Splatting 2D blood in games built with MonoGame

This article explains how to build a very simple 2D texture mapping system for games built with MonoGame

09 Jun 2013

Introduction

If you develop a 2D action game there is likely to be tons of shooting, sword fighting, maiming and killing: if you want it to be properly dark and gory you'll need to be able to add buckets of blood to your scenes, and you will want this blood to stay in the level for as long as possible. That's exactly the same situation we (me and my team-mates) are now in - while developing an engine for our new game we came to the point that we need the blood splatters to stay on the level.

Note: Don't get me wrong. We are not blood hungry monsters at PixelsFURY. It's just when you build a game of some genre you have to obey the rules of this genre. And we are actually building a gory 2D action game for Windows Phone 8.

This seems to be a good job for decals. In computer graphics a decal is just a texture projection on geometry. Under geometry you can assume the single polygon, the surface (bunch of polygons) or entire model (3d mesh). This will work for 3D as discussed in this article. The rest of the article explains how you can do the same sort of thing in 2D for drawing your sprites.

Complete scene rendered using decals. Moody, isn't it? :)

Decals in 2D

Drawing a texture map (decal) in 2D is pretty simple, and you will be reminded of the technique for deferred 2D lighting. The process is:

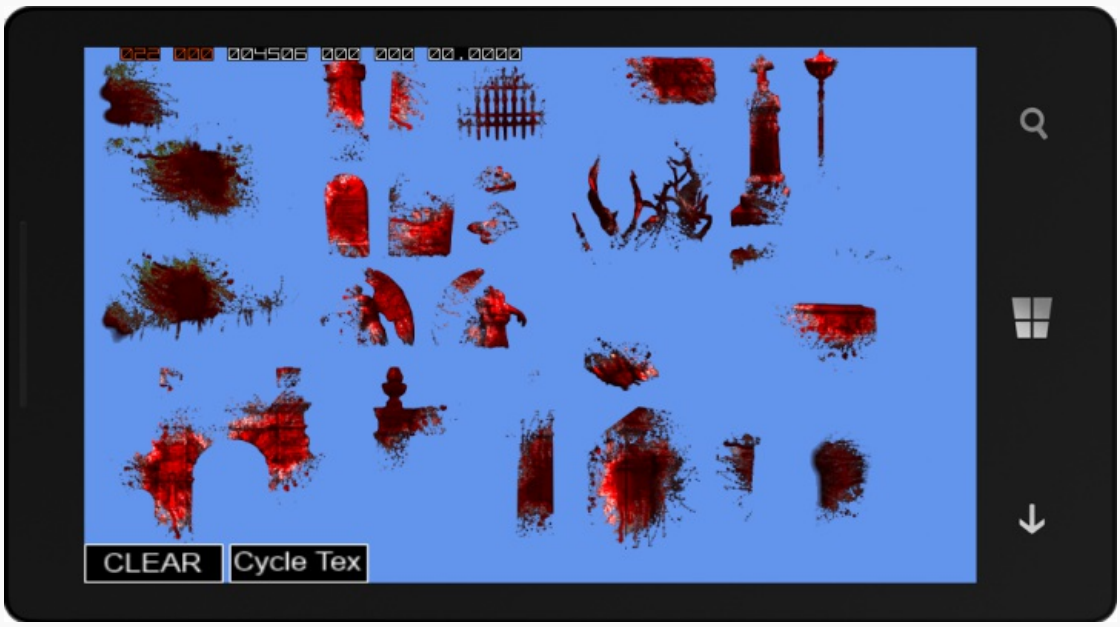
1. Draw your scene to a render target (or only those elements of the scene that can be splattered with blood or any other substance)
2. Draw the actual blood splatter to another render target (if there are really many of them you will have completely filled render target)
3. Track all positions of your splatters for the entire level
4. Combine the two previous render targets with a pixel shader to a final decals render target
5. Combine the scene and the decals render target using alpha blending

Next when our strategy is layered down we will need a shader to do the computation for us. The output of the shader will give the result with only those pixels (or texels if you mind) filled that have passed the alpha test.

