Shaders in Screen Space

In the last article, I introduced you to some super simple shaders that you can get up and running in Orx in minutes.

They were all based on the use of `gl_TexCoord` in order to get the pixel data of the current texture. In this article we will focus on the use of `gl_FragCoord` which works differently.

The differences between them are:

1. `gl_TexCoord[0]` is your primary texture coordinate coming out of the fixed vertex shader used by Orx, for both viewports and objects. Texture coordinates, often called UVs, are always normalized (values always between 0.0 and 1.0), so you don't need to know about the resolution to use it, even for a quad covering the full screen.
2. `gl_FragCoord` is the coordinates of your fragment, in screen space, in pixels. In this case, you need to know the resolution in order to normalize it, and know in which part of the screen you are.

If that's not clear, let's move on and try some things out to see the effect. Hopefully these statements will prove clearer after.

Setup

Let's start as usual by creating a default project using the init tool.

I'm going to make a few little changes to the default project config, making the game area an 800×600 window with a blue background to be able to illustrate things clearly:

```
[Display]
...
Decoration      = true
...
ScreenWidth     = 800
ScreenHeight    = 600
```

```
[MainCamera]
...
...
BackgroundColor         = (0, 128, 255)
```

The Shader

Here is a shader based on one from the previous article. Add the following to your config:

```
[GradientShader]
ParamList = texture # resolution
resolution = (800, 600)
Code = "
void main() {
    vec2 pos = gl_FragCoord.xy/resolution.xy;
    gl_FragColor = vec4(1.0, pos.y, 0.0, 1.0);
}
"
```

Notice the `resolution` parameter has been added next to the `texture` parameter. Then the value of `resolution` value is provided.

Why? Because we are going to be using `gl_FragCoord` which needs to be scaled with the current resolution. This is because `gl_FragCoord` is in screen space. So we will pass this to the shader.

Next, the `pos` coordinates are calculated, using the screen space / resolution.

Here is the result:

Well... there does appear to be some gradient between red and yellow occurring inside the texture rectangle. But the effect is subtle, if there at all. And is certainly not obvious what is happening.

Let's scale the size of the object up to make things clearer:

```
[Object]
...
```

```
...
Scale = 3.0
```

Run it and you'll see:

This should make things clearer. The pixel positions are mapped from the entire screen space and not from the dimensions of the texture as done with `gl_TexCoord[0]`.

This makes for very different and interesting effects.

We'll take a look at another:

```
[GradientShader]
ParamList = texture # resolution
resolution = (800, 600)
Code = "
void main() {
    vec2 pos = gl_FragCoord.xy/resolution.xy;

    vec4 textureFrag = texture2D(texture, pos );

    gl_FragColor = vec4(1.0, pos.y, 0.0, textureFrag.a);
}
"
```

Which looks like this:

The Orx logo texture is calculated in screen space, but is rendered into the texture which gives that great cut-out mask effect.

See also

1. https://orx-project.org/forum/discussion/9065/getting-a-glsl-shader-to-display-anything-at-all/p1#Comment_9554