Onscreen Controller for XNA and MonoGame Games

This article describes an onscreen controller that you can use for XNA and MonoGame games.

Introduction

So you decided to make a game using XNA or MonoGame and you need some sort of controls for your game. Let's assume that your game will require more accuracy than you can get with G-Sensor of the phone no matter how calibrated it is or how will you smooth the peaks of data. You think further, watch around you in the room and see the game console controller. Now you start thinking of the way to get similar input experience to your game.

If your game is a twin-stick top-down shooter game this sort of control is ideal - you will basically need just two virtual sticks placed somewhere at the bottom of your game screen and in an equal distances from sides for each. The left stick will be responsible for navigation of your character and the right will cover the shooting part of things.

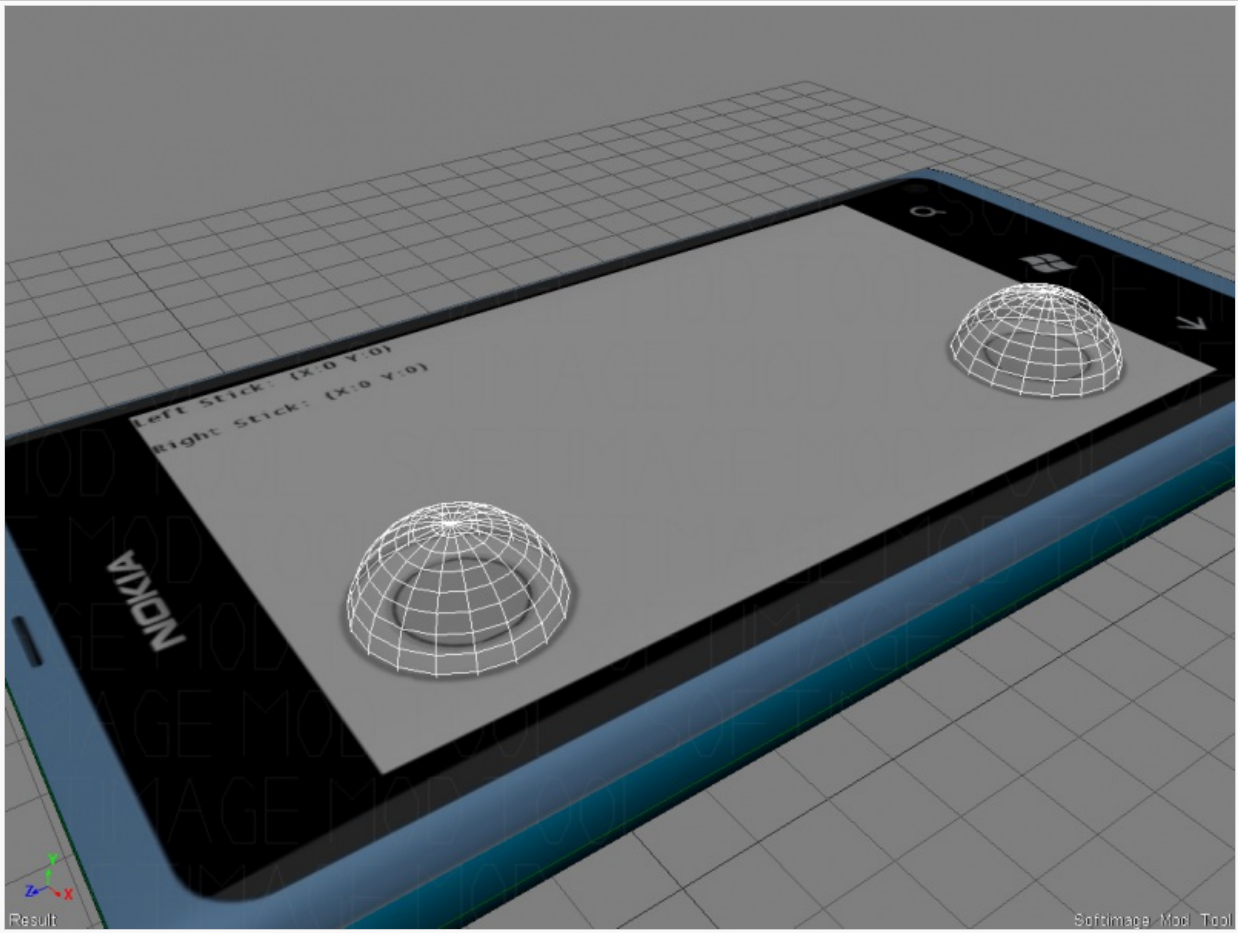
The following sections explain the overall design, and a code download is also linked.