The gallery below shows our original scene and the final render target produced by the shader:

(C) Copyright Nokia 2014. All rights reserved.

Scene render target contents



Final blood decals target contents

The Shader

To start learning about shader compilation and loading please refer to MonoGame on Windows Phone 8 Post-Processing Your Game.

Our shader will be pretty simple. It will take two textures and a float value as parameters.

```
float destinationWeight; // comes from C# code
sampler ColorMap : register(s0); // this is detected automatically as we draw with it
sampler BloodMap : register(s1); // comes from C# code

float4 Blender(float4 pos: POSITION, float4 col: COLOR, float2 coords: TEXCOORD0) :
COLOR0
{
 float4 baseTex = tex2D(ColorMap, coords);
 float4 overlayTex = tex2D(BloodMap, coords);
```


(C) Copyright Nokia 2014. All rights reserved.

```
 float4 overlay = float4(overlayTex.rgb * destinationWeight, overlayTex.a);

 // if our base texture at this pixel has alpha value less than 0.1
 // we will simply discard the same pixels from the overlay texture
 // with setting all of its color chanels to zero
 if (baseTex.a < 0.1)
 {
  overlay.a = 0;
  overlay.rgb = 0;
 }

 float4 final = baseTex * overlay * 2;
 final = saturate(final);

 return final;
}

technique T1
{
 pass P0
 {
  PixelShader = compile ps_4_0_level_9_3 Blender();
 }
}
```

Keeping Track Of Decals

The first idea was to make a decal the same as a particle in our engine. However there was already an overhead in memory when a lot of particles are displayed. So we decided to keep it simple and we went with List of `Vector4` type. The **X** and **Y** are the location coordinates, the **Z** component represents rotation and the **W** component is responsible for the decal texture index.

Local variables of `Game1` class

```
int width, height; // screen size

GraphicsDeviceManager _graphics;
SpriteBatch spriteBatch;

RenderTarget2D bloodTarget;
RenderTarget2D maptarget;
RenderTarget2D finalDecalsTarget;

RenderTarget2D finalImageTarget; // this will handle screen orientation

Vector2 halfScreen; // position of finalImageTarget
Vector2 invHalfScreen; // origin of finalImagetarget

Texture2D[] splaters;
Texture2D map;

Vector2 splatPoint = Vector2.Zero; // cache for blood decal splat position
Vector2 splatOrigin = Vector2.Zero; // cache for blood decal splat origin
List<Vector4> pointsToSplat; // the actual decals list

byte[] shaderCode; // this will be need to load the shader
```


(C) Copyright Nokia 2014. All rights reserved.

```
Effect blender;
private int index = 0;

private TouchLocation prevTouch;
Vector2 touchLocation = new Vector2(-1, -1);
private Texture2D buttonClearTex;
private Texture2D buttonCycleTex;
bool touchDown = false;

Rectangle sRect = new Rectangle(0, 0, 75, 25);
Rectangle clearRect = new Rectangle(0, 480 - 35, 125, 35); // rectangle to test touch on
"Clear" button
Rectangle cycleRect = new Rectangle(130, 480 - 35, 125, 35);// rectangle to test touch
on "Cycle" button
```

Lots of variables here. I agree. However we need all of them.

