CUSTOM GRAPHIC ELEMENTS

STRV

OVERVIEW

01

OVERVIEW

- High-level graphics frameworks
  - Abstracted from hardware
  - Handles complex tasks behind the scenes
  - Easier for developers to use
  - *Examples: UIKit, SwiftUI, RealityKit, ARKit, SpriteKit*
- Low-level graphics frameworks
  - Closer to hardware, with more control
  - Optimized for memory, performance, and resources
  - *Examples: Core Graphics, Metal*

SWIFTUI

02

SWIFTUI

- Mostly UIKit under the hood
- Interoperability UIKit <-> SwiftUI
  - SwiftUI to UIKit: UIHostingController
  - UIKit to SwiftUI: UIViewRepresentable
- Less code, more clarity
- Declarative approach for defining view behavior using view modifiers

SWIFTUI

- Mostly UIKit under the hood
- Interoperability UIKit <-> SwiftUI
  - SwiftUI to UIKit: UIHostingController
  - UIKit to SwiftUI: UIViewRepresentable
- Less code, more clarity
- Declarative approach for defining view behavior using view modifiers

```
class CustomLabelView: UIView {
    private let label: UILabel = {
        let label = UILabel()
        label.text = "Hello world!!!"
        label.translatesAutoresizingMaskIntoConstraints = false
        return label
    }()

    override init(frame: CGRect) {
        super.init(frame: frame)
        setupView()
    }

    required init?(coder: NSCoder) {
        super.init(coder: coder)
        setupView()
    }

    private func setupView() {
        addSubview(label) // Přidání labelu jako subview

        NSLayoutConstraint.activate([  // Nastavení constraintů pomocí NSLayoutConstraint
            label.centerXAnchor.constraint(equalTo: centerXAnchor),
            label.centerYAnchor.constraint(equalTo: centerYAnchor),
            label.leadingAnchor.constraint(greaterThanOrEqualTo: leadingAnchor, constant: 16),
            label.trailingAnchor.constraint(lessThanOrEqualTo: trailingAnchor, constant: -16)
        ])
    }
}
```

SWIFTUI

- Mostly UIKit under the hood
- Interoperability UIKit <-> SwiftUI
  - SwiftUI to UIKit: UIHostingController
  - UIKit to SwiftUI: UIViewRepresentable
- Less code, more clarity
- Declarative approach for defining view behavior using view modifiers

```
struct LoadingView: View {
    var body: some View {
        Text("Hello world!!!")
    }
}
```

SWIFTUI

- Mostly UIKit under the hood
- Interoperability UIKit <-> SwiftUI
  - SwiftUI to UIKit: UIHostingController
  - UIKit to SwiftUI: UIViewRepresentable
- Less code, more clarity
- Declarative approach for defining view behavior using view modifiers

```
Rectangle()
    .foregroundStyle(Color.red)
    .opacity(0.5)
    .frame(width: 100, height: 100)
```

PROPERTY WRAPPERS

- Adds behavior to common properties
  - managing, observing, or validating data, without cluttering code
- Types
  - @State – Manages local view state. Change triggers redrawing view
  - @Binding – Shares state between parent and child views. Change triggers redrawing view
  - @Environment – Accesses shared environment data
  - @StateObject/@ObservedObject: – Tracks observable objects shared across views

VIEW MODIFIERS

- Ability to wrap in another view
  and override behaviour of given content
- Reduces code duplication by grouping frequently
  used modifiers together
- Ability to encapsulate property wrappers
  - @State, @Binding, etc…

```
struct Title: ViewModifier {
    func body(content: Content) -> some View {
        content
            .font(.largeTitle)
            .foregroundStyle(.white)
            .padding()
            .background(.blue)
            .clipShape(.rect(cornerRadius: 10))
    }
}
```

```
Text("hello world!!!")
    .modifier(Title())
```

VIEW MODIFIERS

- Ability to wrap in another view
  and override behaviour of given content
- Reduces code duplication by grouping frequently used modifiers together
- Ability to encapsulate property wrappers
  - @State, @Binding, etc…

```
struct Title: ViewModifier {
    func body(content: Content) -> some View {
        content
            .font(.largeTitle)
            .foregroundStyle(.white)
            .padding()
            .background(.blue)
            .clipShape(.rect(cornerRadius: 10))
    }
}

extension View {
    func title() -> some View {
        self.modifier(Title())
    }
}


Text("hello world!!!")
    .title()
```

VIEW MODIFIERS

- Ability to wrap in another view
  and override behaviour of given content
- Reduces code duplication by grouping frequently used modifiers together
- Ability to encapsulate property wrappers
  - @State, @Binding, etc…

```
struct OnFirstAppear: ViewModifier {
    @State var viewDidAppear = false
    var action: () -> Void

    func body(content: Content) -> some View {
        content
            .onAppear {
                if !viewDidAppear {
                    viewDidAppear = true
                    action()
                }
            }
    }
}
```

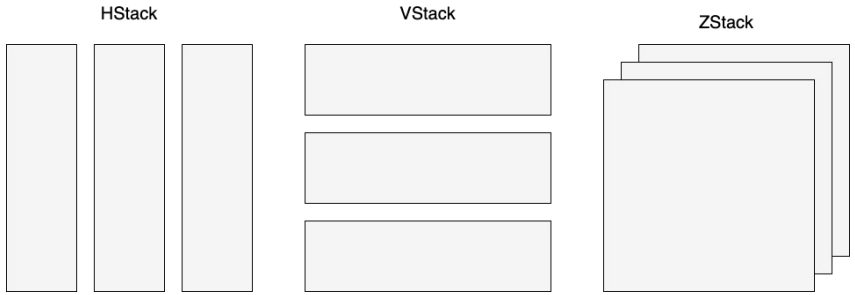
VIEW STYLES

- Modifier for overwriting default style of basic SwiftUI components
  - .buttonStyle, .labelStyle, .toggleStyle, .pickerStyle (not customizable)
- Reduces code duplication by reusing styles across views
- Enhances maintainability of large apps
- Creates a unified design for a consistent and improved UX

```
struct BlueButtonStyle: ButtonStyle {
    func makeBody(configuration: Configuration) -> some View {
        configuration.label
            .padding()
            .background(Color.blue)
    }
}
```

