Creating a Custom Fragment Shader

Write a fragment shader using the set of SpriteKit-exposed symbols, and supply it with custom data.

Overview

Your fragment shader should be written in the OpenGL ES 2.0 shading language. SpriteKit automatically does the necessary work to make sure this shader compiles correctly in both iOS and macOS. The shader object compiles your shader by appending the source code you provide to a preamble created by SpriteKit. This preamble declares all of the standard SpriteKit variables and functions, including declarations for any uniforms you have associated with the shader object.

Use SpriteKit-Provided Data in a Shader

The following table describes the SpriteKit symbols made available to your shader.

Symbol declaration	Type	Description
`sampler2D u_texture;`	Uniform	A sampler associated with the texture used to render the node.
`float u_time;`	Uniform	The elapsed time in the simulation.
`float u_path_length;`	Uniform	Provided only when the shader is attached to an `SKShapeNode` object's `strokeShader` property. This value represents the total length of the path, in points.
`vec2 v_tex_coord;`	Varying	The coordinates used to access the texture. These coordinates are normalized so that the point (0.0,0.0) is in the bottom-left corner of the texture.
`vec4 v_color_mix;`	Varying	The premultiplied color value for the node being rendered.
`float v_path_distance;`	Varying	Provided only when the shader is attached to an `SKShapeNode` object's `strokeShader` property. This value represents the distance along the path in points.
`vec4 SKDefaultShading()`	Function	A function that provides the default behavior used by SpriteKit.

Provide Custom Per-Frame Data to a Shader

To make a global variable available to your shader, you create an `SKUniform` and add it to the shader's `uniforms` array. Uniforms can be modified at any time on the CPU, but they're constant on the GPU. Therefore, uniforms are the way to provide your shader with per-frame data.

The following code shows how to pass a size value to a shader by using a uniform.

```
let uniformBasedShader = SKShader(fileNamed: "UsingUniform.fsh")

let sprite = SKSpriteNode()
sprite.shader = uniformBasedShader

let spriteSize = vector_float2(Float(sprite.frame.size.width),
                               Float(sprite.frame.size.height))
uniformBasedShader.uniforms = [
    SKUniform(name: "u_sprite_size", vectorFloat2: spriteSize)
]
```

Because uniforms are shared across executions in a given frame, every node that executes uniformBasedShader in the example above will read the same value for u_sprite_size.

Provide Custom Per-Primitive Data to a Shader

To associate shader variables with specific nodes, you use SKAttribute. Using attributes rather than uniforms allows a single SKShader to be shared between nodes, each one defining their own value for a shader variable. The following code demonstrates attributes by passing the size of a sprite to a shader as an attribute.

```
let attributeBasedShader = SKShader(fileNamed: "UsingAttributes.fsh")
attributeBasedShader.attributes = [
    SKAttribute(name: "a_sprite_size", type: .vectorFloat2)
]

let sprite = SKSpriteNode()
sprite.shader = attributeBasedShader

let spriteSize = vector_float2(Float(sprite.frame.size.width),
                               Float(sprite.frame.size.height))
sprite.setValue(SKAttributeValue(vectorFloat2: spriteSize),
                forAttribute: "a_sprite_size")
```

Because an attribute defines a shader variable on the sprite node and not the shader object, every execution of attributeBasedShader will read its own value of a_sprite_size.

Define a Function Body

Your shader should define a main() function. This function must set the gl_FragColor variable to a color value to use in the blend stage. Typically, the color value you return in this variable should already be premultiplied by the fragment's alpha value.

The following code shows a very simple fragment shader that provides the default shading behavior.

```
void main()
{
    gl_FragColor = SKDefaultShading();
}
```