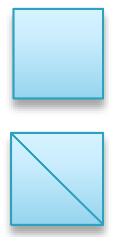
2D Rendering with SpriteBatch

2D Rendering

- Common uses
  - User interface
  - 2D games
- How do we render sprites?
  - Sprite are generally rectangular
  - Rectangles = 2 triangles



- Can use rendering pipeline for 2D rendering

Make our own sprite system?

- You could!

- Sprites are all the same shape: **Rectangles**
  - All sprites can share the exact same mesh
  - Start with a simple 1x1 rectangle
  - Vertex positions in the range (0,0) to (1,1)
  - Very simple to scale up and down

- Change size with world matrix
  - In addition to position and rotation

Efficiency?

- A separate draw call per sprite is inefficient
  - Although good enough for a simple UI

- Could use a dynamic buffer
  - Similar to particle systems
  - Copy vertices of similar sprites into buffer
  - Draw entire buffer once

- Hmmm…seems like a lot of work

Do we have to make our own?

- Nope!
- DirectX Toolkit comes with a **SpriteBatch** class

SpriteBatch

SpriteBatch

- Part of DirectX Toolkit
- Based on SpriteBatch from XNA/MonoGame
- Simplifies 2D rendering
  - You just provide a texture & rectangle
  - Batches similar sprites together for efficiency
  - Hence the name
- Can also draw arbitrary text

Using SpriteBatch

Initializing SpriteBatch

- Header: “SpriteBatch.h”

- Create a single SpriteBatch* object
  - For your whole project

```
// Requires the context
spriteBatch = new SpriteBatch(context);
```

Basic SpriteBatch usage

- Begin a batch
  - Can customize options as necessary
- Draw all of your sprites
- End the batch
  - This is when drawing actually occurs
  - Fewer batches is better!
  - Exactly one is best

Drawing with SpriteBatch

- Basic drawing steps:
  - `spriteBatch->Begin();`
  - `spriteBatch->Draw(…); // Do this many times`
  - `spriteBatch->Draw(…);`
  - `spriteBatch->End();`
- Basic Draw() call requires:
  - Texture (shader resource view)
  - Rectangle (defined in pixel coords)
- Draw() can also rotate, scale, tint, etc.

Important: Cleanup after drawing

- Begin() changes several render states
  - Blend mode, rasterizer state, etc.
  - End() does **NOT** restore them!

- Do it yourself before the next frame!

```
// Reset states that may be changed by sprite batch!
context->OMSetBlendState(0, 0, 0xFFFFFFFF);
context->RSSetState(0);
context->OMSetDepthStencilState(0, 0);
```

Drawing Text with SpriteFont

What about text?

- Drawn much the same way
  - Each character is a small rectangle
  - Textured with an image of that character
  - Requires an image with all text characters

- DirectX Toolkit has a SpriteFont system
  - Works in conjunction with SpriteBatch
  - Performs steps outlined above
  - Requires SpriteFont assets

SpriteFonts

- Special asset that contains:
  - An image of characters at a particular size
  - Info mapping characters to uv coordinates

- Need SpriteFonts for font/size combinations
  - Arial 12
  - Arial 14
  - Times New Roman 11
  - Etc.

- Note: SpriteFonts do not contain EVERY character – just a subset

Creating SpriteFonts

- `MakeSpriteFont.exe`
  - Command line utility
  - Can generate SpriteFont assets from system fonts

- Not included in NuGet package
  - Part of DXTK git repo
  - https://github.com/microsoft/DirectXTK/tree/master/MakeSpriteFont

- I've included a copy on MyCourses

MakeSpriteFont utility usage

- Open command prompt
- Go to folder containing MakeSpriteFont.exe
- Run the utility

- Required parameters:
  - `MakeSpriteFont.exe` *`fontname outputfile`*

- To print help info, run without parameters:
  - `MakeSpriteFont.exe`

MakeSpriteFont examples

- Arial at default size
  - `MakeSpriteFont.exe Arial Arial.spritefont`
- Arial at size 8
  - `MakeSpriteFont.exe Arial /FontSize:8 Arial8.spritefont`
- Times New Roman at default size
  - `MakeSpriteFont.exe "Times New Roman" TNR.spritefont`

Loading SpriteFont assets

- Create a SpriteFont object for each font

```
spriteFont = new SpriteFont(
    device,
    L"Fonts/Arial.spritefont");
```

- Don’t forget to delete in destructor

Drawing static text

- Must happen between Begin()/End()

```
spriteBatch->Begin();
spriteBatch->Draw(…);

font->DrawString(       // Font to use
   spriteBatch,         // Current batch
   "This is some cool text, yo", // Text
   XMFLOAT2(10, 120)); // Location

spriteBatch->End();
```

Drawing dynamic text

```
spriteBatch->Begin();

std::string dynamicText = "Label: " +
   std::to_string(numberVar);

font->DrawString(         // Font to use
   spriteBatch,           // Current batch
   dynamicText.c_str(),   // Text
   XMFLOAT2(10, 120));    // Location

spriteBatch->End();
```

Other SpriteFont methods

- SpriteFonts have several helper methods
- ContainsCharacter – Does this sprite font have the specified character?
- GetLineSpacing – Height of a line in this font
- MeasureString – How many pixels will the specified string take up if drawn?

References

SpriteBatch & SpriteFont references

- SpriteBatch reference
- SpriteFont reference
- MakeSpriteFont reference