STACKS

- Basic layout containers
  - VStack, HStack, ZStack
- Identify primitive elements
  and then group into containers
- Before coding, consider
  - Potential new features
  - Development time
  - Separation of concerns
    (view models for UI components
    not for entire screens)
- Complex layouts with Layout protocol
  - FlowLayout etc…
- More about this at https://www.swiftuifieldguide.com



Primitive element


VStack
Text
Text


HStack
Image
Text
Text
Button

VStack


Image
Text
Text
Button
Image
Text
Text
Button
Image
Text
Text
Button

HStack

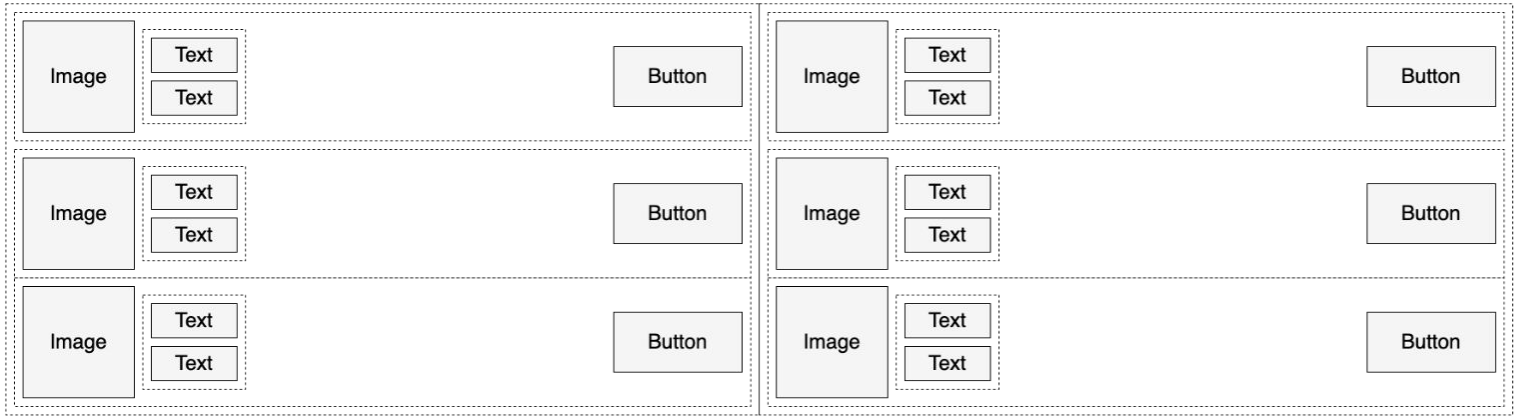
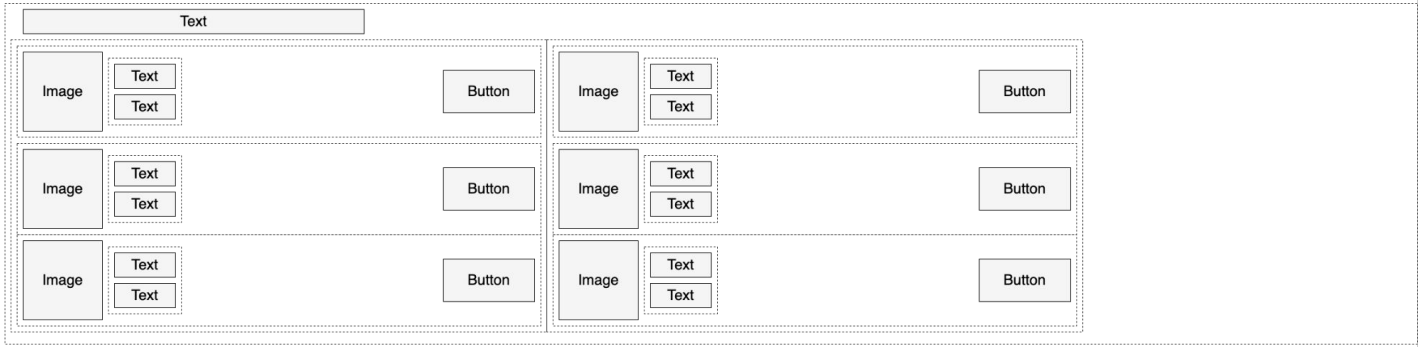
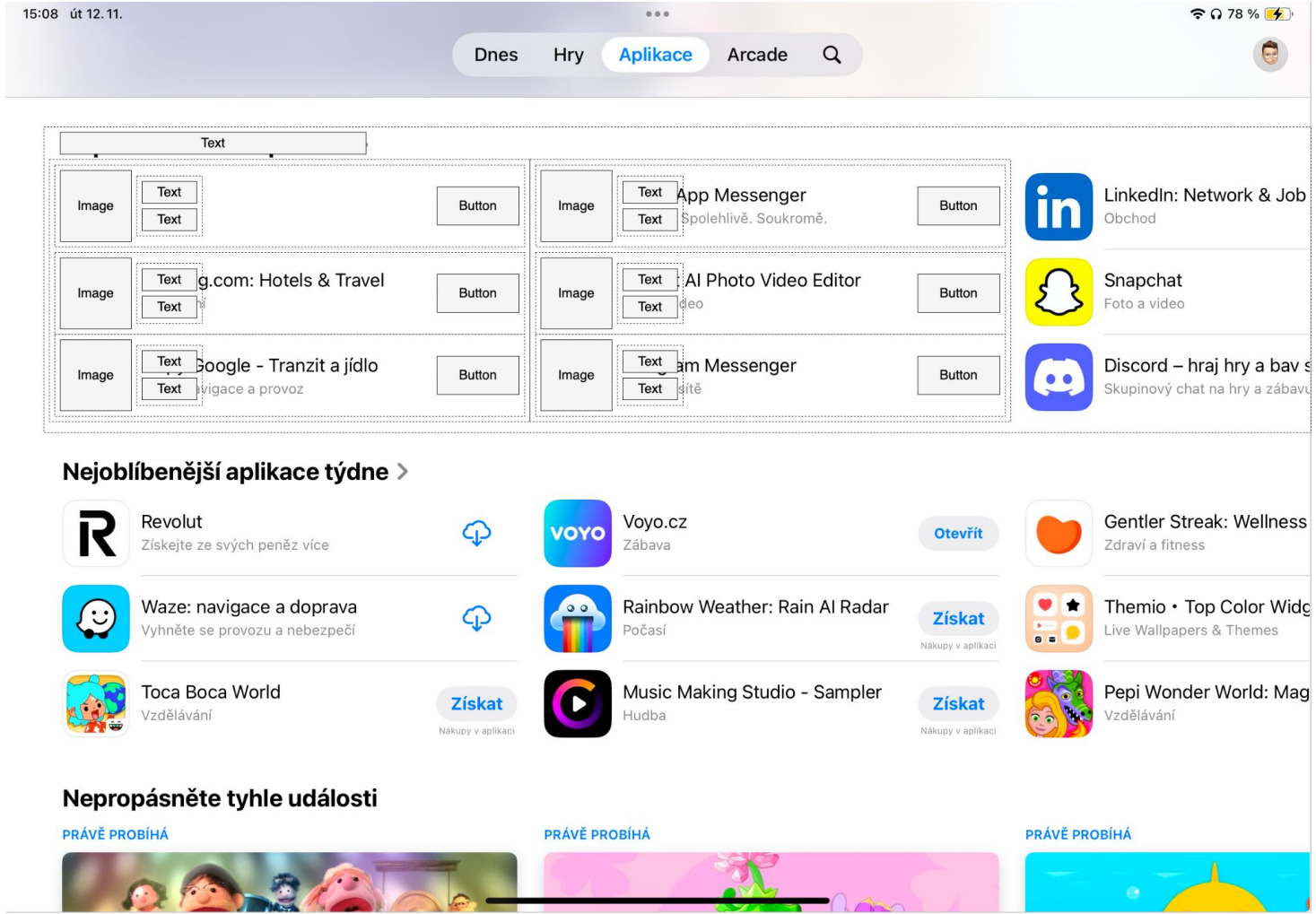


