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

Project triangle1

This chapter is a synoptic summary of three Direct3D-Tutorials from Microsoft: `Tutorial1`, `Tutorial2` und `Tutorial3`. You find the tutorials here: `C:\DXSDK\Samples\Managed\Direct3D\Tutorials`.

Main Menu after starting VS 2005: `File → New Project... → Templates: Windows Application Name: triangle1 → Location: C:\temp → Create directory for solution:` switch it off `→ OK`
Delete the files `Program.cs` and `Form1.Designer.cs` and the content of `Form1.cs`, as descibed in the chapters 2DCisC1 to 2DCisC4.

If You find no `Solution Explorer`-window, open it via the main menu: `View → Solution Explorer`.
Inside the `Solution Explorer`-window click the plus-sign in front of `triangle1`. A tree opens. Look for the branch "`References`". **Right**-click `References` and **left**-click `Add Reference...`. An `Add Reference` dialog box opens. Scroll down to the component name: `Microsoft.DirectX Version 1.0.2902.0`. Highlight this reference by a left-click and (holding the `Strg`-key pressed) the reference `Microsoft.DirectX.Direct3D Version 1.0.2902.0` somewhere below. Quit the `Add Reference` dialog box with `OK`.
Check if both references `Microsoft.DirectX` and `Microsoft.DirectX.Direct3D` are now visible inside the Solution Explorer window underneath `triangle1 → References`.

If You use Visual Studio 2005 Professional You should switch off the vexatious automatic format- and indent- mechanism of the code editor before You copy the following code to `Form1.cs` (otherwise all the code will be reformatted into chaos):
1. Main menu of Visual Studio 2005 Professional: click menu "`Tools`".
2. A DropDown-menu appears. Click "`Options...`".
3. An `Options` dialog box appears.
4. Click the branch "`Projects and Solutions`". Click "`General`". Redirect all three pathes to `C:\temp`.
5. Click the branch "`Text Editor`", then click "`C#`".
6. A sub-tree appears with the branches "`General, Tabs, Advanced, Formatting, IntelliSense`".
7. Click "`Tabs`". Change "`Indenting`" to `None`, "`Tab size`" and "`Indent size`" to `1` and switch on the option "`Insert spaces`".
8. Inside the sub-tree "`C#`" click the plus-sign in front of "`Formatting`" and change all "`Formatting`"-branches as follows:
"`General`": switch off all CheckBoxes, "`Indentation`": switch off all CheckBoxes, "`New Lines`": switch off all CheckBoxes, "`Spacing`": switch off all CheckBoxes, "`Wrapping`": switch **on** both CheckBoxes.
9. Leave the dialog box with button "`OK`".

Form1, OnResize, OnTimer

Write the following code into the empty code window of `Form1.cs`:

```
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
  }
}
```

Click `Debug` → `Start Without Debugging Ctrl F5`. Try to drag all window borders.

Using the VertexBuffer of the graphics card

Until now at any Timer-event `triangle1` reads the triangle which has the form of the structure `CustomVertex.PositionColored[3]` from the main memory via the AGP- (or PCIe-) bus into the graphics card. This is no catastrophe in case of a single triangle but it is highly ineffective with komplex polygons. It is much faster to copy the polygons once into the `VertexBuffer`-memory of the graphics card. Nearly all modern graphics cards have such an on board memory for polygons (dynamically allocated).
Write the following declaration into the head of `public class Form1 : Form` below the line `static float fAngle;`:

```
  VertexBuffer vertexBuffer;
```

Write the following code into the function `protected override void OnResize( System.EventArgs e )` below `device.VertexFormat = CustomVertex.PositionColored.Format;`, but above `myTimer.Start();`.

```
      if ( vertexBuffer != null ) vertexBuffer.Dispose();//Free the old vertexBuffer if any.
      //Create a new vertex buffer on the graphics 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 memory.
      device.SetStreamSource( 0, vertexBuffer, 0 );//Use the vertexBuffer on the graphics card.
```

Inside `protected static void OnTimer( Object myObject, EventArgs myEventArgs )` change the line `device.DrawUserPrimitives(PrimitiveType.TriangleList, 1, v);` to

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

Click `Debug` → `Start Without Debugging Ctrl F5`. There is just a minimal performance benefit of approx. 10%. A more dramatic benefit will occur in case of larger polygons with more than 10 vertices.

