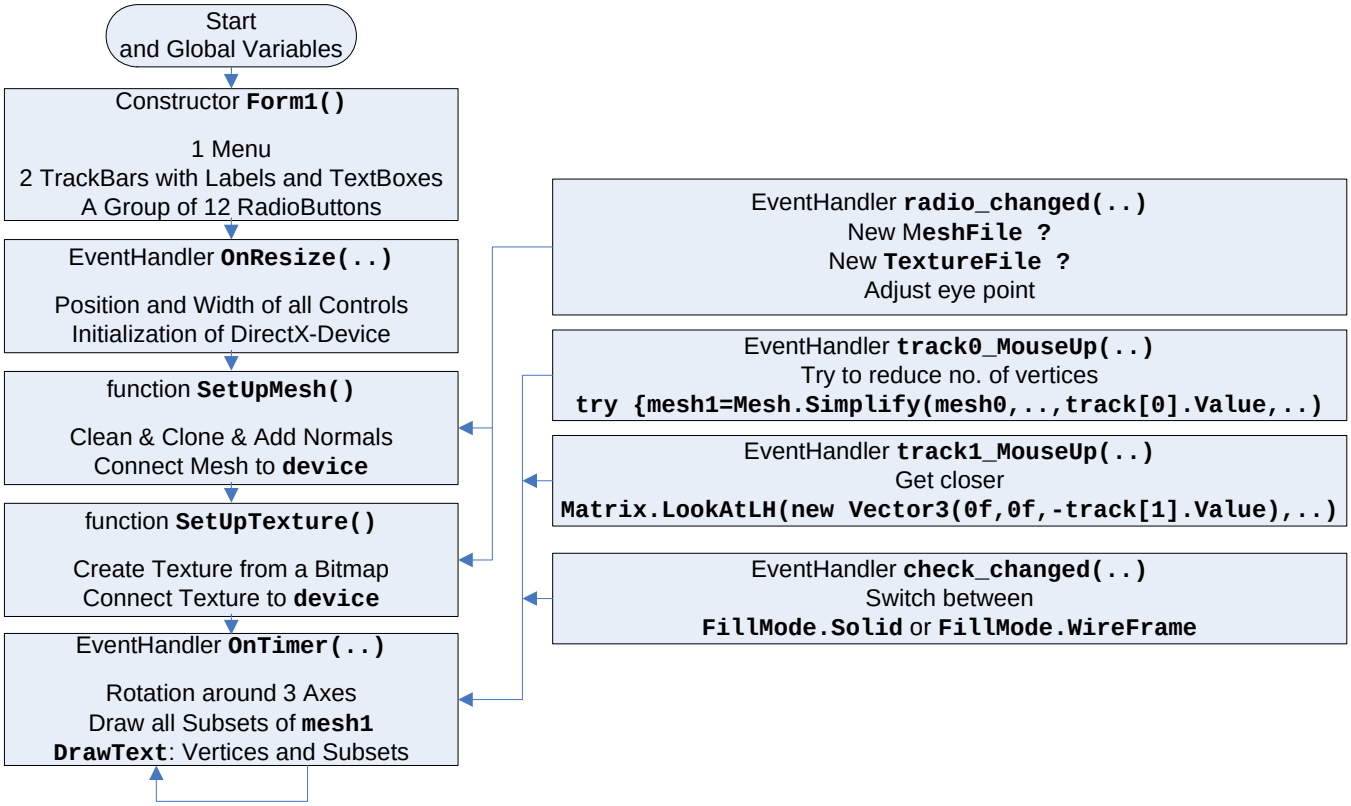
Course 3D_MDX: 3D-Graphics with Managed DirectX 9.0Chapter C5: Comments to the "Mesh Viewer" Project



---

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.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` etc.
`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.

