Version: 2.6.0 – 2.12.0

Pan gesture

A continuous gesture that can recognize a panning (dragging) gesture and track its movement.

The gesture activates when a finger is placed on the screen and moved some initial distance.

Configurations such as a minimum initial distance, specific vertical or horizontal pan detection and number of fingers required for activation (allowing for multifinger swipes) may be specified.

Gesture callback can be used for continuous tracking of the pan gesture. It provides information about the gesture such as its XY translation from the starting point as well as its instantaneous velocity.

Multi touch pan handling

If your app relies on multi touch pan handling this section provides some information how the default behavior differs between the platform and how (if necessary) it can be unified.

The difference in multi touch pan handling lies in the way how translation properties during the event are being calculated. On iOS the default behavior when more than one finger is placed on the screen is to treat this situation as if only one pointer was placed in the center of mass (average position of all the pointers). This applies also to many platform native components that handle touch even if not primarily interested in multi touch interactions like for example UIScrollView component.

The default behavior for native components like scroll view, pager views or drawers is different and hence gesture defaults to that when it comes to pan handling. The difference is that instead of treating the center of mass of all the fingers placed as a leading pointer it takes the latest placed finger as such. This behavior can be changed on Android using `avgTouches` flag.

Note that on both Android and iOS when the additional finger is placed on the screen that translation prop is not affected even though the position of the pointer being tracked might have changed. Therefore it is safe to rely on translation most of the time as it only reflects the movement that happens regardless of how many fingers are placed on the screen and if that number changes

over time. If you wish to track the "center of mass" virtual pointer and account for its changes when the number of finger changes you can use relative or absolute position provided in the event (`x` and `y` or `absoluteX` and `absoluteY`).

Config

Properties specific to `PanGesture`:

`minDistance(value: number)`

Minimum distance the finger (or multiple finger) need to travel before the gesture activates. Expressed in points.

`minPointers(value: number)`

A number of fingers that is required to be placed before gesture can activate. Should be a higher or equal to 0 integer.

`maxPointers(value: number)`

When the given number of fingers is placed on the screen and gesture hasn't yet activated it will fail recognizing the gesture. Should be a higher or equal to 0 integer.

`activateAfterLongPress(duration: number)`

Duration in milliseconds of the `LongPress` gesture before `Pan` is allowed to activate. If the finger is moved during that period, the gesture will fail. Should be a higher or equal to 0 integer. Default value is 0, meaning no `LongPress` is required to activate the `Pan`.

`activeOffsetX(value: number | number[])`

Range along X axis (in points) where fingers travels without activation of gesture. Moving outside of this range implies activation of gesture. Range can be given as an array or a single number. If range is set as an array, first value must be lower or equal to 0, a the second one higher or equal to 0. If only one number `p` is given a range of `(-inf, p)` will be used if `p` is higher or equal to 0 and `(-p, inf)` otherwise.

`activeOffsetY(value: number | number[])`

Range along Y axis (in points) where fingers travels without activation of gesture. Moving outside of this range implies activation of gesture. Range can be given as an array or a single number. If range is set as an array, first value must be lower or equal to 0, a the second one higher or equal to 0. If only one number `p` is given a range of `(-inf, p)` will be used if `p` is higher or equal to 0 and `(-p, inf)` otherwise.

`failOffsetY(value: number | number[])`

When the finger moves outside this range (in points) along Y axis and gesture hasn't yet activated it will fail recognizing the gesture. Range can be given as an array or a single number. If range is set as an array, first value must be lower or equal to 0, a the second one higher or equal to 0. If only one number `p` is given a range of `(-inf, p)` will be used if `p` is higher or equal to 0 and `(-p, inf)` otherwise.

`failOffsetX(value: number | number[])`

When the finger moves outside this range (in points) along X axis and gesture hasn't yet activated it will fail recognizing the gesture. Range can be given as an array or a single number. If range is set as an array, first value must be lower or equal to 0, a the second one higher or equal to 0. If only one number `p` is given a range of `(-inf, p)` will be used if `p` is higher or equal to 0 and `(-p, inf)` otherwise.

