Sample Code

Displaying an updating path of a user's location history

Continually update a MapKit overlay displaying the path a user travels.

Download

iOS 16.0+ | iPadOS 16.0+ | Xcode 16.0+

Overview

Travel History is a sample app that allows a user to track their location history, and displays a live-updating path of that history on a map. The app combines location updates from Core Location with a custom MapKit overlay for the location history. MapKit draws the user's path using a custom MapKit overlay renderer.

For this sample app to show a path, it needs to receive a stream of location updates. Install the app on a device, and take a walk with it recording locations. Alternately, use one of the routes in the iOS simulator, such as Freeway Drive or City Bicycle Ride. To set a route in the iOS simulator, see Set a location or route.

Configure a location manager to receive location updates

Before recording the user's path, the app asks the user for permission to access location data while the app is running. After the user grants location authorization, a `CLLocationManager` receives a stream of location updates as the user travels. The app also receives location updates while in the background by setting the `allowsBackgroundLocationUpdates` property of `CLLocationManager` to `true`. This combination of when-in-use authorization and background updates allows the app to receive location updates while in the background, and the system displays a location service indicator when the app isn't in the foreground.

```
locationManager.requestWhenInUseAuthorization()

/// Enable the app to collect location updates while it's in the background.
locationManager.allowsBackgroundLocationUpdates = true
```

The app sets a default value for the `desiredAccuracy` property on the location manager. This value is customizable in a menu to demonstrate how the value of this property affects the accuracy of the recorded data. Lower accuracy values allow the device to conserve power and have less impact on the device's battery life. For example, if this app is for hiking, it needs high-accuracy data to capture a detailed path for the user's hiking route, requiring `kCLLocationAccuracyBest` to facilitate the required detail. In contrast, if the app tracks overall progress between a start and end point without needing the exact path taken, using a lower accuracy value, such as `kCLLocationAccuracyKilometer` or `kCLLocationAccuracyThreeKilometers`, is a better choice because it conserves power.

The app also sets a default value for the `activityType` property, and is configurable in a menu. This property provides a hint to Core Location about the type of travel the device encounters while monitoring location. For example, if someone uses this app often while running, it sets `activityType` to `fitness`. If they use it for providing driving directions, it sets `activityType` to `automotiveNavigation` so that Core Location makes small adjustments to the reported location to match known roads.

```
locationManager.requestWhenInUseAuthorization()

/// Enable the app to collect location updates while it's in the background.
locationManager.allowsBackgroundLocationUpdates = true
```

After the app calls `startUpdatingLocation` on the location manager, Core Location provides location updates to the location manager's delegate. Then the app forwards the location updates to other functions in the app responsible for maintaining the user's location history.

```
func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CL

    // Play a sound so it's easy to tell when a location update occurs while the app
    if chimeOnLocationUpdate && !locations.isEmpty {
        setSessionActiveWithMixing(true) // Ducks the audio of other apps when playi
        playSound()
    }

    // Always process all of the provided locations. Don't assume the array only cor
    for location in locations {
```

```
            displayNewBreadcrumbOnMap(location)
        }
    }
```

Define a custom map overlay data object

MapKit treats overlay data as static when using the system-provided overlay classes, such as `MKPolyline`. For example, apps that only need to show a path with a static set of coordinates, such as the path a user took in the distant past, can use `MKPolyline`. Because this app continually updates with new data, it implements its own data class, called `BreadcrumbPath`, conforming to the `MKOverlay` protocol, rather than using `MKPolyline`. `BreadcrumbPath` maintains an array of location *crumbs* comprising the user's location history.

When drawing an overlay on the map, MapKit uses the overlay's `boundingMapRect` to determine when the overlay is visible. Because this app can't determine the extent of the path the user might track on the map, `BreadcrumbPath` declares its `boundingMapRect` as `world`. In an app that has well-defined usage patterns, such as a hiking app for use within a national park, the `boundingMapRect` might consist of a large area the path is likely to remain within, such as the bounds of the national park.

When the app adds a new location, `BreadcrumbPath` validates that the information in `CLLocation` is usable for its purposes. Each app that maintains a location history needs to define criteria for ensuring a location update is usable based on the app's specific needs.

```
private func isNewLocationUsable(_ newLocation: CLLocation, breadcrumbData: Breadcru
    /**
     Always check the timestamp of a location value to ensure the location is recent
     location updates, the values that return may reflect cached values while the de
     the accuracy level of the location manager. For some apps, the cached values ma
     a user's travel path, values that are too old may deviate too far from the user
     */
    let now = Date()
    let locationAge = now.timeIntervalSince(newLocation.timestamp)
    guard locationAge < 60 else { return false }

    /**
     An app might keep the first few updates before applying any further filtering,
     locations while waiting for the location accuracy to increase to the requested
     */
    guard breadcrumbData.locations.count > 10 else { return true }

    /**
     Identify locations that shouldn't be part of the breadcrumb data, such as locat
```

```
        Get the distance between this new location and the previous location, and use a
        to determine if keeping the location is useful. For example, a location update
        prior location might represent a user that hasn't moved.

        Your app may apply other criteria. For example, an app tracking a user at walki
        moving at car speeds because that might indicate the user forgot to stop record
        Consider comparing an average value over the last several location updates, suc

        If using the location accuracy properties as criteria for determining a usable
        throw away low-accuracy values by expecting only high-accuracy values. Discardi
        can cause the user's location to appear to jump if the user is moving.
        */
        let minimumDistanceBetweenLocationsInMeters = 10.0
        let previousLocation = breadcrumbData.locations.last!
        let metersApart = newLocation.distance(from: previousLocation)
        return metersApart > minimumDistanceBetweenLocationsInMeters
    }
```