Image
Text
Text
Button
Image
Text
Text
Button
Image
Text
Text
Button
Image
Text
Text
Button
Image
Text
Text
Button
Image
Text
Text
Button

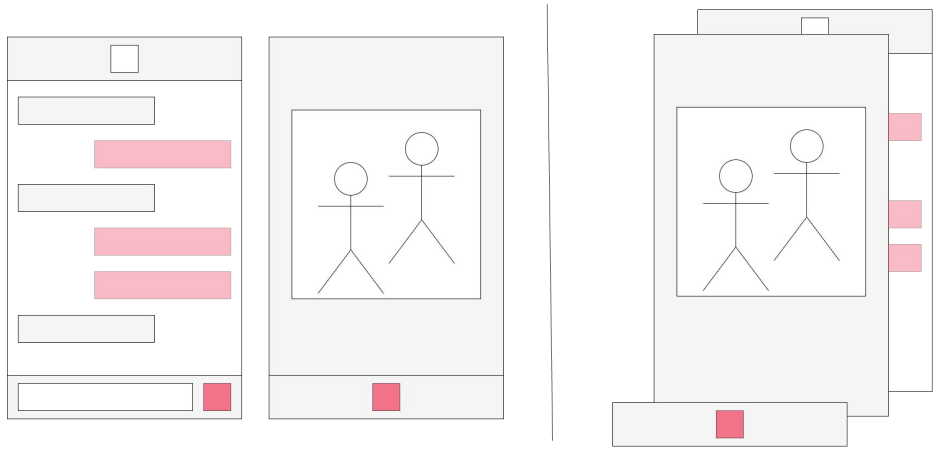
VStack


Text
Image
Text
Text
Button
Image
Text
Text
Button
Image
Text
Text
Button
Image
Text
Text
Button
Image
Text
Text
Button
Image
Text
Text
Button


15:08 út 12. 11.
78 %
Dnes
Hry
Aplikace
Arcade
Text
Image
Text
Text
Button
App Messenger
Společně. Soukromě.
LinkedIn: Network & Job
Obchod
Snapchat
Foto a video
Discord – hraj hry a bav
Skupinový chat na hry a zábavu
Nejoblíbenější aplikace týdne
Revolut
Získejte ze svých peněz více
Waze: navigace a doprava
Vyhněte se provozu a nebezpečí
Toca Boca World
Vzdělávání
Získat
Nákupy v aplikaci
Voyo.cz
Zábava
Otevřít
Rainbow Weather: Rain AI Radar
Počasí
Music Making Studio - Sampler
Hudba
Gentler Streak: Wellness
Zdraví a fitness
Themio • Top Color Widg
Live Wallpapers & Themes
Pepi Wonder World: Mag
Vzdělávání
Nepropásněte tyhle události
PRÁVĚ PROBÍHÁ
PRÁVĚ PROBÍHÁ
PRÁVĚ PROBÍHÁ

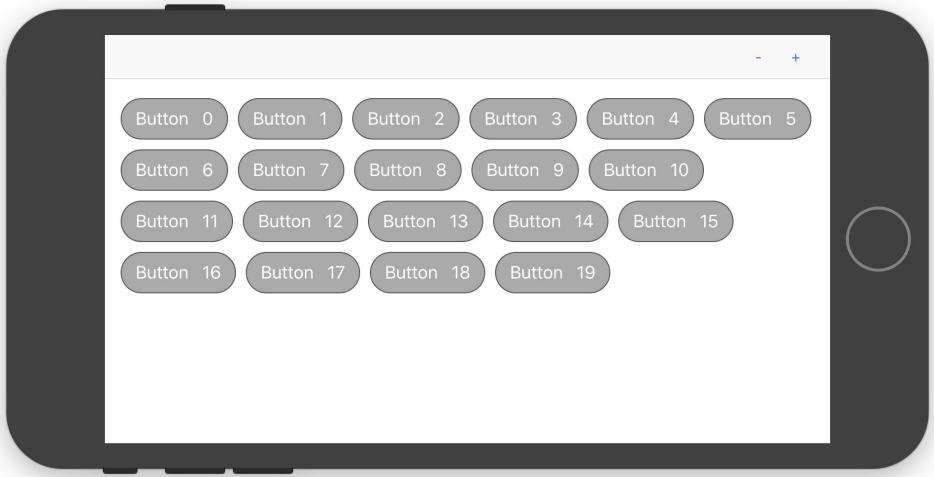
STACKS

- Basic layout containers
  - VStack, HStack, ZStack
- Identify primitive elements
  and then group into containers
- Before coding, consider
  - Potential new features
  - Development time
  - Separation of concerns
    (view models for UI components
    not for entire screens)
- Complex layouts with Layout protocol
  - FlowLayout etc…
- More about this at https://www.swiftuifieldguide.com



