Wouter Devinck

Aero Glass met Windows Presentation Foundation

PROFESSIONEEL OGENDE INTERFACES MET WPF EN GLASS

In Windows Vista is het vrij eenvoudig om gebruik te maken van Aero Glass in jouw applicaties, dit kan namelijk door een API aan te roepen. Met Windows Forms en GDI+ is het echter niet zo eenvoudig om tekst en controls netjes weer te geven op het glas. Maar als we de grafische kracht en eenvoud van WPF met het glaseffect combineren, worden geavanceerde menu's zoals onderaan in Windows Media Player 11 en bovenaan in Internet Explorer 7 plots verassend eenvoudig. In dit artikel legt de auteur de belangrijkste mogelijkheden van Aero Glass uit aan de hand van drie voorbeelden.

In Windows Vista worden alle vensters (zelfs de 'command prompt') standaard van een glazen randje voorzien, ten minste, als dit effect ingeschakeld is (dit is niet mogelijk op Vista Home Basic). Behalve het esthetische voordeel heeft het gebruik van het glaseffect ook functionele voordelen. Zo is de rand van de vensters dikker dan in alle voorgaande versies van Windows. Dit zorgt voor een groter gebruiksgemak doordat je die brede rand veel makkelijker kunt 'vastpakken', om de grootte van een venster aan te passen. Maar dankzij het semitransparante effect ziet de rand er niet 'zwaar' uit. Vista beschikt over een API om ontwikkelaars de kans te geven gebruik te maken van het effect in de client area van hun applicaties. Dit is iets dat zelfs in enkele van de met Vista meegeleverde programma's gedaan wordt, denk bijvoorbeeld maar aan Windows Media Player, Internet Explorer en Windows Photo Gallery; zie afbeelding 1. Door het glaseffect in jouw programma's te gebruiken, zien ze er professioneler uit. Verder zorgt het glas er voor dat ze vertrouwder aanvoelen en maakt het ze, mits de juiste aanpak van de ontwerper, een stuk gebruiksvriendelijker. En dan vergeet ik nog te zeggen dat het er gewoon mooi uitziet.

Visual Studio 2008

Voor het maken van de voorbeelden bij dit artikel heb ik gebruikgemaakt van Microsoft Visual Studio 2008 bèta 2 en Microsoft Expression Blend 2 May Preview. Ik kan iedereen aanraden om de bèta van Visual Studio 2008 te gebruiken, maar voor wie dit (om welke reden dan ook) niet wil, zijn alle voorbeelden ook beschikbaar in het formaat voor Visual Studio 2005. De xaml- en Visual Basic-code zijn voor beide versies identiek, maar in Visual Studio 2005 zit geen designer voor xaml-code ingebouwd.

Afbeelding 1. Glass in IE en WMP

De praktijk

Hoe krijg je nu precies dat glaseffect in een deel van de client area van jouw WPF Window? Ten eerste is het belangrijk om te controleren of het programma wel degelijk op Vista uitgevoerd wordt en, als dat het geval is, of Aero Glass aanstaat. Het eerste is eenvoudig te controleren in .NET-code, maar voor het tweede moeten we de hulp van een API inroepen, 'DwmApi.dll', waarin 'Dwm' een verwijzing is naar de 'Desktop Windows Manager' van Vista. In deze API vinden we een functie, 'DwmIsCompositionEnabled', die deze controle uitvoert. Een goede plaats om deze controles uit te voeren, is in de sub New van het desbetreffende Window, zie codevoorbeeld 1. Merk op dat ik er in deze voorbeelden van uit ga dat de applicatie enkel gemaakt is voor Vista met Aero Glass. In een productieomgeving moet er in de meeste gevallen natuurlijk een alternatieve interface aanwezig zijn, ten minste voor Vista zonder Aero Glass en vaak ook voor oudere besturingssystemen.

```
Public GlassIsEnabled As Boolean
Private Declare Function DwmIsCompositionEnabled Lib "dwmapi.dll" (ByRef
 en As Boolean) As Integer
Public Sub New()
  MyBase.New()
  Me.InitializeComponent()
  If Environment.OSVersion.Version.Major < 6 Then
    MessageBox.Show("This application cannot run on versions " & _
      "of Windows prior to Vista.")
    End
  Else
    DwmIsCompositionEnabled(GlassIsEnabled)
    If Not GlassIsEnabled Then
      MessageBox.Show("This application cannot run on Windows Vista " & _
   "Home Basic or on any other version of Vista with Aero Glass disabled.")
      End
    End If
  End If
End Sub
```

