Course 3D_MDX: 3D-Graphics with Managed DirectX 9.0
Chapter C5: Comments to the "Read a Mesh from File.x" Project

Caution: Mozilla Firefox doesn't show the left side tree of MSDN. Recommended browser for MSDN: Internet Explorer

---

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.Windows.Forms;`
//Home of the "`Form`" class (base class of `Form1)` and its method `Application.Run`.
`using System.Drawing;`
//Home of the "`Graphics`" class and its drawing methods such as `DrawStirng,DrawLine,DrawRectangle,FillClosedCurve`
`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 xAngle, yAngle, zAngle;` //Global movements around the main 3 axes.

`static Mesh mesh = null;` //declaration of the mesh identifier

`String myMeshFile = @"C:\DXSDK\Samples\Media\Tiger\tiger.x";`
//The mesh file `tiger.x` is assumed to be found here.
`String myTextureFile = @"C:\DXSDK\Samples\Media\Tiger\tiger.bmp";`
//The texture image `tiger.bmp` is assumed to be found here.
//When You installed the DirectX SDK in the recommended directory `C:\DXSDK`, You will find `tiger.x` and `tiger.bmp` at this place, otherwise You have to adjust the path.
The character "`@`" in front of a string prevents the "`\`" characters inside the string to be interpreted as Escape-characters. Otherwise You would have to double them all: `"C:\\DXSDK\Samples\\Media\\Tiger\\tiger.x"`.

`BaseTexture texture;` //`BaseTexture` is the base class of `Texture`, which could be used here too.
`Bitmap myTexture;` //`Bitmap` object to be filled by `(Bitmap)Image.FromFile( myTextureFile )`.
(See last function `void MenuFileRead( object obj, EventArgs ea )`.)
`Timer myTimer = new Timer();`
//This Timer sends messages at fixed time intervals to `Form1`, that trigger `Form1` to execute its `OnTimer(..)` method.
`MenuItem miReadMesh;` //Menu object used by the constructor `public Form1()` and by the last function `void MenuFileRead( object obj, EventArgs ea )`.

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

```
        miReadMesh    = new MenuItem( "Read Mesh"   , new EventHandler( MenuFileRead ) );
MenuItem miReadTexture = new MenuItem( "Read Texture", new EventHandler( MenuFileRead ) );
MenuItem miExit        = new MenuItem( "Exit"        , new EventHandler( MenuFileExit ) );
MenuItem miFile        = new MenuItem( "File"        , new MenuItem[] { miReadMesh,
miReadTexture, miExit } );
Menu = new MainMenu( new MenuItem[] { miFile } );
```

//These lines define a drop down menu "`File`" with three entries: `Read Mesh`, `Read Texture` and `Exit`.

`Text = "DirectX3DMesh";` //Title in the blue title bar of `Form1`.

`myTexture = (Bitmap)Image.FromFile( myTextureFile );` //Read the first texture image from the hard disk.

`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 (normally 1000[msec] divided by monitor refresh[≈80Hz] ≈ 13 msec).

`ClientSize = new Size( 1024, 800 ); //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 offers a civilized exit.

```
//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 four 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**

```
presentParams.EnableAutoDepthStencil = true; //with depth buffer
```

//We want a Z-buffer on the graphics board.

```
presentParams.AutoDepthStencilFormat = DepthFormat.D16; //16 bit depth
```

//Z-buffer just needs limited resolution (short integers). Other possible formats see: **http://msdn.microsoft.com/archive**

```
//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 different 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` Pointer to our `Form1`-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 behavior of a graphic board.
//See: **../../Lectures/L05_OpenGL_DirectX**

```
if ( mesh != null ) mesh.Dispose(); //free the old mesh if any
mesh = Mesh.FromFile( myMeshFile, MeshFlags.SystemMemory, device );
```

//Read an `.x`-file from the hard disk.
//1. parameter = String that specifies the file name.
//2. parameter = One or more flags from the `MeshFlags` enumeration that specify creation options for the mesh. See: **msdn.microsoft.com**.
//3. parameter = The current `Device` object.

```
if ( texture != null ) texture.Dispose(); //free the old texture if any
texture = new Texture( device, myTexture, 0, Pool.Managed );
```