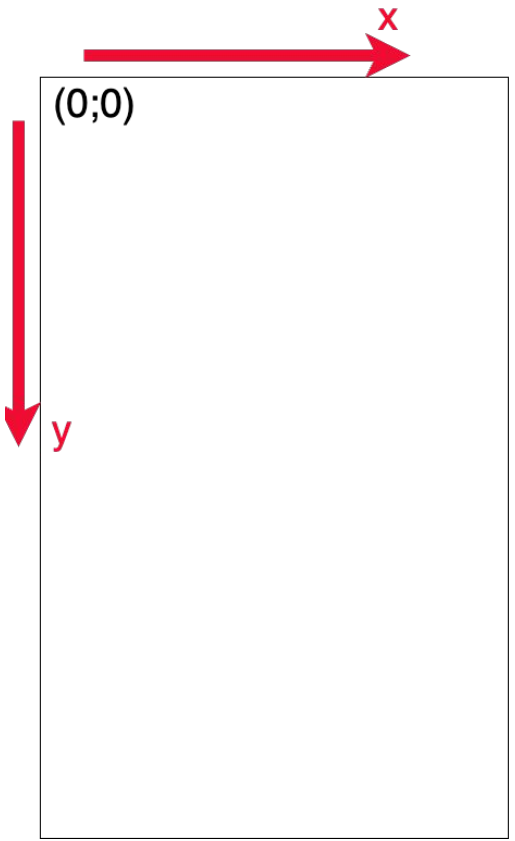
STACKS

- Basic layout containers
  - VStack, HStack, ZStack
- Identify primitive elements
  and then group into containers
- Before coding, consider
  - Potential new features
  - Development time
  - Separation of concerns
    (view models for UI components
    not for entire screens)
- Complex layouts with Layout protocol
  - FlowLayout etc…
- More about this at https://www.swiftuifieldguide.com



GEOMETRYREADER

- Use when basic containers aren’t enough
- Own coordinate system for precise layout control
- Reads proposed size by parent
    - e.g., setting text width to 1/4 of screen

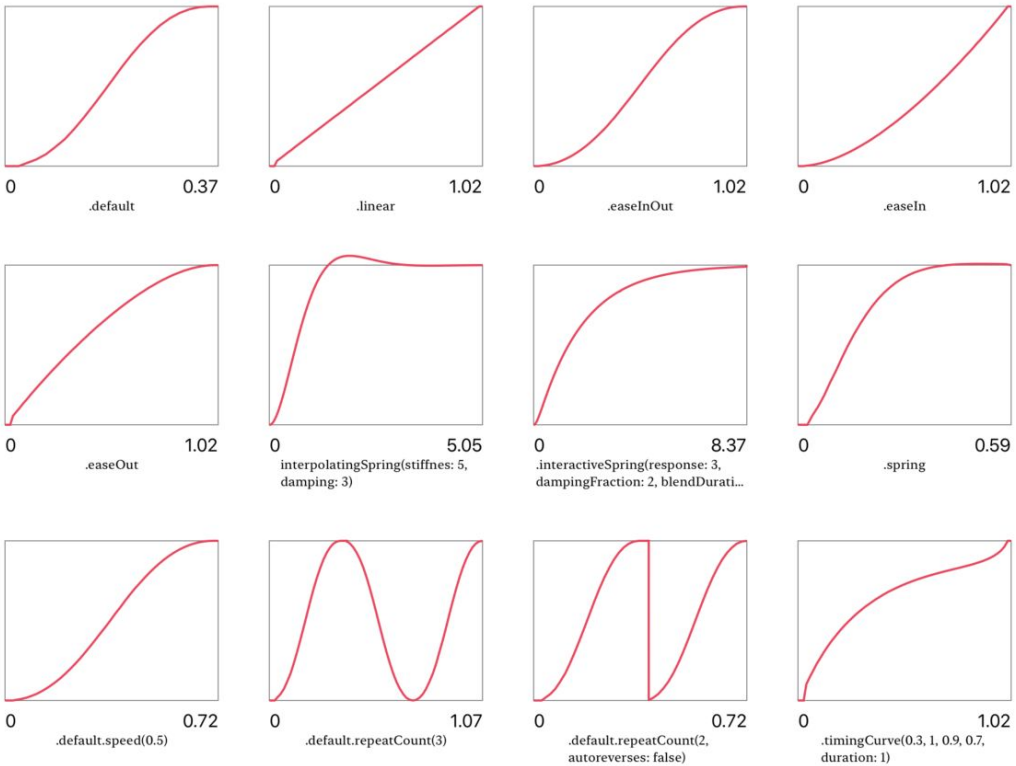


ANIMATIONS

- The change in value is continuous, not discrete.
  - Instead of shift 1 (true) to 0 (false),
    the value gradually decreases.
  - Example: Instead of moving from 1 → 0,
    the value transitions smoothly, e.g., 1 → 0.9 → 0.8 → … → 0
  - Timing curve of animation is adjustable

```
struct AnimateButton: View {
    @State var animatedColor: Color = .red
    var body: some View {
        Button("Start animation") {
            withAnimation(.bouncy.repeatForever(autoreverses: true)) {
                animatedColor = .green
            }
        }
        .padding()
        .background(animatedColor)
    }
}
```

ANIMATIONS


0
0.37
.default
0
1.02
.linear
0
1.02
.easeInOut
0
1.02
.easeIn
0
1.02
.easeOut
0
5.05
interpolatingSpring(stiffnes: 5, damping: 3)
0
8.37
.interactiveSpring(response: 3, dampingFraction: 2, blendDurati...
0
0.59
.spring
0
0.72
.default.speed(0.5)
0
1.07
.default.repeatCount(3)
0
0.72
.default.repeatCount(2, autoreverses: false)
0
1.02
.timingCurve(0.3, 1, 0.9, 0.7, duration: 1)

