# 3D User Interfaces for Games and Virtual Reality

Lecture #4: XNA & Bespoke 3DUI

Spring 2009

Joseph J. LaViola Jr.

Special thanks to Paul Varcholik for developing Bespoke 3DUI and these slides.



# Introductions

- Paul Varcholik: pvarchol@ist.ucf.edu
- Technology Blog: www.bespokesoftware.org

# Software

- Visual Studio 2008 (Standard or Professional)
  or
- Visual C# Express 2008
- Windows Vista Service Pack 1; Windows XP Service Pack 3

- XNA Game Studio 3.0



# Online Resources

- XNA Developer Center
  http://msdn.microsoft.com/xna
- XNA Team Blog
  http://blogs.msdn.com/xna/
- XNA Creators Club
  http://creators.xna.com/

# What is XNA?

- Graphics and Game Development Framework
- Announced: Aug 2006
- 1.0: Dec 2006
- 1.0 Refresh: April 2007
- 2.0: Dec 2007
- 3.0: Oct 2008
- 2D and 3D
- Managed DirectX
- Windows and Xbox 360
- Content Pipeline
- XNA's Not Acronymed



# Why Use XNA?

- Extremely Comprehensive
- Free
- Easy to Use (though game programming is, in general, quite challenging)
- Development and Real-time Debugging on a Retail Xbox 360
- Casual Games
- Game Prototyping
- Access to the .NET Framework Class Libraries

# XNA

- Microsoft.Xna
  - .Framework
  - .Graphics
  - .Content
  - .Input
  - .Audio
- Microsoft.Xna.Framework Classes/Structs
  - Game
  - ContentManager
  - GraphicsDeviceManager
  - GameComponent
  - DrawableGameComponent



# XNA (cont.)

- Microsoft.Xna.Framework Classes/Structs
  - Vector2
  - Vector3
  - Point
  - Matrix
  - BoundingBox
  - BoundingSphere
  - Texture2D
  - SpriteFont

# XNA (cont.)

- Game Class
  - Initialize()
  - LoadContent()
  - UnloadContent()
  - Update()
  - Draw()
- Components
  - Separate out a generic/reusable class library



# Bespoke 3D UI Framework

- Organization:
  - Source Code
    - Framework
    - Samples
      - StereoscopicRendering
      - TrackIRTestbed
      - WiimoteTestbed
  - Executables
  - Documentation

# Bespoke 3D UI Framework

- Namespaces:
  - Bespoke.Common
    General Utilities (not game/XNA specific)
  - Bespoke.Games.Framework
    XNA utility libraries
  - Bespoke.Games.Framework.Content
    Custom XNA Content Processors



# Bespoke.Commom

- Assert (static)
- CommandLineParser
- Library (static)
- LogManager
- Node<T> / NodeCollection <T>
- ProgressIndicator
- XmlHelper

## Bespoke.Common

- .Data
- .LinearAlgebra
- .TrackIR – Requires OptiTrack software
- .Video – Uses DirectShow.NET (wrapper for unmanaged DirectShow)
- .Wiimote – Brian Peek's Wiimote Library from Coding4Fun.com



## Bespoke.Games.Framework.Content

- TerrainContentImporter
- TerrainContentProcessor
- TerrainDataWriter

These classes provide a content pipeline for using a heightmap for terrain. This is used in conjunction with the TerrainComponent.

# Bespoke.Games.Framework

- Actor/ActorList
- DynamicActor
- DynamicActorGroup
- FontManager
- FpsComponent
- GridComponent
- PostProcessor
- ScreenCapture
- CameraComponent
- ChaseCameraComponent
- StereoscopicChaseCameraComponent
- Sprite
- SpriteManager
- SkyBoxComponent
- SoundManager
- TerrainComponent



# Bespoke.Games.Framework

- ScreenManager
- GameScreen
- MenuScreen
- ScreenInputManager
- .UI
  - UIManager
  - UIControl
  - Button
  - XML Configuration
- .Input
  - KeyboardComponent
  - MouseComponent
  - GamepadComponent
  - TrackIRComponent
  - WiimoteComponent

# How To:

- Render a 2D Texture
- Draw Text
- View the Game's Framerate
- Collect Input
  - Keyboard
  - Mouse
  - Gamepad
- Initialize a 3D camera
- Draw a Reference Grid
- Render a 3D Model
- Play Sound
- Render a SkyBox
- Render Terrain



# How To:
## Render a 2D Texture

1. Include the texture in the Content project. Supported formats:
   - bmp, .dds, .dib, .hdr, .jpg, .pfm, .png, .ppm, and .tga
2. Initialize the SpriteManager class within the LoadContent() or Initialize() method:

   ```
   SpriteManager.Initialize(this);
   ```
3. Create data member to store texture:

   ```
   private Texture2D mTexture;
   ```
4. Load the texture in the LoadContent() method:

   ```
   mTexture = Content.Load<Texture2D>(@"Content\Textures\Skybox\back");
   ```
5. Render the texture in the Draw() method:

   ```
   SpriteManager.DrawTexture2D(mTexture, Vector2.Zero, Color.White);
   ```

# How To:
## Draw Text

1. Initialize the SpriteManager class as required before any calls can be made to the SpriteManager.
2. Add a call to SpriteManager.DrawString in the Draw() method:

```
SpriteManager.DrawString("Hello World", 40.0f, 40.0f, Color.White);
```

- Variety of overloads to the DrawString method:
  - Change the font
  - The blend color
  - Rotation
  - Sorting



# How To:
## View the Game's Framerate

1. Add the following statements to the Game-derived constructor, LoadContent(), or Initialize() method:

```
FpsComponent fps = new FpsComponent(this);
fps.Location = FpsComponent.ScreenLocation.TitleBar;
Components.Add(fps);
```

- FpsComponent display locations:
  - Titlebar
  - UpperLeft
  - UpperRight
  - LowerLeft
  - LowerRight

# How To:
## Collect Keyboard Input

1. Add a using statement for the Bespoke.Games.Framework.Input namespace;
2. (Optional) Create a data member to store the keyboard component:

```
private KeyboardComponent mKeyboardComponent;
```

3. Add the following statements to the Game-derived constructor, LoadConent(), or Initialize() method:

```
mKeyboardComponent = new KeyboardComponent(this);
Components.Add(mKeyboardComponent);
```

4. Add keyboard queries to the Update() method:

```
if (mKeyboardComponent.WasKeyPressedThisFrame(Keys.Escape))
 {
 Exit();
 }
```



# How To:
## Initialize a 3D Camera

1. Add the following statements to the Game-derived constructor, LoadContent(), or Initialize() method:

```
mCamera = new CameraComponent(this);
Services.AddService(typeof(ICamera), mCamera);
Components.Add(mCamera);

mCamera.KeyboardComponent = mKeyboardComponent;
 mCamera.GamePadComponent = mGamePadComponent;
 mCamera.Position = new Vector3(0.0f, 20.0f, 200.0f);
mCamera.Orientation = Vector3.Up;
mCamera.Direction = Vector3.Forward;
mCamera.LookAtOffset = Vector3.Forward;
mCamera.NearPlaneDistance = 1.0f;
mCamera.FarPlaneDistance = 100000.0f;
mCamera.FieldOfView = MathHelper.PiOver4;
mCamera.AspectRatio = (float)GraphicsDevice.PresentationParameters.BackBufferWidth /
GraphicsDevice.PresentationParameters.BackBufferHeight;
 mCamera.UpdateProjectionMatrix();
```

# How To:
## Draw a Reference Grid

1. Initialize a camera
2. Add the following statements to the Game-derived constructor, LoadContent(), or Initialize() method:

```
GridComponent grid = new GridComponent(this);
Components.Add(grid);
```

- You can modify the size (number of cells), scale (spacing between each line) and the color of the grid



# How To:
## Render a 3D Model

1. Include the model in your Content project (this is typically a sub-project within your Game project)
   - Supported Formats:
     - .fbx (Autodesk)
     - .x (DirectX Surface)
   - Be certain that associated textures reside in the proper locations.
2. Add the following statements to the Game-derived LoadContent(), or Initialize() method:

```
Model tankModel = Content.Load<Model>(@"Content\Models\Tank");
Actor tankActor = new DynamicActor(this, "Tank", Vector3.Zero, Vector3.Up, 0.05f, 1.0f,
 tankModel, mCamera);
tankActor.Initialize();
```

3. Add tankActor.Update() and tankActor.Draw() calls to the corresponding Game-derived Update() and Draw() methods.

# How To:
## Play Sound

1. Build your sound project using XACT.
2. Include your sound project (.xap) into your Content project.
3. Initialize the SoundManager static class in the LoadContent() or Initialize() method:

```
SoundManager.Initialize(@"Content\Audio\SoundProject.xgs", @"Content\Audio\Wave
Bank.xwb", @"Content\Audio\Sound Bank.xsb");
```

4. Play sounds with SoundManager.Play():
5. Call SoundManager.Update() within the main Update() loop.



# How To:
## Render a Skybox

1. Create your skybox textures (Terragen) and import them into your Content project (front, back, left, right, top)
2. Create a SkyBoxComponent data member.

```
private SkyBoxComponent mSkyBox;
```