The Draw Function

```
protected override void Draw(GameTime gameTime)
{
 GraphicsDevice.SetRenderTarget(maptarget);
 GraphicsDevice.Clear(Color.Transparent);

 spriteBatch.Begin(SpriteSortMode.Deferred, BlendState.AlphaBlend);
 spriteBatch.Draw(map, Vector2.Zero, Color.White);
 spriteBatch.End();

 GraphicsDevice.SetRenderTarget(bloodTarget);
 GraphicsDevice.Clear(Color.Transparent);

 spriteBatch.Begin(SpriteSortMode.Deferred, BlendState.AlphaBlend);

 for (int i = 0; i < pointsToSplat.Count; i++)
 {
  splatPoint.X = pointsToSplat[i].X;
  splatPoint.Y = pointsToSplat[i].Y;

  spriteBatch.Draw(splaters[(int)pointsToSplat[i].W], splatPoint, null, Color.Gray,
pointsToSplat[i].Z, splatOrigin, 1f, SpriteEffects.None, 0f);
 }

 spriteBatch.End();

 GraphicsDevice.SetRenderTarget(finalDecalsTarget);
 GraphicsDevice.Clear(Color.Transparent);

 spriteBatch.Begin(SpriteSortMode.Immediate, BlendState.AlphaBlend);

 blender.Parameters["BloodMap"].SetValue(bloodTarget);
 blender.Parameters["destinationWeight"].SetValue(2.6f);

 blender.Techniques[0].Passes[0].Apply();
 spriteBatch.Draw(maptarget, Vector2.Zero, Color.White);
 spriteBatch.End();
 GraphicsDevice.Textures[1] = null;
```


(C) Copyright Nokia 2014. All rights reserved.

```
 GraphicsDevice.SetRenderTarget(finalImageTarget);
 GraphicsDevice.Clear(Color.CornflowerBlue);

 // when done rendering to the bloodTarget we will simply draw it over the main scene
 spriteBatch.Begin(SpriteSortMode.Deferred, BlendState.AlphaBlend);
 spriteBatch.Draw(maptarget, Vector2.Zero, Color.White);
 spriteBatch.Draw(finalDecalsTarget, Vector2.Zero, Color.White);

 DrawButton();
 DrawCycleButton();

 spriteBatch.End();

 GraphicsDevice.SetRenderTarget(null);
 GraphicsDevice.Clear(Color.CornflowerBlue);

 spriteBatch.Begin(SpriteSortMode.Deferred, BlendState.AlphaBlend);
 spriteBatch.Draw(finalImageTarget, invHalfScreen, null, Color.White,
MathHelper.PiOver2, halfScreen, 1f, SpriteEffects.None, 0f);
 spriteBatch.End();

 base.Draw(gameTime);
}
```

The key point in `Draw()` function is that we have to clear decal and map render targets with transparent colour as we will test pixels against their alpha value.

The Update Function

```
protected override void Update(GameTime gameTime)
{
 if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
  Exit();

 touchDown = false;
 TouchCollection touches = TouchPanel.GetState();

 if (touches.Count > 0)
 {
  TouchLocation curTouch = touches[0];

  bool prevAvailable = curTouch.TryGetPreviousLocation(out prevTouch);
  if (curTouch.State == TouchLocationState.Pressed &&
   !prevAvailable)
  {
   touchDown = true;
   touchLocation = new Vector2(curTouch.Position.Y, height - curTouch.Position.X);
   bool touchClear = UpdateButton((int)touchLocation.X, (int)touchLocation.Y,
touchDown);
   bool touchCycle = UpdateCycleButton((int)touchLocation.X, (int)touchLocation.Y,
touchDown);

   // clear the splatters list if we touch the "Clear" button
   if (touchClear && !touchCycle)
   {
```