Codevoorbeeld 1. Controle

Na deze controle kunnen we het glas uitbreiden. Hiertoe declareren we een tweede functie uit de API met een wel erg toepasselijke naam: 'DwmExtendFrameIntoClientArea'. Aan deze functie moeten we de handle van het venster en de gewenste marges van het glasefect doorgeven. Daarna moeten we het venster transparant maken. Dit zijn drie dingen die moeten gebeuren, waarbij we ook drie problemen moeten oplossen. Ten eerste kunnen we in WPF de handle van het venster niet rechtstreeks verkrijgen zoals in Windows Forms. Ten tweede werkt WPF niet met pixels zoals Windows (en dus ook de API) dat doet, maar maakt het gebruik van 'Device Independent Pixels' (vanaf hier DIP's genoemd). Als we willen dat het glas ook correct geschaald wordt, moeten we de afmetingen ervan omzetten van DIP's naar pixels. De API verwacht namelijk een waarde in pixels. Ten derde is het in WPF niet voldoende om de achtergrond van het window transparant te maken, omdat de WPF-inhoud in feite gehost wordt in een Win32-venster dat niet transparant is.
Om het eerste probleem aan te pakken, importeren we 'System.Windows.Interop' en gebruiken we een 'WindowInteropHelper' om de benodigde handle te verkrijgen. Hierbij moeten we er wel rekening mee houden dat die handle pas beschikbaar is op het moment dat 'SourceInitialized' afgevuurd wordt.
Ook het tweede probleem is eenvoudig op te lossen, maar des te moeilijker te begrijpen. Per definitie is één DIP in WPF gelijk aan één zesennegentigste van een inch, 96 DIP's vormen dus 1 inch op elke monitor (op voorwaarde dat de fysieke DPI-waarde [dots per inch] van de monitor gelijk is aan de ingestelde DPI-waarde in Windows). Als we waarden van DIP's naar gewone pixels willen omrekenen, komt het er dus op neer om de waarde in DIP's te vermenigvuldigen met de in Windows ingestelde DPI-waarde gedeeld door 96.
Om het derde en laatste probleem op te lossen, maken we gebruik van de klasse 'HwndSource' die eveneens beschikbaar is in 'System.Windows.Interop'. Hiermee kunnen we de achtergrond zowel vanuit het standpunt van GDI als van WPF transparant maken. Bijkomend probleem is dat het volledige window transparant wordt en niet alleen het gebied waar we het glaseffect gebruiken. Dit probleem is echter eenvoudig op te lossen in de xaml-code, zie codevoorbeeld 2. In codevoorbeeld 3 zie je de code die samen met de xaml-code van codevoorbeeld 2 het belangrijkste stuk van ons eerste voorbeeld vormt. De drie voorbeelden bij dit artikel zijn beschikbaar op: www.microsoft.nl/netmagazine20

