txtCaja3_KeyPress(KeyAscii As Integer)
        Dim valor As Integer
        valor = Val(txtCaja3.Text)
        If KeyAscii = 13 Then
                If valor <= HScroll1.Max And valor >= HScroll1.Min Then
                    HScroll1.Value = valor 'Asignamos valor del textbox a barra
                ElseIf valor > HScroll1.Max Then
                   HScroll1.Value = HScroll1.Max
                Else
                   HScroll1.Value = HScroll1.Min
                End If
        End If
End Sub


Private Sub VScroll1_Change()
        txtCaja2.Text = Format(VScroll1.Value) 'Leemos valor barra vertical
        pctBox.PSet (HScroll1.Value, VScroll1.Value), vbRed 'Dibujamos punto
End Sub
```

6.7.2 Ejemplo 6.2: Representación gráfica de la solución de la ecuación de segundo grado

En este segundo ejemplo, cuyo formulario se muestra en la Figura 6.10, se representan el lugar de raíces de la ecuación de segundo grado en función de los coeficientes, o más en concreto en función de los cocientes B/A y C/A. El valor de estas relaciones se cambia interactivamente por medio de dos barras de desplazamiento.

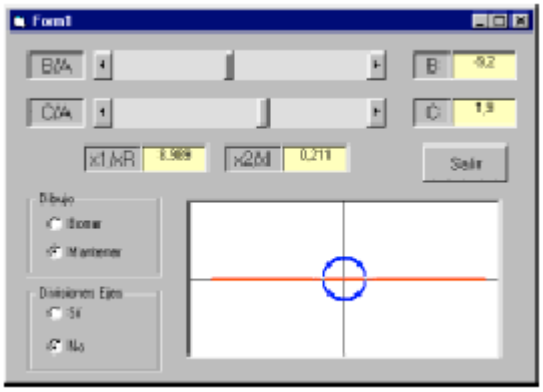


Figura 6.10. Raíces de una ecuación de 2º grado.

El programa permite además la posibilidad de mantener dibujadas las soluciones anteriores de la ecuación, o borrarlas y dibujar sólo las últimas raíces calculadas borrando las anteriores. Para finalizar el programa basta presionar el botón ***Salir***.

La Tabla 6.6 muestra los nombres y los valores de las principales propiedades de los objetos que aparecen en la Figura 6.10.

Control	Propiedad	Valor	Control	Propiedad	Valor
Frame	Name	fraDib	Label	Name Caption	Label2 C/A
	Caption	Dibujo	Label	Name Caption	Label3 B
Frame	Name	fraEjes	Label	Name Caption	Label4 C
	Caption	Divisiones Ejes	Label	Name Caption	Label5 X1/XR
HScrollBar	Name	hsbBA	Label	Name Caption	Label6 X2/X1
	LargeChange	10	CommandButton	Name	CmdSalir
	Max	1000		Caption	Salir
	Min	-1000	Label	Name	lblBA, lblCA, lblX1, lblB2
	SmallChange	1		BackColor	&H00C0FFFF&
HScrollBar	Name	hsbCA	Option	Name	optD1
	LargeChange	10		Caption	Borrar
	Max	100	Option	Name	optD2
	Min	-100		Caption	Mantener
	SmallChange	1	Option	Name	OptNo
PictureBox	Name	pctBox		Caption	No
	BackColor	&H00FFFFFF&	Option	Name	OptSi
Label	Name Caption	Label1 B/A		Caption	Si

Tabla 6.6. Controles y propiedades del Ejemplo 6.3.

Todas las labels que aparecen tienen la propiedad *BorderStyle* igual a *1- Fixed Single*.

El código del programa es el siguiente:

```
Option Explicit
Dim a, b, c As Double
Dim x1, x2, dis, xr, xi As Double

Private Sub divisiones(nx As Integer, ny As Integer)
        Dim i As Integer
        Dim x, xinc, y, yinc As Single
        pctBox.DrawWidth = 1
        xinc = 20 / (nx - 1)
        x = -10
        For i = 1 To nx
                pctBox.Line (x, 0)-(x, -1)
                x = x + xinc
        Next i
        yinc = 10 / (ny - 1)
        y = -5
        For i = 1 To ny
                pctBox.Line (-1, y)-(0, y)
                y = y + yinc
        Next i
        pctBox.DrawWidth = 2
End Sub


