Postprocessing

Prof. Aaron Lanterman
School of Electrical and Computer Engineering
Georgia Institute of Technology

Bloom effect - before


Bloom effect - after


Motion blur in Valve’s Portal - roll

Motion blur in Valve’s Portal - falling


Giant warning

The code in these slides has not been tested.

There may be bugs and/or misconceptions.

Typical XNA setup code

```csharp
GraphicsDeviceManager graphics = new GraphicsDeviceManager(this);

ContentManager content = new ContentManager(Services);

GraphicsDevice device = graphics.GraphicsDevice;
```

Setup for postprocessing

```csharp
SpriteBatch mySpriteBatch;
RenderTarget2D myRenderTarget;
Texture2D beforeProc;
Effect ppEffect;

ppEffect =
    content.Load<Effect>("Content\Effects\CoolEffect");
```

Based on discussion on p. 277-281 of Chad Carter, “Microsoft XNA Unleashed,” 2008
Creating the rendertarget

```csharp
myRenderTarget = new RenderTarget2D(device,
    device.Viewport.Width,
    device.Viewport.Height,
    1, // number of mipmap levels
    device.DisplayMode.Format
    // a SurfaceFormat)

Vector2 offset = new Vector2(0, 1 / device.Viewport.Height);
```

Based on discussion on p. 277-281 of Chad Carter, "Microsoft XNA Unleashed," 2008

Creating the rendertarget (advanced)

```csharp
myRenderTarget = new RenderTarget2D(device,
    device.Viewport.Width,
    device.Viewport.Height,
    1, // number of mipmap levels
    device.DisplayMode.Format, // a SurfaceFormat
    service.PresentationParameters.MultiSampleType,
    service.PresentationParameters.MultiSampleQuality)
```

Based on discussion on p. 277-281 of Chad Carter, "Microsoft XNA Unleashed," 2008

### SurfaceFormat enumeration

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha8</td>
<td>(UnSigned format) 8-bit alpha only.</td>
</tr>
<tr>
<td>Bt233</td>
<td>(UnSigned format) 8-bit BGR texture format using 2 bits for blue, 3 bits for green, and 3 bits for red.</td>
</tr>
<tr>
<td>Bt34</td>
<td>(UnSigned format) 24-bit BGR pixel format with 8 bits per channel.</td>
</tr>
<tr>
<td>Bt52</td>
<td>(UnSigned format) 32-bit BGR pixel format, where 8 bits are reserved for each color.</td>
</tr>
<tr>
<td>Bt444</td>
<td>(UnSigned format) 16-bit BGR pixel format using 4 bits for each color.</td>
</tr>
<tr>
<td>Bt155</td>
<td>(UnSigned format) 16-bit BGR pixel format where 5 bits are reserved for each color.</td>
</tr>
<tr>
<td>Bt565</td>
<td>(UnSigned format) 16-bit BGR pixel format with 5 bits for blue, 6 bits for green, and 5 bits for red.</td>
</tr>
<tr>
<td>Bt1010102</td>
<td>(UnSigned format) 32-bit pixel format using 10 bits each for blue, green, and red, and 2 bits for alpha.</td>
</tr>
<tr>
<td>Bt23333</td>
<td>(UnSigned format) 16-bit BGRA format using 2 bits for blue, 3 bits each for red and green and 3 bits for alpha.</td>
</tr>
</tbody>
</table>


### Direct3D/XNA SurfaceFormat Conversions

<table>
<thead>
<tr>
<th>Direct3D Surface Format</th>
<th>SurfaceFormat equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flst32</td>
<td>D3DFMT_R32F</td>
</tr>
<tr>
<td>D3DFMT_G16R32F</td>
<td>Vector2</td>
</tr>
<tr>
<td>D3DFMT_A16D128P32F</td>
<td>Vector4</td>
</tr>
<tr>
<td>Flst16</td>
<td>D3DFMT_R16F</td>
</tr>
<tr>
<td>D3DFMT_G16R16F</td>
<td>HalfVector2</td>
</tr>
<tr>
<td>D3DFMT_A16B16G16</td>
<td>HalfVector4</td>
</tr>
</tbody>
</table>

