

Course 3D_XNA Chapter C4: Comments to the XBox 360 Controller Project

Copyright © by V. Miszalok, last update: 18-12-2007

namespaces

using System; //Home of the base class of all classes "System.Object".

using Microsoft.Xna.Framework;

//Provides commonly needed game classes such as timers and game loops.

See: <http://msdn2.microsoft.com/en-us/library/microsoft.xna.framework.aspx>

using Microsoft.Xna.Framework.Content; //Contains classes to load a mesh or model from the Content Pipeline and to manage the lifespan of the loaded objects.

See: <http://msdn2.microsoft.com/en-us/library/microsoft.xna.framework.content.aspx>

using Microsoft.Xna.Framework.Graphics;

//Contains classes, structures and enumerations to access the graphics board directly.

See: <http://msdn2.microsoft.com/en-us/library/microsoft.xna.framework.graphics.aspx>

using Microsoft.Xna.Framework.Input; //Contains classes, structures and enumerations to receive input from keyboard, mouse, and Xbox 360 Controller.

See: <http://msdn2.microsoft.com/en-us/library/microsoft.xna.framework.input.aspx>

Entry to start public class Game1 : Microsoft.Xna.Framework.Game

//Our Game1 is derived from Microsoft.Xna.Framework.Game which is the base class of XNA.

See: http://msdn2.microsoft.com/en-us/library/microsoft.xna.framework.game_members.aspx

static class Program {[STAThread] static void Main() {Game1 game=new Game1();game.Run();}
//Create a single thread instance of Game1 and ask the operating system to start it.

GraphicsDeviceManager g;

//A GraphicsDevice performs rendering, creates resources and creates shaders.

The GraphicsDeviceManager handles the configuration and management of the GraphicsDevice.

See: http://msdn2.microsoft.com/en-us/library/microsoft.xna.framework.graphicsdevicemanager_members.aspx

Model model; //A Model contains all parts of a 3D drawing, at least one ModelMesh=set of vertices, one BasicEffect=graphics board settings and one Bone=dependence level.

Models are stored in the Content Pipeline of the game.

See: http://msdn2.microsoft.com/en-us/library/microsoft.xna.framework.graphics.modelmesh_members.aspx

BasicEffect effect; //A BasicEffect is a part of Model and is loaded with Model from the content pipeline of the game (file name extension: .fx). It describes the current settings of the graphics board such as Vertex Shader, Pixel Shader and Technique and it requires a set of world, view, and projection matrices, a vertex buffer and a vertex declaration.

Form1 form; //The main XNA-game window cannot show controls such as buttons, labels, tackbars etc. That's why we need a second window derived from Windows.Form.

float positionX = 0.0f, positionY = 0.0f, positionZ = 0.0f; //tiger's initial position
float scaleX = 1.0f, scaleY = 1.0f, scaleZ = 1.0f; //tiger's initial zoom
float rotationX = 0.0f, rotationY = 0.0f, rotationZ = 0.0f; //tiger's initial rotations
//These variables will change when the controller's buttons, sticks and DPadsThese will be pressed.

public Game1() { g = new GraphicsDeviceManager(this); } //Constructor of Game1.

Create an instance of GraphicsDeviceManager which we need inside

protected override void Initialize() to set some properties of our GraphicsDevice.

Event handler protected override void Initialize() inside public class Game1

//Obligatory event handler of any XNA Game

```
g.PreferredBackBufferWidth = 600; //Allocate output buffer space on the graphics board
g.PreferredBackBufferHeight = 600; //Allocate output buffer space on the graphics board
g.ApplyChanges(); //Force the graphics board to rearrange its memory structure
```

```
g.IsFullScreen = false; //Game1 appears in a resizable window.
Window.AllowUserResizing = true; //The user is allowed to change the window size at run time.
Window.Title = "Moving the Tiger via an Xbox 360 Controller"; //The title of the window.
```

```
base.Initialize(); //Complete initialization of DirectX. Enumerate through any graphics components
that have been added to components and call their Initialize methods.
```

This call is obligatory after the GraphicsDevice is created, but before LoadGraphicsContent.

```
form = new Form1(); //Create a a second window Form1 which we define later in a separate class.
```

```
form.Location = new System.Drawing.Point( Window.ClientBounds.Right+5,
Window.ClientBounds.Top ); //Put the second window closely right of the first one on the screen and
at the same hight.
```

```
Window.ClientBounds.Right //furnishes the screen coordinates of the right border of the client area of
the XNA-game-window.
```

```
Window.ClientBounds.Top //furnishes the screen coordinates of the upper border of the client area of
the XNA-game-window.
```

```
form.Size = new System.Drawing.Size ( 100, Window.ClientBounds.Height );
//Width of Form1 = 100 and height = same as the game window.
```

```
form.Show(); //Form1 appears on the screen.
```

