Course 3D_MDX: 3D-Graphics with Managed DirectX 9.0
Chapter C3: Comments to the Cylinder with Texture 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 `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 xAngle, yAngle, zAngle;` //Global movements of the cylinder around the main 3 axes.

//The following 3 vectors define the 3 axes of rotation of the render loop inside the `static OnTimer(...)`-function. For the sake of simpleness I choosed the main axes.
`static Vector3 xAxis = new Vector3( 1, 0, 0 );` //direction of the x-axis
`static Vector3 yAxis = new Vector3( 0, 1, 0 );` //direction of the y-axis
`static Vector3 zAxis = new Vector3( 0, 0, 1 );` //direction of the z-axis

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

`Bitmap bmp = null;` //image object
`Texture texture = null;` //texture object

`const int N = 100; //N must be an even no. 6, 8, 10, etc` //no. of vertices around the cylinder (50% on top, 50% on bottom). With 6, 8, 10 the cylinder will be rather awkward. It becomes rounder (at raising computation costs) with increasing `N`.

`CustomVertex.PositionNormalTextured[] vv = new CustomVertex.PositionNormalTextured[N];`
//Memory space for `N` vertices each containing 8 float values in 3 groups:
1) `X/Y/Z` = `vv[i].Position` = vertex coordinates,
2) `Nx/Ny/Nz` = `vv[i].Normal` = normal pointing towards the outside world,
3) `Tu/Tv` = `vv[i].Tu` and `vv[i].Tv` with `0.0 ≤ Tu,Tv ≤ 1.0` identifying what pixel is to be fixed at the vertex. The `Tu/Tv`-relative-coordinate system addresses an image as follows:
`Tu/Tv = 0.0/0.0` = upper left corner of the image,
`Tu/Tv = 1.0/1.0` = lower right corner of the image.
Cylinder vertices have the property that all `x/y` values of 1) and 2) are identical: `X == Nx` and `Y == Ny`. Explanation: The vector pointing from the central axis to the vertex is collinear to the vector pointing from the vertex to the outside world.

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

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

//These 4 statements create a menu:
```
MenuItem miRead = new MenuItem( "Read", new EventHandler( MenuFileRead ) ); //sub-item
MenuItem miExit = new MenuItem( "Exit", new EventHandler( MenuFileExit ) ); //sub-item
MenuItem miFile = new MenuItem( "File", new MenuItem[] { miRead, miExit } ); //super-item
Menu = new System.Windows.Forms.MainMenu( new MenuItem[] { miFile } ); //menu bar with super-item
```

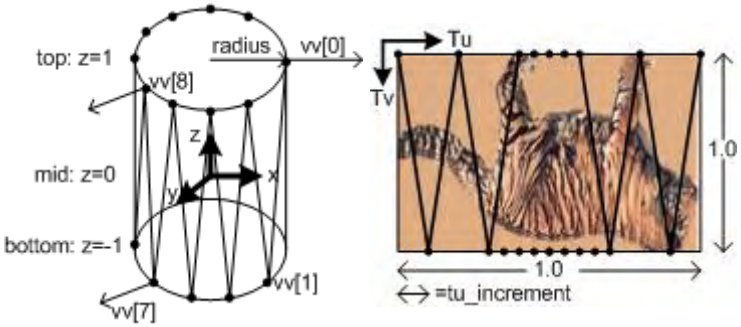
Behavior: At first just the string "`File`" is visible on the left of the new grey menu bar below the blue title bar.
On clicking "`File`" a drop down menu appears containing two items: "`Read`" and "`Exit`".
The 4 program statements are written in bottom-to-top-order:
1) : Create a sub-item "`miRead`"
2) : Create a sub-item "`miExit`"
3) : Create a super-item "`miFile`" containing 1) and 2).
4) : Create a menu bar and attach 3) inside this bar.

//The following statements try to read an image from the local hard disk or from the internet.
If one of the two methods succeeds, the image is stored as `bmp` = instance of the powerful `Bitmap`-class, otherwise `bmp = null` remains undefined.
```
try { bmp = (Bitmap)Image.FromFile( "C:\\DXSDK\\Samples\\Media\\Tiger\\tiger.bmp" ); }
catch { try { //Delete this inner try-catch clause if you have no Internet connection
running.
  String s = "http://www.miszalok.de/Images/tiger.bmp";
  System.Net.WebRequest webreq = System.Net.WebRequest.Create( s );
  System.Net.WebResponse webres = webreq.GetResponse();
  System.IO.Stream stream = webres.GetResponseStream();
  bmp = (Bitmap)Image.FromStream( stream );
} catch {}; }; // end of inner and outer try-catch clauses
```
//do nothing when no image was found

```
Text = "D3DTexture: Use the File menu to read new textures !";
```
//Title in the blue title bar of `Form1`.

