Getting Started with Nodes

Learn about the fundamental properties that provide a foundation for all other nodes.

Overview

Nodes are organized hierarchically into node trees, similar to how views and subviews work. Most commonly, a node tree is defined with a scene node (SKScene) as the root node and other content nodes as descendants. The scene node runs an animation loop that processes actions on the nodes, simulates physics, and renders the contents of the node tree for display.

Every node in a node tree provides a coordinate system to its children. After a child is added to the node tree, you position it inside its parent's coordinate system by setting its zPosition property. A node's coordinate system can be scaled and rotated by changing its xScale, yScale, and zRotation properties. When a node's coordinate system is scaled or rotated, this transformation is applied both to the node's own content and to that of its descendants.

Review Important Properties

The SKNode class does not perform any drawing of its own. However, many SKNode subclasses render visual content and so the SKNode class understands some visual concepts:

>> The frame property provides the bounding rectangle for a node's visual content, modified by the scale and rotation properties. The frame is nonempty if the node's class draws content. Each node subclass determines the size of this content differently. In some subclasses, the size of the node's content is declared explicitly, such as in the SKSpriteNode class. In other subclasses, the content size is calculated implicitly by the class using other object properties. For example, an SKLabelNode object determines its content size using the label's message text and font characteristics.

>> A node's accumulated frame, retrieved by calling the calculateAccumulatedFrame() method, is the largest rectangle that includes the frame of the node and the frames of all its descendants.

>> Other properties, such as the alpha and isHidden properties, affect how the node and its descendants are drawn.

All nodes are responder objects that can respond directly to user interaction with the node onscreen. You can also convert between coordinate systems and perform hit testing to determine which nodes a point lies in, and perform intersections between nodes in the tree to determine if their physical areas overlap.

A node can support a physics body, which is an object that simulates the physical properties of the object. When a node has a physics body, the physics simulation automatically computes a new position for the physics body and then moves and rotates the node to match that position.

A node can supply constraints that express relationships with other nodes or locations in the scene. These constraints are automatically applied by the scene before the scene is rendered. Another set of constraints is used in conjunction with actions to perform inverse-kinematic animations.

Ensure Node Access on the Main Thread

All of the SpriteKit callbacks for `SKViewDelegate`, `SKSceneDelegate`, and `SKScene` occur in the main thread and therefore, these are safe places to access or manipulate nodes. If you are performing other work in the background, you should adopt an approach similar to Listing 1.

LISTING 1 MANIPULATING A NODE WITHIN A SCENE UPDATE CALLBACK

```
class ViewController: UIViewController {
    let queue = DispatchQueue.global()

    var makeNodeModifications = false

    func backgroundComputation() {
        queue.async {
            // Perform background calculations but do not modify
            // SpriteKit objects

            // Set a flag for later modification within the
            // SKScene or SKSceneDelegate callback of your choosing

            self.makeNodeModifications = true
        }
    }
}
extension ViewController: SKSceneDelegate {
    func update(_ currentTime: TimeInterval, for scene: SKScene) {
        if makeNodeModifications {
            makeNodeModifications = false

            // Make node modifications
        }
    }
}
```

Lay Out Your Scene with Basic Nodes

Even though SKNode objects cannot directly draw content, there are useful ways to use them in your app. Here are some ideas to get you started:

>> You have content that is built up from multiple node objects, either sprites or other content nodes. However, you want this content to be thought of as a single object within your game, without promoting any one of the content nodes to be the root. A basic node is appropriate, because you can give it a position in the scene tree and then make all of the other nodes its descendants. These individual pieces can also be moved or adjusted relative to the parent's location.

>> You want to organize your drawn content into a series of layers. For example, many games have a background layer for the world, another layer for characters, and a third layer for text and other game information. Other games have many more layers. Create the layers as basic nodes, and insert them into the scene in order. Then, as necessary, you can make individual layers visible or invisible.

>> You need an object in the scene that is invisible but that performs some other necessary function. For example, in a dungeon-exploring game, an invisible node might be used to represent a hidden trap. When another node intersects it, the trap is triggered. For another example, you might add a node as a child of another node that represents the position of the player's point of view.

There are advantages to having a node in the tree to represent these concepts:

>> You can add or remove entire subtrees by adding or removing a single node. This makes scene management efficient.

>> You can adjust properties of a node in the tree and have the effects of those properties propagate down to the node's descendants. For example, if the basic node has sprite nodes as its children, rotating the basic node also rotates all of the sprite content.

>> You can take advantage of actions, physics contacts, and other SpriteKit features to implement the concept.