Course 3D_MDX: 3D- Graphics with Managed DirectX 9.0
Chapter C1: Comments to the Moving Triangels Project

---

namespaces

`using System;` //Home of the base class of all classes "`System.Object`" and of all primitive data types such as `Int32, Int16, double, string`.
`using System.Drawing;` //Home of the "`Graphics`" class and its drawing methods such as `DrawStirng, DrawLine, DrawRectangle, FillClosedCurve` etc.
`using System.Windows.Forms;` //Home of the "`Form`" class (base class of our main window `Form1)` and its method `Application.Run`.
`using Microsoft.DirectX;` //Utilities including exception handling, simple helper methods, structures for matrix, clipping, and vector manipulation.
`using Microsoft.DirectX.Direct3D;` //Graphics application programming interface (API) with models of 3-D objects and hardware acceleration.
For DirectX see: **http://msdn.microsoft.com/library/default.asp** → Win32 and COM Development → Graphics and Multimedia → DirectX → SDK Documentation → DirectX SDK Managed → DirectX SDK → Namespaces.

Entry to start our .NET Windows program: `public class Form1 : Form`

//We derive our window `Form1` from the class `Form`, which is contained in the `System.Windows.Forms` namespace.

`static void Main() { Application.Run( new Form1() ); }` //Create an instance of `Form1` and ask the operating system to start it as main window of our program.

`static Device device = null;` //The global device object must be static since we need it inside the static Timer event handler.

`static float fAngle;` //Global common basic movement of all traingles around the y-axis. At any Timer event all triangles turn by 0.1 radians ≈ 5.7 degrees.

`VertexBuffer vertexBuffer;` //This structure is necessary to create buffer space for vertices in the graphic board memory.

`static CustomVertex.PositionColored[] v = new CustomVertex.PositionColored[3];` //Memory space for 3 vertices each containing x,y,z plus a RGB-color. The keyword `static` is just needed in version 1 where `v` is directly used by the renderer inside the `static OnTimer`-function. All higher versions of the program use `vertexBuffer` instead of `v` and do not need this keyword anymore.

`Timer myTimer = new Timer();` //This Timer sends messages at fixed time intervals to `Form1`, that trigger `Form1` to execute its `OnTimer(..)` method.

`const Int32 nTriangles = 100;` //The number of triangles. Recommended experiment: Try out other values.

//6 arrays containing 3 shifts and 3 rotation angles for any individual triangle
`static float[] dx = new float[nTriangles];` //individual horizontal position
`static float[] dy = new float[nTriangles];` //individual vertical position
`static float[] dz = new float[nTriangles];` //individual depth position
`static float[] ax = new float[nTriangles];` //individual yaw = rotation around the x-axis
`static float[] ay = new float[nTriangles];` //individual pitch = rotation around the y-axis
`static float[] az = new float[nTriangles];` //individual roll = rotation around the y-axis

`static Random r = new Random();` //Class to generate random numbers. Its method `Random.NextDouble()` furnishes an arbitrary value between 0.0 and 1.0. We use it in the Constructor and in the Timer event handler to give each triangle its individual random position and individual rotation.

Constructor `public Form1()` inside `public class Form1`

`Text = "D3DTriangleAnimation";` //Title in the blue title bar of `Form1`.

`//fill coordinates and colors into an array "v"` //coordinates of a triangle in the xy-plane. z is always 0.
`v[0].X=-1f; v[0].Y=-1f; v[0].Z=0f;` //left lower vertex
`v[1].X= 1f; v[1].Y=-1f; v[1].Z=0f;` //right lower vertex
`v[2].X= 0f; v[2].Y= 1f; v[2].Z=0f;` //upper vertex

```
v[0].Color = System.Drawing.Color.DarkGoldenrod.ToArgb(); //color of left lower vertex
v[1].Color = System.Drawing.Color.MediumOrchid.ToArgb(); //color of right lower vertex
v[2].Color = System.Drawing.Color.Cornsilk.ToArgb(); //color of upper vertex
```

`myTimer.Tick += new EventHandler( OnTimer );` //Obligatory definition of an event handler for the Timer event.
`myTimer.Interval = 1;` //1 millisecond intervals means: as fast as possible. The operating system will raise as many events as possible.

`for ( int i = 0; i < nTriangles; i++ )` //Give any triangle an individual position position and a orientation in space.

```
{ 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 yaw rotation angle
  ay[i] = (float)r.NextDouble(); //random initial pitch rotation angle
  az[i] = (float)r.NextDouble(); //random initial roll rotation angle
}
```

`ClientSize = new Size( 400, 300 ); //Calls OnResize( ... )` //This statement raises an `OnResize(...)` event which leads to the first time initialization of a DirectX-`Device`.

Overridden event handler `protected override void OnResize( System.EventArgs e )` inside `public class Form1`
//Whenever the window changes we have to initialize Direct3D from scratch.

`myTimer.Stop();` //Stop the timer during initialization. It may disturb DirectX-initialization.