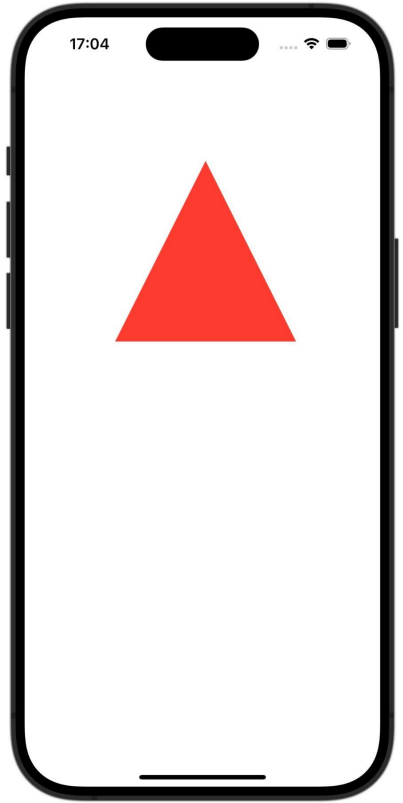
BEZIERPATH

03

PATH

- Custom shapes
  - Eg. Map regions

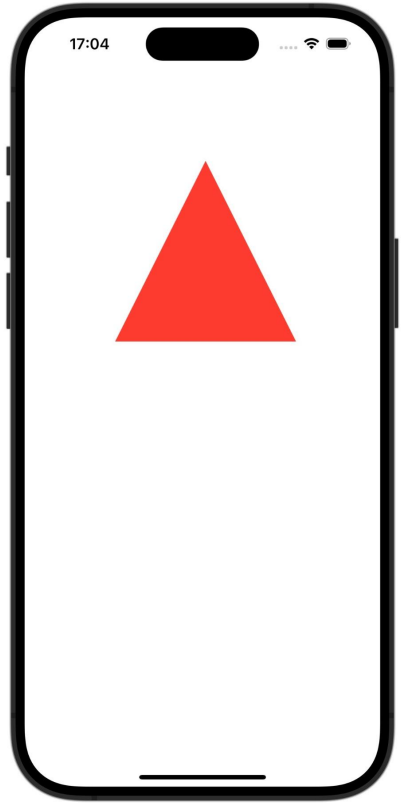
```
Path { path in
    path.move(to: CGPoint(x: 200, y: 100))
    path.addLine(to: CGPoint(x: 100, y: 300))
    path.addLine(to: CGPoint(x: 300, y: 300))
    path.addLine(to: CGPoint(x: 200, y: 100))
}
```



PATH

- Custom shapes
  - Eg. Map regions

```
Path { path in
    path.move(to: CGPoint(x: 200, y: 100))
    path.addLine(to: CGPoint(x: 100, y: 300))
    path.addLine(to: CGPoint(x: 300, y: 300))
    path.addLine(to: CGPoint(x: 200, y: 100))
}
```

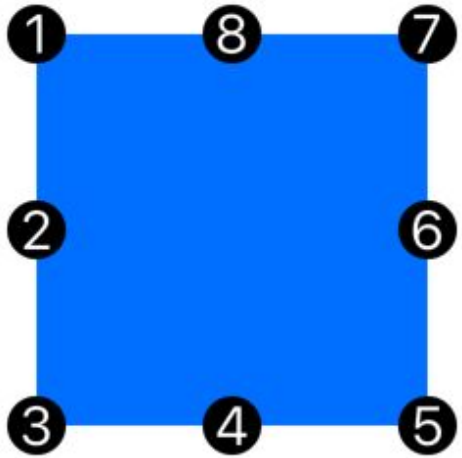


PATH

```
var firstPoint = CGPoint(x: 100, y: 100)
var secondPoint = CGPoint(x: 100, y: 200)
var thirdPoint = CGPoint(x: 100, y: 300)
var fourthPoint = CGPoint(x: 200, y: 300)
var fifthPoint = CGPoint(x: 300, y: 300)
var sixthPoint = CGPoint(x: 300, y: 200)
var seventhPoint = CGPoint(x: 300, y: 100)
var eighthPoint = CGPoint(x: 200, y: 100)
```

SQUARE

```
var square: Path {
    Path { path in
        path.move(to: firstPoint)
        path.addLine(to: secondPoint)
        path.addLine(to: thirdPoint)
        path.addLine(to: fourthPoint)
        path.addLine(to: fifthPoint)
        path.addLine(to: sixthPoint)
        path.addLine(to: seventhPoint)
        path.addLine(to: eighthPoint)
        path.closeSubpath()
    }
}
```

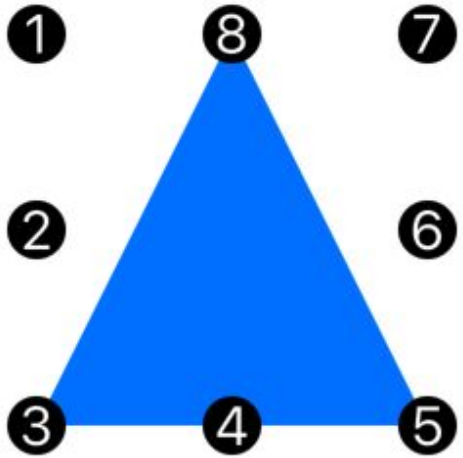


TRIANGLE

```
var triangle: Path {
    Path { path in
        path.move(to: eighthPoint)
        path.addLine(to: thirdPoint)
        path.addLine(to: fifthPoint)
        path.closeSubpath() // back to eightPoint
    }
}
```

TRIANGLE

```
var triangle: Path {
    Path { path in
        path.move(to: eighthPoint)
        path.addLine(to: thirdPoint)
        path.addLine(to: fifthPoint)
        path.closeSubpath() // back to eightPoint
    }
}
```

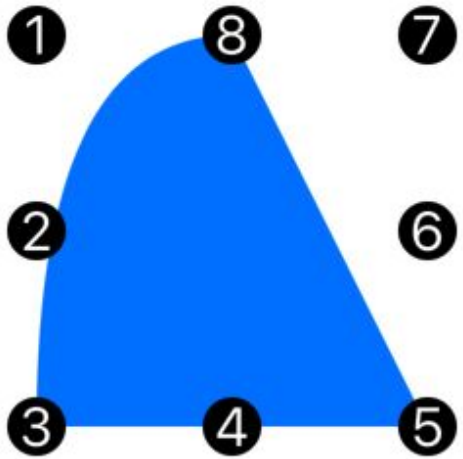


CURVE1

```
var curve1: Path {
    Path { path in
        path.move(to: eighthPoint)
        path.addQuadCurve(
            to: thirdPoint,
            control: firstPoint
        )
        path.addLine(to: fifthPoint)
        path.closeSubpath() // back to eightPoint
    }
}
```

