Version: 2.6.0 – 2.12.0

Composing gestures

Composing gestures is much simpler in RNGH2, you don't need to create a ref for every gesture that depends on another one. Instead you can use `Race`, `Simultaneous` and `Exclusive` methods provided by the `Gesture` object.

Race

Only one of the provided gestures can become active at the same time. The first gesture to become active will cancel the rest of the gestures. It accepts variable number of arguments. It is the equivalent to having more than one gesture handler without defining `simultaneousHandlers` and `waitFor` props.

For example, lets say that you have a component that you want to make draggable but you also want to show additional options on long press. Presumably you would not want the component to move after the long press activates. You can accomplish this using `Race`:

> Note: the `useSharedValue` and `useAnimatedStyle` are part of `react-native-reanimated`.

```
const offset = useSharedValue({ x: 0, y: 0 });
const start = useSharedValue({ x: 0, y: 0 });
const popupPosition = useSharedValue({ x: 0, y: 0 });
const popupAlpha = useSharedValue(0);

const animatedStyles = useAnimatedStyle(() => {
  return {
    transform: [{ translateX: offset.value.x }, { translateY: offset.value.y }],
  };
});

const animatedPopupStyles = useAnimatedStyle(() => {
  return {
    transform: [
      { translateX: popupPosition.value.x },
      { translateY: popupPosition.value.y },
    ],
    opacity: popupAlpha.value,
  };
});
```

```
const dragGesture = Gesture.Pan()
  .onStart((_e) => {
    popupAlpha.value = withTiming(0);
  })
  .onUpdate((e) => {
    offset.value = {
      x: e.translationX + start.value.x,
      y: e.translationY + start.value.y,
    };
  })
  .onEnd(() => {
    start.value = {
      x: offset.value.x,
      y: offset.value.y,
    };
  });

const longPressGesture = Gesture.LongPress().onStart((_event) => {
  popupPosition.value = { x: offset.value.x, y: offset.value.y };
  popupAlpha.value = withTiming(1);
});

const composed = Gesture.Race(dragGesture, longPressGesture);

return (
  <Animated.View>
    <Popup style={animatedPopupStyles} />
    <GestureDetector gesture={composed}>
      <Component style={animatedStyles} />
    </GestureDetector>
  </Animated.View>
);
```

Simultaneous

All of the provided gestures can activate at the same time. Activation of one will not cancel the other. It is the equivalent to having some gesture handlers, each with `simultaneousHandlers` prop set to the other handlers.

For example, if you want to make a gallery app, you might want user to be able to zoom, rotate and pan around photos. You can do it with `Simultaneous`:

> Note: the `useSharedValue` and `useAnimatedStyle` are part of `react-native-reanimated`.

```
  const offset = useSharedValue({ x: 0, y: 0 });
  const start = useSharedValue({ x: 0, y: 0 });
  const scale = useSharedValue(1);
  const savedScale = useSharedValue(1);
  const rotation = useSharedValue(0);
  const savedRotation = useSharedValue(0);

  const animatedStyles = useAnimatedStyle(() => {
    return {
      transform: [
        { translateX: offset.value.x },
        { translateY: offset.value.y },
        { scale: scale.value },
        { rotateZ: `${rotation.value}rad` },
      ],
    };
  });

  const dragGesture = Gesture.Pan()
    .averageTouches(true)
    .onUpdate((e) => {
      offset.value = {
        x: e.translationX + start.value.x,
        y: e.translationY + start.value.y,
      };
    })
    .onEnd(() => {
      start.value = {
        x: offset.value.x,
        y: offset.value.y,
      };
    });

  const zoomGesture = Gesture.Pinch()
    .onUpdate((event) => {
      scale.value = savedScale.value * event.scale;
    })
    .onEnd(() => {
      savedScale.value = scale.value;
    });

  const rotateGesture = Gesture.Rotation()
    .onUpdate((event) => {
      rotation.value = savedRotation.value + event.rotation;
    })
    .onEnd(() => {
      savedRotation.value = rotation.value;
    });

  const composed = Gesture.Simultaneous(
    dragGesture,
    Gesture.Simultaneous(zoomGesture, rotateGesture)
  );

  return (
    <Animated.View>
      <GestureDetector gesture={composed}>
        <Photo style={animatedStyles} />
      </GestureDetector>
    </Animated.View>
  );
```

Exclusive

Only one of the provided gestures can become active, with the first one having a higher priority than the second one (if both gestures are still possible, the second one will wait for the first one to fail before it activates), second one having a higher priority than the third one, and so on. It is equivalent to having some gesture handlers where the second one has the `waitFor` prop set to the first handler, third one has the `waitFor` prop set to the first and the second one, and so on.

For example, if you want to make a component that responds to single tap as well as to a double tap, you can accomplish that using `Exclusive`:

> Note: the `useSharedValue` and `useAnimatedStyle` are part of `react-native-reanimated`.

```
  const singleTap = Gesture.Tap().onEnd((_event, success) => {
    if (success) {
      console.log('single tap!');
    }
  });
  const doubleTap = Gesture.Tap()
    .numberOfTaps(2)
    .onEnd((_event, success) => {
      if (success) {
        console.log('double tap!');
      }
    });

  const taps = Gesture.Exclusive(doubleTap, singleTap);

  return (
    <GestureDetector gesture={taps}>
      <Component />
    </GestureDetector>
  );
```

Composition vs `simultaneousWithExternalGesture` and `requireExternalGestureToFail`

You may have noticed that gesture composition described above requires you to mount all of the composed gestures under a single `GestureDetector`, effectively attaching them to the same underlying component. If you want to make a relation between gestures that are attached to separate `GestureDetectors`, we have a separate mechanism for that: `simultaneousWithExternalGesture` and `requireExternalGestureToFail` methods that are available on every base gesture. They work exactly the same way as `simultaneousHandlers` and `waitFor` on gesture handlers, that is they just mark the relation between the gestures without joining them into single object.