`averageTouches(value: boolean)` (Android only)

`enableTrackpadTwoFingerGesture(value: boolean)` (iOS only)

Enables two-finger gestures on supported devices, for example iPads with trackpads. If not enabled the gesture will require click + drag, with enableTrackpadTwoFingerGesture swiping with two fingers will also trigger the gesture.

Properties common to all gestures:

`enabled(value: boolean)`

Indicates whether the given handler should be analyzing stream of touch events or not. When set to `false` we can be sure that the handler's state will **never** become `ACTIVE`. If the value gets updated while the handler already started recognizing a gesture, then the handler's state it will immediately change to `FAILED` or `CANCELLED` (depending on its current state). Default value is `true`.

`shouldCancelWhenOutside(value: boolean)`

When `true` the handler will cancel or fail recognition (depending on its current state) whenever the finger leaves the area of the connected view. Default value of this property is different depending on the handler type. Most handlers' `shouldCancelWhenOutside` property defaults to `false` except for the `LongPressGesture` and `TapGesture` which default to `true`.

`hitSlop(settings)`

This parameter enables control over what part of the connected view area can be used to begin recognizing the gesture. When a negative number is provided the bounds of the view will reduce the area by the given number of points in each of the sides evenly.

Instead you can pass an object to specify how each boundary side should be reduced by providing different number of points for `left`, `right`, `top` or `bottom` sides. You can alternatively provide `horizontal` or `vertical` instead of specifying directly `left`, `right` or `top` and `bottom`. Finally, the object can also take `width` and `height` attributes. When `width` is set it is only allow to specify one of the sides `right` or `left`. Similarly when `height` is provided only `top` or `bottom` can be set. Specifying `width` or `height` is useful if we only want the gesture to activate on the edge of the view. In which case for example we can set `left: 0` and `width: 20` which would make it possible for the gesture to be recognize when started no more than 20 points from the left edge.

IMPORTANT: Note that this parameter is primarily designed to reduce the area where gesture can activate. Hence it is only supported for all the values (except `width` and `height`) to be non positive (0 or lower). Although on Android it is supported for the values to also be positive and therefore allow to expand beyond view bounds but not further than the parent view bounds. To achieve this effect on both platforms you can use React Native's View hitSlop property.

`withRef(ref)`

Sets a ref to the gesture object, allowing for interoperability with the old API.

`withTestId(testID)`

Sets a `testID` property for gesture object, allowing for querying for it in tests.

`cancelsTouchesInView(value)` (iOS only)

Accepts a boolean value. When `true`, the gesture will cancel touches for native UI components (`UIButton`, `UISwitch`, etc) it's attached to when it becomes `ACTIVE`. Default value is `true`.

`runOnJS(value: boolean)`

When `react-native-reanimated` is installed, the callbacks passed to the gestures are automatically workletized and run on the UI thread when called. This option allows for changing this behavior: when `true`, all the callbacks will be run on the JS thread instead of the UI thread, regardless of whether they are worklets or not. Defaults to `false`.

`simultaneousWithExternalGesture(otherGesture1, otherGesture2, ...)`

Adds a gesture that should be recognized simultaneously with this one.

IMPORTANT: Note that this method only marks the relation between gestures, without composing them. `GestureDetector` will not recognize the `otherGestures` and it needs to be added to another detector in order to be recognized.

`requireExternalGestureToFail(otherGesture1, otherGesture2, ...)`

Adds a relation requiring another gesture to fail, before this one can activate.

IMPORTANT: Note that this method only marks the relation between gestures, without composing them.`GestureDetector` will not recognize the `otherGestures` and it needs to be added to another detector in order to be recognized.