Nu we een window hebben met onderaan 100 DIP's extra glas, kunnen we aan de slag met WPF om op dit glas te tekenen. In het eerste voorbeeld heb ik enkele geanimeerde, semitransparante ellipsen getekend en een label; zie afbeelding 2. Het vraagt uiteraard wat meer inspanning om een menu zoals in Windows Media Player te maken, maar het is heel goed te realiseren.

```
<Window
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   x:Class="winOne" Title="Voorbeeld 1" Height="300" Width="500"
   WindowStartupLocation="CenterScreen" ResizeMode="CanMinimize">
   <Grid>
      <Grid.RowDefinitions>
         <RowDefinition Height="*" />
         <RowDefinition Height="100" />
      </Grid.RowDefinitions>
      <Grid Grid.Row="0" Margin="0" Name="ClientArea"
         Background="White"/><!-- Background hier instellen -->
      <Grid Grid.Row="1" x:Name="GlassArea" >
      </Grid>
   </Grid>
</Window>
```

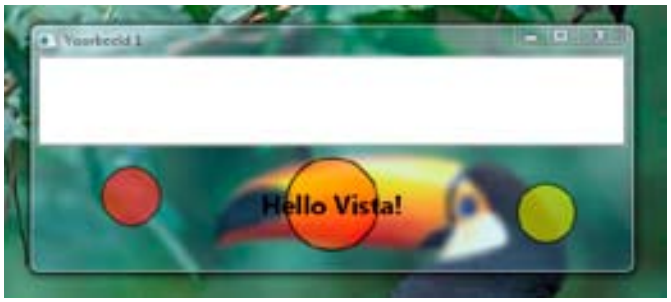
Codevoorbeeld 2. De xaml-code van Vb1

```
Imports System.Windows.Interop
Partial Public Class winOne

   Private Structure MARGINS
      Public Left As Integer
      Public Right As Integer
      Public Top As Integer
      Public Bottom As Integer
   End Structure

   Private _margins As MARGINS
   Public GlassIsEnabled As Boolean
   Dim DesktopDpi As Single

   Private Declare Function DwmIsCompositionEnabled Lib "dwmapi.dll" _
      (ByRef en As Boolean) As Integer
   Private Declare Function DwmExtendFrameIntoClientArea Lib _
"dwmapi.dll" ( ByVal hWnd As IntPtr, ByRef margins As MARGINS) As Integer

   Public Sub New()
      'codevoorbeeld 1
   End Sub

   Private Sub winOne_SourceInitialized(ByVal sender As Object,
       ByVal e As System.EventArgs) Handles Me.SourceInitialized
      SetGlassRegion(0, 0, 100, 0)
   End Sub

   Public Sub SetGlassRegion(ByVal top As Integer, ByVal left As Integer,_
   ByVal bottom As Integer, ByVal right As Integer)
      If GlassIsEnabled Then
         'Probeer de handle te verkrijgen
         Dim hwnd As IntPtr = New WindowInteropHelper(Me).Handle
         If hwnd = IntPtr.Zero Then Throw New InvalidOperationException( _
            "The Window must be shown before extending glass.")
         'Maak de achtergrond voor zowel WPF als Win32 transparant
         HwndSource.FromHwnd(hwnd).CompositionTarget.BackgroundColor =
            Colors.Transparent 'win32
         Me.Background = Brushes.Transparent 'wpf
         'Stel de margins in, maar converteer WPF DIPs (device
         'independent pixels) naar pixels
         Dim desktop As System.Drawing.Graphics = System.Drawing.Grap
hics.FromHwnd(hwnd)
         DesktopDpi = desktop.DpiX
         _margins = New MARGINS()
         _margins.Top = CInt(top * (DesktopDpi / 96))
         _margins.Left = CInt(left * (DesktopDpi / 96))
         _margins.Bottom = CInt(bottom * (DesktopDpi / 96))
         _margins.Right = CInt(right * (DesktopDpi / 96))
         'Geef alles door aan de API
         DwmExtendFrameIntoClientArea(hwnd, _margins)
      End If
   End Sub
End Class
```

