Advances in Collection View Layout
Complex layouts made simple

Steve Breen, UIKit Framework Engineer
Troy Stephens, AppKit Framework Engineer
Dersu Abolfathi, App Store Engineer
Current state-of-the-art
A new approach
Demos
Advanced layouts
Current State-of-the-Art
UICollectionViewFlowLayout
In the beginning, there was flow

Useful for many common designs

Line-based

What about today’s apps?
Today’s App Designs

Complex

Custom layouts
Building Custom Layouts

Boilerplate code

Performance considerations

Supplementary and decoration view challenges

Self-sizing challenges
Compositional Layout
Compositional Layout

Composable

Flexible

Fast
Compositional Layout

Composing small layout groups together

Layout groups are line-based

Composition instead of subclassing
Show me some code already!
// Create a List by Specifying Three Core Components: Item, Group and Section

let size = NSCollectionLayoutSize(widthDimension: .fractionalWidth(1.0),
                                   heightDimension: .absolute(44.0))

let item = NSCollectionLayoutItem(layoutSize: size)

let group = NSCollectionViewGroup.horizontal(layoutSize: size, subitems: [item])

let section = NSCollectionViewSection(group: group)

let layout = UICollectionViewCompositionalLayout(section: section)
Item > Group > Section > Layout
Item > Group > Section > Layout
Item > Group > Section > Layout
Item > Group > Section > Layout
Item > Group > Section > Layout
Core Concepts

NSCollectionLayoutSize

NSCollectionLayoutItem

NSCollectionLayoutGroup

NSCollectionLayoutSection
Everything has an explicit size

Size = Width + Height dimension
NSCollectionLayoutDimension

Axis independent

Four ways to define

class NSCollectionLayoutDimension {
    class func fractionalWidth(_ fractionalWidth: CGFloat) -> Self
    class func fractionalHeight(_ fractionalHeight: CGFloat) -> Self
    class func absolute(_ absoluteDimension: CGFloat) -> Self
    class func estimated(_ estimatedDimension: CGFloat) -> Self
}

let widthDimension = NSCollectionLayoutDimension.fractionalWidth(0.5)
let heightDimension = NSCollectionLayoutDimension.fractionalHeight(0.3)
let size = NSCollectionLayoutDimension(widthDimension: .fractionalWidth(0.25), heightDimension: .fractionalWidth(0.25))
let heightDimension = NSCollectionLayoutDimension.absolute(200)
let heightDimension = NSCollectionLayoutDimension.estimated(200)
let heightDimension = NSCollectionLayoutDimension.estimated(200)
NSCollectionLayoutItem

Cell or Supplementary

class NSCollectionLayoutItem {
    convenience init(layoutSize: NSCollectionLayoutSize)
    var contentInsets: NSDirectionalEdgeInsets
}
NSCollectionLayoutGroup

Basic unit of layout

Three forms: Horizontal, vertical, and custom

class NSCollectionLayoutGroup: NSCollectionLayoutItem {
    class func horizontal(layoutSize: NSCollectionLayoutSize,
                           subitems: [NSCollectionLayoutItem]) -> Self
    class func vertical(layoutSize: NSCollectionLayoutSize,
                        subitems: [NSCollectionLayoutItem]) -> Self
    class func custom(layoutSize: NSCollectionLayoutSize,
                       itemProvider: NSCollectionLayoutGroupCustomItemProvider) -> Self
}
class NSCollectionLayoutSection {
    convenience init(layoutGroup: NSCollectionLayoutGroup)

    var contentInsets: NSDirectionalEdgeInsets
}

Section’s layout

Additional cool features
UICollectionViewCompositionalLayout

Same API for iOS, tvOS, and macOS

Repeating

Per-section

class UICollectionViewCompositionalLayout: UICollectionViewLayout {
    init(section: NSCollectionLayoutSection)
    init(sectionProvider: @escaping SectionProvider)
}
Demo

Show me some more code already!
Advanced Layouts
NSCollectionLayoutSupplementaryItem

