Maximizing Node Drawing Performance

Structure your nodes for maximum performance.

Overview

As a high-level graphics framework, you configure your scene by laying out nodes, and SpriteKit builds the Metal draw calls for you that actually render the content. Thus, you have little control over the amount of low-level graphics work that SpriteKit does, but this article gives you hints that can speed up SpriteKit’s efforts.

Show Draw Count

Each draw call is an expensive operation; the more graphical elements you can pack into a single draw, the faster your scene can be rendered, and thus the higher frames per second it can achieve. You can see the number of draw calls it’s taking to render your scene live on the SKView by enabling showsDrawCount. In general, all scene nodes can be rendered in a single draw call that came from the same SKTexture, for example:

>> Sprite nodes that use the same SKTexture

>> Sprite nodes whose texture specifies a subrectangle of the same texture through the SKTexture init(rect:in:).

> **Important:** Node zPosition impacts draw call count because, generally, nodes at different zPositions must be rendered on a subsequent pass. For example, nodes that overlap each other must be done in separate draw calls. Overlapping nodes at the same zPosition can be rendered in the same draw call if you enable the SKView ignoresSiblingOrder property.

For all other SpriteKit nodes that draw, pay attention to the showsDrawCount statistic as you develop your app to gain an understanding of which configurations affect draw count.

Show Node Count

The number of nodes in your scene relates to the number of draw calls. Therefore, minimizing the number of nodes in your scene is a good practice to keep the draw call count down. Showing the node count in your scene is done through the SKView `showsNodeCount` property.

The number of nodes in your scene can also affect performance if you enumerate all of your nodes within the update life cycle. While the SKView `shouldCullNonVisibleNodes` property can prune offscreen nodes out of an issued draw call, SpriteKit must still enumerate all nodes in your scene to figure out which ones are offscreen, and this is an expensive operation. As a result, eliminating offscreen nodes yourself by using `removeFromParent()` is the fastest way to cull non visible nodes. Once a node is removed from the node tree, SpriteKit no longer needs to consider whether it’s onscreen or offscreen every frame.