Start and global variables: `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;` //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 mesh0, mesh1;` //declaration of two mesh identifiers

`static ExtendedMaterial[] materials;` //This is an "`out`"-parameter of the `Mesh.FromFile`-function which returns an array of information about the different subsets in the mesh. We just use the `materials.Length` property (inside `onTimer(..)`) in order to obtain the no. of subsets of the mesh. All other informations inside `ExtendedMaterial[]` about the subset(s) are ignored here.

`static Microsoft.DirectX.Direct3D.Font myfont;` //Direct3D-class to draw 3D-fonts. see: http://msdn2.microsoft.com/en-us/library/microsoft.windowsmobile.directx.direct3d.font.aspx

---

```
struct m{ public String title; public String mesh; public String texture; public Int32
eyedistance;
          public m(String a, String b, String c, Int32 d) //Constructor
          { title = a; mesh = b; texture = c; eyedistance = d; }
        }; // This is a structure plus constructor of 3 strings and 1 integer to describe 1) the abbreviation, 2) the path of
```

the `.x`-file, 3) the path of the texture image and 4) an eye distance that fits to this mesh.

`static String s = @"C:\DXSDK\Samples\";` //When You installed the DirectX SDK in the recommended directory `C:\DXSDK`, You will find `tiger.bmp` and all other predefined meshes 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"`.
`static m[] meshes =` //array of structures to be intialized:
//It follows a four column table: 1) the texts of the radio buttons, 2) the absolute path of the mesh, 3) the absolute path of the texture image, 4) the proposed eye distance.

```
{ new m("tiger"  ,s+@"Media\Tiger\tiger.x"            ,s+@"Media\Tiger\tiger.bmp", 5),
  new m("bigship",s+@"Media\misc\bigship1.x"          ,s+@"Media\Tiger\tiger.bmp", 50),
  new m("knot"   ,s+@"Media\misc\knot.x"              ,s+@"Media\Tiger\tiger.bmp", 5),
  new m("shapes" ,s+@"Media\misc\shapes1.x"           ,s+@"Media\Earth\earth.bmp", 50),
  new m("scull"  ,s+@"Media\misc\skullocc.x"          ,s+@"Media\Tiger\tiger.bmp", 50),
  new m("shark"  ,s+@"Media\Prt Demo\LandShark.x"     ,s+@"Media\Tiger\tiger.bmp",1000),
  new m("car"    ,s+@"C++\Direct3D\EffectParam\car2.x",s+@"C++\Direct3D\EffectParam\EffectParam.jpg",2000),
  new m("tiny"   ,s+@"Media\Tiny\tiny.x"              ,s+@"Media\Tiger\tiger.bmp",2000),
  new m("dwarf"  ,s+@"Media\Dwarf\dwarf.x"            ,s+@"Media\Tiger\tiger.bmp", 10),
  new m("airplan",s+@"Media\Airplane\airplane 2.x"    ,s+@"Media\Airplane\bihull.bmp",50),
  new m("headsad",s+@"Media\Prt Demo\Head_Sad.x"      ,s+@"Media\Tiger\tiger.bmp",1000),
  new m("virus"  ,s+@"C++\Direct3D\EffectParam\cytovirus.x",s+@"Media\Tiger\tiger.bmp",2000)
};
```

```
String myMeshFile = meshes[0].mesh; //Default mesh is "tiger.x".
String myTextureFile = meshes[0].texture; //Default texture is "tiger.bmp".
```

```
Bitmap          myBitmap  = null; //Bitmap object used in the event handler radio_changed(..).
BaseTexture     myTexture = null; //BaseTexture is the base class of Texture-class.
GraphicsStream  adjacency = null; //This is an "out"-parameter of the Mesh.FromFile & Mesh.Clean-
```

functions. Not important here.

```
GroupBox group = new GroupBox(); //Create a box grouping all radio buttons
TrackBar[] track = new TrackBar[2]; //Declare an array of two trackbars
Label[] label = new Label[2]; //Declare an array of two labels for trackbar titles
TextBox[] text = new TextBox[2]; //Declare an array of two text boxes for trackbar values
CheckBox check = new CheckBox(); //Create a checkbox for "Wire Frame yes/no"
Panel panel = new Panel(); //Create a canvas covering the complete Form1 client size area right of the buttons
Timer myTimer = new Timer(); //This Timer sends messages at fixed time intervals to Form1, that trigger Form1 to
```

execute its `OnTimer(..)` event handler.

