How to draw pixel perfect sprites on the PS2

James Lee (jbit)

June 6th 2006

1 How to draw pixel perfect sprites on the PS2

1.1 Overview

I've noticed that there's a lack of information detailing correct use of the PS2s GS. Since the GS has a few undocumented peculiarities, like an unusual DDA system, I think some documents would be helpful. In this doc I'll explain why the obvious ways of drawing sprites using linear mapping may cause unnecessary blurrynes. All examples will use the sprite shown below, and all images are the actual result of GS drawing, which have been scaled by 500% for easy viewing. You're assumed to know how to write PS2 drawing code, I won't cover that here.

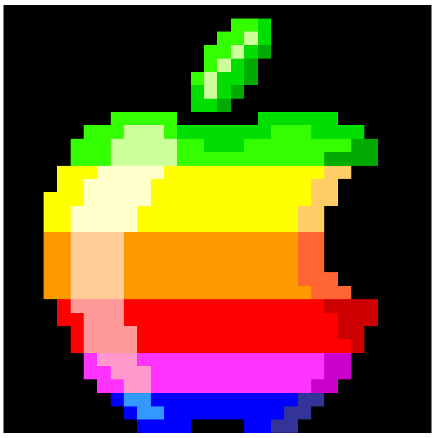

Figure 1: The golden sprite.

1.2 The obvious, but wrong, ways

To draw sprites on the ps2, most people will try specifying $UV(0,0)$ $XY(0,0)$ to $UV(31,31)$ $XY(31,31)$, this is what alot of other systems use, but as you can see below, the GS has a slight problem with this.

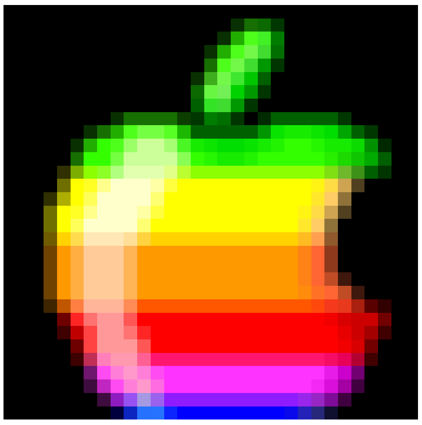

Figure 2: Image is blurry and shows only 31x31 pixels have been drawn.

Oh no! It seems the GS got lazy and only drew 31x31 pixels, and also decided to blur the sprite! Let's tackle the lack of pixels first. Since the GS is really for 3D drawing, it will not draw bottom/right pixels, this is so triangles will fit together nicely. Most people will now think that just expanding the entire sprite by one pixel/texel will be enough, resulting with the coords $UV(0,0)$ $XY(0,0)$ to $UV(32,32)$ $XY(32,32)$. This will actually work fine for nearest texel sampling. However, as you can see below, with linear sampling the sprite is still blurry!

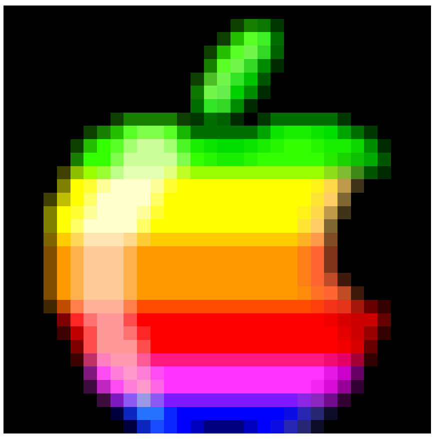

Figure 3: Image is blurry.

So, why doesn't this work as expected? Well, on the GS, the centre of a pixel is *coord.0*, but the centre of a texel is *coord.5*. This means in the above case the UV mapping is off by about half a pixel all the time, not good! So let's adjust this to $UV(0.5,0.5)$ $XY(0,0)$ to $UV(32.5,32.5)$ $XY(32,32)$ and see what happens.

