iOS Photos App Adaptive Grid Navigation in SwiftUI

Overview

The iOS Photos app features a distinctive adaptive grid layout that responds to pinch-to-zoom gestures. As users zoom in, the grid transitions from a dense small thumbnail view to larger thumbnails and eventually to a single-column layout. This guide explains how to replicate this behavior in SwiftUI.

Key Behaviors to Replicate

- Small Grid: Many columns (typically 5-7) with tiny thumbnails
- Medium Grid: Fewer columns (3-4) with medium-sized thumbnails
- Large Grid: Even fewer columns (2) with larger thumbnails
- Single Column: Full-width images in a vertical list
- Smooth transitions between states using pinch gestures

Core Concepts

1. MagnificationGesture

SwiftUI's MagnificationGesture is the foundation for detecting pinch-to-zoom interactions. This gesture provides a scale value that increases as the user spreads their fingers apart and decreases as they pinch together.

```
@State private var scale: CGFloat = 1.0
@State private var lastScale: CGFloat = 1.0

var magnificationGesture: some Gesture {
    MagnificationGesture()
        .onChanged { value in
            let delta = value / lastScale
            lastScale = value
            scale *= delta
        }
        .onEnded { _ in
            lastScale = 1.0
            // Snap to nearest zoom level
            withAnimation(.spring()) {
                scale = snapToNearestLevel(scale)
```

```
            }
        }
}
```

2. Dynamic Column Count

The number of columns in the grid should be calculated based on the current zoom scale. As the scale increases, the column count decreases, making each item larger.

```
func columnCount(for scale: CGFloat, in width: CGFloat) -> Int {
    // Define zoom thresholds
    let baseColumns = 5

    if scale < 0.8 {
        return 7  // Most zoomed out - tiny thumbnails
    } else if scale < 1.2 {
        return 5  // Default view
    } else if scale < 2.0 {
        return 3  // Medium zoom
    } else if scale < 3.0 {
        return 2  // Large thumbnails
    } else {
        return 1  // Single column / full width
    }
}
```

3. LazyVGrid with Adaptive Columns

Use LazyVGrid with GridItem arrays that change based on the zoom level. The .adaptive modifier won't work well here because we need discrete snap points.

```
func gridColumns(count: Int) -> [GridItem] {
    Array(repeating: GridItem(.flexible(), spacing: 2), count: count)
}

// In your view:
LazyVGrid(columns: gridColumns(count: currentColumnCount), spacing: 2) {
    ForEach(photos) { photo in
        PhotoThumbnail(photo: photo)
            .aspectRatio(1, contentMode: .fill)
    }
}
```

Complete Implementation

AdaptivePhotoGrid View

Here's a complete implementation that combines all the concepts into a reusable view:

```
import SwiftUI

struct AdaptivePhotoGrid: View {
    let photos: [Photo]

    @State private var scale: CGFloat = 1.0
    @State private var lastScale: CGFloat = 1.0
    @State private var columnCount: Int = 5

    // Define zoom levels for snapping
    private let zoomLevels: [(scale: CGFloat, columns: Int)] = [
        (0.6, 7),   // Most zoomed out
        (1.0, 5),   // Default
        (1.6, 3),   // Medium
        (2.5, 2),   // Large
        (4.0, 1)    // Single column
    ]

    var body: some View {
        GeometryReader { geometry in
            ScrollView {
                LazyVGrid(
                    columns: Array(
                        repeating: GridItem(.flexible(), spacing: 2),
                        count: columnCount
                    ),
                    spacing: 2
                ) {
                    ForEach(photos) { photo in
                        PhotoThumbnailView(photo: photo)
                            .aspectRatio(1, contentMode: .fill)
                            .clipped()
                    }
                }
                .padding(.horizontal, 2)
            }
            .gesture(magnificationGesture)
        }
    }

    private var magnificationGesture: some Gesture {
        MagnificationGesture()
            .onChanged { value in
                let delta = value / lastScale
                lastScale = value

                // Update scale with bounds
                let newScale = scale * delta
```

```
                scale = min(max(newScale, 0.5), 5.0)

                // Update column count in real-time
                columnCount = columnsForScale(scale)
            }
            .onEnded { _ in
                lastScale = 1.0

                // Snap to nearest zoom level
                withAnimation(.spring(response: 0.3, dampingFraction: 0.7)) {
                    let snapped = snapToNearestLevel(scale)
                    scale = snapped.scale
                    columnCount = snapped.columns
                }
            }
    }

    private func columnsForScale(_ scale: CGFloat) -> Int {
        for (index, level) in zoomLevels.enumerated() {
            if index == zoomLevels.count - 1 {
                return level.columns
            }
            let nextLevel = zoomLevels[index + 1]
            let midpoint = (level.scale + nextLevel.scale) / 2
            if scale < midpoint {
                return level.columns
            }
        }
        return zoomLevels.last?.columns ?? 5
    }

    private func snapToNearestLevel(_ scale: CGFloat)
        -> (scale: CGFloat, columns: Int) {
        var closest = zoomLevels[0]
        var minDistance = abs(scale - closest.scale)

        for level in zoomLevels {
            let distance = abs(scale - level.scale)
            if distance < minDistance {
                minDistance = distance
                closest = level
            }
        }
        return closest
    }
}
```