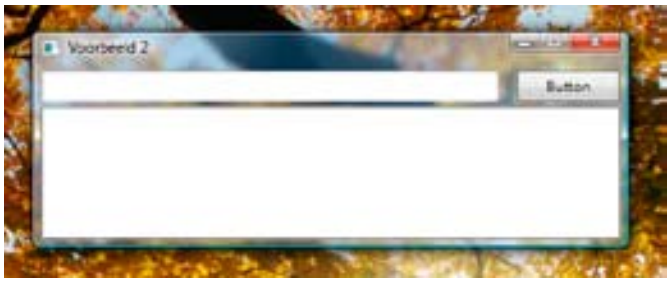
Codevoorbeeld 3. De Visual Basic-code van Vb1

Venster verplaatsen

In het tweede voorbeeld breiden we het glaseffect uit aan de bovenkant van het venster en plaatsen we er een tekstvak en een knop op; afbeelding 3. Standaard kunnen we het venster niet verplaatsen (door de client area te slepen) zoals we dat wel met de titelbalk kunnen. Daardoor kunnen we het venster ook niet verplaatsen door het strookje extra glas onder de titelbalk te slepen. Als we dat wel wensen, zoals in Internet Explorer, komt er behoorlijk wat programmeerwerk aan te pas.
Er bestaan verschillende mannieren om dit aan te pakken. Ik



Afbeelding 2. Voorbeeld 1 in actie



Afbeelding 3. Voorbeeld 2 in actie

```
Public Sub SetGlassRegion(…)
  If GlassIsEnabled Then
    '… zie codevoorbeeld 3 …
    HwndSource.FromHwnd(hwnd).AddHook(New HwndSourceHook( _
      AddressOf WndProc))
  End If
End Sub

Private Const HTCAPTION As Integer = 2
Private Const WM_NCHITTEST As Integer = &H84

Function WndProc(ByVal hwnd As IntPtr, ByVal msg As Integer, _
          ByVal wParam As IntPtr, ByVal lParam As IntPtr, _
          ByRef handled As Boolean) As IntPtr
  If msg = WM_NCHITTEST AndAlso IsOnGlass(lParam) Then 'beweegt de muis
    'over het extra glas ...
    handled = True
    Return New IntPtr(HTCAPTION) '... lieg dan tegen windows, zeg dat
    'dit glas de titelbalk is
  End If
End Function
```

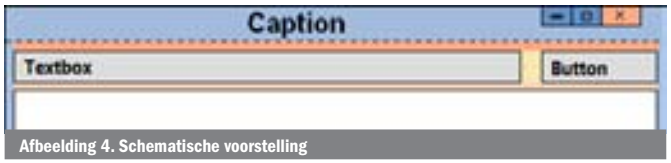
Codevoorbeeld 4. Liegen tegen Windows

kies hier niet voor de eenvoudigste methode, maar wel voor de methode met de beste resultaten. Het komt er op neer dat we moeten liegen tegen Windows. Als de cursor zich op ons stukje glas bevindt, vertellen we Windows dat de cursor op de titelbalk staat. Zo moeten we ons van het eigenlijke verplaatsen niets aantrekken en volgt het venster ook als we de muis zo snel wegslepen dat events zoals 'MouseMove' al niet meer reageren.

Liegen tegen Windows

We kunnen met 'WndProc' luisteren naar Windows Messages en ook zelf berichten naar het besturingssysteem versturen. In WPF kunnen we het beste met 'HwndSource' een 'HwndSourceHook' toevoegen om de berichten aan de de sub 'WndProc' te adresseren, zie codevoorbeeld 4. Eenmaal in 'WndProc' aangekomen, filteren we de berichten. Als het om een muisbeweging gaat (verplaatsen of klikken), vangen we het bericht 'WM_NCHITTEST' op. Als dat het geval is, controleren we of de muisaanwijzer op het glas staat, en, indien ja, retourneren we het bericht 'HTCAPTION'. Let op het gebruik van 'AndAlso'. Deze operator die nieuw is sinds Visual Basic 2005, zorgt dat de tweede voorwaarde pas geëvalueerd wordt als de eerste waar is.