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

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

```
for ( int i = 0; i < track.Length; i++ ) //Create two TrackBars togehter with their Labels and TextBoxes and
```

add them to `Form1`.

```
{ label[i] = new Label();    Controls.Add( label[i] ); //Create a Label and add it to the control collection
```

of `Form1`.

```
  track[i] = new TrackBar(); Controls.Add( track[i] ); //Create a TrackBar and add it to the control
```

collection of `Form1`.

```
  text [i] = new TextBox();  Controls.Add( text [i] ); //Create a TextBox and add it to the control
```

collection of `Form1`.

```
  label[i].BackColor = track[i].BackColor = Color.Gray; //Set properties.
  track[i].Minimum = 1; //Both TrackBars have the same minimum = 1.
  track[i].TickStyle = TickStyle.None; //TrackBars need no rules.
  label[i].TextAlign = ContentAlignment.MiddleCenter; //Centered titles.
  text [i].TextAlign = HorizontalAlignment.Center; //Horizontally centered values.
}
```

```
Controls.Add( group ); //Add a grouping rectangle to Form1 that will contain 12 RadioButtons.
for ( int i = 0; i < meshes.Length; i++ ) //Create 12 RadioButtons to select 12 different mesh files.
{ radio[i] = new RadioButton(); Controls.Add( radio[i] );
  radio[i].Parent = group; //Put it into the Group "group".
  radio[i].Text = meshes[i].title; //Assign a short string indicating the content of the mesh.
  radio[i].Location = new Point( 5, Convert.ToInt32((0.6 + i*1.2)*FontHeight) ); //Assign a Left=5 and a Top property.
  radio[i].Size = new Size( 60, Convert.ToInt32(1.2*Font.Height) ); //Assign a Width=60 and a Height property.
  radio[i].TextAlign = ContentAlignment.MiddleCenter; //Adjust the strings in the center of the text space.
  radio[i].CheckedChanged += new EventHandler( radio_changed ); //All RadioButtons obtain a common event handler function.
  if ( i == 0 ) radio.Checked = true; //Default check: the first RadioButton.
}
```

```
label[0].Text = "Reduce Vertices"; label[1].Text = "Eye Distance"; //Write TrackBar titles into both Labels above both TrackBars.
```

```
Controls.Add( check ); //Add the CheckBox "Wire Frame" to the control collection of Form1.
Controls.Add( panel ); //Add the big canvas which covers most of the client area of Form1 to the control collection of Form1.
```

```
track[0].MouseUp       += new EventHandler( track0_MouseUp ); //Assign an event handler function to the 1st TrackBar.
track[1].MouseUp       += new EventHandler( track1_MouseUp ); //Assign an event handler function to the 2nd TrackBar.
check   .CheckedChanged += new EventHandler( check_changed ); //Assign an event handler function to the CheckBox "Wire Frame".
```

```
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).
```

```
myBitmap = (Bitmap)Image.FromFile( meshes[0].texture ); //Read the default first texture image (tiger.bmp) from the hard disk.
track[1].Value = track[1].Maximum = meshes[0].eyedistance; //Set the default first eye distance = 5 for tiger.x.
```

```
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.
```

```
//The following statements adjust the positions and size properties of all controls to a new window size.
for ( int i = 0; i < track.Length; i++ ) //For both Trackbars with both Labesls and both TextBoxes
  label[i].Width = track[i].Width = text[i].Width = ClientSize.Width / 10; //new Width
text[0].Text = "vertices = " + track[0].Value.ToString(); //new text
text[1].Text = "eye = - " + track[1].Value.ToString(); //new text
check .Text = "Wire Frame"; //new text
group.Size = new Size( ClientSize.Width / 10, meshes.Length*radio[0].Height + 6 ); //new Width and Height
check.Size = new Size( ClientSize.Width / 10, radio[0].Height ); //new Width and Height
panel.Size = new Size( ClientSize.Width - label[0].Width - 2, ClientSize.Height - 2 ); //new Width and Height
group   .Location = new Point( 2, 1 ); //new Left and Top
label[0].Location = new Point( 2, group .Location.Y + group .Height + 20 ); //new Left and Top
track[0].Location = new Point( 2, label[0].Location.Y + label[0].Height + 1 ); //new Left and Top
text [0].Location = new Point( 2, track[0].Location.Y + track[0].Height + 2 ); //new Left and Top
label[1].Location = new Point( 2, text[ 0].Location.Y + text [0].Height + 20 ); //new Left and Top
track[1].Location = new Point( 2, label[1].Location.Y + label[1].Height + 1 ); //new Left and Top
text [1].Location = new Point( 2, track[1].Location.Y + track[1].Height + 2 ); //new Left and Top
check   .Location = new Point( 2, text [1].Location.Y + text [1].Height + 20 ); //new Left and Top
panel   .Location = new Point( 2 + group.Width + 2, group.Location.Y ); //new Left and Top
```