Enhanced Version with Smooth Transitions

For even smoother transitions that more closely match the iOS Photos app, you can add animated layout changes and haptic feedback:

```
import SwiftUI

struct EnhancedPhotoGrid: View {
    let photos: [Photo]

    @State private var scale: CGFloat = 1.0
    @State private var lastScale: CGFloat = 1.0
    @State private var columnCount: Int = 5
    @State private var lastColumnCount: Int = 5

    private let zoomLevels: [(scale: CGFloat, columns: Int)] = [
        (0.6, 7), (1.0, 5), (1.6, 3), (2.5, 2), (4.0, 1)
    ]

    private let hapticFeedback = UIImpactFeedbackGenerator(style: .light)

    var body: some View {
        GeometryReader { geometry in
            ScrollView {
                LazyVGrid(
                    columns: gridColumns,
                    spacing: spacing
                ) {
                    ForEach(photos) { photo in
                        PhotoThumbnailView(photo: photo)
                            .aspectRatio(1, contentMode: .fill)
                            .clipped()
                            .id(photo.id)
                    }
                }
                .padding(.horizontal, spacing)
                .animation(.easeInOut(duration: 0.2), value: columnCount)
            }
            .gesture(magnificationGesture)
        }
        .onAppear {
            hapticFeedback.prepare()
        }
    }

    private var gridColumns: [GridItem] {
        Array(
            repeating: GridItem(.flexible(), spacing: spacing),
            count: columnCount
        )
    }

    private var spacing: CGFloat {
        columnCount == 1 ? 8 : 2
```

```
    }

    private var magnificationGesture: some Gesture {
        MagnificationGesture()
            .onChanged { value in
                let delta = value / lastScale
                lastScale = value

                let newScale = min(max(scale * delta, 0.5), 5.0)
                scale = newScale

                let newColumns = columnsForScale(newScale)
                if newColumns != lastColumnCount {
                    // Provide haptic feedback on column change
                    hapticFeedback.impactOccurred()

                    withAnimation(.easeInOut(duration: 0.15)) {
                        columnCount = newColumns
                    }
                    lastColumnCount = newColumns
                }
            }
            .onEnded { _ in
                lastScale = 1.0

                withAnimation(.spring(response: 0.3, dampingFraction: 0.8)) {
                    let snapped = snapToNearestLevel(scale)
                    scale = snapped.scale
                    columnCount = snapped.columns
                    lastColumnCount = snapped.columns
                }
            }
    }

    // ... (columnsForScale and snapToNearestLevel same as before)
}
```

Supporting Components

Photo Model

```
struct Photo: Identifiable {
    let id: UUID
    let image: UIImage
    let date: Date

    init(image: UIImage, date: Date = Date()) {
        self.id = UUID()
        self.image = image
        self.date = date
    }
}
```

