# Getting started with Shaders in Orx

This guide is designed to help you to get some simple shaders quickly working in Orx in order to build confidence using them. I will touch on a few key concepts but I recommend afterwards using a guide like The Book Of Shaders to understand fragment shaders in more detail.

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

## Setup

I'm going to make a few little changes to the default project config, making the game area an 800×600 window to that I can easily see the Orx console output for problems:

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

## The Shader

Let's start with a really simple shader. Add the following to your config:

```
[GradientShader]
ParamList = texture
Code = "
void main() {
    float y = gl_TexCoord[0].y;
    gl_FragColor = vec4(1.0, y, 0.0, 1.0);
}
"
```

And set the default object to use that shader:

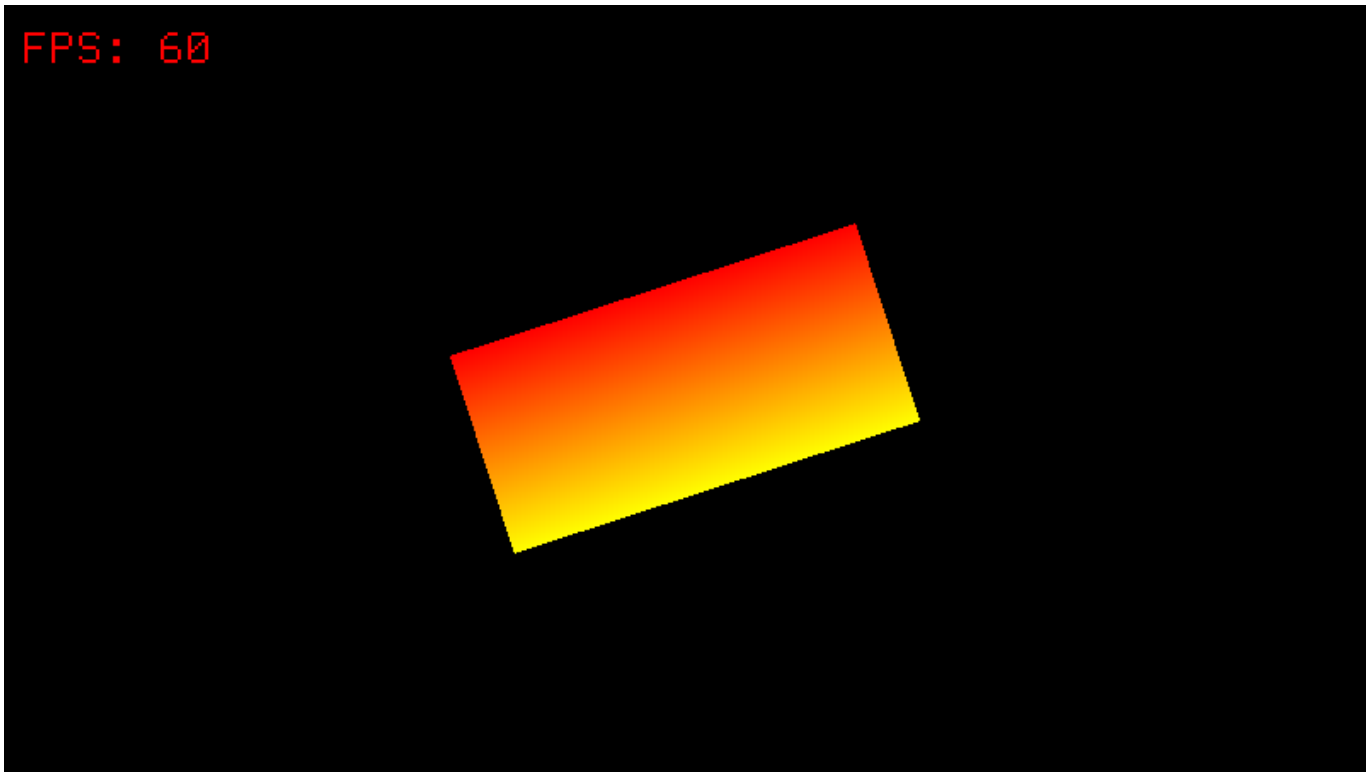
```
[Object]
Graphic         = @
Texture         = logo.png
...
...
ShaderList = GradientShader
```

I'll explain the above sections. We start with a parameter list in the shader section. We can pass parameters to our shader. In this case, we are only supplying the `texture` parameter. But notice that I didn't specify an actual argument for texture? So which texture is the shader using?

In this case, Orx determines it automatically. `Object` uses the `GradientShader` so the texture used will be the one found in the Object's `Graphic` section. That texture is `logo.png`. To read more about this, see orxSHADER structure in the Wiki Config.

The `Code` is the actual shader code itself which looks a little like C. In the first line we determine the y-coordinate of the current texture using `gl_TexCoord[0].y`. This will be a float between 0.0 and 1.0. We then plug this value into the green value of `gl_FragColor`. Red, blue and alpha are full 1.0, the result which changes colours from red to yellow as the texture renders.

Run this and you will find the logo is replaced by a gradient texture from red to yellow.




Nice and simple, using only two lines of shader code.

Let's now have a play with the alpha values. The red to yellow fade is based on the value of the y-coordinate. So let's make the texture progressively more transparent based on the x-coordinate.

