# Course 3D_XNA
# Chapter C1: Comments to the Triangle Project



Recommended for early beginners:
XNA Overview: http://msdn2.microsoft.com/en-us/library/bb200104.aspx
Video Training: http://msdn2.microsoft.com/en-us/xna/bb245766.aspx

## 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 functions to load objects.
See: http://msdn2.microsoft.com/en-us/library/microsoft.xna.framework.content.aspx
```
using Microsoft.Xna.Framework.Graphics; //Contains methods to access the GPU.
```
See: http://msdn2.microsoft.com/en-us/library/microsoft.xna.framework.graphics.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.

---

```
private 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-....framework.graphicsdevicemanager_members.aspx

---

```
private VertexBuffer vbuffer;
```
//`VertexBuffer vbuffer` will transport the array `VertexPositionColor[] v` to the graphics card.
See: http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.graphics.vertexbuffer.aspx

---

```
private BasicEffect effect;
```
//A `BasicEffect` describes the current settings of the GPU and it requires a set of world, view
and projection matrices, a vertex buffer and a vertex declaration.
See: http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.graphics.basiceffect.aspx

---

```
private EffectPass pass;
```
//An `EffectPass` defines a vertex- and a pixel-shader program that the GPU should execute while drawing an effect. See: http://creators.xna.com/en-US/article/shader_primer

---

```
private float rotAngle = 0f; //No rotation at start.
```

---

```
private Matrix rot       = Matrix.CreateRotationY( 0f );
private Matrix zoom      = Matrix.CreateScale( 0.5f );
private Matrix leftshift  = Matrix.CreateTranslation( new Vector3( -1f, 1f, 0f ) );
private Matrix rightshift = Matrix.CreateTranslation( new Vector3( 1f, 0f, 0f ) );
```
//Some predefined transform matrices for later use. See:
http://www.gamedev.net/reference/articles/article877.asp

---

```
private const Int32 nTriangles = 100; //We want to render one triangle 100 times.
```

```
private float[] dx = new float[nTriangles];
private float[] dy = new float[nTriangles];
private float[] dz = new float[nTriangles];
private float[] ax = new float[nTriangles];
private float[] ay = new float[nTriangles];
private float[] az = new float[nTriangles];
```
//Each of the 100 rendered triangles will be have its own dx,dy,dz-position in space and 3 individual ax,ay,az-angles of rotations around the 3 axes. We define just the space to store theses values but not the values yet.

---

```
private Random r = new Random(); //An object that delivers random numbers.
```

---

Constructor `public Game1()` inside `public class Game1`

```
Window.Title = "3DTriangleAnimation";
Window.AllowUserResizing = true; //The user is allowed to zoom the window at run time.
g = new GraphicsDeviceManager( this );
```
//Create an instance of `GraphicsDeviceManager`. See above.
```
g.IsFullScreen = false; //Let's everything happen inside a relatively small window.
g.PreferredBackBufferWidth  = 600; //Initial width
g.PreferredBackBufferHeight = 600; //Initial height
```

---

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

//Obligatory event handler of any XNA Game

```
VertexPositionColor[] v = new VertexPositionColor[3]; //space for 3 colored vertices
v[0].Position.X=-1f; v[0].Position.Y=-1f; v[0].Position.Z=0f; //lower left vertex
v[1].Position.X= 0f; v[1].Position.Y= 1f; v[1].Position.Z=0f; //upper vertex
v[2].Position.X= 1f; v[2].Position.Y=-1f; v[2].Position.Z=0f; //lower right vertex
v[0].Color = Color.DarkGoldenrod; //color of the lower left vertex
v[1].Color = Color.Cornsilk;      //color of the upper vertex
v[2].Color = Color.MediumOrchid;  //color of the lower right vertex
```

---