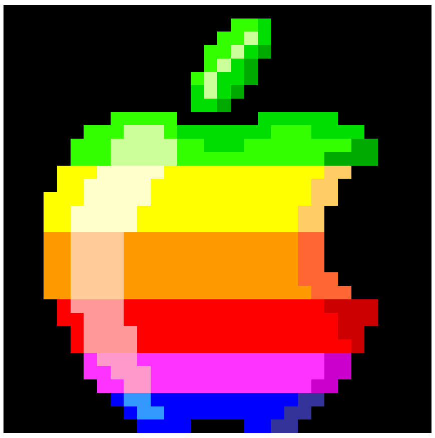

Figure 4: Image is correct.

Wow, that looks perfect, mission successful.... Actually no. For certain sizes of sprite, these coord offsets will work fine, but for others, you'll get slight blurring, see this 24x24 sprite using $UV(0.5, 0.5)$ $XY(0, 0)$ to $UV(24.5, 24.5)$ $XY(24, 24)$...

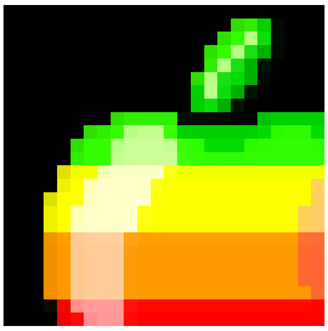

Figure 5: Image is slightly blurry.

If you look closely, you can still see some blurring! I'm sure most people won't care about this level of blurring, but for nice crisp sprites and font glyphs, a better way is needed. This blurring is being caused because the actual primitive being drawn is one pixel (on each axis) larger than the framebuffer pixels that it's filling, this causes the GS' DDA to screw up the texel to pixel mapping.

1.3 The correct way

The moment you've been waiting for, specifying $UV(0.5, 0.5)$ $XY(0, 0)$ to $UV(31.625, 31.625)$ $XY(31.0625, 31.0625)$ and $UV(0.5, 0.5)$ $XY(0, 0)$ to $UV(23.625, 23.625)$ $XY(23.0625, 23.0625)$ gives these results...

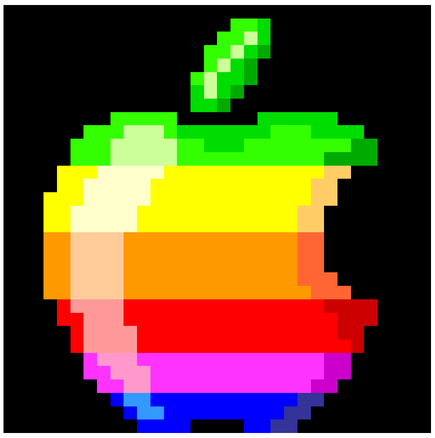

Figure 6: 32x32 Image is correct.

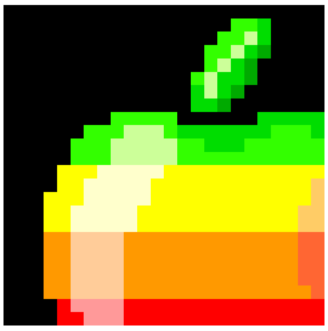

Figure 7: 24x24 Image is correct.

Wow, they're both perfect! And you'll find that pretty much all sprite sizes give perfect results too! (I haven't found any that don't work, but I'm sure they exist). So what's with these magic numbers? Why do they work? Well instead of adding an entire pixel to the XY coords, we add the smallest value possible, which is 1/16th of a pixel (0.0625), for some reason we have to add twice this to the texture coords to get correct mapping, I may write up why this is at a later date.

1.4 Wrapping up

So now we know that in order to get pixel perfect sprites using linear mapping on the GS, we need to use $UV(0.5, 0.5)XY(0, 0)$ to $UV(W - 0.375, H - 0.375)XY(W - 0.9375, H - 0.9375)$. I have tested these results quite a bit and everything seems to come out perfect in all my tests, if you find differently please contact me.