

# Course 3D\_XNA: 3D-Computer Graphics with XNA

## Chapter C5: Comments to Dice with Texture

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](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](http://msdn2.microsoft.com/en-us/library/microsoft.xna.framework.graphicsdevicemanager_members.aspx)

ContentManager content;

//ContentManager loads binary files from the content pipeline of the game.

It loads and prepares your graphic to be drawn, and will reload your graphic if the graphics device is reset (such as in the case of the game window being resized).

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

Matrix world, view, proj; //Three matrices for transformations of vertices.

The Matrix-class provides methods for performing standard matrix operations such as addition, subtraction, and multiplication, in addition to helper methods for creating scale, translation, and rotation matrices.

float positionX = 0.0f, positionY = 0.0f, positionZ = 0.0f; //initial position of the dice  
float scaleX = 1.0f, scaleY = 1.0f, scaleZ = 1.0f; //initial zoom of the dice  
float rotationX = 0.0f, rotationY = 0.0f, rotationZ = 0.0f; //initial rotations of the dice  
//These variables will change when the player presses the controller's buttons, sticks and DPads.

public Game1() //Constructor of Game1

g = new GraphicsDeviceManager( this ); } //Create an instance of GraphicsDeviceManager which we need below inside Initialize() to set some properties of our GraphicsDevice.

content = new ContentManager( Services ); //Constructor of ContentManager. See above.

```
Event handler protected override void Initialize() inside public class Game1
//Obligatory event handler of any XNA Game
```

```
g.PreferredBackBufferWidth = 400; //Allocate output buffer space on the graphics board
g.PreferredBackBufferHeight = 400; //Allocate output buffer space on the graphics board
g.ApplyChanges(); //Force the graphics board to rearrange its memory structure
```

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

```
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`.

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

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

```
Event handler protected override void UnloadContent() inside public class Game1
```

```
// This function is symmetrical to LoadContent()
```

```
content.Unload(); //Free the memory space occupied by the "dice" model.
```

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

// This function is derived from a virtual function `protected virtual void Update( 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 )`-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( PlayerIndex.One );
// Get the status of player's 1 controller. (PlayerIndex.One just represents value 0.)
```

```
if ( !s.IsConnected ) { Exit(); } //Stop the game if there is no Xbox 360 controller.
```

```
Vector2 ts = s.ThumbSticks.Left; //ts holds the X/Y-coordinates that ThumbSticks.Left furnishes.
rotationY += 0.05f * ts.X; //The dice rotates around the Y-axis
rotationX -= 0.05f * ts.Y; //The dice rotates around the X-axis
ts = s.ThumbSticks.Right; //ts holds the X/Y-coordinates that ThumbSticks.Right furnishes.
rotationY += 0.05f * ts.X; //The dice rotates around the Y-axis
positionZ -= 0.1f * ts.Y; //The dice rotates around the Z-axis
// The thumbstick values are between -1.0 and +1.0. Such increments produce too fast rotations.
Therefore we reduce them to 5%.
```

```
if ( s.DPad.Left == ButtonState.Pressed ) positionX -= 0.05f; //The dice moves left.
if ( s.DPad.Right == ButtonState.Pressed ) positionX += 0.05f; //The dice moves right.
if ( s.DPad.Up == ButtonState.Pressed ) positionY += 0.05f; //The dice moves up.
if ( s.DPad.Down == ButtonState.Pressed ) positionY -= 0.05f; //The dice moves down.
if ( s.Buttons.Y == ButtonState.Pressed ) scaleY *= 1.01f; //The dice grows 1% taller.
if ( s.Buttons.A == ButtonState.Pressed ) scaleY *= 0.99f; //The dice's height shrinks by 1%.
if ( s.Buttons.X == ButtonState.Pressed ) scaleX *= 0.99f; //The dice's width shrinks by 1%.
if ( s.Buttons.B == ButtonState.Pressed ) scaleX *= 1.01f; //The dice thickens by 1%.
```

```

if ( s.Buttons.Back == ButtonState.Pressed ) //Back button serves as reset button.
positionX = positionY = positionZ = 0f; //Reset all positions to their initial value 0.
scaleX = scaleY = scaleZ = 1f; //Reset all zooms to their initial value 1.
rotationX = rotationY = rotationZ = 0f; //Reset all rotations to their initial value 0.

world = Matrix.CreateScale( scaleX, scaleY, scaleZ ) * //zoom matrix
Matrix.CreateRotationX( rotationX ) * //This matrix rotates around the X-axis.
Matrix.CreateRotationY( rotationY ) * //This matrix rotates around the Y-axis.
Matrix.CreateRotationZ( rotationZ ) * //This matrix rotates around the Z-axis.
Matrix.CreateTranslation( positionX, positionY, positionZ ); //This matrix shifts.

view = Matrix.CreateLookAt( new Vector3(0f, 0f, 4f), Vector3.Zero, Vector3.Up );
//Sets the camera to position 4.0 in front of the display,
pointing from the players nose to the center of the game window. The Y-axis goes up.

proj = Matrix.CreatePerspectiveFieldOfView( MathHelper.Pi/4, 1f, 0.1f, 200f );
//Sets the projection of the 3D-model to the 2D-screen: viewing angle = 45 degrees; aspect ratio 1:1;
front clipping plane close at z=0.1; back clipping plane far away at z=200f.

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

```

```

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 all 6 effects of the dice and to set there transformation matrices.

```

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

foreach ( BasicEffect effect in model.Meshes[0].Effects )
//Iterate through all 6 effects of model dice. Dice has 6 effects because it has 6 surfaces each having
its own material and its own texture. See file dice.x.
effect.EnableDefaultLighting(); //Switch on some default ambient and directional light.
effect.World; //All 6 effects are transformed by the same World-matrix.
effect.View; //All 6 effects are transformed by the same View-matrix.
effect.Projection; //All 6 effects are transformed by the same Projection-matrix.
model.Meshes[0].Draw(); //Render the dice and then flip back-buffer to front-buffer to show the scene.

```