ECE4893A/CS4803MPG: Multicore and GPU Programming for Video Games

Postprocessing in XNA

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

Giant warning

The code in these slides has not been tested.

There may be bugs and/or misconceptions.

Typical XNA setup code

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

ContentManager content =
  new ContentManager(Services);

GraphicsDevice device =
  graphics.GraphicsDevice;
```

Setup for postprocessing

```
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

```
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)

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

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

SurfaceFormat enumeration

Member name	Description
Alpha8	(Unsigned format) 8-bit alpha only.
Bgr233	(Unsigned format) 8-bit BGR texture format using 2 bits for blue, 3 bits for green, and 3 bits for red.
Bgr24	(Unsigned format) 24-bit BGR pixel format with 8 bits per channel.
Bgr32	(Unsigned format) 32-bit BGR pixel format, where 8 bits are reserved for each color.
Bgr444	(Unsigned format) 16-bit BGR pixel format using 4 bits for each color.
Bgr555	(Unsigned format) 16-bit BGR pixel format where 5 bits are reserved for each color.
Bgr565	(Unsigned format) 16-bit BGR pixel format with 5 bits for blue, 6 bits for green, and 5 bits for red.
Bgra1010102	(Unsigned format) 32-bit pixel format using 10 bits each for blue, green, and red, and 2 bits for alpha.
Bgra2338	(Unsigned format) 16-bit BGRA format using 2 bits for blue, 3 bits each for red and green, and 8 bits for alpha.

Screenshot from msdn2.microsoft.com/en-us/library/microsoft.xna.framework.graphics.surfaceformat.aspx

Direct3D/XNA SurfaceFormat Conversions

	Direct3D Surface Format	SurfaceFormat equivalent
Floating Point		
Float32	D3DFMT_R32F	Single
	D3DFMT_G32R32F	Vector2
	D3DFMT_A32B32G32R32F	Vector4
Float16	D3DFMT_R16F	HalfSingle
	D3DFMT_G16R16F	HalfVector2
	D3DFMT_A16B16G16R16F	HalfVector4
Unsigned Normalized		
64 bpp	D3DFMT_A16B16G16R16	Rgba64
32 bpp	D3DFMT_A8R8G8B8	Color
	D3DFMT_X8R8G8B8	Bgr32
	D3DFMT_A8B8G8R8	Rgba32
	D3DFMT_X8B8G8R8	Rgb32
	D3DFMT_A2R10G10B10	Bgra1010102

Screenshot from msdn2.microsoft.com/en-us/library/microsoft.xna.framework.graphics.surfaceformat.aspx

Rendering the preprocessed scene

```
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);
// needed on Xbox 360 to copy eDRAM contents to main RAM

beforeProc = myRenderTarget.GetTexture();

// Set render target to the usual backbuffer
device.SetRenderTarget(0, null);
```

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

Georgia Institute of Technology

9

Setting up the postprocessing effect

```
// Next sort of line only seems necessary if you have
// more than one technique
myEffect.CurrentTechnique = effect.Techniques["BlurEffect"];

myEffect.Parameters["offset"].SetValue(offset);
```

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

Georgia Institute of Technology

10

Drawing the processed scene

```
device.Clear(Color.Black);
myEffect.Begin();
mySpriteBatch.Begin(SpriteBlendMode.None,
SpriteSortMode.Immediate, SpriteStateMode.None);
EffectPass pass = effect.CurrentTechnique.Passes[0]
pass.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

Georgia Institute of Technology

11

Just need a pixel shader

```
// 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

Georgia Institute of Technology

12

Just need a pixel shader

```
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;
```

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

Multiple textures

- In your C# code:

```
graphics.GraphicsDevice.Textures[0] = firstTexture;
graphics.GraphicsDevice.Textures[1] = secondTexture;
```

- In your shader code:

```
sampler firstSampler : register(s0);
sampler secondSampler : register(s1);
```

From msdn2.microsoft.com/en-us/library/microsoft.xna.framework.graphics.graphicsdevice.textures.aspx

Rendertarget semantics on Windows

- If not multisampling, a single area of video memory can be used for rendering or as a texture
  - Resolve is a no-op
  - Contents of rendertarget not lost
- 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

From Shawn Hargreaves, "XNA rendertarget semantics," blogs.msdn.com/shawnhar/archive/2007/02/04/xna-rendertarget-semantics.aspx

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, "XNA rendertarget semantics," blogs.msdn.com/shawnhar/archive/2007/02/04/xna-rendertarget-semantics.aspx

Shawn Hargraves' chart

	Rendertarget and texture share video memory (resolve is a no-op)	*Changing rendertarget destroys existing contents*	*Resolve destroys existing contents*
Windows (not multisampled)	yes	no	no
Windows (multisampled)	no	no	no
Xbox	no	yes	yes

Shawn's advice: "If you play it safe by always assuming your buffer contents will be lost when you call SetRenderTarget or Resolve, that code will work consistently on all platforms."

From Shawn Hargreaves, "XNA rendertarget semantics," blogs.msdn.com/shawnhar/archive/2007/02/04/xna-rendertarget-semantics.aspx

Georgia Institute of Technology

17