Solve These Irregular Problems

by Karl E. Peterson

Click & Retrieve Source CODE!

CREATING A NONRECTANGULAR USERCONTROL

Q I want to build an ActiveX control, similar to the shape component provided with VB, to draw five-sided polygons. I have found the Polygon API useful for drawing these shapes on the UserControl, but when I set the BackStyle of the UserControl to Transparent, no drawing appears on the control. VB methods alone don't allow me to fill the polygon. How can I build a polygon control with a transparent BackStyle?

A VB UserControls' support of transparent backgrounds can be confusing and at times inconsistent. The trick is to designate a given area of the control you intend to draw on. You do this using the MaskPicture and MaskColor properties of the UserControl.

Start by creating a bitmap that represents which area(s) you want opaque and which area(s) you want transparent. Due to a flaw in Windows 95, this bitmap should be monochrome, using white for transparent and black for opaque. Assign white as the MaskColor and this bitmap as the MaskPicture. Now, if the BackStyle is set to Transparent, areas colored white in the MaskPicture bitmap are invisible (see Figure 1).

This approach is fairly limited, in that you must decide the size and shape of your controls at design time. A more flexible approach is to generate the MaskPicture bitmap at run time, allowing the user to resize your control at will. The first step is to calculate the coordinates of each vertex in the polygon. You've probably got this angle covered, because you're already passing this information to the Polygon API, but I've worked up a little routine that performs these calculations for any size polygon, having any number of sides, and at any rotation angle (see Listing 1).

Once you have these coordinates, create a memory device context (DC) and a memory-based monochrome bitmap. Select the bitmap into the DC, flood the background with white, then use the Polygon API to draw the polygon in black (see Listing 2).

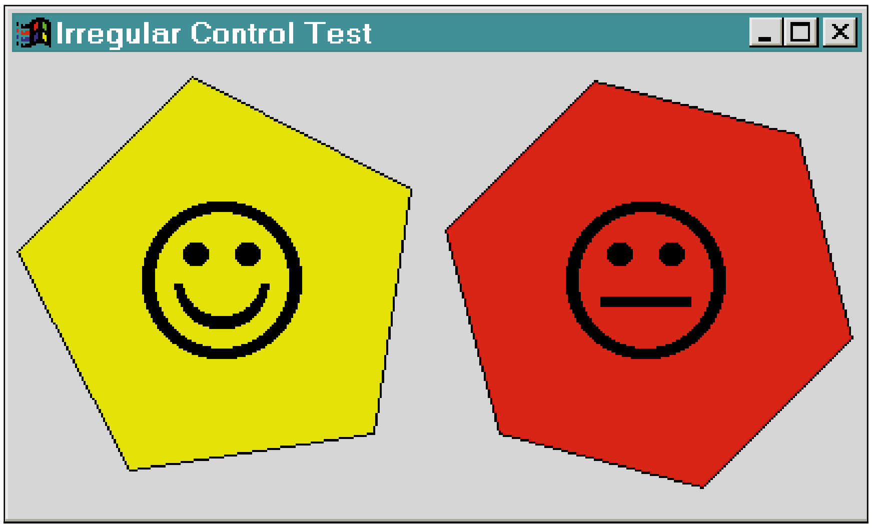

FIGURE 1 ***Avoiding the Rectangular Rut.*** *Controls can be just about any shape you can draw. Pictured here are two polygonal button controls, whose MaskPictures were generated using the code in Listings 1-3. These buttons, each "sized" to fill half the form, don't react to mouse clicks except within their visible areas.*

Karl E. Peterson is a GIS analyst with a regional transportation planning agency and serves as a member of the Visual Basic Programmer's Journal *Technical Review and Editorial Advisory Boards. Based in Vancouver, Wash., he's also an independent programming consultant specializing in ActiveX controls. In addition to contributing to various journals, Karl coauthored* Visual Basic 4 How-To *from Waite Group Press. Online, he's a Microsoft MVP, and a section leader in several* VBPJ *online forums. Find more of Karl's VB samples at http://www.mvps.org/vb.*

Ask the VB Pro provides you with free advice on programming obstacles, techniques, and ideas. Read more answers from our crack VB pros on the Web at http://www.inquiry.com/thevbpro. You can submit your questions, tips, or ideas on the site, or access a comprehensive database of previously answered questions.