After `BreadcrumbPath` validates that a location is usable, it updates its data structures to include the new location. One of the properties of `BreadcrumbPath` is `pathBounds`, the bounding rectangle containing all of the locations for the overlay. Unlike `boundingMapRect`, `pathBounds` changes its value as the app adds locations to the overlay. A custom overlay renderer reads the value of `pathBounds` from multiple threads concurrently, so `BreadcrumbPath` uses `OSAllocatedUnfairLock` to protect reading of this property from data races.

```
    /// This is a lock protecting the `locations` and `bounds` properties that define th
    private let protectedBreadcrumbData = OSAllocatedUnfairLock(initialState: Breadcrumb

    /**
     This is a rectangle encompassing the breadcrumb path, including a reasonable amount
     when adding a new location to the breadcrumb path that's outside of the existing `p
     */
    var pathBounds: MKMapRect {
        /**
         The app accesses this property from the main thread in `BreadcrumbViewControlle
         background threads running in parallel, through the `canDraw(_:zoomScale)` meth
         Using a lock for access avoids data races.
         */
        return protectedBreadcrumbData.withLock { breadcrumbData in
            return breadcrumbData.bounds
        }
    }
```

Implement a custom overlay renderer to display the path

To implement a custom overlay renderer object that complements the custom overlay data object, the app subclasses `MKOverlayRenderer`. To determine whether an overlay has content to draw, MapKit queries the `canDraw(_:zoomScale:)` method of an overlay renderer. The `BreadcrumbPathRenderer` class in this sample overrides this method to check whether the provided `MKMapRect` intersects the `pathBounds` property of the `BreadcrumbPath` object.

```
override func canDraw(_ mapRect: MKMapRect, zoomScale: MKZoomScale) -> Bool {
    return crumbs.pathBounds.intersects(mapRect)
}
```

A custom overlay renderer also needs to implement the `draw(_:zoomScale:in:)` method to draw the requested section of the overlay. The app draws only the portion of the overlay within the bounds of the provided `MKMapRect` into the provided `CGContext`.

```
override func draw(_ mapRect: MKMapRect, zoomScale: MKZoomScale, in context: CGConte
    /// Scale the width of the line to match the width of a road.
    let lineWidth = MKRoadWidthAtZoomScale(zoomScale)

    /// Outset `mapRect` by the line width to include points just outside of the cur
    let clipRect = mapRect.insetBy(dx: -lineWidth, dy: -lineWidth)

    /**
     Because the system might call this function on multiple background threads simu
     and the `locations` property of the `BreadcrumbPath` updates frequently,
     `locations` needs to guard against data races. See the comments in `BreadcrumbP
     */
    let points = crumbs.locations.map { MKMapPoint($0.coordinate) }
    if let path = pathForPoints(points, mapRect: clipRect, zoomScale: zoomScale) {
        context.addPath(path)
        context.setStrokeColor(UIColor.systemBlue.withAlphaComponent(0.5).cgColor)
        context.setLineJoin(.round)
        context.setLineCap(.round)
        context.setLineWidth(lineWidth)
        context.strokePath()
    }
}
```

Request that the map update the overlay

When the sample app receives a location update, it adds the location to the custom overlay by calling a method that `BreadcrumbPath` defines. This method returns information indicating the result of adding the location to the overlay. When successful, the app calls `setNeedsDisplay(_:)` on the custom overlay renderer to trigger a redraw of the overlay within the region of the map where the user traveled.

```
/**
 If the `BreadcrumbPath` model object determines that the current location moves far
 use the returned updateRect to redraw just the changed area.
*/
let result = breadcrumbs.addLocation(newLocation)

/**
 If the `BreadcrumbPath` model object sucessfully adds the location to the path,
 update the rendering of the path to include the new location.
 */
 if result.locationAdded {
    // Compute the currently visible map zoom scale.
    let currentZoomScale = mapView.bounds.size.width / mapView.visibleMapRect.size.w

    /**
     Find out the line width at this zoom scale and outset the `pathBounds` by that
     This covers situations where the new location is right on the edge of the provi
     is within the bounds.
    */
    let lineWidth = MKRoadWidthAtZoomScale(currentZoomScale)
    var areaToRedisplay = breadcrumbs.pathBounds
    areaToRedisplay = areaToRedisplay.insetBy(dx: -lineWidth, dy: -lineWidth)

    /**
     Tell the overlay view to update just the changed area, including the area that
     Use `setNeedsDisplay(_:)` to only redraw the changed area of a breadcrumb overl
     the changed area includes the entire overlay because if the app was recently in
     that's visible when the app returns to the foreground might change significantl

     In general, avoid calling `setNeedsDisplay()` on the overlay renderer without a
     pass for the entire visible map, only some of which may contain updated data in

     To avoid an expensive operation, call `setNeedsDisplay(_:)` instead of removing
     adding it back to trigger a render pass when the data is changing often. The re
     map is not instantaneous, so removing and adding an overlay may cause a visual
     without the overlay, and then updates it again with the overlay. This is especi
```

```
     one overlay or updating the overlay data often, such as on each location update
    */
    breadcrumbPathRenderer?.setNeedsDisplay(areaToRedisplay)
}
```

See Also

Samples

{} Displaying overlays on a map

Add regions of layered content to a map view.