Maar hoe bepalen we nu of de muis op het extra glas staat? Wel, de coördinaten van de muisaanwijzer kunnen we uit één van de parameters van WndProc berekenen. Met 'PointFromScreen' kunnen we deze coördinaten ten opzichte van het venster bekijken, in plaats van ten opzichte van het volledige scherm (vergelijkbaar met 'PointToClient' in Windows Forms). Let op: hierdoor moeten we de coördinaten niet manueel omzetten naar DIP's. Om te bepalen of de coördinaten zich op het glas bevinden, moeten we het gebied eerst met behulp van GDI+ onderverdelen in enkele rechthoeken. We willen namelijk niet dat de controls op het glas onbruikbaar worden door ze als titelbalk te beschouwen (in het voorbeeld: een tekstvak en een knop). Zie het schema in afbeelding 4, waarop de drie rechthoeken met rood, geel en oranje weergegeven worden. Deze verdeling in rechthoeken is natuurlijk in elk project verschillend. Codevoorbeeld 5 toont de functie IsOnGlass.



Afbeelding 4. Schematische voorstelling

Volledig glas

Soms kan het ook leuk zijn om het glaseffect op een volledig venster toe te passen. In het derde en laatste voorbeeld nemen we een oud WPF-project en met een minimum aan code maken we de achtergrond wat mooier; zie afbeelding 5. Het is voldoende om één van de marges op -1 in te stellen om dit effect te bereiken.

Afbeelding 5. Voorbeeld 3

```
Private Function IsOnGlass(ByVal l As Integer) As Boolean
   'De coördinaten uit de parameter halen
   Dim x As Integer = (l << 16) >> 16
   Dim y As Integer = l >> 16
   'Met de coördinaten een punt maken en dat punt ten opzichte van
   'de client bekijken ipv ten opzichte van het volledige scherm
   Dim p As New System.Windows.Point(x, y)
   p = Me.PointFromScreen(p)
   '
   Dim wi As Double = GlassArea.ActualWidth
   Dim he As Double = GlassArea.ActualHeight
   'Maak zoveel rechthoeken aan als nodig, zorg dat alle gebieden
   'waar je glas kan slepen in een rechthoek zitten, maar de controls
   'op het glas niet. Zie schema (afbeelding 4)
   Dim r(2) As System.Drawing.Rectangle
   r(0) = New System.Drawing.Rectangle(0, 0, wi, ((he - 25) / 2))
   r(1) = New System.Drawing.Rectangle(TextBox1.ActualWidth, ((he - 25) _
   / 2), (wi - TextBox1.ActualWidth - Button1.ActualWidth), 25)
   r(2) = New System.Drawing.Rectangle(0, (he - ((he - 25) / 2)), wi, _
   ((he - 25) / 2))
   'Test alle rechthoeken
   Dim flag As Boolean
   For i As Integer = 0 To 2
      flag = flag Or r(i).Contains(New System.Drawing.Point(p.X, p.Y))
   Next
   Return flag
End Function
```

Codevoorbeeld 5. Functie IsOnGlass()

Tot besluit

Om het typische glaseffect van Windows Vista in de client area van jouw eigen software te gebruiken, is helemaal niet zo moeilijk als het lijkt. De voorbeelden bij dit artikel zijn eveneens beschikbaar in de vorm van een webcast op Microsoft Chopsticks. Chopsticks is een recent project van Microsoft België, bestaande uit twee websites waarop korte on-demand webcasts worden aangeboden. De video bij dit artikel is beschikbaar op: http://www.microsoft.com/belux/msdn/nl/chopsticks/default.aspx?id=169.

Wouter Devinck is een 17-jarige student die graag op de hoogte blijft van de nieuwste technologieën en een zwak heeft voor het maken van oogverblindende interfaces. Met vragen en opmerkingen kun je terecht op info@wouterdevinck.net of zie www.wouterdevinck.net.

Referenties

http://msdn2.microsoft.com/en-us/default.aspx
http://www.danielmoth.com/blog/
http://channel9.msdn.com/