`try` //All the following things crash when DirectX is not properly installed. In this case the `try-catch` clause offers a civilized exit.

`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.
`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**

```
if ( device != null ) device.Dispose(); //Free the old canvas if any.
device = new Device( 0, DeviceType.Hardware, panel, CreateFlags.MixedVertexProcessing,
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 = `panel` Pointer to our `Panel`-Control being the target of any graphical output.
//4. parameter = `CreateFlags.MixedVertexProcessing` is a flag that allows the driver to decide at run time whether to use the vector graphics part of the graphics card via HAL or to use HEL instead.
//5. parameter = `presentParams` is a structure of status flags describing the behavior of a graphic board.
//see: **../../Lectures/L05_OpenGL_DirectX**

```
//turn on some white directional light from the left to right
device.Lights[0].Type = LightType.Directional; //Directional light.
device.Lights[0].Diffuse = Color.White; //with white color
device.Lights[0].Direction = new Vector3( 1, 0, 0 ); //Light points horizontally from left to right.
device.Lights[0].Enabled = true; //Pure bureaucracy.
```

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

```
device.RenderState.Ambient = Color.FromArgb( 0x00303030 ); //Dim gray light.
```

```
device.Transform.Projection = Matrix.PerspectiveFovLH((float)Math.PI/4, 1f, 1f, 5000f );.
```
//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.

```
device.Transform.View = Matrix.LookAtLH(
  new Vector3( 0f, 0f,-track[1].Value ), //Eye point in front of the canvas depends on the 2nd TrackBar.
  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
```

```
device.RenderState.Lighting = true; //Switch on the diffuse directional white light.
```

//The CheckBox switches the triangles of the mesh from being filled (default=`FillMode.Solid`) to being left open (`FillMode.WireFrame`).

```
if ( check.Checked ) device.RenderState.FillMode = FillMode.WireFrame;
else                 device.RenderState.FillMode = FillMode.Solid;
```

```
myfont = new Microsoft.DirectX.Direct3D.Font( device, new System.Drawing.Font( "Arial", 12,
FontStyle.Bold ) ); //Creates a DirectX-font-object (This is an alternative to a .NET font object Font myfont = new
Font( "Arial", 12 );).
```

```
SetUpMesh();     //Load the current mesh and connect it to the device.
SetUpTexture(); //Load the current texture and connect it to the device.
```

```
myTimer.Start(); //The Timer 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.

