CS 5410

Intro to 2D MonoGame Rendering

MonoGame 2D Basics

- MonoGame provides both 2D and 3D rendering; we focus on 2D rendering in this class
  - Behind the scenes, MonoGame uses either DirectX or OpenGL for rendering
- Features
  - Rectangles
  - Polygons
  - Textures
  - Text/Fonts
  - Effects (this is where the real power comes from)
    - Use this to draw polygons
  - Transformations (e.g., rotations, scaling)
  - Render to (non-visible) surface

Defining the Window

- MonoGame creates a window in which to render. You have some control over this…
  - Set the window size
  - Full screen or windowed
  - Many other things...
- In the Initialize method, specify the settings you want
  - `GraphicsDeviceManager.PreferredBackBufferWidth`
  - `GraphicsDeviceManager.PreferredBackBufferHeight`
  - `GraphicsDeviceManager.IsFullScreen`
  - `GraphicsDeviceManager.ApplyChanges()`

```
m_graphics.PreferredBackBufferWidth = 1920;
m_graphics.PreferredBackBufferHeight = 1080;
m_graphics.ApplyChanges();
```

Defining the Window

- Have ability to enumerate the different display modes…

```
foreach (DisplayMode mode in m_graphics.GraphicsDevice.Adapter.SupportedDisplayModes)
{
    System.Console.WriteLine("({0}, {1}", mode.Width, mode.Height);
}
```

Coordinate System

- Upper Left: (0, 0)
- Lower Right: (width – 1, height – 1)
- Increasing Y moves down
- Increasing X moves right

- Note
  - This is not typically what you are used to in a Cartesian coordinate system where lower left is (0, 0)

Clearing The Rendering Buffer

- Once something is drawn to the rendering buffer, it persists until cleared.
- Use the GraphicsDevice to clear the rendering buffer
  - `GraphicsDevice.Clear(Color.CornflowBlue);` // can use any color
  - This is performed as the first step in the `Draw` method
- Why `CornflowerBlue` instead of `Black`?
  - When rendering goes back, typically it renders in black
  - Want a background that isn't black, so you can know if you at least did something, even if it wasn't right
- When not debugging, clear it to `Black`, or whatever is appropriate for your game

Drawing 2D Things

- MonoGame uses something called a `SpriteBatch` to group related 2D drawing calls
  - It is created during the `LoadContent()` initialization
- During the `Draw()` method, it looks like
  - `m_spriteBatch.Begin();`
  - . . . draw various things . . .
  - `m_spriteBatch.End();`
- You can create more than one `SpriteBatch` instance and there are reasons to do so, but not yet in this class
  - You can also use the same `SpriteBatch` instance with multiple `.Begin`/`.End` calls

Drawing a Rectangle

- MonoGame doesn’t support drawing shapes (like rectangles) directly. Instead, you draw rectangles that have a texture (image) applied to them.
- Things you need
  - The location and dimensions of the rectangle: `Rectangle`
  - A texture: `Texture`
- Define a rectangle
  - `Rectangle myBox = new Rectangle(100, 100, 400, 400);`
- Define and load a texture
  - `Texture2D myTexture = this.Content.Load<Texture2D>(“my-texture”);`
- Draw the texture
  - `m_spriteBatch.Draw(myTexture, myBox, Color.White);`

Drawing a Rectangle - Code

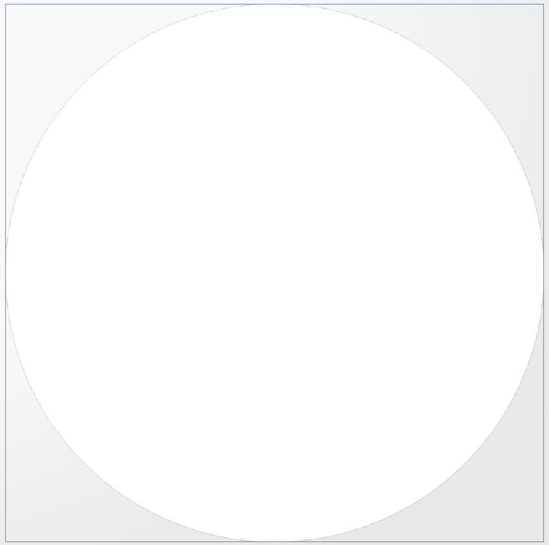
Must first add the "square" texture using the MGCB editor

```
protected override void LoadContent()
{
    m_spriteBatch = new SpriteBatch(GraphicsDevice);
    m_myBox = new Rectangle(100, 100, 400, 400);
    m_myTexture = this.Content.Load<Texture2D>("square");
}
```

```
protected override void Draw(GameTime gameTime)
{
    m_spriteBatch.Begin();

    m_spriteBatch.Draw(m_myTexture, m_myBox, Color.White);

    m_spriteBatch.End();

    base.Draw(gameTime);
}
```