PhotoThumbnailView

```
struct PhotoThumbnailView: View {
    let photo: Photo

    var body: some View {
        GeometryReader { geometry in
            Image(uiImage: photo.image)
                .resizable()
                .scaledToFill()
                .frame(
                    width: geometry.size.width,
                    height: geometry.size.width
                )
                .clipped()
        }
        .aspectRatio(1, contentMode: .fit)
    }
}
```

Advanced Techniques

1. Scroll Position Preservation

When changing zoom levels, preserve the user's scroll position relative to the content they were viewing. Use ScrollViewReader and programmatic scrolling to maintain context:

```
@State private var scrollTarget: UUID?

ScrollViewReader { proxy in
    ScrollView {
        LazyVGrid(...) {
            ForEach(photos) { photo in
                PhotoThumbnailView(photo: photo)
                    .id(photo.id)
```

```
                }
            }
        }
        .onChange(of: columnCount) { _ in
            if let target = scrollTarget {
                withAnimation {
                    proxy.scrollTo(target, anchor: .center)
                }
            }
        }
    }
```

2. Performance Optimization

For large photo libraries, optimize performance with these techniques:

- Use thumbnail images instead of full-resolution photos in the grid
- Implement image caching with NSCache or a library like Kingfisher
- Use drawingGroup() modifier for complex grid layouts
- Consider using PHCachingImageManager for Photos framework integration

3. Gesture Conflict Resolution

If your grid is inside a NavigationView or has tap gestures, you may need to manage gesture priorities:

```
.simultaneousGesture(magnificationGesture)
// or
.highPriorityGesture(magnificationGesture)
```

Integration with Photos Framework

To build a complete photo gallery app, integrate with the Photos framework to access the user's photo library:

```
import Photos
import SwiftUI

class PhotoLibraryManager: ObservableObject {
    @Published var photos: [Photo] = []

    private let imageManager = PHCachingImageManager()
    private var assets: [PHAsset] = []

    func requestAuthorization() {
        PHPhotoLibrary.requestAuthorization(for: .readWrite) { status in
            if status == .authorized {
                self.fetchPhotos()
            }
        }
    }

    private func fetchPhotos() {
        let fetchOptions = PHFetchOptions()
        fetchOptions.sortDescriptors = [
            NSSortDescriptor(key: "creationDate", ascending: false)
        ]

        let results = PHAsset.fetchAssets(with: .image, options: fetchOptions)

        results.enumerateObjects { asset, _, _ in
            self.assets.append(asset)
        }

        // Start caching thumbnails
        let targetSize = CGSize(width: 200, height: 200)
        imageManager.startCachingImages(
            for: assets,
            targetSize: targetSize,
            contentMode: .aspectFill,
            options: nil
        )
    }

    func thumbnail(for index: Int, size: CGSize) async -> UIImage? {
        guard index < assets.count else { return nil }

        let asset = assets[index]
        let options = PHImageRequestOptions()
        options.deliveryMode = .opportunistic
        options.isNetworkAccessAllowed = true

        return await withCheckedContinuation { continuation in
            imageManager.requestImage(
```

```
                for: asset,
                targetSize: size,
                contentMode: .aspectFill,
                options: options
            ) { image, _ in
                continuation.resume(returning: image)
            }
        }
    }
}
```

Summary

Replicating the iOS Photos app's adaptive grid navigation involves combining several SwiftUI features: MagnificationGesture for detecting pinch-to-zoom, LazyVGrid for efficient grid layouts, and careful state management for smooth transitions. The key is to define discrete zoom levels with corresponding column counts and implement snapping behavior for a polished feel.

Key Takeaways:

- Use MagnificationGesture to detect pinch gestures and track scale
- Calculate column count dynamically based on zoom scale
- Implement snap-to-level behavior for discrete zoom states
- Add haptic feedback for a native iOS feel
- Use animations for smooth transitions between grid layouts
- Optimize performance with thumbnail caching for large libraries