```
vbuffer = new VertexBuffer(g.GraphicsDevice, typeof(VertexPositionColor),
                           3, BufferUsage.WriteOnly);
```
//instance of a vertex buffer containing 3 colored vertices
```
vbuffer.SetData< VertexPositionColor >( v );
```
//Transport the array `v[]` from main memory to the graphics card and store it into the graphics card vertex input buffer. See: **VertexBuffer.SetData Method**

---

```
effect = new BasicEffect( g.GraphicsDevice, null );
```
//instance of an `BasicEffect` describing the current graphics card
```
effect.View = Matrix.CreateLookAt( new Vector3(0, 0, 3), Vector3.Zero, Vector3.Up );
```
//effect requires a view matrix. See: **How to: Rotate and Move a Camera** and **Matrix.CreateLookAt Method**
```
effect.Projection = Matrix.CreatePerspectiveFieldOfView( MathHelper.Pi/4, 1f, 1f, 10f );
```
//effect requires a perspective matrix. See: **Matrix.CreatePerspectiveFieldOfView Method**
```
effect.VertexColorEnabled = true;
```
//Advice the effect to interpolate the vertex colors over the triangle area.
```
pass = effect.CurrentTechnique.Passes[0];
```
//Tell the effect that there is just one pass.

---

```
for ( int i = 0; i < nTriangles; i++ )
```
//In order to expand a heterogeneous 3D-cloud of triangles
//  fill all dx,dy,dz with random values between -0.5 and +0.5
//and all ax,ay,az with random angles between 0 and 180/Pi = 57.3 degrees.
```
{ dx[i] = (float)r.NextDouble() - 0.5f; //random permanent translation dx
  dy[i] = (float)r.NextDouble() - 0.5f; //random permanent translation dy
  dz[i] = (float)r.NextDouble() - 0.5f; //random permanent translation dz
  ax[i] = (float)r.NextDouble(); //random initial pitch rotation angle
  ay[i] = (float)r.NextDouble(); //random initial yaw rotation angle
  az[i] = (float)r.NextDouble(); //random initial roll rotation angle
}
```

---

Event handler `protected override void Update( GameTime gameTime )` inside `public class Game1`

//Obligatory event handler of any XNA game. It is invoked once at start and whenever something happens with the window, the keyboard, the mouse or the game controller.

```
g.GraphicsDevice.Vertices[0].SetSource( vbuffer, 0,
VertexPositionColor.SizeInBytes );
```
//Reset the vertex shader start pointer to the first address of the vertex buffer.
```
g.GraphicsDevice.VertexDeclaration = new VertexDeclaration( g.GraphicsDevice,
VertexPositionColor.VertexElements );
```
//Reset the vertex shader offset to the `sizeof(VertexPositionColor`.
```
g.GraphicsDevice.RenderState.CullMode = CullMode.None;
```
//Tell the GPU to render all triangles even if they show their reverse sides.

---

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

---

```
effect.Begin();
```
//Initialize the GPU.
```
for ( int i = 0; i < nTriangles; i++ )
```
//Draw the sole and only triangle 100 times at different positions.
```
{ pass.Begin();
```
//Start the GPU shaders.
```
  rot = Matrix.CreateFromYawPitchRoll( ax[i] += 0.01f, ay[i] += 0.01f, az[i] += 0.01f );
```
  //Create a rotation transform matrix which rotates the triangle continuously around the x-,y- and z-axes.
```
  leftshift = Matrix.CreateTranslation( dx[i], dy[i], dz[i] );
```
  //Create a shift transform matrix which places the triangle to its random position.
```
  effect.World = zoom * rot * leftshift;
```
  //Concatenate the zoom-matrix with the rotation- and the shift-matrix to one single complex transformation.
```
  try { g.GraphicsDevice.DrawPrimitives( PrimitiveType.TriangleList, 0, 1 ); } catch {}
```
  //Draw the sole and only triangle. (This can go wrong at first start, when there is nothing in the vertex buffer.)
```
  pass.End();
```
//Stop the GPU shaders.
```
}
effect.End();
```
//Stop the GPU.

---