Private function `SetUpMesh()` inside `public class Form1`

//This function is called:
1) by `OnResize(..)` after having created a new `device`,
2) by a user click on one of the grouped RadioButtons named `tiger, bigship, .., virus`.
Purpose:
1) Read a mesh from an `.x`-file, connect it to `device` and check its health.
2) Create two identical meshes `mesh0` and `mesh1` by cloning forth and back (needed for further simplification).
3) Both meshes must carry vertices of type `CustomVertex.PositionNormalTextured`. If there are no normals, compute them (needed to reflect directional light).

```
Cursor.Current = Cursors.WaitCursor; //change mouse pointer to hour glass
```
//The following `Mesh`-methods can be time consuming.
```
if ( mesh0 != null ) mesh0 .Dispose(); //free the old mesh if any
mesh0 = Mesh.FromFile( myMeshFile, MeshFlags.Managed, device, out adjacency, out materials );
```
//Read an `.x`-file.
```
mesh0 = Mesh.Clean( CleanType.Optimization, mesh0, adjacency, adjacency );
```
//Make sure that `mesh0` is correct (It may have syntax bugs and geometrical loops).
```
if ( mesh1 != null ) mesh1.Dispose(); //free the old mesh if any
//make a copy mesh1 from mesh0 //
mesh1 = mesh0.Clone( mesh0.Options.Value, mesh0.VertexFormat | VertexFormats.Normal, device );
```
//Make sure that `mesh1` contains space for vertex normals.
```
//if mesh0 has no normals, compute them within mesh1 and copy them back to mesh0
if ( (mesh0.VertexFormat & VertexFormats.Normal) == 0 )
```
//There are no normals
```
{ mesh1.ComputeNormals();
  mesh0.Dispose();
  mesh0 = mesh1.Clone( mesh1.Options.Value, mesh1.VertexFormat, device );
```
//Make sure that both meshes are identical.
```
}
track[0].Value = track[0].Maximum = mesh0.NumberVertices;
```
//Set TrackBar.Maximum of the 1st TrackBar.
```
text[0].Text = "vertices = " + track[0].Value.ToString();
```
//Show TrackBar.Maximum in the 1st TextBox.
```
Cursor.Current = Cursors.Arrow; //change mouse pointer back to normal arrow
```

Private function `SetUpTexture()` inside `public class Form1`

//This function is called:
1) by `OnResize(..)` after having created a new `device`,
2) by a user click on one of the grouped RadioButtons named `tiger, bigship, .., virus`.
Purpose: Create a `Texture` object from an existing `Bitmap` object and connect it to `device`.

```
if ( myTexture != null ) myTexture.Dispose(); //free the old texture if any
myTexture = new Texture( device, myBitmap, 0, Pool.Managed );
```
//Create a `Texture`-object.
```
device.SetTexture( 0, myTexture );
```
//Connect the `Texture`-object to `device`.

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

```
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.
```
device.Transform.World = Matrix.RotationYawPitchRoll( yAngle += 0.02f, xAngle += 0.02f, zAngle += 0.02f );
```

```
device.BeginScene(); //Open the render clause
for ( int i=0; i < materials.Length; i++ ) mesh1.DrawSubset( i ); //Render the mesh with all subsets.
myfont.DrawText( null, "This mesh has " + mesh0.NumberVertices.ToString() + " vertices",
```
//"This mesh has xxx vertices"
```
                 new Rectangle( 0, 0, 100, 20 ), DrawTextFormat.NoClip, Color.Red );
```
//Left upper corner of `Panel`.
```
myfont.DrawText( null, "divided into " + materials.Length.ToString() + " subsets",
```
//"divided into x subsets"
```
                 new Rectangle( 0, 20, 100, 20 ), DrawTextFormat.NoClip, Color.Red );
```
//Left upper corner of `Panel` but below the 1st string
```
device.EndScene(); //Close the render clause
device.Present(); //show the canvas
```
//Flip the front and the back buffer of the graphic board.

Event handler `private void radio_changed( Object sender, EventArgs e )` inside `public class Form1`

//This function is called by clicking on one of the grouped RadioButtons named `tiger, bigship, .., virus`.