#### Unsigned Normalized

- 64 bit: D3DFMT_A16H16G16              | RGBA64 |
- 32 bit: D3DFMT_A16R5G5B5            | Color  |
- D3DFMT_X4R8G8B8                 |Argb32  |
- D3DFMT_A8R8G8B8                 |Argb32  |
- D3DFMT_X8R8G8B8                 |Argb32  |
- D3DFMT_A2R10G10B10            |Argb10102 |

Rendering the preprocessed scene

```csharp
device.SetRenderTarget(0, myRenderTarget);
// On Xbox 360, first argument must be set to zero
// since you only can set one Render Target on the 360
// PUT CODE TO DRAW STUFF HERE

 device.ResolveRenderTarget(0);
```


Setting up the postprocessing effect

```csharp
myEffect.CurrentTechnique = effect.Techniques["BlurEffect"];
myEffect.Parameters["offset"].SetValue(offset);
```

Based on discussion on p. 277-281 of Chad Carter, “Microsoft XNA Unleashed,” 2008

Drawing the processed scene

```csharp
device.Clear(Color.Black);
myEffect.Begin();
mySpriteBatch.Begin(SpriteBlendMode.None, SpriteSortMode.Immediate, SpriteStateMode.None);
EffectPass pass = effect.CurrentTechnique.Passes[0].Begin();
mySpriteBatch.Draw(beforeProc, Vector2.Zero, Color.White);
pass.End();
mySpriteBatch.End();
myEffect.End();
```

Based on discussion on p. 277-281 of Chad Carter, “Microsoft XNA Unleashed,” 2008

Just need a pixel shader

```csharp
// CoolEffect.fx
sampler textureSampler;
float2 offset;

float4 threeWayBlurPS(texCoord : TEXCOORD0) : COLOR0
{
    float4 color =
        (tex2D(textureSampler, texCoord) + tex2D(textureSampler, texCoord + offset) + tex2D(textureSampler, texCoord - offset)) / 3;
    return color;
}

technique BlurEffect
{
    pass P0 {
        PixelShader = compile ps_2_0 threeWayBlurPS();
    }
}
```

Based on discussion on p. 277-281 of Chad Carter, “Microsoft XNA Unleashed,” 2008
Just need a pixel shader

```csharp
sampler textureSampler

is sort of implicitly

sampler textureSampler : register(S0);

mySpriteBatch.Draw(beforeProc, Vector.Zero, Color.White);

was sort of doing this somewhere:

device.Textures[0] = beforeProc;
```

Multiple textures

- In your C# code:
  ```csharp
graphics.GraphicsDevice.Textures[0] = firstTexture;
```
- In your shader code:
  ```csharp
  sampler firstSampler : register(s0);
sampler secondSampler : register(s1);
  ```

Rendertarget semantics on Windows

- If not multisampling, a single area of video memory can be used for rendering or as a texture
  - In XNA 1.0:
    - Contents of rendertarget not lost in XNA 1.0 on Windows
    - Resolve is a no-op
    - Rendertarget cleared in XNA 2.0 to emulate Xbox 360 behavior
- If multisampling, need large area to render into and small area to copy into
  - Resolve copies, downsampling as it goes
  - Contents of both buffers not lost (in XNA 1.0)

Rendertarget semantics on Xbox 360

- Xenos GPU renders into only one physical rendertarget: 10 MB eDRAM
  - Cannot texture from eDRAM
  - Cannot render into main 512M RAM
- Must "resolve" to copy rendering in eDRAM back to main memory
  - Hardware designed to make this fast
  - Note from Shawn: "It is less obvious why the resolve call needs to clear the special memory, but apparently there is a performance gain from doing this: I don’t pretend to understand why but I’m not going to complain as long as this keeps my Xbox running as fast as it does!"

From Shawn Hargreaves, "Rendertarget changes in XNA Game Studio 2.0,”
rendertarget-changes-in-xna-game-studio-2-0.aspx

From Shawn Hargreaves, "XNA rendertarget semantics,”