Course 3D_MDX: 3D-Graphics with Managed DirectX 9.0
Chapter C2: Comments to Cylinder with Directional Light

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 movement of the cylinder (around the 3 main axes).

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

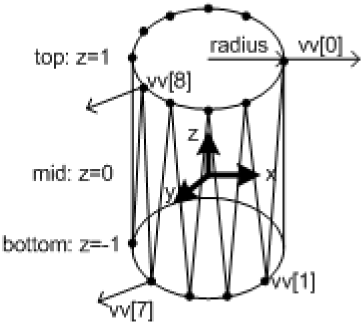
`const int N = 100; //N must be an even no. 6, 8, 10, etc` //no. of vertices around the cylinder (50% on top, 50% on bottom). With 6, 8, 10 the cylinder will be rather awkward. It becomes rounder (at raising computation costs) with increasing `N`.

`CustomVertex.PositionNormal[] vv = new CustomVertex.PositionNormal[N];` //Memory space for `N` vertices each containing 6 float values in 2 groups:
1) `X/Y/Z` = `vv[i].Position` = vertex coordinates,
2) `Nx/Ny/Nz` = `vv[i].Normal` = normal pointing towards the outside world.
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`

`Text = "D3DLights";` //Title in the blue title bar of `Form1`.



`//TriangleStrip forming a cylinder` //See: **../../Lectures/L06_3DVector/3D_Vertex/3DVertex_deutsch.htm**

`//radius = 1; axis = Z-axis; top = 1; bottom = -1; → height = 2;`

`//in order to see the wireframe, replace the the "TriangleStrip" by a "LineStrip" in OnTimer(...)` //recommended experiment

`float arcus_increment = (float)( 2 * Math.PI / (N-2) );` //360 degree divided by the no. of triangles. The no. of triangles = `(N-2)` 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 and `N` normals.
`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` //Top and bottom normals need no `z` and we give both of them `z=0`.
`vv[ i ].Normal = v; //normal = (cos,sin,0)` //copy the intermediary variable `v` into the array `vv`.

`//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 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 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.DarkTurquoise; //Arbitrary exotic color
device.Lights[0].Direction = new Vector3( 1, 1, 5 ); //Light comes from upper right in front of the monitor.
Experiments: Change to upper left = -1,1,5; to lower left = -1,-1,5; to backside = 1,1,-5 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.
Experiments: a) Switch it off: 0x000000; b) dim it heavily: 0x020202; c) turn it on: 0xFFFFFF.
```

```
//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 );.
//Describes 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.
Experiment 1: Enlarge Math.PI/4 to Math.PI/2 = 90°. 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: Shift 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
device.VertexFormat = CustomVertex.PositionNormal.Format; //We have to tell the device that any vertex carries a normal.
```

```
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.PositionNormal), N, device, 0, CustomVertex.PositionNormal.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, Color.Blue, 1f, 0 ); //Erase any former content from the canvas.
Experiment: Kick out the Clear-statement and observe how the new cylinders cover the old ones.
```

```
//rotate with an angular velocity = 5.7°/timer event //0.1 radians ≈ 5.7 degrees.
fAngle += 0.1f; //Experiment: Change this value in order to accelerate or slow down the animation.
device.Transform.World = Matrix.RotationAxis( new Vector3(1, 1, 1), fAngle ); //Rotation axis is 45° oblique to all three coordinate system axes.
```

```
//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.
```