Course 3D_MDX: 3D- Graphics with Managed DirectX 9.0
Chapter C1: Moving Triangles

Copyright © by V. Miszalok, last update: 26-04-2007

Projekt triangle1

Diese Übungsaufgabe ist eine kurze, übersichtliche Zusammenfassung von drei Direct3D-Tutorials von Microsoft: `Tutorial1`, `Tutorial2` und `Tutorial3`. Sie finden diese unter `C:\DXSDK\Samples\Managed\Direct3D\Tutorials`.

Main Menu nach dem Start von VS 2005: `File → New Project... → Templates: Windows Application`
`Name: triangle1 → 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 `triangle1`. 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.DirectX Version 1.0.2902.0` sehen.
Markieren Sie durch Linksklick diese Referenz und (bei gedrückter der Strg-Taste) die weiter unten stehende Referenz `Microsoft.DirectX.Direct3D Version 1.0.2902.0`. Verlassen Sie die `Add Reference` Dialog Box mit `OK`.
Kontrollieren Sie, ob jetzt im Solution Explorer Fenster unter `triangle1 → References` (unter anderen) die beiden Referenzen `Microsoft.DirectX` und `Microsoft.DirectX.Direct3D` stehen.

Wenn Sie nicht Visual C# Express sondern Visual Studio 2005 Professional benutzen sollten Sie die nerventötende Formatier- und Einrückautomatik des Code-Editors ausschalten bevor Sie den unten vorgegebenen Code durch kopieren nach `Form1.cs` transferieren:
1. Hauptmenu von Visual Studio 2005 Professional: Klick auf Menupunkt "`Tools`".
2. Es erscheint ein DropDown-Menu. Klick auf "`Options...`".
3. Es erscheint eine `Options` Dialog Box.
4. Klick auf den Ast "`Projects and Solutions`". Klick auf "`General`". Alle drei Pfade auf `C:\temp` stellen.
5. Klick auf den Ast "`Text Editor`", dann auf "`C#`".
6. Es erscheint ein Unterbaum mit den Ästen "`General, Tabs, Advanced, Formatting, IntelliSense`".
7. Klick auf "`Tabs`". Stellen Sie "`Indenting`" auf `None`, "`Tab size`" und "`Indent size`" auf `1` und schalten Sie die Option "`Insert spaces`" ein.
8. Klicken sie im Unterbaum "`C#`" auf das Pluszeichen vor "`Formatting`" und ändern Sie alle "`Formatting`"-Äste:
"`General`": alle CheckBoxes ausschalten, "`Indentation`": alle CheckBoxes ausschalten, "`New Lines`": alle CheckBoxes ausschalten, "`Spacing`": alle CheckBoxes ausschalten, "`Wrapping`": beide CheckBoxes **ein**schalten.
9. Verlassen Sie den Dialog mit Button "`OK`".

Form1, OnResize, OnTimer

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

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

public class Form1 : Form
{ static void Main() { Application.Run( new Form1() ); }
  static Device device = null;
  static float fAngle;
  static CustomVertex.PositionColored[] v = new CustomVertex.PositionColored[3];
  Timer myTimer = new Timer();
  public Form1()
  { Text = "D3DTriangleAnimation";
    //fill coordinates and colors into an array "v"
    v[0].X=-1f; v[0].Y=-1f; v[0].Z=0f;
    v[1].X= 1f; v[1].Y=-1f; v[1].Z=0f;
    v[2].X= 0f; v[2].Y= 1f; v[2].Z=0f;
    v[0].Color = System.Drawing.Color.DarkGoldenrod.ToArgb();
    v[1].Color = System.Drawing.Color.MediumOrchid.ToArgb();
    v[2].Color = System.Drawing.Color.Cornsilk.ToArgb();
    myTimer.Tick += new EventHandler( OnTimer );
    myTimer.Interval = 1;
    ClientSize = new Size( 400, 300 ); //Calls OnResize( ... )
  }