Event handler protected override void LoadContent() inside public class Game1

```
model = (new ContentManager(Services)).Load< Model >( "tiger" );
//Creates a ContentManager which loads model "tiger" from the content pipeline of the game.
```

```
effect = (BasicEffect)model.Meshes[0].Effects[0];
//Extract from model "tiger" the first part of the first mesh and write it into the vertex buffer of the graphics board.
Since model "tiger" just has one single part in one single mesh this line loads the complete tiger.
```

```
effect.View= Matrix.CreateLookAt( new Vector3(0.0f,0.0f,4.0f), Vector3.Zero, Vector3.Up );
//Overrides the empty values of effect by the following settings: camera position 4.0 in front of the display,
pointing from the players nose to the center of the window. The y-axis goes up.
```

```
effect.Projection = Matrix.CreatePerspectiveFieldOfView( MathHelper.Pi/4,1f,0.1f,1000.0f );
//Overrides the empty values of effect by the following settings: viewing angle = 45 degrees; aspect ratio 1:1;
front clipping plane close at z=0.1; back clipping plane far away at z=1000.0f.
```

Update method `protected override void Update(GameTime gameTime)`
 = Game State Manager inside `public class Game1`

// This function is derived from a virtual function `protected virtual void Update(GameTime gameTime)` of the parent class `Microsoft.Xna.Framework.Game` which calls this function automatically as often as possible together with the `protected override void Draw(GameTime gameTime)`-function below. Here is the right place to check the player's actions and to respond to his movements, collisions etc.

```
GamePadState s = GamePad.GetState( 0 ); //Get the status of player's 1 controller
```

```
if ( !s.IsConnected )
{ System.Windows.Forms.MessageBox.Show( "Can't find a Xbox 360 Controller" ); Exit(); }
//Stop the game if there is no player's 1 controller.
```

```
ButtonState bp = ButtonState.Pressed; //bp is just an acronym for "ButtonState.Pressed"
```

```
if ( s.Buttons.A != bp) form.checkbox[ 0].Checked = false; //deplete checkbox 0
else { scaleY *= 0.99f; form.checkbox[ 0].Checked = true; }
//Button A: 1% Y-down-zoom → cut short the tiger
```

```
if ( s.Buttons.B != bp) form.checkbox[ 1].Checked = false; //deplete checkbox 1
else { scaleX *= 1.01f; form.checkbox[ 1].Checked = true; }
//Button B: 1% X-up-zoom → thicken the tiger
```

```
if ( s.Buttons.X != bp) form.checkbox[ 2].Checked = false; //deplete checkbox 2
else { scaleX *= 0.99f; form.checkbox[ 2].Checked = true; }
//Button X: 1% X-down-zoom → slender the tiger
```

```
if ( s.Buttons.Y != bp) form.checkbox[ 3].Checked = false; //deplete checkbox 3
else { scaleY *= 1.01f; form.checkbox[ 3].Checked = true; }
//Button Y: 1% Y-up-zoom → tiger gets taller
```

```
if ( s.Buttons.LeftShoulder != bp) form.checkbox[ 4].Checked = false; //deplete checkbox 4
else { scaleZ *= 0.99f; form.checkbox[ 4].Checked = true; }
//Button LeftShoulder: 1% Z-down-zoom → tiger moves back
```

```
if ( s.Buttons.RightShoulder!= bp) form.checkbox[ 5].Checked = false; //deplete checkbox 5
else { scaleZ *= 1.01f; form.checkbox[ 5].Checked = true; }
//Button RightShoulder: 1% Z-up-zoom → tiger moves forward
```

```
if ( s.Buttons.Start != bp) form.checkbox[ 6].Checked = false; //deplete checkbox 6
else { reset(); form.checkbox[ 6].Checked = true; }
//Button Start: reset all positions and rotations to 0 and all zooms to 1.
```

```
if ( s.Buttons.Back != bp) form.checkbox[ 7].Checked = false; //deplete checkbox 7
else { reset(); form.checkbox[ 7].Checked = true; }
//Button Reset: reset all positions and rotations to 0 and all zooms to 1.
```

```
if ( s.Buttons.LeftStick != bp) form.checkbox[ 8].Checked = false; //deplete checkbox 8
else { form.checkbox[ 8].Checked = true; }
//do nothing except an ok in checkbox 8
```

```
if ( s.Buttons.RightStick != bp) form.checkbox[ 9].Checked = false; //deplete checkbox 9
else { form.checkbox[ 9].Checked = true; }
//do nothing except an ok in checkbox 9
```

```

if (s.DPad.Left!= bp)      form.checkbox[10].Checked = false; //deplete checkbox 10
else { positionX -= 0.01f; form.checkbox[10].Checked = true; }
//DPad.Left: tiger moves left

if (s.DPad.Right != bp)   form.checkbox[11].Checked = false; //deplete checkbox 11
else { positionX += 0.01f; form.checkbox[11].Checked = true; }
//DPad.Right: tiger moves right

if (s.DPad.Up != bp)      form.checkbox[12].Checked = false; //deplete checkbox 12
else { positionY += 0.01f; form.checkbox[12].Checked = true; } //DPad.Up: tiger moves up

if (s.DPad.Down != bp)   form.checkbox[13].Checked = false; //deplete checkbox 13
else { positionY -= 0.01f; form.checkbox[13].Checked = true; }
//DPad.Down: tiger moves down

Vector2 ts1 = s.ThumbSticks.Left;
//ts1 is just an acronym for the X/Y-coordinates that ThumbSticks.Left furnishes.
Vector2 ts2 = s.ThumbSticks.Right;
//ts2 is just an acronym for the X/Y-coordinates that ThumbSticks.Right furnishes.

rotationZ -= 0.1f * ts1.X; form.trackbar[0].Value=(int)( 50f*(ts1.X +1f) );
//tiger rotates around the Z-axis
rotationX -= 0.1f * ts1.Y; form.trackbar[1].Value=(int)( 50f*(ts1.Y +1f) );
//tiger rotates around the X-axis
rotationZ -= 0.1f * ts2.X; form.trackbar[2].Value=(int)( 50f*(ts2.X +1f) );
//tiger rotates around the Z-axis
rotationY -= 0.1f * ts2.Y; form.trackbar[3].Value=(int)( 50f*(ts2.Y +1f) );
//tiger rotates around the Y-axis
//The thumbstick values are between -1.0 and +1.0. Such increments produce too fast rotations.
Therefore we reduce them to 10%.
//The trackbars range from 0 to 100. In order to picture the thumbstick values we first shift them into
the range 0.0 to 2.0 and then multiply them by 50.

float tr1 = s.Triggers.Left ; form.trackbar[4].Value=(int)( 100f*tr1 );
GamePad.SetVibration(0,tr1,0f);
//Switch on the first forced feedback eccentric motor inside the controller.
float tr2 = s.Triggers.Right; form.trackbar[5].Value=(int)( 100f*tr2 );
GamePad.SetVibration(0,0f,tr2);
//Switch on the second forced feedback eccentric motor inside the controller.

base.Update(gameTime); //Call the base entity Update()-method which will check the players actions.

private function void reset() called from s.Buttons.Start and s.Buttons.Back
inside protected override void Update( gameTime gameTime )

positionX = positionY = positionZ = rotationX = rotationY = rotationZ = 0f;
//reset all positions and rotations to their initial value 0.
scaleX = scaleY = scaleZ = 1f; //reset all zooms to their initial value 1.