`try` //All the following things crash when DirectX is not properly installed. In this case the `try-catch` clause allows a decent suicide.

```
//Get information from the operating system about its current graphics properties.
PresentParameters presentParams = new PresentParameters(); //This structure is an obligatory parameter for creating a new Device. It carries several flags such as Windowed = true; and SwapEffect.Discard; = status flags controlling the behavior of the Device.
//we have to set two flags
presentParams.Windowed = true; //We want a program in a window not a full screen program.
presentParams.SwapEffect = SwapEffect.Discard; //This flag tells the graphic board how to handle the backbuffer(s) after front-back flipping. Many graphic boards need this flag, but I do not really know why. See:
```

http://msdn.microsoft.com/library/.../D3DSWAPEFFECT.asp

```
//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 );
```

//1. parameter = `0` = default device. (The computer can have differnt devices f.i. two graphic boards.)
//2. parameter = `DeviceType.Hardware` allows rasterization by the graphic board (HAL=first choice), software (HEL) or mixed.
//3. parameter = `this` means a pointer to a `System.Windows.Forms.Control` being the target of any graphical output.
//4. parameter = `CreateFlags.SoftwareVertexProcessing` is a flag that switches off the vector graphics part of the graphic board to avoid any risk from old graphic boards and/or old DirectX-drivers = all vector graphics via HEL. Disadvantage: Waste of the powerful HAL vector pipelines of a modern graphic board.
//5. parameter = `presentParams` is a structure of status flags describing the bevahior of a graphic board.
//see: **../../Lectures/L05_OpenGL_DirectX**

```
//set up the transformation of world coordinates into camera or view space //Comment
device.Transform.View = Matrix.LookAtLH( //
  new Vector3( 0.0f, 0.0f,-4.0f ), // eye point 4.0 in front of the canvas
  new Vector3( 0.0f, 0.0f, 0.0f ), // camera looks at point 0,0,0
  new Vector3( 0.0f, 1.0f, 0.0f ) ); // worlds up direction is the y-axis. See:
```

http://msdn.microsoft.com/archive

```
//set up the projection transformation using 4 parameters:
//1.: field of view = 45 degrees; 2.: aspect ratio = height/width = 1.0 = square window;
//3.: near clipping distance = 0; 4.: far clipping distance = 10;
device.Transform.Projection = Matrix.PerspectiveFovLH((float)Math.PI/4, 1f, 0f, 10f );.
```

//Describe the truncated viewing pyramid = frustum. 1. is the viewing angle in radians (PI/4=45°), 2. is the ratio height/width, 3. is the z-value of the front plane of the viewing volume and 4. the z-value of its back plane.
//See: **http://msdn.microsoft.com/archive**
//See: **www.lighthouse3d.com/opengl/viewfrustum/** Please mail me if this link is dead.
Recommended experiment 1: Enlarge `Math.PI/4` to `Math.PI/2 = 90`°. The scene will appear shifted away.
Recommended experiment 2: Distort the ratio to a) 0.5 and b) to 2.0.
Recommended experiment 3: Shift the front plane away from you in steps of 0.5.
Recommended experiment 4: Shift the back plane nearer to you in steps of 1.0 until it cuts through the convention of triangles.

```
//Turn off culling, so the user sees the front and back of the triangle //
device.RenderState.CullMode = Cull.None; //Culling means excluding (mostly back) surfaces from being rendered.
```

```
//Turn off lighting, since the triangle provides its own colors
device.RenderState.Lighting = false; //Switch off any external light. The triangles carry their own color.
```

```
//set up the property that fills the triangle with colors
device.VertexFormat = CustomVertex.PositionColored.Format; //We have to tell the device that any vertex carries a color.
```

```
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 ); //
// See: ../../Lectures/L06_3DVector/3D_Vertex/3DVertex_deutsch.htm#a3
vertexBuffer.SetData( v, 0, LockFlags.None ); //Copy the vertices from main memory to graphic card memory.
device.SetStreamSource( 0, vertexBuffer, 0 ); //Tell the device to use the vertexBuffer on the graphic card.
```

```
myTimer.Start(); //Start the timer again that has been stopped by the first statment of this function.
```

```
catch (DirectXException) { MessageBox.Show( "Could not initialize Direct3D." ); return; }
//Emergency exit when DirectX 9.0 was not found and/or new Device crashed. End of the try-clause = second statement of this function.
```

Event handler `protected static void OnTimer( Object myObject, EventArgs myEventArgs )` inside `public class Form1`

```
if (device == null) return; //Emergency exit if the DirectX Initialisation has gone wrong.
```

```
//throw the old image away
device.Clear( ClearFlags.Target, Color.Blue, 1f, 0 ); //Erase any former content from the canvas.
Recommended experiment: Kick out the Clear-statement and observe how the new cylinders cover the old ones.
```

```
//rotate with an angular velocity = 0.1 //The angular increment is 0.1 radians ≈ 5.7°.
fAngle += 0.1f; //Recommended experiment: Change this value in order to accelerate or slow down the animation.
device.Transform.World = Matrix.RotationY( fAngle ); //Presets two initial values sinus(fAngle) and cosinus(fAngel) into the world transform matrix = common basic orientation of all traingles.
```

```
//draw on the canvas
device.BeginScene(); //Open the render clause
  for ( int i = 0; i < nTriangles; i++ ) //Loop covering all triangles
  { device.Transform.World = Matrix.Scaling( 0.5f, 0.5f, 0.5f ) //reduce it to half size
      * Matrix.Translation( dx[i], dy[i], dz[i] ) //shift it to its individual position
      * Matrix.RotationYawPitchRoll(  //Simultaneous change of all 3D angles
                        ax[i] += 0.01f * (float)r.NextDouble() , //Slight individual random change of the pitch angle
                        ay[i] += 0.01f * (float)r.NextDouble() , //Slight individual random change of the yaw angle
                        az[i] += 0.01f * (float)r.NextDouble() ); //Slight individual random change of the roll angle
//See: ../../Lectures/L06_3DVector/3D_Basics/3DBasics_deutsch.htm#a2
  device.DrawPrimitives( PrimitiveType.TriangleList, 0, 1 ); //
  } //
device.EndScene(); //Close the render clause
device.Present(); //show the canvas // = Command to flip the front and the back buffer of the graphic board.
```