Private Sub cmdSalir_Click()
        End
End Sub

Private Sub Form_Load()
        pctBox.Scale (-10, 5)-(10, -5)
End Sub

Private Sub hsbBA_Change()
```

```
        a = 1
        b = hsbBA.Value / 10#
        c = hsbCA.Value / 10#
        lblBA.Caption = b
        lblCA.Caption = c
        dis = b ^ 2 - 4 * a * c
        If optD2.Value = True Then           'mantener
           pctBox.AutoRedraw = True
        Else                                 'borrar
           pctBox.AutoRedraw = False
           pctBox.Cls
        End If
        If dis > 0 Then
               x1 = (-b + Sqr(dis)) / (2 * a)
               x2 = (-b - Sqr(dis)) / (2 * a)
               lblX1.Caption = Format(x1, "###0.000")
               lblX2.Caption = Format(x2, "###0.000")
               pctBox.PSet (x1, 0), vbRed
               pctBox.PSet (x2, 0), vbRed
        ElseIf dis = 0 Then
               x1 = -b / (2 * a)
               x2 = x1
               lblX1.Caption = Format(x1, "###0.000")
               lblX2.Caption = ""
               pctBox.PSet (x1, 0), vbGreen
        Else
           xr = -b / (2 * a)
           xi = Sqr(-dis) / (2 * a)
           lblX1.Caption = Format(xr, "###0.000")
           lblX2.Caption = Format(xi, "###0.000")
           pctBox.PSet (xr, xi), vbBlue
           pctBox.PSet (xr, -xi), vbBlue
        End If
        If optSi = True Then
               Call divisiones(10, 5)
        End If
End Sub

Private Sub hsbCA_Change()
        a = 1
        b = hsbBA.Value / 10#
        c = hsbCA.Value / 10#
        lblBA.Caption = b
        lblCA.Caption = c
        dis = b ^ 2 - 4 * a * c
        If optD2.Value = True Then    'mantener
          pctBox.AutoRedraw = True
        Else                           'borrar
               pctBox.AutoRedraw = False
               pctBox.Cls
        End If
        If dis > 0 Then
          x1 = (-b + Sqr(dis)) / (2 * a)
          x2 = (-b - Sqr(dis)) / (2 * a)
          lblX1.Caption = Format(x1, "###0.000")
          lblX2.Caption = Format(x2, "###0.000")
          pctBox.PSet (x1, 0), vbRed
          pctBox.PSet (x2, 0), vbRed
        ElseIf dis = 0 Then
               x1 = -b / (2 * a)
               x2 = x1
               lblX1.Caption = Format(x1, "###0.000")
               lblX2.Caption = ""
               pctBox.PSet (x1, 0), vbGreen



        Else
               xr = -b / (2 * a)
               xi = Sqr(-dis) / (2 * a)
               lblX1.Caption = Format(xr, "###0.000")
               lblX2.Caption = Format(xi, "###0.000")
               pctBox.PSet (xr, xi), vbBlue
               pctBox.PSet (xr, -xi), vbBlue
        End If
        If optSi = True Then
               Call divisiones(10, 5)
        End If
End Sub

Private Sub optD1_Click()
        pctBox.AutoRedraw = True
        pctBox.Cls
        pctBox.DrawWidth = 1
        pctBox.Line (-90, 0)-(90, 0), vbBlack
        pctBox.Line (0, -45)-(0, 45), vbBlack
        pctBox.DrawWidth = 2
```

```
End Sub

Private Sub pctBox_Paint()
        pctBox.AutoRedraw = True
        pctBox.Line (-90, 0)-(90, 0), vbBlack
        pctBox.Line (0, -45)-(0, 45), vbBlack
        pctBox.DrawWidth = 2
End Sub
```

6.8 BARRAS DE HERRAMIENTAS (TOOLBARS)

Con ***Visual Basic 6.0*** es fácil crear ***barras de herramientas*** constituidas por botones clicables, al estilo de las aplicaciones de ***Windows***. De ordinario las barras de herramientas dan acceso a las funciones o comandos más comunes de los menús de la aplicación.

Se puede crear una barra de herramientas por medio de un ***PictureBox*** colocado en un formulario. En este ***PictureBox*** se pueden colocar controles ***CommandButton*** o ***Image*** en los que se programa el evento ***click***. La propiedad ***Picture*** del control ***Image*** puede contener la dirección de alguno de los iconos estándar que vienen con ***Visual Basic*** (extensión ****.ico***) o la de un icono construido por el programador.

En el caso de los formularios MDI se puede colocar una barra de herramientas en el ***MDIform***, que automáticamente adquiere la anchura del formulario.

Continuará.....

Nota de Radacción: El lector puede descargar este capítulo y capítulos anteriores del curso desde la sección ***"Artículos Técnicos"*** en el sitio web de **EduDevices (www.edudevices.com.ar)**