```
RadioButton radio = (RadioButton)sender; //Find out from which RadioButton the call comes from.
Int32 i; //Remember the fitting RadioButton no.
for ( int i = 0; i < meshes.Length; i++ ) //Investigate all RadioButtons until You find the right one.
{ if ( meshlist[i,0] == radio.Text ) break; //Stop the loop, when the no. has been found.
```

```
if ( myMeshFile != meshes[i].mesh ) //Is the new mesh file the same as the current one ?
{ myMeshFile = meshes[i].mesh; //Read it from struct-array static m[] meshes.
  SetUpMesh(); //Call a subroutine (see above).
}
```

```
if ( myTextureFile != meshes[i].texture ) //Is the new texture file the same as the current one ?
{ myBitmap = (Bitmap)Image.FromFile( myTextureFile = meshes[i].texture ); //Read it from struct-array static m[] meshes and convert it to a Bitmap-object.
  SetUpTexture(); //Call a subroutine (see above).
}
```

```
track[1].Value = track[1].Maximum = meshes[i].eyedistance; //Set the 2nd TrackBar to an reasonable eye distance.
text[1].Text = "eye = - " + track[1].Value.ToString(); //Display the current eye distance inside the 2nd TextBox.
```

```
device.Transform.View = Matrix.LookAtLH( //Set up the transformation of world coordinates into camera-space (=view-space).
  new Vector3( 0f, 0f, -meshes[i].eyedistance ), //Camera position depends on the 2nd TrackBar. Negative Z-value means a position in front of the screen.
  new Vector3( 0f,0f,0f ), //Camera axis points to the middle of the client area.
  new Vector3( 0f,1f,0f ) ); //Y-axis points upwards.
```

Event Handler `private void track0_MouseUp(object sender, System.EventArgs e)` inside `public class Form1`

= Reduce no. of vertices TrackBar

```
if ( materials.Length > 1 ) //Since each subset has its own material, the no. of materials indicates the no. of subsets.
{ MessageBox.Show( "This mesh has more than one subset. It can't be simplified !" ); //No chance to simplify such a mesh at once.
  track[0].Value = track[0].Maximum; //Reset the 1st Trackbar to its default value.
  return; }
```

```
Cursor.Current = Cursors.WaitCursor; //Simplification of a mesh can be time consuming.
if ( mesh1 != null ) mesh1.Dispose(); //Throw the old mesh1 away.
try { mesh1 = Mesh.Simplify( mesh0, adjacency, null, track[0].Value,
MeshFlags.SimplifyVertex ); } //one of the most powerful methodes of the Mesh-class.
//If simplification doesn't work clone mesh1 from mesh0 and inform the user:
catch { mesh1 = mesh0.Clone( mesh0.Options.Value, mesh0.VertexFormat, device );
        MessageBox.Show( "This mesh cannot be simplified !" ); }
```

```
track[0].Value = mesh1.NumberVertices; //Update the 1st TrackBar.
text[0].Text = "vertices = " + mesh1.NumberVertices.ToString(); //Update the 1st TextBox.
Cursor.Current = Cursors.Default; //Reset the mouse cursor to its normal arrow-form.
```

Event Handler `private void track1_MouseUp(object sender, System.EventArgs e)` inside `public class Form1`

= Eye distance TrackBar

```
device.Transform.View = Matrix.LookAtLH( //Set up the transformation of world coordinates into camera-space (=view-space).
        new Vector3( 0f, 0f, -track[1].Value ), //eye point 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
text[1].Text = "eye = - " + track[1].Value.ToString(); //Update the 2nd TextBox.
```

Event Handler `private void check_changed( Object sender, EventArgs e )` inside `public class Form1`

= WireFrame CheckBox

```
if ( check.Checked ) device.RenderState.FillMode = FillMode.WireFrame; //Only the outlines of the triangles are rendered.
else                 device.RenderState.FillMode = FillMode.Solid; //Normal rendering with filled triangles.
```