```

Render method `protected override void Draw(GameTime gameTime)`.

// This function is derived from a virtual function `protected virtual void Draw(GameTime gameTime)` of the parent class `Microsoft.Xna.Framework.Game`, which calls this function automatically as often as possible together with the `protected override void Update(GameTime gameTime)`-function above. Here is the right place to light the single effect of the tiger and to set its World-matrix.

```
g.GraphicsDevice.Clear( Color.DarkBlue ); //Erase any former content
```

```
effect.EnableDefaultLighting(); //Switch on some default ambient and directional light.
```

```
effect.World = Matrix.CreateScale( scaleX, scaleY, scaleZ ) *
                Matrix.CreateRotationX( modelRotation ) *
                Matrix.CreateRotationY( modelRotation ) *
                Matrix.CreateRotationZ( modelRotation ) *
                Matrix.CreateTranslation( positionX, positionY, positionZ );
```

//This statement moves the tiger. It replaces the former world transform matrix of `effect`

by a new one which is the result of a matrix multiplication of 5 single matrices.

Matrix 1 zooms the tiger, matrices 2 to 4 rotate it and matrix 5 shifts it. The result is a simultaneous zoom, rotation and translation.

```
model.Meshes[0].Draw(); //Render the tiger and then flip back-buffer to front-buffer to show the scene.
```

Windows Form Class `FileForm1.cs`.

// This file contains all necessary code to display a Windows Form with 25 controls: 14 CheckBoxes, 6 TrackBars and 6 Labels.

