Sample Code

Integrating pointer interactions into your iPad app

Support touch interactions in your iPad app by adding pointer interactions to your views.

Download

iOS 13.4+ | iPadOS 13.4+ | Xcode 12.5+

Overview

This sample code project shows how to use the `UIPointerInteraction` class. A pointer interaction enables support for adding effects to a view, and for customizing the pointer's appearance within a region of an app. It enhances user experience with mouse and trackpad devices and reduces the need for users to move their hands between a hardware keyboard and the touchscreen of an iPad. The sample places four shape views in a canvas within the app's `View Controller`, each of which can be moved around. The shape views are a rectangle, oval, rounded rectangle, and triangle. Each of these takes on a different pointer effect when the user tracks the cursor over them. In addition, the sample shows a custom `UIControl` subclass, to illustrate how controls can adopt pointer interaction.

Add a pointer interaction to a button

The sample adds a pointer interaction to a `UIButton`. Custom pointer interactions require a `UIButtonPointerStyleProvider` function. When applying a style provider function, UIKit hands a pointer effect and pointer shape within that function to an app; the app then returns a pointer style for that particular button. Developers can pick and choose between the proposed effect and shape that the system recommends, replace one or the other, or create an entirely custom style. The sample applies four different pointer effects to its buttons: automatic, highlight, lift, and hover. For the fifth button a custom pointer hover effect is applied, so its effect remains the same size while hovered, and uses a custom `UIPointerShape`.

Provide a custom view for a pointer interaction

The sample implements shape views, which are a subclass of `UIView` called `ShapeView`. They interact with touch events with both the device's touchscreen and touch pad. To visually interact with a shape view a `UIPointerInteraction` object is assigned to it. View controllers adopt `UIPointerInteractionDelegate` to help describe how that pointer interaction operates.

Add a pointer interaction to a view

A pointer interaction is described by a pointer style or visual representation. The pointer style is made up of both a `UIPointerEffect` with a `UITargetedPreview`, and a `UIPointerShape`. A commonly used shape for pointer effects is a rounded rectangle. A pointer shape or `UIPointerShape` requires a `UIBezierPath`, which describes that shape. The sample associates a pointer interaction to a shape view by adding one like this:

```
shapeView.addInteraction(UIPointerInteraction(delegate: self))
```

Create a targeted preview for a pointer interaction

For a shape view to describe its appearance during a pointer interaction, it must provide a `UITargetedPreview`. `UITargetedPreview` gives UIKit a view to which to apply an effect during pointer interactions:

```
func targetedPreview() -> UITargetedPreview? {
    let parameters = UIPreviewParameters()

    // Use the entire view's shape for the preview.
    let visiblePath = viewPath
    parameters.visiblePath = visiblePath

    return UITargetedPreview(view: self, parameters: parameters)
}
```

As the user moves the points within a shape view, the pointer interaction displays this targeted preview. Note that targeted previews are also used with context menus through the use of `UIContextMenuConfiguration`. In the sample, shape views have context menus.

Create a pointer effect for a pointer interaction

A pointer interaction needs a visual effect to describe how it will render the shape view's targeted preview. The oval shape view, for example, uses `UIPointerLiftEffect`, which slightly lifts the targeted preview and slides it around as the pointer is moved within the oval shape. The other shape views have their own pointer effects. The rectangle view uses a `UIPointerEffect`, a rounded rectangle view uses a `UIPointerHighlightEffect`, and a triangle view uses a `UIPointerHoverEffect`, which allows for its `UIPointerShape` to be revealed as a triangle.

Create a shape for a pointer interaction

The Sample creates either region or shape, defined for each of its shape views, so a pointer interaction detects where to interact. The sample also implements `pointerInteraction(_:regionFor:defaultRegion:)` as the `UIPointerInteractionDelegate`. This delegate is called by UIKit as the pointer moves within the pointer interaction's view. Returning a `UIPointerRegion` in which to apply a pointer style or returning `nil` indicates that this interaction does not customize the pointer for the current location.

```
func pointerInteraction(_ interaction: UIPointerInteraction,
                        regionFor request: UIPointerRegionRequest,
                        defaultRegion: UIPointerRegion) -> UIPointerRegion? {
    var pointerRegion: UIPointerRegion? = nil

    if let view = interaction.view as? ShapeView {
        // Pointer has entered one of the shape views.

        // Check for modifiers keys pressed while inside the view path.
        if request.modifiers.contains(.command) && request.modifiers.contains(.alte
            // Command + Option was both pressed, dim the view.
            view.alpha = 0.50
        } else {
            if view.alpha != 1.0 { view.alpha = 1.0 }
        }

        // The user interacted with the inner path region.
        pointerRegion =
            UIPointerRegion(rect: view.innerPath.bounds,
                            identifier: ShapeView.regionIdentifier)
    } else if let view = interaction.view as? AlphaControl {
        // Pointer has entered the alpha control.
        if view.bounds.contains(request.location) {
            pointerRegion =
                UIPointerRegion(rect: view.bounds,
                                identifier: AlphaControl.regionIdentifier)
        }
    }

    return pointerRegion
}
```

The oval shape view, for example, interacts with the pointer from within its oval shaped UIBezierPath.

Interact with views using gesture recognizers

To make the shape views more interactive and provide custom behaviors driven by a mouse or trackpad, the sample attaches four different gesture recognizers to each shape view, to work alongside their pointer interactions.

A UIPanGestureRecognizer moves a shape view and the frame color changes to orange when the command key is pressed during the pan gesture.

A UIPinchGestureRecognizer changes the shape's size with two-finger pinch gesture.

A UITapGestureRecognizer brings the tapped shape view to the front.

A UIHoverGestureRecognizer changes the frame color of a shape view to blue when the cursor is positioned over it. If the user presses the command key while hovering, the frame color toggles to pink.

Below is an example of handling the UIHoverGestureRecognizer to a ShapeView:

```
func hoverShape(_ gestureRecognizer: UIHoverGestureRecognizer) {
    guard let shapeViewToUse = gestureRecognizer.view as? ShapeView else { return }

    switch gestureRecognizer.state {
    case .began, .changed:
        // User hovered within the view's path, change the view's border color.
        var pathViewColor = UIColor.systemTeal
        if gestureRecognizer.modifierFlags.contains(.command) {
            // If the command key is pressed while hovering change frame color to pi
            pathViewColor = UIColor.systemPink
        }
        shapeViewToUse.viewPathColor = pathViewColor
        shapeViewToUse.setNeedsDisplay()
        case .ended, .cancelled:
            // User left the view, restore the border color.
            shapeViewToUse.restoreOuterFrameColor()

        default:
            break
        }
    }
```

Create a gesture recognizer for continuous scrolling

The `UIPanGestureRecognizer` recognizes continuous scrolling that originates from devices like the trackpad. The sample adds a pan gesture recognizer to a custom `UIControl` subclass `AlphaControl`, that recognizes two-finger scroll gesture to change the alpha value of a given color. With `allowedScrollTypesMask` set to `continuous`, apps recognize continuous scrolling. The control's color swatch changes as the user performs a pan scroll gesture, or through a direct touch.

See Also

Essentials

`class UIPointerInteraction`

An interaction that enables support for effects on a view or customizes the pointer's appearance within a region of an app.

`protocol UIPointerInteractionDelegate`

An interface for handling pointer movements within the interaction's view.

{} Enhancing your iPad app with pointer interactions

Provide a great user experience with pointing devices, by incorporating pointer content effects and shape customizations.