//1. parameter = The current `Device` object.
//2. parameter = `myTexture = (Bitmap)Image.FromFile( myTextureFile );` //Inside constructor `Form1()`.
//3. parameter = `Usage` enumeration that specify creation options for the mesh. See: **msdn.microsoft.com**.
//4. parameter = `Pool` Memory class that holds buffers for a resource. See: **msdn.microsoft.com**.

```
device.SetTexture( 0, texture );
```

//See: **msdn.microsoft.com**.

```
Material myMaterial = new Material();
myMaterial.Diffuse = myMaterial.Ambient = Color.White;
```
//Since all material properties are white, the meshes will reflect any sort of light.
`device.Material = myMaterial;` //Copy the material properties to the device.

```
//turn on some white ambient light that scatters the object evenly
device.RenderState.Ambient = Color.White;
```

```
//set up the transformation of world coordinates into camera or view space
device.Transform.View = Matrix.LookAtLH(
  new Vector3( 0f, 0f,-5f ), // eye point 5.0 in front of the canvas
  new Vector3( 0f, 0f, 0f ), // camera looks at point 0,0,0
  new Vector3( 0f, 1f, 0f ) ); // worlds up direction is the y-axis. //See:
```
http://msdn.microsoft.com

```
//set up the projection transformation using 4 parameters:
//1.: field of view = 45 degrees; 2.: aspect ratio = height / width = 1 = square window;
//3.: near clipping distance = 0; 4.: far clipping distance = 10;
device.Transform.Projection = Matrix.PerspectiveFovLH((float)Math.PI/4, 1f, 1f, 100f );.
```
//Describe the truncated viewing pyramid = frustum:
1. is the viewing angle in radians (PI/4=45^{o}),
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.
Experiment 1: Enlarge `Math.PI/4` to `Math.PI/2 = 90`^{o}. The scene will appear shifted away.
Experiment 2: Distort the ratio to a) 0.5 and b) to 2.0.
Experiment 3: Shift the front plane away from You towards the cylinder in steps of 0.5.

`device.RenderState.CullMode = Cull.None;` //Culling is a method to accelerate rendering by excluding (mostly back-) surfaces from the render process.

`device.RenderState.Lighting = true;` //Switch on the ambient light.

`xAngle = yAngle = zAngle = 0f; //start angles` //You can start with any arbitrary angle.

`myTimer.Start(); //start the timer again` //It has been stopped by the first statement 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 = 2nd 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 initialization has gone wrong.

```
//throw the old image away
device.Clear( ClearFlags.Target | ClearFlags.ZBuffer, Color.Gray, 1f, 0 );
```
//Erase any former content from the canvas and the Z-buffer.
Recommended experiment: Kick out this `Clear`-statement and observe what happens.

```
//rotate with 3 angular velocities
```
//The meshes rotate around three main axes with the same velocity of 0.02/step ≈ 3.6^{o}/step.
`Matrix m = Matrix.RotationYawPitchRoll(yAngle += 0.02f, xAngle += 0.02f, zAngle += 0.02f );`

`device.BeginScene();` //Open the render clause
`mesh.DrawSubset( 0 );` //Render the mesh. If the Mesh has more than one subset, You will just see the first.
`device.EndScene();` //Close the render clause
`device.Present(); //show the canvas` // = Command to flip the front and the back buffer of the graphic board.

Event handler `void MenuFileRead( object obj, EventArgs ea )` inside `public class Form1`

```
OpenFileDialog dlg = new OpenFileDialog(); //A file select dialog box appers in front of Form1

dlg.InitialDirectory = @"C:\DXSDK\Samples\Media";"; //default directory to start with.

if ( (MenuItem)obj == miReadMesh ) dlg.Filter = "meshes (*.x)|*.x| All files (*.*)|*.*";
//If the function has been started by the "ReadMesh" menu item, the dialog box looks for *.x-files.

else dlg.Filter = "Bitmaps (*.bmp)|*.bmp| All files (*.*)|*.*";
//If the function has been started by the "ReadTexture" menu item, the dialog box looks for *.bmp-files.

if ( dlg.ShowDialog() != DialogResult.OK ) return; //Forget everything if no file has been selected.

if ( (MenuItem)obj == miReadMesh ) myMeshFile = dlg.FileName;
//Remember the name of the new mesh file.

else myTexture = (Bitmap)Image.FromFile( dlg.FileName ); //Read a new texture image.

OnResize(null);//Initialize everything
```

Event handler `void MenuFileExit( object obj, EventArgs ea )` inside `public class Form1`

```
Application.Exit(); //Exit of Form1
```