XNA-Windows are derived from `Microsoft.Xna.Framework.Game` which has no controls.

That is the reason why we open a second window derived from `System.Windows.Forms.Form`.

The window is started from inside the `Initialize()` event handler of `Game1`.

```
using System; //Home of the base class of all classes "System.Object".
using System.Drawing; //Home of the "Graphics"-class and its drawing methods.
using System.Windows.Forms; //Home of the "Form"-class.
```

```
public class Form1 : System.Windows.Forms.Form //Derive Form1 from
System.Windows.Forms.Form
public const Int32 nCheckBoxes = 14, nTrackBars = 6;
//Integers that never change their values indicating the numbers of CheckBoxes, TrackBars and Labels.
public CheckBox[] checkbox = new CheckBox[nCheckBoxes]; //Array of 14 CheckBoxes.
public TrackBar[] trackbar = new TrackBar[nTrackBars ]; //Array of 6 TrackBars.
public Label [] label = new Label [nTrackBars ]; //Array of 6 Labels.
```

end of global variables declaration and start of Form1() //Constructor

```
public Form1() //Constructor
BackColor = Color.White; //White background
Text = "Xbox 360 Controller Buttons"; //Title bar text
Int32 i; //Needed as index in 2 for-loops
```

```
for ( i=0; i < nCheckBoxes; i++ ) //Iterate through the 14 CheckBoxes
checkbox[i] = new CheckBox(); Controls.Add( checkbox[i] );
//Create and connect them to Form1
checkbox[i].TextAlign = ContentAlignment.MiddleCenter; //Center the text of the CheckBoxes
```

```

checkbox[ 0].Text = "A";
checkbox[ 1].Text = "B";
checkbox[ 2].Text = "X";
checkbox[ 3].Text = "Y";
checkbox[ 4].Text = "Left Shoulder";
checkbox[ 5].Text = "Right Shoulder";
checkbox[ 6].Text = "Start";
checkbox[ 7].Text = "Back";
checkbox[ 8].Text = "Left Stick";
checkbox[ 9].Text = "Right Stick";
checkbox[10].Text = "Left";
checkbox[11].Text = "Right";
checkbox[12].Text = "Up";
checkbox[13].Text = "Down";
// Texts of the CheckBox-stickers

for ( i=0; i < nTrackBars; i++ ) //Iterate through the 6 Trackbars and their Labels
trackbar[i] = new TrackBar(); Controls.Add( trackbar[i] );
//Create and connect them to Form1
label [i] = new Label(); Controls.Add( label[i] );
//Create and connect them to Form1
trackbar[i].AutoSize = false; //We want to set Minimum and Maximum.
trackbar[i].TickStyle = TickStyle.None; //We want no ticks underneath the Trackbars.
trackbar[i].Minimum = 0; //Left value
trackbar[i].Maximum = 100; //Right value
label [i].TextAlign = ContentAlignment.TopCenter; //Vertically on top, horizontally centered

label[0].Text = "X-Axis 1";
label[1].Text = "Y-Axis 1";
label[2].Text = "X-Axis 2";
label[3].Text = "Y-Axis 2";
label[4].Text = "Left Trigger"; label[5].Text = "Right Trigger";
// Texts of the Labels indicate the purposes of the TrackBars.

foreach ( Control c in Controls ) c.BackColor = Color.Gray; //All controls are grey.

StartPosition = FormStartPosition.Manual;
//Switch off any automatic positioning and sizing of Form1.
Function Initialize() inside public class Game1 will do that.

end of Form1() //Constructor and start of OnResize( EventArgs e )

protected override void OnResize( EventArgs e ) //Override the default event handler
OnResize() of Form1. //We need it to position and size the controls inside the client area of Form1.
Int32 w = ClientRectangle.Width; //Client width of Form1
Int32 h = ClientRectangle.Height / Controls.Count;
//Complete client height of Form1 divided by 25
Int32 i, top = 1; //Index variable and variable which shifts down the upper borders of the controls

for ( i=0; i < Controls.Count; i++ ) //Iterate through all controls.
Controls[i].Top = top; //Start at the highest position inside Form1.
Controls[i].Left = 2;
//Leave two pixels between the left margin of Form1 and the left borders of the controls. (That looks better.)
Controls[i].Width = w; //All controls fill the complete width of Form1.
Controls[i].Height = h - 2; //All controls have the same height = overall height / 25.
top += h; //Shift the next control down by one control height.

for ( i=0; i < nTrackBars; i++ ) trackbar[i].Height = h;
//TrackBars should have the same height as any other control.
By default they have a different height of their own.

end of OnResize( EventArgs e )

```