Drawing a Circle

- (again) MonoGame doesn't support drawing shapes like circles. Instead, you draw rectangles that have a texture (image) applied to them.
  - There IS a way to draw polygons, but I'll cover that later
- To draw a circle, it is the same as drawing a rectangle, but the texture you create is a circle
  - Color the texture white, then when rendering, set the color then
  - Any part of the texture outside the circle must use transparency
  - Then drawing is the same as a rectangle



Drawing a Circle - Code

Must first add the "square" texture using the MGCB editor

```
protected override void LoadContent()
{
    m_spriteBatch = new SpriteBatch(GraphicsDevice);
    m_myCircle = new Rectangle(100, 100, 400, 400);
    m_myTexture = this.Content.Load<Texture2D>("circle");
}
```

```
protected override void Draw(GameTime gameTime)
{
    m_spriteBatch.Begin();

    m_spriteBatch.Draw(m_myTexture, m_myCircle, Color.Blue);

    m_spriteBatch.End();

    base.Draw(gameTime);
}
```

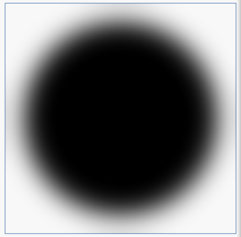
Rotating Shapes

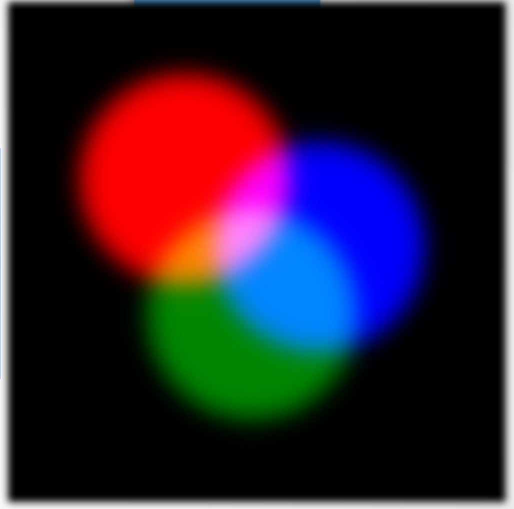
- For each object
  - maintain its center
  - maintain a rotation (radians)
- When rendering, use an overload of the `SpriteBatch.Draw` method where rotation is specified

```
m_spriteBatch.Draw(
    m_myTexture,
    m_myBox,
    null,   // Drawing the whole texture, not a part
    Color.Blue,
    m_rotationBox,
    new Vector2(m_myTexture.Width / 2, m_myTexture.Height / 2),
    SpriteEffects.None,
    0);
```

Blending

- Want to blend transparent/translucent parts of images
- A few simple steps
  - Author source textures with transparent/translucent pixels
  - Specify blending with the `SpriteBatch.Begin`
  - Draw like normal





```
//m_spriteBatch.Begin(SpriteSortMode.Immediate, BlendState.Additive);
m_spriteBatch.Begin(SpriteSortMode.Deferred, BlendState.Additive);

m_spriteBatch.Draw(m_texCircleBlur, m_rectCircleRed, Color.Red);
m_spriteBatch.Draw(m_texCircleBlur, m_rectCircleGreen, Color.Green);
m_spriteBatch.Draw(m_texCircleBlur, m_rectCircleBlue, Color.Blue);

m_spriteBatch.End();
```