Design

The basic idea for sticks in this onscreen controller came to me when I was holding the X360 controller and was thinking about the way XNA covers the interaction with it. When you pole the `GamePad` for its state you get an instance of `GamePadState` structure that already holds all the values of `GamePad` at current time stamp. This includes the values for `ThumbSticks`, `Triggers`, and `Buttons`. This part is pretty easy. We know we need a state, so we make a class or structure for it. Our `GamePad` returns the instance of it to us when we call the `GetState()` function. So we also need a function like this. The next step was how to detect that user touches the round stick. First thing that came to my mind was method of 2D collision detection for round objects. The way using the radius of the objects. However 2D is actually 3D with Orthographic projection. So I decided to the 3D way.

For each round control in controller I place a Bounding Sphere with its center being at the same position as object's center X/Y wise and at 0 Z-wise and make its radius as object's texture width / 2.

So all the described above in reality looks like this:



Bounding Spheres

(C) Copyright Nokia 2014. All rights reserved.

When user puts his finger onto the screen we start getting touch points. Each touch point has a `Location` value that is of a `Vector2` type. To check this point against our spheres we add the Z component of `Vector3` as 0:

```
Vector3 vect = new Vector3(points[i].Location, 0);
```

Next we check this new 3D point for containment inside our sphere

```
bool inside = false;
ContainmentType t = ContainmentType.Disjoint;
Vector3 point = new Vector3(x, y, 0);
leftStickCollision.Contains(ref point, out t);

if (t != ContainmentType.Disjoint)
    inside = true;
```

This way we get true if our point was actually inside the sphere and false if it was not. Ok this method would work perfect with buttons. However with thumb sticks we need also check the distance from center of sphere to our point. As we do not need the Z component of distance we simply check against X and Y values of current touch point location and original touch thumb stick position. We also use these values to draw the stick and its base. Than we check the stick value again for distance and if it's longer than 1 we use a method `Vector2.Normalize()`:

```
/// <summary>
/// The Value From the Right Stick. Vector2
/// </summary>
public Vector2 RightStick
{
    get
    {
        Vector2 scaledVector = (RightThumbPosition - RightThumbOriginalPosition) /
(padTexture.Width / 2);
        // when drawing graphics with spriteBatch we use Orthographic projection matrix.
        // This swaps the direction of Y axis. We need to put Y axis to point UP again
to have correct input values.
        // So we just multiply the calculated Y value of stick offset by -1
        scaledVector.Y *= -1;
        if (scaledVector.Length() > 1f)
            scaledVector.Normalize();
        return scaledVector;
    }
}
```

Well, that's it. You can find the code in **The Code** section. Comments and reviews are granted :)

Notes

This onscreen controller works the same way when you use MonoGame for WP8. Latest WP8 MonoGame code supports the landscape orientation of screen when using `DrawingSurface` instead of `DrawingSurfaceBackgroundGrid` but performance will suffer. However keep in mind that in case of using `DrawingSurfaceBackgroundGrid` you will have to deal with the screen rotation on your own including touch locations. That is a pretty simple stuff to do so I will leave it for you as an exercise.

Summary

When solving some problem try keeping the solution as simple as possible. This will help you instantly come back to the code after long periods of time and also simplify the process of enhancement.

(C) Copyright Nokia 2014. All rights reserved.

The Code

There you go! Try using the overloaded constructor of the `ScreenPad` class in `LoadContent()` method. This will give a completely different controller.

ScreenControlsSample.zip This code was tested on ALL Windows Phone devices :) . This is a bit simplified version of our OSC from Undead Carnage: Redemption. And this game is out for more than a year now.

I've just removed

```
#ifdef
```

compiler directives for Windows Desktop code paths. This was used to control the OSC using XBOX 360 gamepad.

(C) Copyright Nokia 2014. All rights reserved.