Our code now becomes:

```
[GradientShader]
ParamList = texture
Code = "
void main() {
    vec2 pos = gl_TexCoord[0].xy;
    gl_FragColor = vec4(1.0, pos.y, 0.0, 1.0-pos.x);
```

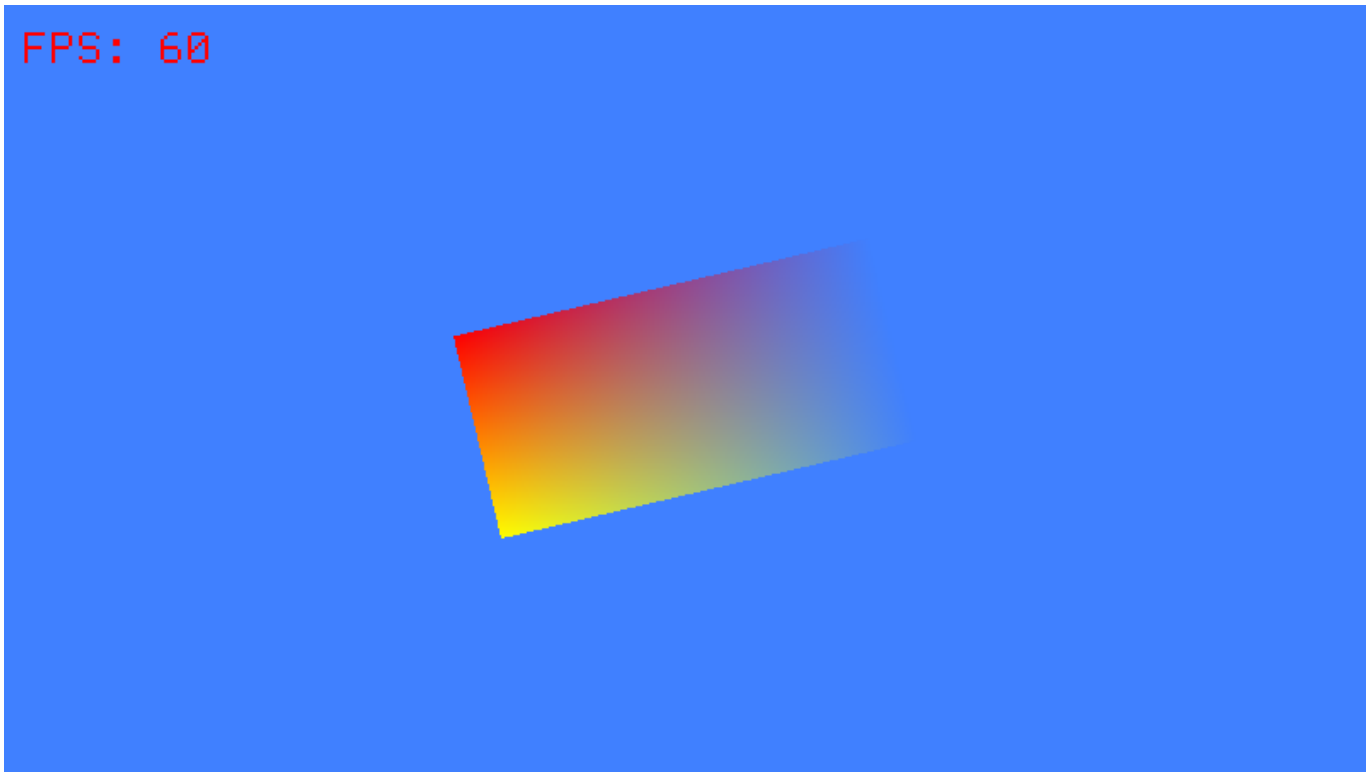
```
}
"
```

Instead of a float to get just the y-coordinate, we'll use a vec2 variable called `pos` to hold both the current x and y-coordinate values. The x-coordinate is then used as the alpha argument for `gl_FragColor`. The texture alpha starts fully opaque (solid) and becomes progressively transparent to the right with `1.0-pos.x`.

To make the effect easier to see, add a nice blue background for the viewport:

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

Run this and you will see the right hand side of the texture fading out, and the blue background starting to show through.




So far, we have made the shader repaint every pixel and nothing of the original underlying texture has been preserved. What if we could blend or mix our colours with the original `logo.png` texture?

It would be nice to use the white area of the logo to paint our colours.

We can do it with a small change:

```
[GradientShader]
ParamList = texture
```

```
Code = "
void main() {
    vec2 pos = gl_TexCoord[0].xy;

    vec4 textureFrag = texture2D(texture, pos );

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

This time, we're using the `texture2D` function to return the texture pixel under our current coordinates. This function returns a vec4 type with four values: `rgba`. We'll just use the `a` (alpha) value from the texture pixel and plug that into the `gl_FragColor`. This is effectively using colours from one source and alpha from another source together.


That's pretty cool right?

And a simple alpha inversion can be done with `1.0-textureFrag.a`:

```
[GradientShader]
ParamList = texture
Code = "
void main() {
    vec2 pos = gl_TexCoord[0].xy;

    vec4 textureFrag = texture2D(texture, pos );
```

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


Really effective!

That will do for the moment. Hopefully that gets you pretty keen for shaders and what simple colour effects are possible.

In the next article, I'll cover the use of `gl_FragCoord`. You will see this in many GLSL examples out on the web. So far, we have covered the use of `gl_TexCoord`, and I'll show you why these two are used in very different ways.