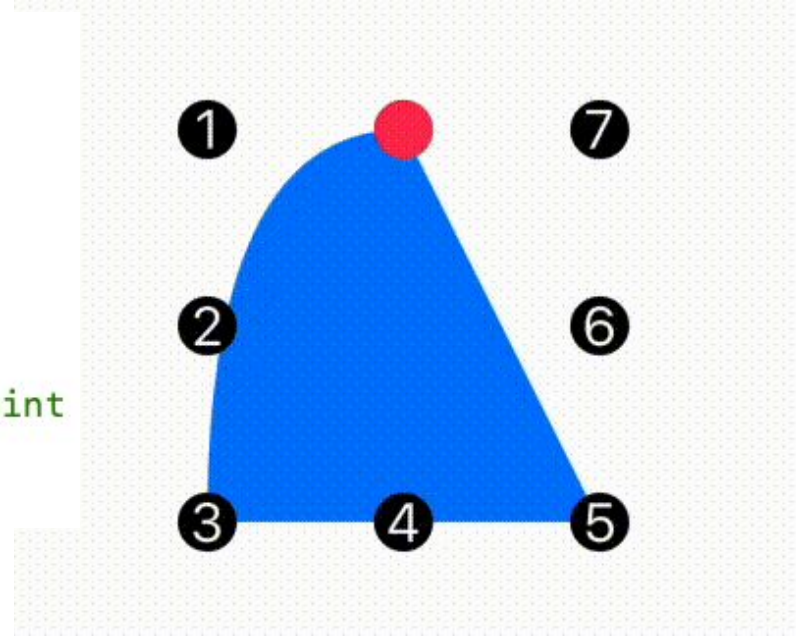
CURVE1

```
var curve1: Path {
    Path { path in
        path.move(to: eighthPoint)
        path.addQuadCurve(
            to: thirdPoint,
            control: firstPoint
        )
        path.addLine(to: fifthPoint)
        path.closeSubpath() // back to eightPoint
    }
}
```



CURVE1

```
var curve1: Path {
    Path { path in
        path.move(to: eighthPoint)
        path.addQuadCurve(
            to: thirdPoint,
            control: firstPoint
        )
        path.addLine(to: fifthPoint)
        path.closeSubpath() // back to eightPoint
    }
}
```

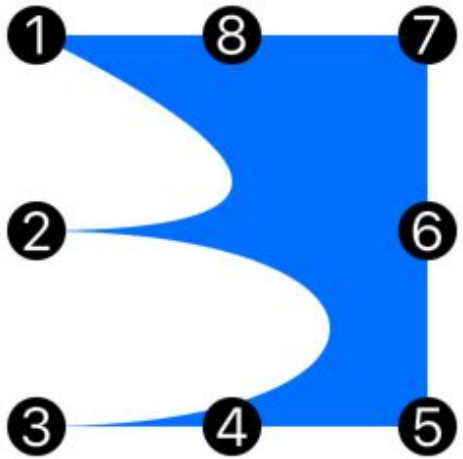


CURVE2

```
var curve2: Path {
    Path { path in
        path.move(to: firstPoint)
        path.addQuadCurve(
            to: secondPoint,
            control: sixthPoint
        )
        path.addCurve(
            to: thirdPoint,
            control1: sixthPoint,
            control2: fifthPoint
        )
        path.addLine(to: fourthPoint)
        path.addLine(to: fifthPoint)
        path.addLine(to: sixthPoint)
        path.addLine(to: seventhPoint)
        path.addLine(to: eighthPoint)
        path.closeSubpath()
    }
}
```

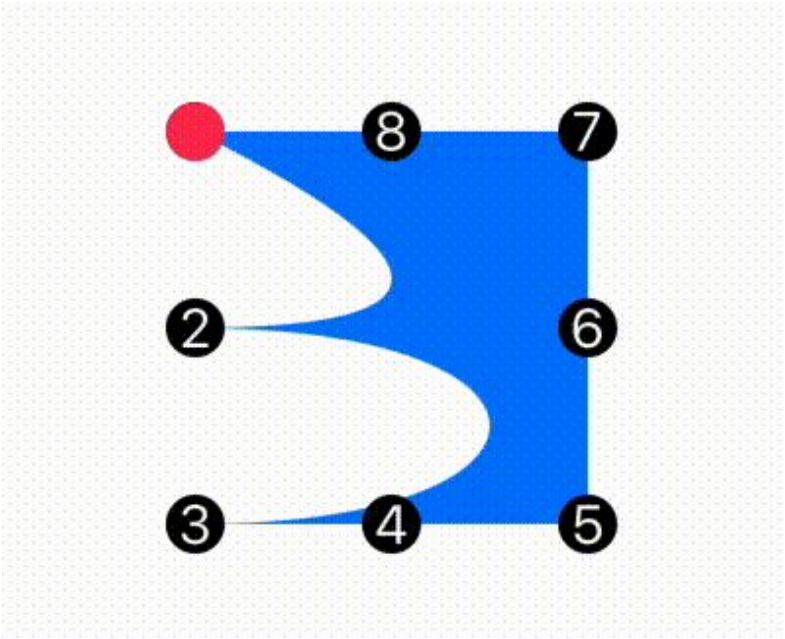
CURVE2

```
var curve2: Path {
    Path { path in
        path.move(to: firstPoint)
        path.addQuadCurve(
            to: secondPoint,
            control: sixthPoint
        )
        path.addCurve(
            to: thirdPoint,
            control1: sixthPoint,
            control2: fifthPoint
        )
        path.addLine(to: fourthPoint)
        path.addLine(to: fifthPoint)
        path.addLine(to: sixthPoint)
        path.addLine(to: seventhPoint)
        path.addLine(to: eighthPoint)
        path.closeSubpath()
    }
}
```



CURVE2

```
var curve2: Path {
    Path { path in
        path.move(to: firstPoint)
        path.addQuadCurve(
            to: secondPoint,
            control: sixthPoint
        )
        path.addCurve(
            to: thirdPoint,
            control1: sixthPoint,
            control2: fifthPoint
        )
        path.addLine(to: fourthPoint)
        path.addLine(to: fifthPoint)
        path.addLine(to: sixthPoint)
        path.addLine(to: seventhPoint)
        path.addLine(to: eighthPoint)
        path.closeSubpath()
    }
}
```