VB4 32-bit VB5

```
Private Sub PreparePoints()
  Dim Angle As Double
  Dim Slice As Double
  Dim X1 As Long, Y1 As Long
  Dim radius As Long
  Dim theta As Double
  Dim n As Long, i As Long
  '
  ' Some useful constants.
  '
  Const Pi# = 3.14159265358979
  Const DegToRad# = Pi / 180
  '
  ' Calc angle between each point, centerpoint, and
  'radius.
  '
  Slice = 360 / m_Sides
  X1 = m_Width \ 2
  Y1 = m_Height \ 2
  If X1 > Y1 Then
     radius = Y1 - 1
  Else
     radius = X1 - 1
  End If
  '
  ' Calculate endpoints for each vertex
  '
  ReDim m_Pts(0 To m_Sides - 1) As POINTAPI
  Angle = 180# - m_Offset
  For i = 0 To m_Sides - 1
     theta = Angle * DegToRad
     m_Pts(i).X = X1 + (radius * (Sin(theta)))
     m_Pts(i).Y = Y1 + (radius * (Cos(theta)))
     Angle = Angle + Slice
  Next i
End Sub
```

LISTING 1 ***Prepare an Array of Polygon Corner Points.*** *Use this routine to generate an array containing the X, Y coordinates of each vertex in an n-sided polygon. Inputs—module-level variables, in this case—include the height and width (m_Height, m_Width), number of sides (m_Sides), and degrees of offset from a vertical orientation (m_Offset). The centerpoint is assumed to be one-half Width and one-half Height.*

The next step: Select the bitmap back out of the DC and transform it into a standard OLE picture object (see Listing 3). For more details on this procedure, see the Microsoft Knowledge Base article Q161299. Once you have the bitmap stored in a picture object, you can delete the memory DC and assign the control's MaskPicture property.

I've wrapped all this functionality into a class module, which you can download from the free, Registered Level of The Development Exchange (for details, see the Code Online box at the end of the column). Using this class requires few steps. Initialize it as your control initializes:

```
Private Sub UserControl_Initialize()
   Set m_Mask = New CPolygonMask
   m_Mask.Parent = ObjPtr(Me)
End Sub
```

And assign a new MaskPicture as needed, thereafter:

```
Call m_Mask.RenderMask
UserControl.MaskPicture = m_Mask.Mask
```

Use these last two steps only when the size or shape of the polygon changes.

Q OPENING UP SYSTEM WIZARDS

How would I open up the Add Printer wizard in VB? I'd like to offer my users this option directly, not just give them instructions on how to do it the "normal" way.

A Windows 95 and NT offer a variety of system wizards and control panel applets you may find useful to call from your applications. In most cases, calling them is a simple Shell call—simple, that is, after you determine the magic string to call. To fire off this wizard, do something like:

```
Call Shell("rundll32.exe shell32.dll,_
   SHHelpShortcuts_RunDLL AddPrinter", _
   vbNormalFocus)
```

Don't put a space after the first comma in the *pathname* parameter. I've collected 30 or so similar calls that open up most control panel applets and many system wizards, and posted them on my Web page. Feel free to swing by http://www.mvps.org/vb and hit the Tipsheets link to see them all.

See "Spooling, Shelling, and Hooking" (*VBPJ* February 1998) for an explanation of what's returned by the Shell function, and how you can use that to track when the wizard has completed. The Shell32.zip sample on my Web site uses several of these techniques to "shell and wait" in 32-bit VB.

Q CALCULATING LISTINDEX UNDER CURSOR

I want my list box to behave like a menu. As the user moves the cursor over the list, I'd like the selected item to move with the cursor. How would I go about this?

A The key to this problem is that each item in a standard list box is the same height. If you know that height (the index value for the topmost visible item) and the y-coordinate of the cursor, determining the item under the cursor boils down to simple math.

You can use the SendMessage API to request the height, in pixels, of each item in the list. Send the LB_GETITEMHEIGHT message to your list box, passing 0 for both wParam and lParam. The return value is the height of each item.

The ListBox_MouseMove event passes the y-coordinate of the cursor using the same ScaleMode as the form because this property is not directly exposed by the list box. You can convert the value returned by SendMessage to the ScaleMode in effect using the ScaleY method.

When you divide the passed Y value by the converted item height then add the TopIndex, you get the item under the cursor. Finally, ensure that your calculated ListIndex value doesn't equal or exceed the ListCount to avoid errors when setting the new value:

```
Private Sub List1_MouseMove(Button As _
   Integer, Shift As Integer, X As _
   Single, Y As Single)
   Dim ItemHeight As Long
   Dim NewIndex As Long
   With List1
      ItemHeight = SendMessage(.hWnd, _
         LB_GETITEMHEIGHT, 0, ByVal 0&)
      ItemHeight = ScaleY(ItemHeight, _
         vbPixels, vbTwips)
      NewIndex = .TopIndex + (Y \ _
         ItemHeight)
      If NewIndex < .ListCount Then
         .ListIndex = NewIndex
      End If
   End With
End Sub
```

The same technique could be used to

calculate where to position a dragged item when inserting it into a list, what options to present in a context menu, or any number of other situations. A sample project demonstrating this technique is available on the free, Registered Level of The Development Exchange (for details, see the Code Online box at the end of the column).