`//TriangleStrip forming a cylinder` //For `TriangleStrip` see: **../../Lectures/.../3DVertex_deutsch.htm**

```
//radius=1; axis=Z-axis; top=1; bottom=-
1; → height=2;
```
//recommended experiment:
```
//in order to see the wireframe, replace
the the "TriangleStrip" by a "LineStrip"
in OnTimer(...)
```
//cylinder angular increment:
```
float arcus_increment = (float)(
2*Math.PI / (N-2) );
```
//texture horizontal increment:
```
float    tu_increment = (float)(
1.0       / (N-2) );
```
//Explanation of `(N-2)`:
`N-2` is the no. of triangles of the cylinder (`N` being the no. of vertices of the cylinder) because the last two vertices `vv[N-2]` and `vv[N-1]` must be identical to vertices `vv[0]` and `vv[1]` in order to close the strip.



```
Vector3 v = new Vector3(); //for intermediary X/Y/Z - variables
for (int i = 0; i < N; i++) //Fill up coordinates and normal vectors
```
//Fill the array `vv[N]` with `N` positions, `N` normals and `N` relative texture coordinates.
```
float arcus = i * arcus_increment; //This is the current angle.
v.X = (float)Math.Cos( arcus ); //next x on the circle
v.Y = (float)Math.Sin( arcus ); //next y on the circle
if ( i%2 == 0 ) v.Z = 1f; //If this is a even no. put it on top of the cylinder.
else            v.Z = -1f; //zigzag between top and bottom
```
//If this is a odd no. put it on the bottom of the cylinder.
```
vv[ i ].Position = v; //vertex = (cos,sin,+1) or (cos,sin,-1)
```
//copy the intermediary variable `v` into the array `vv`.
```
v.Z = 0; //cylinder normals are parallel to the xy-plane
```
//We give all normals a `z=0` value.
```
vv[ i ].Normal = v; //normal = (cos,sin,0)
```
//copy the intermediary variable `v` into the array `vv`.
```
vv[i].Tu = i * tu_increment; //horizontal texture position
if ( i%2 == 0 ) vv[i].Tv = 0f; //If this is a even no. put it on top of the texture image.
else            vv[i].Tv = 1f; //vertical zigzag on texture image
```
//If this is a odd no. put it on the bottom.

```
//set up the timer
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( 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 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
```

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

```
//Create a single, white, directional, diffuse light source and a gray ambient light.
//Many lights may be active at a time. (Notice: Each one slows down the render process.)
device.Lights[0].Type = LightType.Directional; //See: http://msdn.microsoft.com/archive
device.Lights[0].Diffuse = System.Drawing.Color.White; //Just white color
device.Lights[0].Direction = new Vector3( 0, 1, 1 ); //Light comes from upper mid in front of the monitor = roughly from the spectators forehead.
Recommended experiments: Change to upper left = -1,1,1; to lower left = -1,-1,1; to backside = 1,1,-1 etc.
device.Lights[0].Enabled = true; //We have to set the D3DRS_LIGHTING renderstate to enable lighting.
//Finally, turn on some ambient light that scatters and lights the object evenly
device.RenderState.Ambient = System.Drawing.Color.FromArgb( 0x202020 ); //0x202020 is moderate gray.
Recommended experiments: a) Switch it off: 0x000000; b) dim it heavily: 0x020202; c) turn it on: 0xFFFFFF.
```

```
//setup texture
if ( texture != null ) texture.Dispose(); //Throw away any old texture resource.
if ( bmp != null) texture = Texture.FromBitmap( device, bmp, 0, Pool.Managed ); //If there is an image, use it as texture
```
1. parameter: `device` = the current device
2. parameter: `bmp` = a `Bitmap`-object
3. parameter: `0` = no need to specify a `usage` type
4. parameter: `Pool.Managed` = type of storage; see: **http://msdn.microsoft.com/archive/...**

```
device.SetTexture( 0, texture ); //Hook the texture resource to the device stage no. 0.
```

```
//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 ) ); // 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 = square window;
//3.: near clipping distance = 0; 4.: far clipping distance = 10;
device.Transform.Projection = Matrix.PerspectiveFovLH((float)Math.PI/4, 1f, 1f, 10f );.
```
//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.
Experiment 4: Pull the back plane nearer to You in steps of 1.0 until it cuts through the cylinder.

```
//Turn off culling in order to render both the front and back sides of the triangle(s).
device.RenderState.CullMode = Cull.None; //Culling is a method to accelerate rendering by excluding (mostly back-) surfaces from the render process.
```

```
//Turn on lighting, otherwise the cylinder is an invisible white object in total darkness.
device.RenderState.Lighting = true; //Switch on the directional and the ambient light.
```

```
//set up the property that the cylinder has normals and texture coordinates
device.VertexFormat = CustomVertex.PositionNormalTextured.Format; //We have to tell the device that any vertex carries a normal and a texture coordinate Tu/Tv.
```

```
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.PositionNormalTextured), N, device, 0,
CustomVertex.PositionNormalTextured.Format, Pool.Default );
```
// See: **../../Lectures/L06_3DVector/3D_Vertex/3DVertex_deutsch.htm#a3**

```
vertexBuffer.SetData( vv, 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 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 cylinder rotates around three axis (here the main axes, but any other will do too).
//The x-rotation is set to 5.7° but the y+z-rotations much slower to 1.14° per timer event.
`xAngle += 0.1f;` //0.1 radians ≈ 5.7°.
`yAngle += 0.02f;` //0.02 radians ≈ 1.14°.
`zAngle += 0.02f;` //0.02 radians ≈ 1.14°.
//Compose this complicated simultaneous rotation:
`device.Transform.World  = Matrix.RotationAxis( xAxis, xAngle );` //Describe a first rotation by axis and angle.
`device.Transform.World *= Matrix.RotationAxis( yAxis, yAngle );` //Concatenate one more rotation by axis and angle.
`device.Transform.World *= Matrix.RotationAxis( zAxis, zAngle );` //Concatenate a third rotation by axis and angle.

```
//draw on the canvas
device.BeginScene(); //Open the render clause
  device.DrawPrimitives( PrimitiveType.TriangleStrip, 0, N-2 );
```
//Show the complete strip with `N-2` triangles.
```
  //Experiment: Replace the TriangleStrip by a LineStrip as follows:
  //device.DrawPrimitives( PrimitiveType.LineStrip, 0, N-2 );
device.EndScene(); //Close the render clause
device.Present(); //show the canvas
```
// = Command to flip the front and the back buffer of the graphic board.