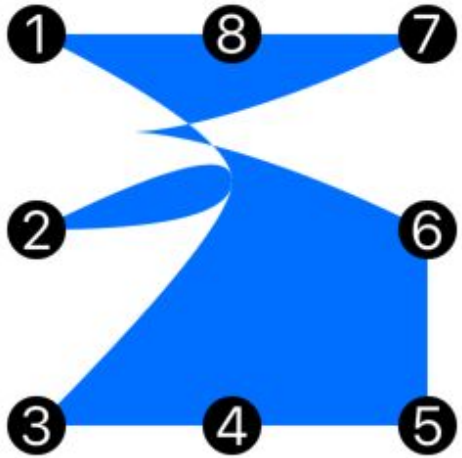


CURVE3

```
var curve3: Path {
    Path { path in
        path.move(to: firstPoint)
        path.addQuadCurve(
            to: secondPoint,
            control: sixthPoint
        )
        path.addQuadCurve(
            to: thirdPoint,
            control: seventhPoint
        )
        path.addLine(to: fourthPoint)
        path.addLine(to: fifthPoint)
        path.addLine(to: sixthPoint)
        path.addCurve(
            to: seventhPoint,
            control1: firstPoint,
            control2: secondPoint
        )
        path.addLine(to: eighthPoint)
        path.addLine(to: firstPoint)
        path.closeSubpath()
    }
}
```

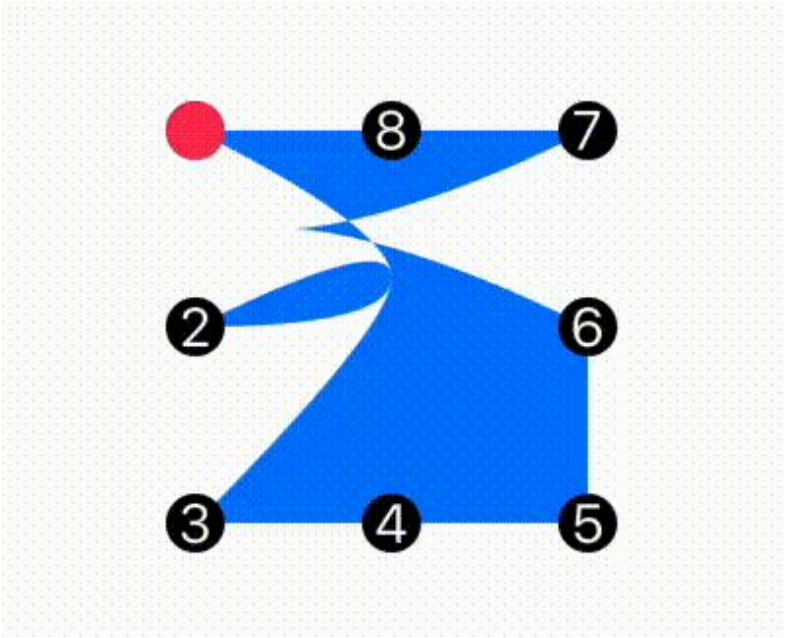
CURVE3

```
var curve3: Path {
    Path { path in
        path.move(to: firstPoint)
        path.addQuadCurve(
            to: secondPoint,
            control: sixthPoint
        )
        path.addQuadCurve(
            to: thirdPoint,
            control: seventhPoint
        )
        path.addLine(to: fourthPoint)
        path.addLine(to: fifthPoint)
        path.addLine(to: sixthPoint)
        path.addCurve(
            to: seventhPoint,
            control1: firstPoint,
            control2: secondPoint
        )
        path.addLine(to: eighthPoint)
        path.addLine(to: firstPoint)
        path.closeSubpath()
    }
}
```



CURVE3

```
var curve3: Path {
    Path { path in
        path.move(to: firstPoint)
        path.addQuadCurve(
            to: secondPoint,
            control: sixthPoint
        )
        path.addQuadCurve(
            to: thirdPoint,
            control: seventhPoint
        )
        path.addLine(to: fourthPoint)
        path.addLine(to: fifthPoint)
        path.addLine(to: sixthPoint)
        path.addCurve(
            to: seventhPoint,
            control1: firstPoint,
            control2: secondPoint
        )
        path.addLine(to: eighthPoint)
        path.addLine(to: firstPoint)
        path.closeSubpath()
    }
}
```



SPRITEKIT

03

SPRITEKIT

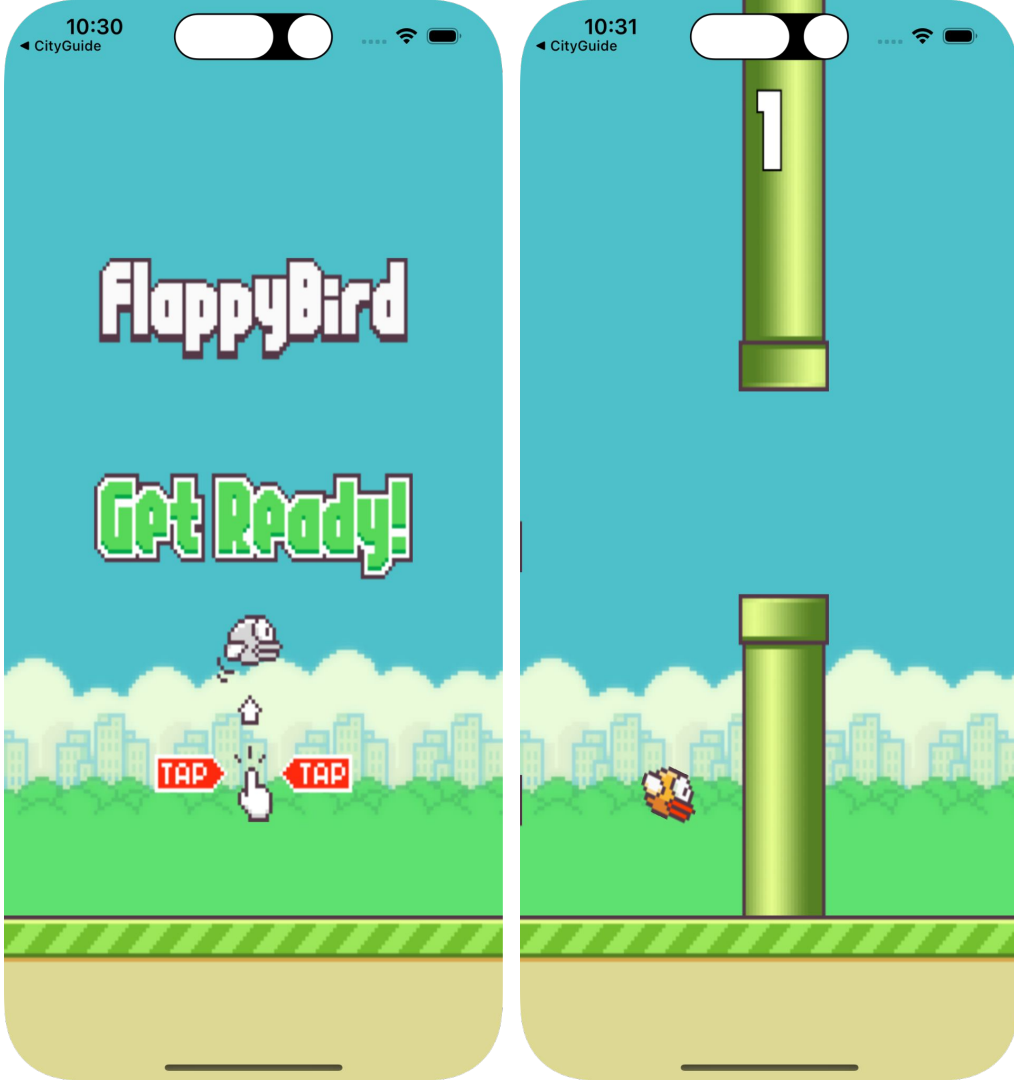
- Ideal for building both 2D games and animations
- Supports realistic object behavior