Badges

Headers

Footers
NSCollectionLayoutSupplementaryItem

Simplifies using supplementaries

Anchored to item or group
NSCollectionLayoutAnchor

- [.trailing, .top]
- [.top]
- [.bottom]

Defines position relative to the host geometry
// NSCollectionLayoutAnchor

let badgeAnchor = NSCollectionLayoutAnchor(edges: [.top, .trailing],
    fractionalOffset: CGPoint(x: 0.3, y: -0.3))

let badgeSize = NSCollectionLayoutSize(widthDimension: .absolute(20),
    heightDimension: .absolute(20))

let badge = NSCollectionLayoutSupplementaryItem(layoutSize: badgeSize,
    elementKind: "badge",
    containerAnchor: badgeAnchor)

let item = NSCollectionLayoutItem(layoutSize: itemSize, supplementaryItems: [badge])
What About Headers and Footers?

Boundary supplementary item

Section or entire layout

Pinning
let header = NSCollectionViewBoundarySupplementaryItem(layoutSize: headerSize,
    elementKind: "header",
    alignment: .top)

let footer = NSCollectionViewBoundarySupplementaryItem(layoutSize: footerSize,
    elementKind: "footer",
    alignment: .bottom)

header.pinToVisibleBounds = true
section.boundarySupplementaryItems = [header, footer]
// Section Background Decoration Views

let background = NSCollectionLayoutDecorationItem.background(elementKind: "background")
section.decorationItems = [background]

// Register Our Decoration View with the Layout
layout.register(MyCoolDecorationView.self, forDecorationViewOfKind: "background")
Estimated Self-Sizing

Fast

Per-axis

Supplementary items

Dynamic text
// Estimated Self-Sizing

let headerSize = NSCollectionLayoutSize(widthDimension: .fractionalWidth(1.0),
                                        heightDimension: .estimated(44.0))

let header = NSCollectionLayoutBoundarySupplementaryItem(layoutSize: headerSize,
                                                          elementKind: "header",
                                                          alignment: .top)

header.pinToVisibleBounds = true

section.boundarySupplementaryItems = [header, footer]
Nested NSCollectionViewLayoutGroup

NSCollectionViewLayoutGroup is-a NSCollectionViewLayoutItem

No limit to nesting depth

Unlocks new designs
// Nested NSCollectionViewGroup

let leadingItem = NSCollectionViewItem(layoutSize: leadingItemSize)

let trailingItem = NSCollectionViewItem(layoutSize: trailingItemSize)

let trailingGroup = NSCollectionViewGroup.vertical(layoutSize: trailingGroupSize)
                        subitem: trailingItem,
                        count: 2)

let containerGroup = NSCollectionViewGroup.horizontal(layoutSize: containerGroupSize,
                                                      subitems: [leadingItem,
                                                                trailingGroup])
Nested CollectionViews

Challenging

Lots of bookkeeping

Common pattern
Nested CollectionViews

Challenging

Lots of bookkeeping

Common pattern
// Orthogonal Scrolling Sections

section.orthogonalScrollingBehavior = .continuous
// Orthogonal Scrolling Sections

enum UICollectionViewLayoutSectionOrthogonalScrollingBehavior: Int {
    case none
    case continuous
    case continuousGroupLeadingBoundary
    case paging
    case groupPaging
    case groupPagingCentered
}
Case Study
App Store adoption
RTL Support for Free
Compositional Layout on the App Store

No more horizontal collection views

Less code, easy to reason about
Demo
Compositional layout on macOS
Compositional Layout Has It All

iOS, tvOS, and macOS

Custom CollectionView layouts for a fraction of the work

Makes CollectionView much more versatile

Tighter UI iteration
More Information

developer.apple.com/wwdc19/215

<table>
<thead>
<tr>
<th>Event</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advances in UI Data Sources</td>
<td>Wednesday, 2:00</td>
</tr>
<tr>
<td>UIKit and Collection Lab</td>
<td>Thursday, 9:00</td>
</tr>
</tbody>
</table>