VB4 32-bit VB5

```
Private Sub CreateMask()
  Dim hWndScn As Long
  Dim hDCScn As Long
  Dim hDC As Long
  Dim hBmp As Long
  ' Bitmap stores mask (monochrome)
  Dim hBmpPrev As Long
  Dim hPenPrev As Long
  Dim hBrushPrev As Long
  Dim rMask As RECT
  '
  ' Get desktop DC, and create compatable DCs.
  '
  hWndScn = GetDesktopWindow()
  hDCScn = GetDC(hWndScn)
  hDC = CreateCompatibleDC(hDCScn)
  Call ReleaseDC(hWndScn, hDCScn)
  '
  ' Create mono BMP
  '
  hBmp = CreateBitmap(m_Width, m_Height, 1, 1, ByVal _
     0&)
  '
  ' Select BMP into DC, storing previous bitmap.
  '
  hBmpPrev = SelectObject(hDC, hBmp)
  '
  ' Flood bitmap with white.
  '
  rMask.Right = m_Width
  rMask.Bottom = m_Height
  Call FillRect(hDC, rMask, GetStockObject(WHITE_BRUSH))
  '
  ' Draw polygon in black
  '
  hPenPrev = SelectObject(hDC, GetStockObject(BLACK_PEN))
  hBrushPrev = SelectObject(hDC, GetStockObject(BLACK_BRUSH))
  Call Polygon(hDC, m_Pts(0), m_Sides)
  Call SelectObject(hDC, hPenPrev)
  Call SelectObject(hDC, hBrushPrev)
  '
  ' Remove the new copy of the bitmap.
  '
  hBmp = SelectObject(hDC, hBmpPrev)
  '
  ' Create a picture object from memory bitmap.
  '
  Call CreateBitmapPicture(hBmp)
  '
  ' Clean up
  '
  Call DeleteDC(hDC)
End Sub
```

LISTING 2 ***Create a MaskPicture on the Fly.*** *This routine uses the polygon points generated in Listing 1 to paint a black polygon on a white background. The routine creates the monochrome bitmap in memory, using an hDC compatible with the screen. After painting, the routine selects the bitmap out of the hDC, and deletes the hDC. Before letting the bitmap handle go out of scope, this routine passes it to the CreateBitmapPicture routine, which transforms it into a picture object.*

Code Online

You can find all the code published in this issue of VBPJ *on The Development Exchange (DevX) at http://www.vbpj.com. For details, please see "Get Extra Code in DevX's Premier Club" in Letters to the Editor.*

Solve These Irregular Problems

Locator+ Codes

Listings for the entire issue, plus a class module for drawing multisided polygons, plus a sample project demonstrating a technique for moving selected list-box items with the cursor (free Registered Level): VBPJ0998

✪ *Listings for this article only, the module and sample project described above, plus a complete skeleton for a polygonal button control (subscriber Premier Level): AP0998*

VB5

```
Private Declare Function OleCreatePictureIndirect Lib _
  "olepro32.dll" (PicDesc As PicBmp, RefIID As GUID, _
  ByVal fPictureOwnsHandle As Long, IPic As IPicture) _
  As Long

Private Type PicBmp
  Size As Long
  Type As Long
  hBmp As Long
  hPal As Long
  Reserved As Long
End Type

Private Type GUID
  Data1 As Long
  Data2 As Integer
  Data3 As Integer
  Data4(7) As Byte
End Type

Private Sub CreateBitmapPicture(ByVal hBmp As Long)
  Dim pic As PicBmp
  Dim IPic As IPicture
  Dim IID_IDispatch As GUID
  '
  ' Fill in with IDispatch Interface ID
  '
  With IID_IDispatch
     .Data1 = &H20400
     .Data4(0) = &HC0
     .Data4(7) = &H46
  End With
  '
  ' Fill PicBmp struct with necessary parts
  '
  With pic
     .Size = Len(pic)            ' Length of structure
     .Type = vbPicTypeBitmap     ' Type of Picture
     .hBmp = hBmp                ' Handle to bitmap
     .hPal = 0                   ' Handle to palette
                                 ' (may be null)
  End With
  '
  ' Clear old instance of picture object.
  '
  Set m_Pic = Nothing
  '
  ' Create Picture object
  '
  OleCreatePictureIndirect pic, IID_IDispatch, 1, m_Pic
End Sub
```

LISTING 3 ***Transform Bitmap Into Picture Object.*** *This routine converts an in-memory bitmap into a standard OLE picture object. The resulting object may be assigned to any VB property, such as MaskPicture, that requires that object type. The handle to the bitmap is no longer yours after this conversion, and must not be deleted!*