  for smooth animations
  - Physics (gravitation, object interaction)
  - Lighting / shadows
- Optimized for Apple devices

```
class LoadingScene: SKScene {
    override func didMove(to view: SKView) {
        let loadingSprite = SKSpriteNode(imageNamed: "loadingIcon")
        loadingSprite.position = CGPoint(x: size.width / 2, y: size.height / 2)

        addChild(loadingSprite)

        let rotateAction = SKAction.rotate(byAngle: .pi * 2, duration: 1.0)
        let repeatRotation = SKAction.repeatForever(rotateAction)

        loadingSprite.run(repeatRotation)
    }
}
```

SPRITEKIT

- Ideal for building both 2D games and animations
- Supports realistic object behavior for smooth animations
  - Physics (gravitation, object interaction)
  - Lighting / shadows
- Optimized for Apple devices



LOW-LEVEL

03

CORE GRAPHICS

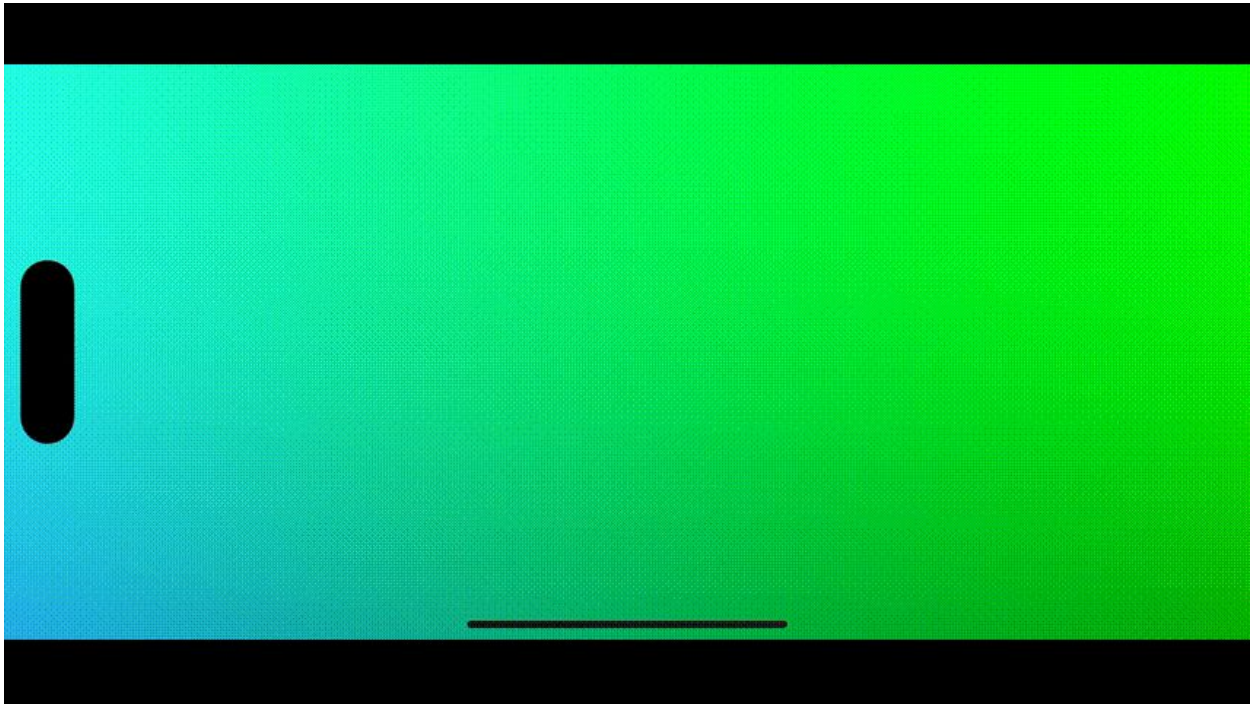
- Used for 2D vector graphics
- Working directly with paths, shapes, colors, and images
- Heavily integrated into higher-level frameworks
    - Essential data types like CGFloat, CGSize, CGColor (reduction of type conversions)

METAL

- Low-level graphics API
- Customization of elements is done by shaders
  - GPU programs for high-performance graphics and parallel data processing
- Provides direct access to the GPU
  - Designed for maximum graphics performance and optimized for Apple hardware
  - Unlike OpenGL, it is built to reduce CPU overhead
- Applications:
  - Game graphics
  - Machine learning

METAL EXAMPLES

Try handling individual pixels with swiftUI



THANK YOU!

Martin Vidovič, Róbert Oravec

STRV

QUESTIONS

STRV