SwiftUI Pills

GeometryReader

Simple Concepts
What you need to know at first in order to handle the complexity of more advanced concepts later on.

Beginner Level
Minimal-complexity explanations to focus on what really matters.

Test Code
Short, simple code to help you try out ideas and concepts.

Real Cases
Application situations in real cases to help you see the example as a ready-to-use tool.

Clear Explanations
Explanations that clarify, not complicate.

Interview Questions
Questions and answers on the topic that commonly appear in technical interviews.

Knowledge in programming is like constructing a building: the strength of the entire structure depends on the quality of each brick. Each unit of knowledge must be clear, solid, and understandable on its own, independent of the others. Over time, these blocks consolidate and creatively connect with others, forming a robust and harmonious structure.

GeometryReader is an advanced tool in SwiftUI that's often introduced in the final chapters of books, yet its use is valuable even in the early stages of interface creation. GeometryReader can be as simple as providing graphical context information to position a component or as sophisticated as controlling complex animations. If you're just starting to build your "structure" in Swift, this is the ideal time to incorporate and refine this small yet powerful "brick" into your foundation.

What is GeometryReader?

GeometryReader is a special view in SwiftUI that provides information about the size and position of the available space in the view it wraps. By using it, you gain access to a GeometryProxy object, which contains data on the view's size and coordinates. This allows for dynamically adjusting the layout based on the available space, similar to how AutoLayout worked in UIKit.

When and Why Did It Appear?

GeometryReader was introduced in 2019 with the launch of SwiftUI in iOS 13. Its inclusion was driven by the need for adaptive design tools in SwiftUI, enabling developers to create fluid and adaptable user interfaces for different screen sizes. Before SwiftUI, adaptive design in UIKit was achieved through Auto Layout and other visual tools in Interface Builder. However, with SwiftUI's declarative approach, a programmatic way to access layout space and adjust views dynamically was essential.

The introduction of GeometryReader helped address these needs by providing an efficient way to read a view's size and position in real-time, allowing developers to respond to environmental changes and thus create more responsive interfaces.

What Is It Used For?

GeometryReader allows you to build views that automatically adjust to different screen sizes. It's useful for writing logic that reorganizes or resizes elements when the device orientation changes (such as switching from portrait to landscape on an iPhone) or when an app runs on devices of different sizes (for example, from an iPhone to an iPad). This makes views more adaptive and responsive to various display spaces. Without the help of GeometryReader, this task can be quite challenging to implement and complex to maintain as Apple releases new devices with different display spaces.

When Should It Be Used?

GeometryReader should be used when you need a view or several elements within it to automatically adapt to the available size and position on the screen. This is especially useful in situations such as:

- Adapting to different screen sizes: When the same interface needs to look good on both an iPhone and an iPad, or on any device with varying dimensions.
- Responding to orientation changes: When rotating the device from portrait to landscape, GeometryReader enables layout adjustments to better use the available space.
- Positioning elements based on available space: If you want to place or size elements relative to the screen or their container, GeometryReader provides the data to achieve this.

Contraindications?

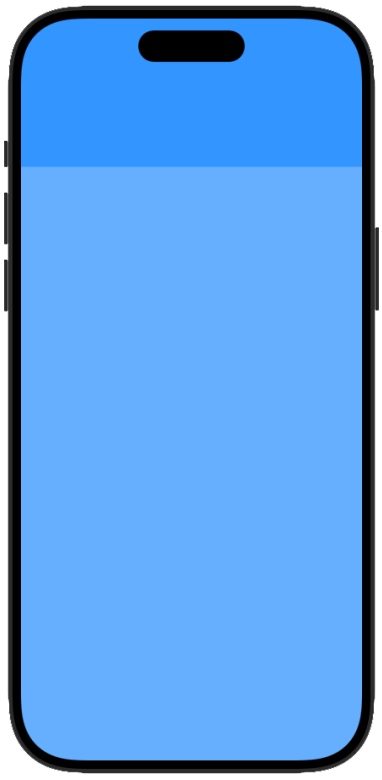
Yes, GeometryReader can be costly in terms of computational power, which results in higher battery consumption. Try to structure your view functionality without relying on GeometryReader and use it only when absolutely necessary. Keep in mind that a view with multiple GeometryReader instances could slow down performance on some devices.

Let's look at a simple use case to see how it works

Suppose you define a header in your view that should always occupy 30% of the total height, regardless of the device or whether it's rotated to landscape mode.
An initial approach would be the following:

```swift
struct View001: View {
    var body: some View {
        VStack(spacing: 0) {
            VStack {
                Rectangle()
                    .fill(Color.blue.opacity(0.8))
                    .frame(height: UIScreen.main.bounds.height * 0.20)
            }
            VStack {
                Rectangle()
                    .fill(Color.blue.opacity(0.6))
            }
        }
        .ignoresSafeArea()
    }
}
```

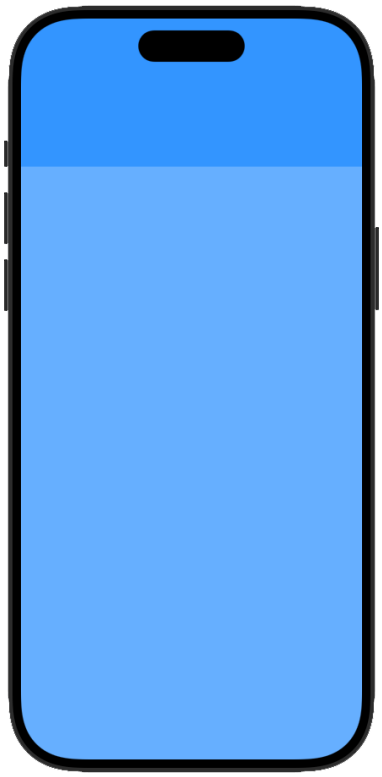
The result in portrait mode is correct; however, the result in landscape mode is not as expected. This happens because the height referenced in the frame takes the height of the "physical screen," which remains the same whether it is in portrait or landscape orientation.



To solve this problem, we can wrap our stacks in a GeometryReader and then use the geometry reading to refer to the view's height instead of the screen height, achieving the expected result.

```swift
struct View001: View {
    var body: some View {
        VStack(spacing: 0) {
            GeometryReader { geometry in
                VStack {
                    Rectangle()
                        .fill(Color.blue.opacity(0.8))
                        .frame(height: geometry.size.height * 0.20)
                }

                VStack {
                    Rectangle()
                        .fill(Color.blue.opacity(0.6))
                }
            }
        }
        .ignoresSafeArea()
    }
}
```



GeometryReader is a key tool for creating responsive layouts in SwiftUI. Many of the questions we raised at the beginning could come up in technical interview tests. You don't need to be an expert in GeometryReader, but knowing it exists and being able to explain its basic functionality adds great value as a developer and makes it easier to dive deeper into the topic when the time comes.

Thank you for downloading this content. If you found it useful, follow me on LinkedIn for more SwiftUI tips.

Example Code: https://github.com/edd709/P20241105151449.git