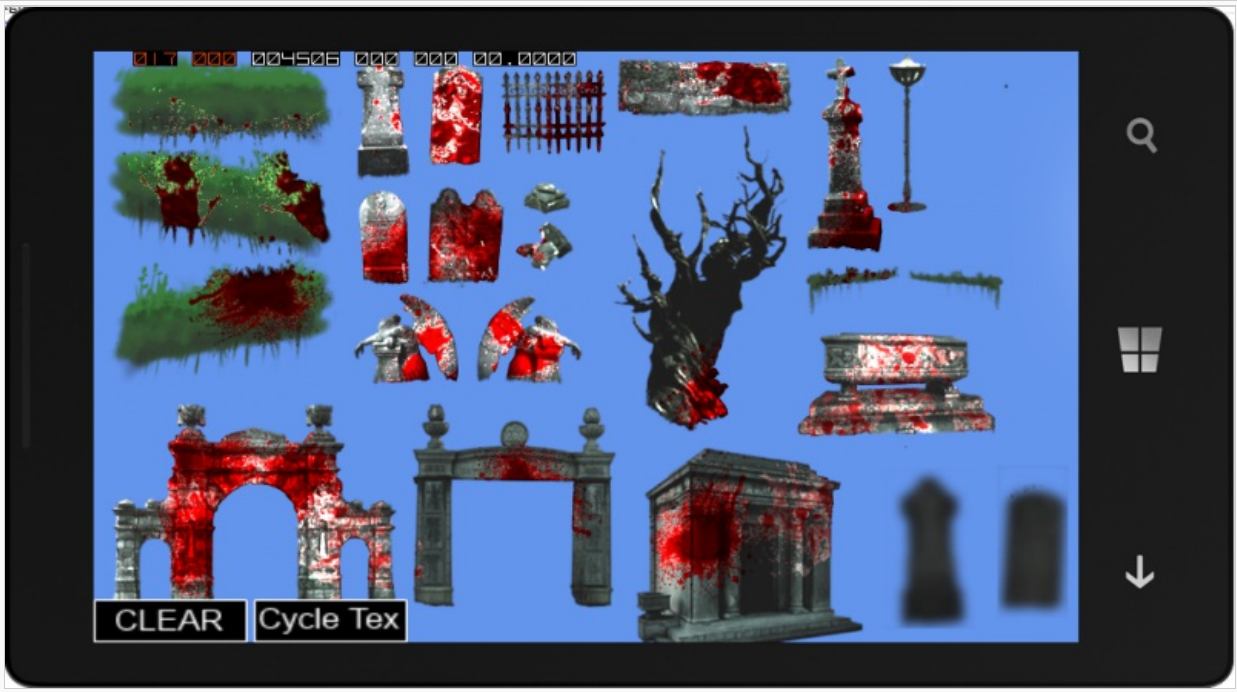

(C) Copyright Nokia 2014. All rights reserved.

```
     pointsToSplat.Clear();
    }
    // cycle the splatter pattern texture if we touch "Cycle"  button
    else if (!touchClear && touchCycle)
    {
     //index = 1 - index;
     index = (index + 1) % splaters.Length;
    }
    // if we do not touch the buttons
    // at the bottom of the screen we
    // will add a new splat point
    else if (!touchCycle && !touchClear)
    {
     pointsToSplat.Add(new Vector4(touchLocation.X, touchLocation.Y,
 Rand.GetRandomFloat(0, MathHelper.TwoPi), index));
    }
   }
  }

  base.Update(gameTime);
 }
```

The `Update()` function is very simple. It just keeps track of splatters List, clears the splatters list and cycles the splatters textures. All the rest code (for drawing and updating buttons you can find inside the attached code sample)

So when you run the sample you will see something like this:



Complete scene rendered. Moody, isn't it? :)

Code Example

The code can be downloaded from here: Media:AlphaSplatters.zip

Notes:

1. To run this sample you will need MonoGame Framework referenced from your project. Download the latest from here and SharpDX binaries here
2. If you use Express edition of Visual Studio 2012 for Windows Phone, you will also need Visual Studio 2012 Express for Windows Desktop to compile 2MGFX project.
3. You will need XNA framework and VS 2010 to compile graphics assets to XNB format.

(C) Copyright Nokia 2014. All rights reserved.

4. You can refer to this forum post on how to enable XNA projects in Visual Studio 2012

Hardware

This code was tested on Nokia Lumia 820

Summary

Even the simplest Decal system can add much depth to a 2D game. Combining this with 2D dynamic lighting and normal mapping can produce really "scary" results.

So who is the one to build a 2D side scrolling version of Doom 3 for Windows Phone?

(C) Copyright Nokia 2014. All rights reserved.