  protected override void OnResize( System.EventArgs e )
  //Whenever the window changes we have to initialize Direct3D from scratch
  { myTimer.Stop();// stop the timer during initialization
    try
    { //get information from the operating system about its current graphics properties
      PresentParameters presentParams = new PresentParameters();
      //we have to set two extra flags
      presentParams.Windowed = true;                 //no full screen display
      presentParams.SwapEffect = SwapEffect.Discard; //no swap buffer
      //create a new D3D-device that serves as canvas
      if ( device != null ) device.Dispose(); //free the old canvas if any
      device = new Device( 0, DeviceType.Hardware, this,
                           CreateFlags.SoftwareVertexProcessing, presentParams );
      //set up the transformation of world coordinates into camera or view space
      device.Transform.View = Matrix.LookAtLH(
        new Vector3( 0f, 0f,-4f ),   //eye point 4.0 in front of the canvas
        new Vector3( 0f, 0f, 0f ),   //camera looks at point 0,0,0
        new Vector3( 0f, 1f, 0f ) ); //world's up direction is the y-axis
      //set up the projection transformation using 4 parameters:
      //1.: field of view = 45 degrees; 2.: aspect ratio = heigth / width = 1 = square window;
      //3.: near clipping distance = 0; 4.: far clipping distance = 10;
      device.Transform.Projection = Matrix.PerspectiveFovLH((float)Math.PI/4, 1f, 0f, 10f );
      //Turn off culling, so the user sees the front and back of the triangle
      device.RenderState.CullMode = Cull.None;
      //Turn off lighting, since the triangle provides its own colors
      device.RenderState.Lighting = false;
      //set up the property that fills the triangle with colors
      device.VertexFormat = CustomVertex.PositionColored.Format;
      myTimer.Start();//start the timer again
    }
    catch (DirectXException) { MessageBox.Show( "Could not initialize Direct3D." ); return; }
  }

  protected static void OnTimer( Object myObject, EventArgs myEventArgs )
  { if (device == null) return;
    //throw the old image away
    device.Clear( ClearFlags.Target, Color.Blue, 1f, 0 );
    //rotate with an angular velocity = 0.1
    fAngle += 0.1f;
    device.Transform.World = Matrix.RotationY( fAngle );
    //draw on the canvas
    device.BeginScene();
      device.DrawUserPrimitives( PrimitiveType.TriangleList, 1, v );
    device.EndScene();
    device.Present(); // show the canvas
  }
}
```

Klicken Sie `Debug` → `Start Without Debugging Ctrl F5`. Erproben Sie das Programm durch ziehen an allen Fensterrändern.

VertexBuffer der Graphikkarte benutzen

`triangle1` liest bisher bei jedem Timer Event das Dreieck in Form der Datenstruktur `CustomVertex.PositionColored[3]` aus dem Hauptspeicher über den AGP-Bus in die Graphikkarte ein. Das mag bei einem Dreieck noch angehen, aber bei komplexeren Polygonen wäre dieser dauernde Kopiervorgang über den AGP-Bus eine schwere Bremse. Es ist viel effektiver, die Polygone dauerhaft in einem `VertexBuffer`-Speicher der Graphikkarte abzulegen. So gut wie alle modernen Graphikkarten besitzen so einen dynamisch konfigurierbaren Speicher.
Schreiben Sie in den Kopf von `public class Form1 : Form` unter die Zeile `static float fAngle;` folgende Deklaration:

```
VertexBuffer vertexBuffer;
```

Schreiben Sie in die Funktion `protected override void OnResize( System.EventArgs e )` unterhalb von `device.VertexFormat = CustomVertex.PositionColored.Format;`, aber noch vor die Zeile `myTimer.Start();`

```
if ( vertexBuffer != null ) vertexBuffer.Dispose();//Free the old vertexBuffer if any.
//Create a new vertex buffer on the graphic card and connect it to the device.
vertexBuffer = new VertexBuffer( typeof(CustomVertex.PositionColored), 3,
                                 device, Usage.WriteOnly, CustomVertex.PositionColored.Format,
                                 Pool.Default );
vertexBuffer.SetData( v, 0, LockFlags.None );//Copy vertices from main memory to graphics card.
device.SetStreamSource( 0, vertexBuffer, 0 );//Use vertexBuffer on the graphics card.
```

Verändern Sie in `protected static void OnTimer( Object myObject, EventArgs myEventArgs )` den Befehl `device.DrawUserPrimitives(PrimitiveType.TriangleList, 1, v);` in

```
device.DrawPrimitives( PrimitiveType.TriangleList, 0, 1 );
```

Klicken Sie `Debug` → `Start Without Debugging Ctrl F5`. Erproben Sie das Programm. Sie werden kaum ein Änderung feststellen, der Performancegewinn beträgt nur ca. 10%. Erst bei Polygonen mit mehr als 10 Vertices würde der Gewinn dramatisch deutlich.

Weitere Dreiecke

Verändern Sie in `protected static void OnTimer( Object myObject, EventArgs myEventArgs )` die `device.BeginScene(); - device.EndScene();` - Klammer folgendermaßen:

```
device.BeginScene();
  device.DrawPrimitives( PrimitiveType.TriangleList, 0, 1 );
  device.Transform.World = Matrix.Scaling( 0.5f, 0.5f, 0.5f )
                         * Matrix.Translation( 1f, 0f, 0f );
  device.DrawPrimitives( PrimitiveType.TriangleList, 0, 1 );
  device.Transform.World = Matrix.Scaling( 0.5f, 0.5f, 0.5f )
                         * Matrix.Translation( -1f, 0f, 0f );
  device.DrawPrimitives( PrimitiveType.TriangleList, 0, 1 );
device.EndScene();
device.Present(); //show the canvas
```

Klicken Sie `Debug` → `Start Without Debugging Ctrl F5`. Erproben Sie das Programm. Sie werden zwei neue kleine Dreiecke vorfinden.

Weitere Bewegungen

Verändern Sie in `protected static void OnTimer( Object myObject, EventArgs myEventArgs )` die `device.BeginScene(); - device.EndScene();` - Klammer folgendermaßen:

```
device.BeginScene();
  device.DrawPrimitives( PrimitiveType.TriangleList, 0, 1 );
  device.Transform.World = Matrix.Scaling( 0.5f, 0.5f, 0.5f )
                         * Matrix.Translation( 1f, 0f, 0f )
                         * Matrix.RotationYawPitchRoll( fAngle, fAngle, fAngle );
  device.DrawPrimitives( PrimitiveType.TriangleList, 0, 1 );
  device.Transform.World = Matrix.Scaling( 0.5f, 0.5f, 0.5f )
                         * Matrix.Translation(-1f, 0f, 0f )
                         * Matrix.RotationYawPitchRoll( fAngle, fAngle, fAngle );
  device.DrawPrimitives( PrimitiveType.TriangleList, 0, 1 );
device.EndScene();
device.Present(); //show the canvas
```

Klicken Sie `Debug → Start Without Debugging Ctrl F5`. Erproben Sie das Programm. Die beiden kleinen Dreiecke tanzen um das erste Dreieck herum.

Viele Dreiecke

Ergänzen Sie im Kopf von `public class Form1 : Form` folgende Deklarationen unterhalb der Zeile: `Timer myTimer = new Timer();`

```
const Int32 nTriangles = 100;
static float[] dx = new float[nTriangles];
static float[] dy = new float[nTriangles];
static float[] dz = new float[nTriangles];
static float[] ax = new float[nTriangles];
static float[] ay = new float[nTriangles];
static float[] az = new float[nTriangles];
static  Random r = new Random();
```

Ergänzen Sie am Ende des Konstruktors `public Form1()` folgende Initialisierungen unterhalb der Zeile: `myTimer.Interval = 1;`

```
for ( int i = 0; i < nTriangles; i++ )
{ dx[i] = (float)r.NextDouble();  //random permanent translation dx
  dy[i] = (float)r.NextDouble();  //random permanent translation dy
  dz[i] = (float)r.NextDouble();  //random permanent translation dz
}
```

Verändern Sie in `protected static void OnTimer( Object myObject, EventArgs myEventArgs )` die `device.BeginScene(); - device.EndScene();` - Klammer folgendermaßen:

```
device.BeginScene();
  for ( int i = 0; i < nTriangles; i++ )
  { device.Transform.World = Matrix.Scaling( 0.5f, 0.5f, 0.5f )
                           * Matrix.Translation( dx[i], dy[i], dz[i] )
                           * Matrix.RotationYawPitchRoll( fAngle, fAngle, fAngle );
    device.DrawPrimitives( PrimitiveType.TriangleList, 0, 1 );
  }
device.EndScene();
device.Present(); //show the canvas
```

Klicken Sie `Debug → Start Without Debugging Ctrl F5`. Erproben Sie das Programm. Hundert kleine Dreiecke tanzen exakt gemeinsam wie ein Fischschwarm.

Chaos

Ergänzen Sie am Ende des Konstruktors `public Form1()` die Initialisierungen der Drehwinkel `ax[i], ay[i], az[i]` innerhalb der Schleife: `for ( int i = 0; i < nTriangles; i++ )`, indem Sie die ganze Schleife neu schreiben:

```
for ( int i = 0; i < nTriangles; i++ )
{ dx[i] = (float)r.NextDouble(); //random permanent translation dx
  dy[i] = (float)r.NextDouble(); //random permanent translation dy
  dz[i] = (float)r.NextDouble(); //random permanent translation dz
  ax[i] = (float)r.NextDouble(); //random initial pitch rotation angle
  ay[i] = (float)r.NextDouble(); //random initial yaw   rotation angle
  az[i] = (float)r.NextDouble(); //random initial roll  rotation angle
}
```

Verändern Sie in `protected static void OnTimer( Object myObject, EventArgs myEventArgs )` die `device.BeginScene(); - device.EndScene();` - Klammer folgendermaßen:

```
device.BeginScene();
  for ( int i = 0; i < nTriangles; i++ )
  { device.Transform.World = Matrix.Scaling( 0.5f, 0.5f, 0.5f )
                           * Matrix.Translation( dx[i], dy[i], dz[i] )
                           * Matrix.RotationYawPitchRoll( ay[i] += 0.01f * (float)r.NextDouble(),
                                                          ax[i] += 0.01f * (float)r.NextDouble(),
                                                          az[i] += 0.01f * (float)r.NextDouble() );
    device.DrawPrimitives( PrimitiveType.TriangleList, 0, 1 );
  }
device.EndScene();
device.Present(); //show the canvas
```

Klicken Sie `Debug` → `Start Without Debugging Ctrl F5`. Erproben Sie das Programm. Hundert kleine Dreiecke tanzen wie fliegende Papierblätter im Wind.

Weitere Aufgaben

1. Bremsen Sie die Animation durch hoch setzen von `myTimer.Interval = 100;` im Konstruktor.
2. Beschleunigen Sie die Animation durch hoch setzen von `fAngle += 0.5f;` in `OnTimer(..)`.
3. Drehen Sie das Dreieck um die X-Achse bzw. um die Z-Achse mit
`device.Transform.World = Matrix.RotationX( fAngle );` bzw.
`device.Transform.World = Matrix.RotationZ( fAngle );` in `OnTimer(..)`.
4. Ändern Sie die Farben an den Ecken `v[0]`, `v[1]` und `v[2]` im Konstruktor.
5. Treten Sie mit dem Augenpunkt weiter zurück von `( 0f, 0f,-4f )` nach `( 0f, 0f,-100f )` und näher heran mit `( 0f, 0f,-2f )`.
6. Kommentieren Sie probeweise folgende Zeile in `OnTimer(..)` aus:
`device.Clear(ClearFlags.Target, Color.Blue, 1f, 0);`.
7. Variieren Sie die Konstante `nTriangles` zwischen `10` und `1000`.
8. Variieren Sie im Konstruktor die random initial pitch+yaw+roll rotation angles `ax[i], ay[i], az[i]` zwischen `0f` und `(float)(2 * Math.PI * r.NextDouble())` um das anfängliche Chaos zu verkleinern oder zu vergrößern.
9. Variieren Sie in `Matrix.RotationYawPitchRoll` die Bremsfaktoren `0.01f` der Winkelinkremente von `ay[i], ax[i]` und `az[i]` zwischen `0.001f` und `0.1f`, um das Chaos zu verkleinern oder zu vergrößern.
10. Programmieren Sie Hektik, indem Sie die 3 Matrix.Scaling-Parameter von `Matrix.Scaling( 0.5f, 0.5f, 0.5f )` durch Zufallszahlen `(float)r.NextDouble()` ersetzen.
11. Lesen Sie die Kommentare **C3DCisC1_Comment.htm** und versuchen Sie, den Sinn der Befehle zu verstehen.
12. Sie finden Erklärungen und Kommentare zu dieser Übung unter **http://msdn.microsoft.com/library/default.asp**.
Vorsicht: Mozilla Firefox zeigt den Baum auf der linken Seite nicht richtig an. Benutzen Sie den Internet Explorer hier !
12a: Verzweigen Sie in dem Baum auf der linken Seite auf:
Win32 and COM Development → Graphics and Multimedia → DirectX → SDK Documentation → DirectX SDK Managed → DirectX SDK → Introducing DirectX 9.0 → Direct3D Graphics → Getting Started with Direct3D → Direct3D Tutorials → Tutorial 1: Creating a Device.
12b: Verzweigen Sie in dem Baum auf der linken Seite auf:
Win32 and COM Development → Graphics and Multimedia → DirectX → SDK Documentation → DirectX SDK Update Managed → DirectX SDK → Introducing DirectX 9.0 → Direct3D Graphics → Getting Started with Direct3D → Devices.
12c: Verzweigen Sie in dem Baum auf der linken Seite auf:
Win32 and COM Development → Graphics and Multimedia → DirectX → SDK Documentation → DirectX SDK Managed → DirectX SDK → Introducing DirectX 9.0 → Direct3D Graphics → Getting Started with Direct3D → Direct3D Tutorials → Tutorial 2: Rendering Vertices.
13: Lesen Sie: **../../Lectures/L05_OpenGL_DirectX/OGL_DX_deutsch.htm#a5**