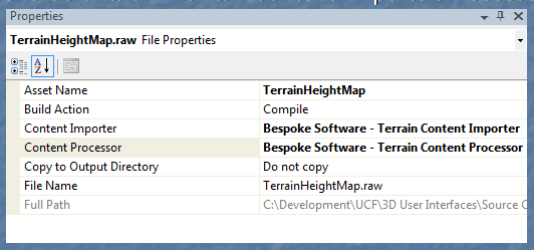
3. Add the following code to your Initialize() or LoadContent() method:

```
Texture2D front = Content.Load<Texture2D>(@"Content\Textures\SkyBox\front");
Texture2D back = Content.Load<Texture2D>(@"Content\Textures\SkyBox\back");
Texture2D left = Content.Load<Texture2D>(@"Content\Textures\SkyBox\left");
Texture2D right = Content.Load<Texture2D>(@"Content\Textures\SkyBox\right");
Texture2D top = Content.Load<Texture2D>(@"Content\Textures\SkyBox\top");
mSkyBox = new SkyBoxComponent(this, "SkyBox", front, back, left, right, top, 1000.0f,
mCamera);
mSkyBox.Initialize();
```

4. Call mSkyBox.Draw() from the main draw loop. Call this as the first object to be rendered.

# How To:
## Render Terrain

1. Reference Bespoke.Games.Framework.Content.dll from your Content project.
2. Import the heightmap (.raw) into your Content project and choose the Bespoke Software – Terrain Content Importer/Processor




3. Import the associated texture into your Content project.



# How To:
## Render Terrain (cont.)

4. Create a TerrainComponent data member.

```
private TerrainComponent mTerrain;
```

5. Add the following code to your Initialize() or LoadContent() method:

```
TerrainData terrainData = Content.Load<TerrainData>(@"Content\Other\TerrainHeightMap");
Texture2D terrainTexture = Content.Load<Texture2D>(@"Content\Textures\Terrain");
mTerrain = new TerrainComponent(this, terrainData, terrainTexture, 513, 513, 4.0f, 6000.0f,
                Color.White, -1000.0f, mCamera);
mTerrain.Initialize();
```

5. Call mTerrain.Draw() from the main draw loop.

# How To:
## Collect Wiimote Input

From Brian Peek's Wiimote.NET Article

**Getting Connected**

This will likely be the biggest sticking point. The Wiimote will not pair and communicate successfully with every Bluetooth device and stack on the planet. There's little I can do to help get you connected if the following steps do not work. Either it's going to work, or it isn't. Cross your fingers...

1. Start up your Bluetooth software and have it search for a device.
2. Hold down the 1 and 2 buttons on the Wiimote. You should see the LEDs at the bottom start flashing. **Do not let go of these buttons until this procedure is complete.**
3. The device should show up in the list of devices found as **Nintendo RVL-CNT-01**. If it's not there, start over and try again.
4. Click **Next** to move your way through the wizard. If at any point you are asked to enter a security code or PIN, leave the number blank or click **Skip**. Do not enter a number.
5. You may be asked which service to use from the Wiimote. Select the keyboard/mouse/HID service if prompted (you should only see one service available).
6. Finish the wizard.

That's it. The LEDs at the bottom should continue to flash and you should see the device listed in your list of connected Bluetooth devices. If you run the test application included with the source code and you see the numbers change, you are all set. If you don't see them change or you get an error, try the above again. If it continues to not function, you are likely stuck with an incompatible device or stack.



# How To:
## Collect Wiimote Input (cont.)

1. (Optional) Create a data member to store the Wiimote component:

```
private WiimoteComponent mWiimoteComponent;
```

2. Add the following statements to the Game-derived constructor, LoadConent(), or Initialize() method:

```
mWiimoteComponent = new WiimoteComponent(this);
Components.Add(mWiimoteComponent);
```

3. Add Wiimote queries to the Update() method:

```
if (mWiimoteComponent.CurrentState.ButtonState.B)
{
        rotationAmount.X = mWiimoteComponent.Y;
        rotationAmount.Y = -mWiimoteComponent.Z;
}
```

# Controls:
## CameraComponent

- Keyboard
  - WASD (forward, turn left, backward, turn right)
  - Up Arrow (turn up), Down Arrow (turn down)
- GamePad
  - Left Thumbstick (turn up, down, left, right)
  - Right Trigger (forward)
  - Left Trigger (reverse)



# Controls:
## StereoScopicChaseCameraComponent

- Keyboard
  - PageUp/PageDown (increase/descrease IPD)
  - End (toggle stereoscopic rendering)

# Controls:

## StereoScopicChaseCameraComponent

- Keyboard
  - PageUp/PageDown (increase/descrease IPD)
  - End (toggle stereoscopic rendering)