Properties common to all continous gestures:

`manualActivation(value: boolead)`

When `true` the handler will not activate by itself even if its activation criteria are met. Instead you can manipulate its state using state manager.

Callbacks

Callbacks common to all gestures:

`onBegin(callback)`

Set the callback that is being called when given gesture handler starts receiving touches. At the moment of this callback the handler is not yet in an active state and we don't know yet if it will recognize the gesture at all.

`onStart(callback)`

Set the callback that is being called when the gesture is recognized by the handler and it transitions to the active state.

`onEnd(callback)`

Set the callback that is being called when the gesture that was recognized by the handler finishes. It will be called only if the handler was previously in the active state.

`onFinalize(callback)`

Set the callback that is being called when the handler finalizes handling gesture - the gesture was recognized and has finished or it failed to recognize.

`onTouchesDown(callback)`

Set the `onTouchesDown` callback which is called every time a finger is placed on the screen.

`onTouchesMove(callback)`

Set the `onTouchesMove` callback which is called every time a finger is moved on the screen.

`onTouchesUp(callback)`

Set the `onTouchesUp` callback which is called every time a finger is lifted from the screen.

`onTouchesCancelled(callback)`

Set the `onTouchesCancelled` callback which is called every time a finger stops being tracked, for example when the gesture finishes.

Callbacks common to all continous gestures:

`onUpdate(callback)`

Set the callback that is being called every time the gesture receives an update while it's active.

`onChange(callback)`

Set the callback that is being called every time the gesture receives an update while it's active. This callback will receive information about change in value in relation to the last received event.

Event data

Event attributes specific to `PanGesture`:

`translationX`

Translation of the pan gesture along X axis accumulated over the time of the gesture. The value is expressed in the point units.

`translationY`

Translation of the pan gesture along Y axis accumulated over the time of the gesture. The value is expressed in the point units.

`velocityX`

Velocity of the pan gesture along the X axis in the current moment. The value is expressed in point units per second.

`velocityY`

Velocity of the pan gesture along the Y axis in the current moment. The value is expressed in point units per second.

`x`

X coordinate of the current position of the pointer (finger or a leading pointer when there are multiple fingers placed) relative to the view attached to the `GestureDetector`. Expressed in point units.

`y`

Y coordinate of the current position of the pointer (finger or a leading pointer when there are multiple fingers placed) relative to the view attached to the `GestureDetector`. Expressed in point units.

`absoluteX`

X coordinate of the current position of the pointer (finger or a leading pointer when there are multiple fingers placed) relative to the window. The value is expressed in point units. It is recommended to use it instead of `x` in cases when the original view can be transformed as an effect of the gesture.

`absoluteY`

Y coordinate of the current position of the pointer (finger or a leading pointer when there are multiple fingers placed) relative to the window. The value is expressed in point units. It is recommended to use it instead of `y` in cases when the original view can be transformed as an effect of the gesture.

Event attributes common to all gestures:

`state`

Current state of the handler. Expressed as one of the constants exported under `State` object by the library.

`numberOfPointers`

Represents the number of pointers (fingers) currently placed on the screen.

Example

```
const END_POSITION = 200;
const onLeft = useSharedValue(true);
const position = useSharedValue(0);

const panGesture = Gesture.Pan()
  .onUpdate((e) => {
    if (onLeft.value) {
      position.value = e.translationX;
    } else {
      position.value = END_POSITION + e.translationX;
    }
  })
  .onEnd((e) => {
    if (position.value > END_POSITION / 2) {
      position.value = withTiming(END_POSITION, { duration: 100 });
      onLeft.value = false;
    } else {
      position.value = withTiming(0, { duration: 100 });
      onLeft.value = true;
    }
  });

const animatedStyle = useAnimatedStyle(() => ({
  transform: [{ translateX: position.value }],
}));

return (
  <GestureDetector gesture={panGesture}>
    <Animated.View style={[styles.box, animatedStyle]} />
  </GestureDetector>
);
```