Course 3D_MDX: 3D-Graphics with Managed DirectX 9.0
Chapter C7: The Graphics Card Project

Projekt graphics_card1

Diese Übung ist eine kurze, übersichtliche Zusammenfassung von Kapitel 2 aus: MILLER, Tom: Managed DirectX 9 Graphics and Game Programming. SAMS 2003, ISBN: 0-672-32596-9. US$ 34.99.
Die Übung zeigt die verfügbare Information über Ihre Grapikkarte und deren DirectX-Treiber an. Man braucht diese Information zur Programmierung von Computerspielen, wo es darauf ankommt, möglichst alle Fähigkeiten der Graphikkarte zu nutzen.

Main Menu nach dem Start von VS 2005: `File` → `New Project...` → `Templates: Windows Application` `Name: graphics_card1` → `Location: C:\temp` → `Create directory for solution`: ausschalten → `OK`
Löschen Sie die Files `Program.cs` und `Form1.Designer.cs` und den Inhalt von `Form1.cs`, wie es in den Kapiteln 2DCisC1 bis 2DCisC4 beschrieben wurde.

Falls das `Solution Explorer` - Fenster nicht schon offen ist, öffnen Sie es über das Hauptmenü: `View` → `Solution Explorer`.
Im `Solution Explorer` - Fenster klicken Sie auf das Pluszeichen vor `graphics_card1`. Es öffnet sich ein Baum. Ein Ast heißt "`References`". Klicken Sie mit der **rechten** Maustaste auf `References` und dann mit der **linken** Maustaste auf `Add Reference...`. Es öffnet sich eine `Add Reference` Dialog Box. Scrollen Sie abwärts, bis Sie den Component Name: `Microsoft.DirectX3D Version 1.0.2902.0` sehen.
Markieren Sie durch Linksklick diese Referenz. Verlassen Sie die `Add Reference` Dialog Box mit `OK`.
Kontrollieren Sie, ob jetzt im Solution Explorer Fenster unter `graphics_card1` → `References` (unter anderen) die Referenz `Microsoft.DirectX.Direct3D` steht.

Form1

Schreiben in das leere Codefenster `Form1.cs` folgenden Code:

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Text;
using Microsoft.DirectX.Direct3D;

public class Form1 : Form
{ static void Main() { Application.Run( new Form1() ); }
  public Form1()
  { string s1 = "", s2= "";
    StringBuilder s = new StringBuilder();
    TextBox TB = new TextBox();
    TB.Size = ClientSize = new Size( 400, 800 );
    TB.Font = new System.Drawing.Font( "Courier New", 8 );
    TB.Multiline  = true;
    TB.ScrollBars = ScrollBars.Both;
    Controls.Add( TB );
    if ( Manager.Adapters.Count > 1 )
      s.Append( "You have two graphics cards or one dual monitor card.\r\n" +
                "The infos are about your primary card.\r\n\r\n" );
    AdapterInformation ai = Manager.Adapters[0];
    s.Append( "Graphic Board: " + ai.Information.Description   + "\r\n" );
    s.Append( "Driver       : " + ai.Information.DriverName    + "\r\n" );
    s.Append( "Driver Vers. : " + ai.Information.DriverVersion + "\r\n\r\n" );
    s.Append( "current display mode:\r\n" );
    s.Append( ai.CurrentDisplayMode.Width .ToString() + "\tx " );
    s.Append( ai.CurrentDisplayMode.Height.ToString() + "\t" );
    s.Append( ai.CurrentDisplayMode.Format.ToString() + "\r\n\r\n" );
    s.Append( s1 + "possible display modes:\r\n" );
    foreach ( DisplayMode dm in ai.SupportedDisplayModes)
    { s2 = string.Format( "{0}\tx {1}\t{2}\r\n", dm.Width, dm.Height, dm.Format );
      if ( s2 != s1 ) { s.Append( s2 ); s1 = s2; }
    }
    s.Append( "\r\n\r\nCapabilities of this graphics card:\r\n\r\n" );
    Caps caps = Manager.GetDeviceCaps( 0, DeviceType.Hardware );
    s1 = caps.DeviceCaps.ToString();
    s1 = s1.Replace("Caps: ", "Caps:\r\n" );// Insert carriage return and line feed
    s1 = s1.Replace( "\n", "\r\n");          //Insert carriage return
    s.Append( s1 );
    TB.Text = s.ToString();
  }
}
```

Klicken Sie `Debug` → `Start Without Debugging Ctrl F5`.

Weitere Information

1. Multimon capability: New graphic cards all have dual head support, which allow two monitors to be attached to a single graphic card. Direct3D treats both hookups (regardless if DVI or VGA) as two adapters.
2. "`Manager`" is a static class in the Direct3D assemblies to enumerate adapters, driver info and to retrieve its capabilities.
3. `AdapterInformation` is a structure of structures:

```
public struct AdapterInformation
{ int                   Adapter;                //No. of an adapter in the system
  AdapterDetails        Information;            //structure cont. much adapter/driver info
  AdapterDetails        GetWhqlInformation();   //additional Windows Hardware Quality Labs info
  DisplayMode           CurrentDisplayMode;     //struct. cont. Width, Height, Format, Refresh Rate
  DisplayModeEnumerator SupportedDisplayModes;  //collection of DiplayMode structures
}
```

4. The `AdapterInformation.DisplayMode.Format` structure follows this pattern: A letter represents the type of data and the number denotes the number of bits. letters: R=red, G=green, B=blue, X=unused, A=alpha, L=luminance, P=palette. Examples: `X8R8G8B8` denotes a 32 bit color format, `R5G6B5` denotes a 16 bit color format.
5. While the list of formats is quite large, there are only a few that are valid and can be used as a Direct3D display or back buffer format. Valid Direct3D formats are: `X8R8G8B8`, `R5G6B5`, `X1R5G5B5`, `A2R10G10B10`. DirectX3D is not possible when the adapter does not support one of these.
6. You could use the Manager class to find out, whether or not your graphics card supports a particular format. Example: bool b = `Manager.CheckDeviceFormatConversion(0, DeviceType.Hardware, Format.X8R8G8B8, Format.A8R8G8B8, true);` (The 3. and 4. parameter refer to the front buffer and the back buffer. The 5. denotes WindowedMode=`true` or FullScreenMode=`false`)
7. The function `Manager.GetDeviceCaps` generates a `Cap` structure containing every possible capability a graphic card and its driver can have. These hundreds of capabilities are broken down mainly into Boolean values (i.e. feature supported or not) and integer values (i.e. max. no. of features).
8. see: **http://csharp-home.com/index/tiki-read_article.php?articleId=105&page=2**