More triangles

Change the `device.BeginScene(); - device.EndScene();` - clause inside `protected static void OnTimer( Object myObject, EventArgs myEventArgs )` as follows:

```
    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
```

Click `Debug` → `Start Without Debugging Ctrl F5`. Two new small triangles will appear.

More movement

Change the `device.BeginScene(); - device.EndScene();` - clause inside `protected static void OnTimer( Object myObject, EventArgs myEventArgs )` as follows:

```
    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
```

Click `Debug` → `Start Without Debugging Ctrl F5`. The two small triangles will dance around the first one.

Many triangles

Add the following declarations to the head of `public class Form1 : Form` below the line: `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();
```

Add the following intialisations to `public Form1()` below the line: `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
    }
```

Change the `device.BeginScene(); - device.EndScene();` - clause inside `protected static void OnTimer( Object myObject, EventArgs myEventArgs )` as follows:

```
    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
```

Click `Debug` → `Start Without Debugging Ctrl F5`. Hundred small triangles will dance synchronously as fishes do.

Chaos

Add the initializations of the rotating angles `ax[i], ay[i], az[i]` inside the loop `for ( int i = 0; i < nTriangles; i++ )` at the end of the constructor `public Form1()` by rewriting the loop from scratch:

```
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
}
```

Change the `device.BeginScene(); - device.EndScene();` - clause inside `protected static void OnTimer( Object myObject, EventArgs myEventArgs )` as follows:

```
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
```

Click `Debug` → `Start Without Debugging Ctrl F5`. Hundred small triangles will dance as flying paper sheets do.

Exercises

1. Slow the animation down by increasing `myTimer.Interval = 100;` inside the constructor.
2. Accelerate the animation by increasing `fAngle += 0.5f;` inside `OnTimer(..)`.
3. Rotate the triangle around the X-axis resp. around the Z-axis with `device.Transform.World = Matrix.RotationX( fAngle );` resp. `device.Transform.World = Matrix.RotationZ( fAngle );` inside `OnTimer(..)`.
4. Change the colors of the edges `v[0]`, `v[1]` and `v[2]` inside the constructor.
5. Step back with the eye point from `( 0f, 0f,-4f )` to `( 0f, 0f,-100f )` and get closer with `( 0f, 0f,-2f )`.
6. Experiment: Comment out the line `device.Clear(ClearFlags.Target, Color.Blue, 1f, 0);` inside `OnTimer(..)` and observe what happens.
7. Vary the constant `nTriangles` between `10` and `1000`.
8. Vary the random initial pitch+yaw+roll rotation angles `ax[i], ay[i], az[i]` inside the constructor between `0f` and `(float)(2 * Math.PI * r.NextDouble())` in order to decrease/increase the initial disorder.
9. Vary the retarding factors `0.01f` inside `Matrix.RotationYawPitchRoll` of the angular increments `ay[i], ax[i]` and `az[i]` between `0.001f` and `0.1f` in order to decrease/increase the chaos.
10. Make some bustling excitement by replacing the `0.5f` Matrix.Scaling-parameters `Matrix.Scaling( 0.5f, 0.5f, 0.5f )` by random numbers: `(float)r.NextDouble()`.
11. Read the comments **C3DCisC1_Comment.htm** and try to understand the sense of the code lines.
12. You can find more explanations and comments about this chapter here: **http://msdn.microsoft.com/library/default.asp**.
Caution: Mozilla Firefox doesn't correctly display the tree on the left side. Use the Internet Explorer here !
12a: Click trough the tree on the left side:
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: Click trough the tree on the left side:
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: Click trough the tree on the left side:
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: Read: **../../Lectures/L05_OpenGL_DirectX/OGL_DX_deutsch.htm#a5**